

UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
E DE COMPUTAÇÃO

**Sistema Automatizado de Produção de Cerveja,
Monitorado e Controlado Remotamente**

Autor: Leonardo Graboski Veiga

Orientador: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos

2016

Leonardo Graboski Veiga

Sistema Automatizado de Produção de Cerveja, Monitorado e Controlado Remotamente

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica

ORIENTADOR: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos

2016

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

V426s Veiga, Leonardo Graboski
 Sistema Automatizado de Produção de Cerveja,
 Monitorado e Controlado Remotamente / Leonardo Graboski
 Veiga; orientador Evandro Luis Linhari Rodrigues. São
 Carlos, 2016.

 Monografia (Graduação em Engenharia Elétrica com
 ênfase em Eletrônica) -- Escola de Engenharia de São
 Carlos da Universidade de São Paulo, 2016.

 1. BeagleBone Black. 2. Node.js. 3. brassagem. 4.
 cerveja. 5. automatizado. 6. internet. 7. remoto. 8.
 Linux. I. Título.

FOLHA DE APROVAÇÃO

Nome: Leonardo Graboski Veiga

Título: "Sistema automatizado de produção de cerveja, monitorado e controlado remotamente"

Trabalho de Conclusão de Curso defendido e aprovado

em 14 / 06 / 2016,

com NOTA 9,7 (NOVE, SETE), pela Comissão Julgadora:

Prof. Associado Evandro Luís Linhari Rodrigues - (Orientador - SEL/EESC/USP)

Prof. Dr. José Roberto Boffino de Almeida Monteiro - (SEL/EESC/USP)

Mestre Leonardo Mariano Gomes - (Doutorando - SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Dr. José Carlos de Melo Vieira Júnior

Dedicatória

Dedico este trabalho aos meus pais.

Leonardo Graboski Veiga.

Agradecimentos

Tenho convicção de que quem somos pode ser sintetizado pela soma das interações sociais que acumulamos com o passar dos anos. É por isso que agradeço a todos que passaram por minha vida — cada um de vocês é parte daquilo que hoje sou. Não obstante, alguns nomes em específico devem ser lembrados.

Agradeço à minha família pelo suporte incondicional: aos meus genitores Dirce Graboski e Gustavo Veiga, pela vida e por tudo aquilo que se sucedeu desde então; à minha avó Sofia Karachinski e o meu avô Victor Graboski, que me ensinaram que as virtudes e a inteligência de um ser humano vão muito além do estudo; aos meus padrinhos Luis Fernando Busnardo e Cinderela Busnardo, com os quais me diverti e aprendi um bocado; às minhas tias Inez Graboski e Glaci Edwiges Graboski, que foram minhas segundas mães durante a maior parte da minha vida; aos meus primos e primas André Luiz Graboski, Felipe Busnardo, Ana Paula Busnardo e Maria Fernanda Busnardo e; à esposa do meu pai, Elisabete Veiga.

Agradeço aos meus amigos de Curitiba pelas inúmeras situações que compartilhamos: Albert Faria, Pedro Henrique Bertolin, Douglas Auchewski, Eduardo Henrique Fasolin, Isabela Deolindo, Lucas Lupatini, Pamella Prohmann, Massimo Karly, Moacir Costa Filho, Lucas Vieira e Paulo Jacques Filho, dentre vários outros não menos importantes. Desde os tempos do Santa Maria e, posteriormente da UTFPR, até os dias de hoje, nunca aceitaram ver senão alegria em meus olhos, motivando o meu desenvolvimento técnico e pessoal.

Agradeço aos meus amigos de São Carlos pela caminhada que fizemos juntos ao longo destes anos de faculdade: Rafael Hellman, com quem dividi não somente moradia, mas também as dificuldades da engenharia e muitos momentos de festa entre uma prova e outra; Rafael Oishi, Vitor Martins e Gabriel Moreira, da Rep Hot Chili Peppers, minha segunda casa sancarlense; Renato Nunes Moraes e Renata de Camilo, pelo duradouro companheirismo e; a todos os outros amigos e colegas de classe que compartilharam experiência e conhecimento em prol de atingir um objetivo coletivo almejado por todos, chamado graduação.

Agradeço aos meus professores: em especial ao meu professor orientador Prof. Dr. Evandro Luis Linhari Rodrigues, que me incentivou a sempre buscar o melhor e me apresentou ao incrível mundo do Linux embarcado.

Não menos importantes, agradeço àqueles que contribuíram para o resultado final que é este trabalho de conclusão de curso, dentre os quais o Rui Bertho, responsável pela oficina mecânica do departamento. E a todos os outros que porventura eu tenha esquecido por um lapso de memória, mas não por falta de merecimento.

Leonardo Graboski Veiga.

*"O sucesso é ir de fracasso em fracasso
sem perder o entusiasmo."*

Winston Churchill

Resumo

Este trabalho consistiu na implementação de um sistema elétrico automatizado de produção de cerveja, baseado na plataforma de Linux embarcado BeagleBone Black. Esta foi escolhida para a implementação do controle do sistema e interface gráfica de usuário. Com relação à programação do servidor, foi utilizado o ambiente Node.js. Para a interface entre o sistema embarcado e a parte mecânica foi desenvolvido um circuito com optoacopladores, transistores MOSFET e relés de estado sólido. Foram empregados sensores digitais DS18B20 para monitoramento e realimentação do sistema de controle de temperatura: um controle liga-desliga com quatro níveis de potência de saída (0%, 33%, 66% e 100%). Os elementos atuadores foram dois resistores de 2200W em 110Vac. Como resultados, foram realizados testes da interface gráfica de usuário para a qual foi observado que esta funcionou conforme o esperado. A comprovação do algoritmo de automação foi realizada para uma receita de cerveja desenvolvida para propósito de simulação. O circuito de acionamentos de potência foi comprovado visualmente em bancada, com LEDs ligados às saídas. O funcionamento da estrutura mecânica foi o último teste realizado, somente com água, no qual observou-se que este é capaz de cumprir os requisitos para produção de cerveja. Em síntese, este teste possibilitou a validação de todas as partes do projeto operando em conjunto. Pôde-se concluir que, embora o sistema desenvolvido cumpra às especificações do projeto, possibilitando o controle automático e supervisão de uma brassagem pela internet, diversos fatores não contemplados são essenciais para transformá-lo efetivamente em um produto. Apesar dos pontos de melhorias apontados para trabalhos futuros e, em função dos recursos disponíveis para a elaboração e execução do projeto, este foi considerado um sucesso dados os resultados obtidos.

Palavras-Chave: BeagleBone Black, Node.js, brassagem, cerveja, automatizado, computador de placa única, web, internet, remoto.

Abstract

This work had consisted in the implementation of an automated electric brewing system, based on the BeagleBone Black embedded Linux platform. It was chosen to implement the control system and the graphical user interface. For the server-side programming, the Node.js environment was employed. For the interface between the embedded system and the mechanical structure, a power drive circuit was designed, by using optocouplers, MOSFET transistors and solid-state relays. Digital sensors DS18B20 were employed as monitoring and feedback devices to the temperature control system: an on-off control with four output power levels (0%, 33%, 66% e 100%). The actuators were two 2200W at 110Vac resistors. As results, the graphical user interface was tested and it was noticed that it worked as expected. The automation algorithm validation was tested while running a beer recipe tailored for simulation purposes. The power drive circuit bench test was approved by means of a visual inspection, with LEDs coupled to the outputs. The mechanical structure operation was the last test, with water only, in which it was observed that the system is capable of fulfill the requisites to brew beer. In synthesis, this test allowed for the validation of all the project parts operating together. It was concluded that, even though the developed system comply to the project specification by allowing the automatic brewing online control and supervision via web, there are still many factors not contemplated in this project that are essential to transform it into a product. Although there are many improvements pointed out to future works and, because of the restricted resources available for the project design and execution, it was considered a success given the presented results.

Keywords: BeagleBone Black, Node.js, brewing, beer, automated, single board computer, web, internet, remote.

Lista de Figuras

1.1	Número de cervejarias norte-americanas por categoria.	38
2.1	Programa de temperaturas típico de uma brassagem.	45
2.2	Linha do tempo de uma brassagem simples	46
2.3	Dispositivos de mistura de grãos à brassagem.	47
2.4	Esquema de panela de brassagem industrial moderna.	48
2.5	Esquema de panela de mistura do mosto.	49
2.6	Diferentes configurações de caldeirões de fervura.	51
2.7	Arranjo para recuperação de energia.	53
2.8	Sistemas de clarificação.	54
2.9	Esquema de funcionamento do trocador de calor de placas.	55
3.1	Peças empregadas na construção da estrutura mecânica	65
4.1	Diagrama em blocos do projeto	69
4.2	Representação esquemática da estrutura mecânica	70
4.3	Sistema CIP básico	73
4.4	Diagrama conceitual das conexões mecânicas	75
4.5	Diagrama da panela de mostura e conexões	76
4.6	Detalhes da vedação da panela de mostura	76
4.7	Projeto da estrutura metálica	78
4.8	Dimensões da estrutura de adição de lúpulos	79
4.9	Corpo da estrutura de adição de lúpulos	80
4.10	Tampas da estrutura de adição de lúpulos	80
4.11	Esquema de ligação das varetas à tampa de correr	81
4.12	Estrutura de adição de lúpulos	82
4.13	Estrutura de diretórios da aplicação da BBB	83

4.14	Exemplo de uso do módulo <i>Debug</i>	90
4.15	Resultado de chamada de função sem criação de escopo x com criação de escopo	94
4.16	Representação em fluxograma do algoritmo de implementação da função <i>getTemperatureReading</i>	101
4.17	Representação em fluxograma do algoritmo de implementação da função <i>re-arrange</i>	115
4.18	Representação em fluxograma do algoritmo de implementação da função <i>errorWarningHandler</i>	116
4.19	Módulo do circuito de acionamentos de potência	118
4.20	Módulo do circuito de acionamento do servo-motor	119
4.21	Circuito detector de passagem por zero da senóide da rede	121
5.1	Arquivo CSV com registro de temperaturas gerado em Python	123
5.2	Conformidade da atualização do arquivo de registro da última temperatura capturada pelo script Python	124
5.3	Gráfico com 7200 pontos gerado em Python	125
5.4	Geração de gráfico em Python múltiplas vezes para obter o tempo médio	126
5.5	Geração de gráfico em Python a partir de um volume elevado de dados	126
5.6	Teste de funcionamento do Node.js	128
5.7	Observação do chaveamento de dois pinos de GPIO consecutivos com tempo em nível alto esperado de 10ms	129
5.8	Carga da CPU dedicada ao chaveamento de GPIO em Node.js com tempo de chaveamento de 10ms	130
5.9	Verificação de funcionamento do PWM configurado por meio do Node.js, com $f=50\text{Hz}$ e $duty-cycle=50\%$	131
5.10	Consumo de recursos do computador do cliente para 200 abas abertas no navegador Google Chrome	133
5.11	Excertos relevantes das mensagens de depuração do processo de simulação da brassagem (parte 1)	135
5.12	Excertos relevantes das mensagens de depuração do processo de simulação da brassagem (parte 2)	136

5.13	Momentos relevantes da simulação observados na GUI e nos circuitos de interface de potência	137
5.14	Gráfico de consumo de recursos de CPU e memória RAM da BBB em função do tempo, durante uma simulação de brassagem	139
5.15	Gráfico de consumo de recursos de rede da BBB em função do tempo, durante uma simulação de brassagem	139
5.16	Página inicial da GUI	142
5.17	Página de apresentação do trabalho	142
5.18	Página de configuração de data/hora	143
5.19	Tentativa de configuração de data/hora negada	143
5.20	Tentativa de configuração de data/hora aceita	144
5.21	Gráfico dinâmico de temperatura em função do tempo	145
5.22	Aplicação do filtro de médias móveis com diferentes intensidades	145
5.23	Interface gráfica para configuração de início da brassagem	146
5.24	Interface gráfica do painel de controle do sistema mecânico	146
5.25	Gerenciador de receitas da UI	147
5.26	Gráfico de consumo de recursos de CPU e memória RAM da BBB em função do tempo, durante estresse do gerenciador de receitas	148
5.27	Teste de criação de receita com nome inválido	149
5.28	Editor de receitas	150
5.29	Análise de execução do código Javascript da interface de edição de receitas	150
5.30	Implementação em <i>protoboard</i> dos circuitos de interface de potência, servomotor e detector de passagem por zero	153
5.31	Resultado da simulação do módulo de acionamentos de potência com carga simulando a bomba de recirculação	154
5.32	Resultado da simulação do módulo de acionamentos de potência com carga simulando uma válvula solenóide	154
5.33	Resultado da simulação do módulo de acionamentos de potência com carga simulando o SSR	155
5.34	Ensaio do módulo de acionamentos de potência com $R_{porta} = 10k\Omega$	155
5.35	Ensaio do módulo de acionamentos de potência com $R_{porta} = 22k\Omega$	156
5.36	Ensaio do módulo de acionamentos de potência com SSR ligado à saída	156

5.37	Ensaio do módulo de acionamentos de potência com válvula solenóide ligada à saída	157
5.38	Implementação do módulo de acionamentos de potência em placa de protótipo	158
5.39	Resposta ao degrau do acionamento da válvula solenóide	158
5.40	Resposta ao degrau do acionamento da válvula solenóide, em close	159
5.41	Instabilidade da frequência da rede em função do tempo	160
5.42	Acionamento do SSR em diferentes instantes da senóide da rede	160
5.43	Resultado da simulação do módulo de acionamentos de potência com carga simulando o servo-motor	161
5.44	Comportamento da corrente de alimentação do servo-motor durante operação sem carga	162
5.45	Comportamento da corrente de alimentação do servo-motor durante operação com carga	162
5.46	Teste do servo-motor com carga	163
5.47	Comportamento da corrente de alimentação do servo-motor ao aplicar um torque ao seu eixo	163
5.48	Descrição da simulação do circuito detector de <i>zero-crossing</i>	164
5.49	Comportamento do circuito detector de <i>zero-crossing</i> para diferentes valores de atenuação da senóide retificada	164
5.50	Verificação do comportamento do circuito detector de <i>zero-crossing</i> para diferentes valores de R2	165
5.51	Simulação da saída do circuito detector de <i>zero-crossing</i> para diferentes valores de tensão de alimentação	166
5.52	Curvas de tensão, corrente e potência sobre o resistor de potência do circuito detector de <i>zero-crossing</i>	166
5.53	Formas de onda de tensão sobre a base (verde) e o coletor (amarelo) do transistor de detecção de passagem por zero	167
5.54	Entrada e saída do circuito detector de <i>zero-crossing</i> para diferentes valores de tensão de alimentação	168
5.55	Forma de onda de tensão sobre o resistor de potência e a saída do optoacoplador do circuito detector de <i>zero-crossing</i>	168
5.56	Processo de rosqueamento e vedação da tubulação e elementos relacionados .	170
5.57	Paneles e tubos conectados repousando sobre a estrutura de suporte	171

5.58	Detalhe de conexão das painelas com a tubulação	171
5.59	Acomodação do sistema embarcado e acionamentos de potência à estrutura de suporte	172
5.60	Detalhes de ligações elétricas dos elementos de potência	172
5.61	Estrutura mecânica finalizada e pronta para testes	173
5.62	Testes do sistema mecânico com controle pela GUI	174
5.63	Simulação de produção para teste da parte mecânica	175
A.1	Diagrama de blocos do subsistema PRU-ICSS	186
A.2	Pinos da BeagleBone Black acessíveis pela PRU-ICSS	187
A.3	Fluxo de desenvolvimento de software da PRU	190
B.1	Arquitetura interna do DS18B20	192
B.2	Mapa de memória do DS18B20	193
B.3	Registradores de temperatura do DS18B20	193
C.1	BeagleBone Black e principais componentes	196
C.2	Diagrama de blocos de alto nível da BeagleBone Black	198
C.3	Configuração padrão dos pinos da BeagleBone Black	198
C.4	Últimas imagens pré-compiladas disponíveis para a BBB	200
C.5	Configuração do <i>Putty</i> para acesso SSH via USB	201
C.6	Tentativa de login bem sucedida pelo <i>Putty</i>	201
C.7	Acesso à BBB via SSH	202
C.8	Estrutura do serviço de data e hora do Ntp.br	204
C.9	Leitura do sensor DS18B20 pelo terminal	205
D.1	Ambiente completo do IDE <i>Cloud9</i>	209
D.2	Criação de repositório no <i>GitHub</i>	210
D.3	Fazendo git commit após mudanças incrementais	211
D.4	Ramo <i>master</i> do projeto no github	212
D.5	Registro de contribuições para o projeto na plataforma <i>GitHub</i>	213
D.6	Gráfico das contribuições para o projeto na plataforma <i>GitHub</i> em função do tempo	213
D.7	Distribuição das contribuições para o projeto no <i>GitHub</i> ao longo da semana	214
I.1	Servo-motor desmontado	273

I.2	Circuito de caracterização do servo-motor	274
I.3	Forma de onda de acionamento do servo-motor	275
I.4	Limites de excursão do servo-motor	276
I.5	Simulação para obtenção do valor mínimo de incremento Δt	277
I.6	Gráfico de Δt em função de <i>ciclo</i>	277
K.1	Histograma de distribuição do tempo de amostragem de temperatura	286

Lista de Tabelas

2.1	Características de diferentes sistemas de fervura.	52
2.2	Descrição dos campos do registrador de controle de <i>pads</i>	58
3.1	Lista de materiais que compõem o núcleo do projeto	63
3.2	Lista de materiais utilizados na confecção do subsistema mecânico	64
3.3	Lista de componentes eletrônicos diversos empregados na confecção de PCBs e cabeamento do sistema	66
3.4	Lista de equipamentos, instrumentos e <i>softwares</i> de suporte ao desenvolvi- mento do projeto	67
4.1	Dimensões elementos mecânicos relevantes para o projeto da altura da estru- tura metálica	77
4.2	Lista de diretórios e arquivos da aplicação da BBB	84
4.3	Correspondência entre o valor numérico e o significado dos códigos usados para registro da brassagem	104
4.4	Parâmetros de receita de cerveja utilizados para <i>debug</i>	109
4.5	Padrão de data/hora no formato ISO 8601	111
5.1	Estatísticas referentes ao tempo de geração de gráfico na BBB, para 7200 pontos em Python	125
5.2	Estatísticas referentes ao tempo de um único chaveamento de GPIO usando o módulo <i>gpio_config</i> em Node.js	129
5.3	Estatísticas referentes ao chaveamento de GPIO usando o módulo <i>gpio_config</i> em Node.js	130
5.4	Consumo de recursos de memória e CPU da aplicação em função do número de clientes conectados à BBB	132

5.5	Apresentação dos campos <i>código</i> , <i>msg</i> e <i>timestamps:curr</i>) convertida de <i>epoch time</i> para data/hora, das situações notáveis e importantes da brassagem	134
5.6	Consumo de recursos de memória e processamento da BBB ao longo do processo de brassagem	138
5.7	Instantes notáveis do estresse do gerenciador de receitas	148
5.8	Tempos relativos a processamento do editor de receitas, no cliente	151
5.9	Tensões de operação da válvula solenóide	157
5.10	Largura do pulso que indica passagem por zero em função do valor de R2 . .	165
5.11	Tempos de drenagem e bombeamento do sistema mecânico para 20 litros de água	175
A.1	Mapa de Memória Local da PRU	188
A.2	Mapa de Memória Global da PRU	189
C.1	Especificações Gerais da BeagleBone Black	197
I.1	Limites de temporização do servo-motor TowerPro MG995	278
K.1	Estatísticas referentes ao tempo de amostragem de temperatura na BBB, para 500 pontos em Python	285
II.1	Registradores do módulo de controle	303

Lista de Códigos-fonte

2.1	Sobreposição de <i>device tree</i>	57
2.2	Implementação incorreta de código em Node.js	60
2.3	Implementação de código em Node.js orientada a callback	60
2.4	Instalação de pacote usando o NPM	61
2.5	Descrição e dependências de um projeto usando NPM	61
2.6	Instalação de dependências usando o NPM	61
4.1	Função de <i>log</i> da temperatura em Python	86
4.2	Script para gravação das leituras de temperatura em Python	86
4.3	Instalação da biblioteca Matplotlib	87
4.4	Instalação do ecossistema SciPy	87
4.5	Importação das bibliotecas para geração de gráfico em Python	87
4.6	Instalação e/ou atualização do Node.js	88
4.7	Criação do diretório <i>/var/www/node_modules</i>	89
4.8	Instalação do módulo <i>Debug</i>	89
4.9	Exemplo de uso do módulo <i>Debug</i>	89
4.10	Instalação do Octalbonescript	91
4.11	Modificações feitas ao template de <i>device tree</i> do Octalbonescript	91
4.12	Uso de clausura para criação de escopo	93
4.13	Instalação dos módulos necessários para implementar o servidor web em Node.js	94
4.14	Configuração do PHP-express	95
4.15	Configuração do servidor web com o <i>framework</i> Express.js	95
4.16	Exemplo de uso dos argumentos <i>req</i> e <i>res</i> definidos pelo Express.js	97
4.17	Declaração do objeto <i>environmentVariables</i>	102
4.18	Uso da função <i>JSON.stringify</i> para codificar um objeto em string JSON	103

4.19	Função que passa a URL da página atual para um módulo em PHP	110
4.20	Instalação da biblioteca D3.js e do framework Rickshaw	112
C.1	Passos para configuração do Wi-Fi	202
C.2	Configuração para reset do Wi-Fi após o boot	202
C.3	Instalação e configuração do NTP	203
C.4	Servidores brasileiros do NTP	204
C.5	Instalação do DTC	205
C.6	Fragmento de device tree para o DS18B20	205
C.7	Compilação de <i>device tree</i>	205
C.8	Desabilitando servidor web Apache	206
D.1	Primeira tentativa de instalação do <i>Cloud9 IDE core</i> na BBB	207
D.2	Instalação bem sucedida do <i>Cloud9 IDE core</i> na BBB	208
D.3	Execução do <i>Cloud9</i>	208
D.4	Instalação do <i>Git</i> e cópia do repositório do <i>GitHub</i> para a BBB	210
D.5	Primeiro <i>git commit</i>	211
E.1	Módulo com as funções para <i>log</i> da temperatura	217
E.2	Script para plotagem de gráfico	219
F.1	Código-fonte raiz da aplicação em Node.js	221
F.2	Módulo de gerenciamento de GPIO e PWM	222
F.3	Módulo de roteamento do servidor web	224
F.4	Módulo de registros e verificações em geral	226
F.5	Código-fonte raiz da aplicação em Node.js	229
G.1	Módulo PHP que cria a barra de navegação da interface web	235
G.2	Função que passa a URL da página atual para um módulo em PHP	236
G.3	Página inicial da interface web	236
G.4	Página <i>sobre</i> da interface web	236
G.5	Página de estatísticas	237
G.6	Gráfico dinâmico de temperatura	238
G.7	Página gerenciadora de receitas da interface web	242
G.8	Funções auxiliares do gerenciador de receitas	244
G.9	Código HTML do editor de receitas	244
G.10	Código PHP do editor de receitas	248
G.11	Código javascript do editor de receitas	254

G.12	Gerenciador de brassagem	255
G.13	Painel de controle de brassagem	257
G.14	CSS do formulário	260
G.15	CSS específico para alguns botões	262
G.16	CSS do template da aplicação web	262
H.1	/etc/network/interfaces	271
H.2	w1.dts	272
I.1	Código-fonte de caracterização do funcionamento do servo-motor com AT89S52278	

Siglas

ABV	<i>Alcohol by Volume</i> - Porcentagem do volume de álcool na cerveja
API	<i>Application Programming Interface</i> - Interface de Programação de Aplicação
BBB	BeagleBone Black
BJT	<i>Bipolar Junction Transistor</i> - Transistor Bipolar de Junção
BK	<i>Brewing Kettle</i> - Panela de Fervura
CI	Circuito Integrado - equivalente a IC (<i>Integrated Circuit</i>) e muitas vezes referido como <i>chip</i> .
CIP	<i>Clean in Process</i> - Limpeza no Local, é a limpeza automatizada dos equipamentos
CRC	<i>Cyclic Redundancy Check</i> - Verificação de Redundância Cíclica
DDP	Diferença de potencial
DT	<i>Device Tree</i> - Uma estrutura de dados em árvore usada para descrever <i>hardware</i>
DTC	<i>Device Tree Compiler</i> - Compilador de <i>Device Tree</i>
FET	<i>Field Effect Transistor</i> - Transistor de Efeito de Campo
GMT	<i>Greenwich Mean Time</i> - Hora Média de Greenwich
GPIO	<i>General Purpose Input/Output</i> - Entrada/Saída de Propósito Geral
GUI	<i>Graphical User Interface</i> - Interface Gráfica de Usuário
HID	<i>Human Interface Device</i>
HLT	<i>Hot Liquor Tank</i> - Tanque de Água Quente
IBU	<i>International Bitterness Unit</i> - Unidade de Amargor Internacional
IDE	<i>Integrated Development Environment</i> - Ambiente de Desenvolvimento Integrado
IEP	<i>Industrial Ethernet Peripheral</i> - Periférico Ethernet Industrial
IoT	<i>Internet of Things</i> - Internet das Coisas
LSB	<i>Least Significant Bit</i> - Bit Menos Significativo
MOSFET	<i>Metal-Oxide Semiconductor Field Effect Transistor</i> - Transistor de Efeito de Campo Metal-Óxido Semicondutor

LT	<i>Lauter Tun</i> - Panela de Filtragem
MSB	<i>Most Significant Bit</i> - Bit Mais Significativo
MT	<i>Mash Tun</i> - Panela de Mostura
NPM	<i>Node Package Manager</i> - Gerenciador de Pacotes do Node
OG	<i>Original Gravity</i> - Gravidade Original. É a densidade do mosto após o cozimento dos grãos
PID	<i>Proportional-Integral-Derivative-Controller</i> - Controlador Proporcional Integral Derivativo
PCB	<i>Printed Circuit Board</i> - Placa de Circuito Impresso. Embora PCI seja a abreviação em português, o termo PCB foi escolhido para uso no intuito de não confundir o leitor com o barramento de computadores <i>Peripheral Component Interconnect</i>
REST	<i>Representational State Transfer</i> - Transferência de Estado Representacional
RISC	<i>Reduced Instruction Set Computer</i> - Computador com um conjunto reduzido de instruções
SaaS	<i>Software as a Service</i> - Software como um Serviço
SO	<i>Sistema Operacional</i> - também conhecido como OS (<i>Operating System</i>)
SoC	<i>System on Chip</i> - Sistema em um Chip
SBC	<i>Single Board Computer</i> - Computador de Placa Única
SDK	<i>Software Development Kit</i> - Kit de Desenvolvimento de Software
SSH	<i>Secure Shell</i> - Protocolo de comunicação entre computadores
UI	<i>User Interface</i> - Interface de Usuário
UTC	<i>Universal Time Coordinated</i> - Tempo Universal Coordenado
VCS	<i>Version Control System</i> - Sistema de Controle de Versão

Sumário

1	Introdução	37
1.1	Motivação	37
1.2	Objetivos	39
1.3	Justificativas/relevância	40
1.4	Organização do Trabalho	40
2	Embasamento Teórico	43
2.1	Processo de Produção de Cerveja	43
2.2	Estruturas mecânicas de produção de cerveja	47
2.2.1	Panela de mostura	48
2.2.2	Caldeirão de fervura	50
2.2.3	Adição de lúpulos	53
2.2.4	Clarificação e resfriamento do mosto	53
2.3	<i>Device-tree</i>	56
2.4	Programação <i>server-side</i>	58
2.4.1	Node.js	59
2.4.2	<i>Node Package Manager - NPM</i>	60
2.4.3	<i>MEAN Stack</i>	61
2.4.4	Servidor Express.js	62
3	Materiais	63
3.1	Núcleo do projeto	63
3.2	Sistema mecânico	64
3.3	Componentes eletrônicos	66
3.4	Equipamentos, instrumentos, <i>softwares</i> e miscelânea	67

4	Métodos	69
4.1	Estrutura mecânica	69
4.1.1	Funcionamento da estrutura	70
4.1.2	Dimensionamento da parte funcional	73
4.1.3	Estrutura metálica de suporte	77
4.2	Estrutura de adição de lúpulos	78
4.2.1	Controle do servo-motor	82
4.3	Organização do diretório da aplicação	83
4.4	IDE e sistema de controle de versão	85
4.5	Geração de gráfico e registro de temperatura em Python	85
4.5.1	Registro de temperatura	85
4.5.2	Gráfico de temperatura	86
4.6	Aplicação <i>server-side</i> em Node.js	88
4.6.1	Acesso a GPIO e PWM	90
4.6.2	Servidor Express.js	94
4.6.3	Registros e verificações em geral	98
4.6.4	Controle do processo de brassagem	104
4.6.5	Simulação do controle do processo de brassagem	108
4.7	Interface de usuário	109
4.7.1	Aspectos gerais da GUI	110
4.7.2	Boas vindas e informações do projeto	111
4.7.3	Configurações gerais da BBB	111
4.7.4	Estatísticas e gráfico dinâmico	112
4.7.5	Gerenciador de receitas	113
4.7.6	Editor de receitas	114
4.7.7	Gerenciador de brassagem	116
4.7.8	Painel de controle de brassagem	117
4.8	Circuitos de interface entre a BBB e sensores/atuadores	117
4.8.1	Acionamentos de potência	117
4.8.2	Acionamentos de potência	119
4.8.3	Detector de <i>zero-crossing</i>	120
5	Resultados e Discussões	123

5.1	Geração de gráfico e registro de temperatura em Python	123
5.1.1	Registro de temperatura	123
5.1.2	Gráfico de temperatura	124
5.2	Aplicação <i>server-side</i> em Node.js	128
5.2.1	Acesso a GPIO e PWM	128
5.2.2	Servidor Express	131
5.2.3	Registros e verificações em geral	133
5.2.4	Simulação do controle do processo de brassagem	134
5.3	Interface de usuário	141
5.3.1	Página inicial	141
5.3.2	Página de apresentação	142
5.3.3	Interface de configurações	143
5.3.4	Página de estatísticas	144
5.3.5	Interface de brassagem	146
5.3.6	Gerenciador e editor de receitas	147
5.4	Circuitos de interface entre a BBB e sensores/atuadores	153
5.4.1	Acionamentos de potência	153
5.4.2	Servo-motor	161
5.4.3	Detector de <i>zero-crossing</i>	164
5.5	Estrutura mecânica	169
5.5.1	Montagem da mecânica e interface com o sistema embarcado	169
5.5.2	Integração entre GUI, sistema embarcado, algortimo de automação e sistema mecânico	173
6	Conclusões	177
6.1	Considerações gerais	178
6.2	Trabalhos futuros	178
A	<i>Programmable Real-time Unit – PRU-ICSS</i>	185
B	Sensor de temperatura DS18B20	191
C	BeagleBone Black	195
C.0.1	<i>Programmable Real-time Unit – PRU-ICSS</i>	198
C.0.2	Configuração da BeagleBone Black	199

C.0.3	Ajustes de rede para uso do adaptador Wi-Fi/USB	201
C.0.4	Data/hora	203
C.0.5	Sensor DS18B20	204
C.0.6	Webserver Apache	206
D	IDE e sistema de controle de versão	207
D.1	<i>Cloud9</i>	207
D.2	<i>Git/GitHub</i>	209
D.3	Considerações finais sobre o uso das ferramentas Git e GitHub	211
E	Scripts Python	217
E.1	Módulo com as funções para <i>log</i> da temperatura	217
E.2	Script para plotagem de gráfico	219
F	Códigos-fonte Node.js	221
F.1	Código-fonte raiz da aplicação em Node.js	221
F.2	Módulo de gerenciamento de GPIO e PWM	222
F.3	Módulo de roteamento do servidor web	224
F.4	Módulo de registros e verificações em geral	226
F.5	Módulo de controle do processo de brassagem	229
G	Códigos-fonte da aplicação web	235
G.1	Módulo PHP que cria a barra de navegação da interface web	235
G.2	Função que passa a URL da página atual para um módulo em PHP	236
G.3	Página inicial	236
G.4	Página <i>sobre</i>	236
G.5	Página de estatísticas	237
G.6	Gráfico dinâmico de temperatura	238
G.7	Página gerenciadora de receitas da interface web	242
G.8	Funções auxiliares do gerenciador de receitas	244
G.9	Código HTML do editor de receitas	244
G.10	Código PHP do editor de receitas	248
G.11	Código javascript do editor de receitas	254
G.12	Gerenciador de brassagem	255
G.13	Painel de controle de brassagem	257

G.14	CSS do formulário	260
G.15	CSS específico para alguns botões	262
G.16	CSS do template da aplicação web	262
H	Arquivos de configuração do sistema	271
H.1	Arquivo de configuração de rede	271
H.2	Device Tree para o DS18B20	272
I	Caracterização do funcionamento do servo-motor com AT89S52	273
I.1	Simulação e ensaio em bancada	275
I.2	Código-fonte	278
J	Cálculo de OG e IBU de uma receita de cerveja	281
J.1	Estimação do IBU para uma receita de 20l	282
K	Análise do tempo de leitura do sensor DS18B20 em Python	285
L	Diagrama esquemático do circuito de acionamentos de potência	287
I	Diagrama de conexões elétricas da BeagleBone Black	291
II	Registradores do Módulo de Controle	303
III	Válvula Danfoss EV210BD 032U3620	309
IV	Bobina Danfoss 042N7550	317
V	Bomba centrífuga Topsflo B08H121006	321
VI	Transistor IRF540N	323
VII	Servo-motor TowerPro MG995	327

Capítulo 1

Introdução

O objetivo deste Trabalho de Conclusão de Curso é o projeto de um sistema elétrico automatizado de produção de cerveja, monitorado e controlado remotamente. A plataforma de Linux embarcado BeagleBone Black foi escolhida para ser o núcleo de processamento de dados, cuja linguagem de programação escolhida foi o Javascript em conjunto com o interpretador Node.js; comunicação via *web* desenvolvida em HTML, CSS, Javascript e PHP e; controle do processo, de tal modo que foi necessário projetar circuitos de interface entre a placa e os sensores e atuadores do sistema.

1.1 Motivação

Há algumas décadas começou nos Estados Unidos da América um movimento espontâneo no qual diversos cidadãos norte-americanos começaram a produzir cervejas por conta própria. Estes foram denominados *homebrewers*, ou cervejeiros caseiros, e eles criaram um mercado de insumos e equipamentos para brassar - ou seja, produzir cerveja - em pequenas quantidades e de forma artesanal, utilizando-se de processos e ferramentas improvisados para tal [1].

Apesar das ferramentas de produção rudimentares empregadas no início do movimento de produção caseira de cerveja — e que são utilizadas até hoje — percebeu-se que a qualidade do resultado final podia ser tão boa quanto a de cervejas comerciais de alta qualidade. O movimento cervejeiro cresceu e a partir dele nasceram diversas microcervejarias: segundo dados da *Brewers Association*, em 1994 existiam 192 microcervejarias e 329 *brewpubs* (bares que produzem a própria cerveja) nos Estados Unidos, enquanto em 2014 este número cresceu para 1871 microcervejarias e 1412 *brewpubs* [2], indicando uma tendência de mercado no que diz respeito às cervejas artesanais: a figura 1.1 ilustra os números que apoiam esta tendência.

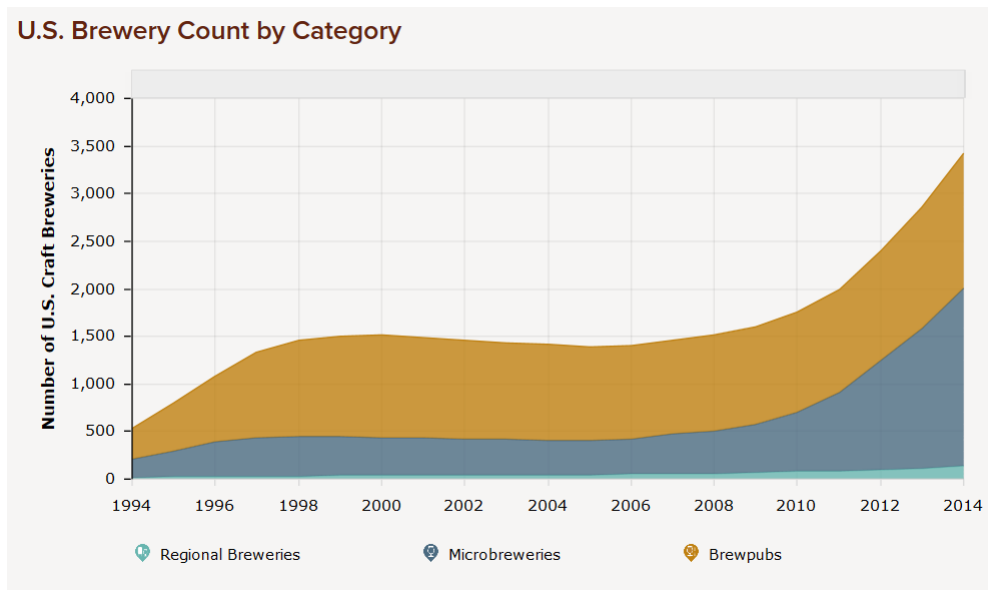


Figura 1.1: Número de cervejarias norte-americanas por categoria.
 Fonte: Site da Brewers Association¹

No Brasil, o movimento cervejeiro artesanal caseiro é considerado por algumas fontes um mercado incipiente, porém com bom ritmo de crescimento [3], devido em grande parte à tendência de mercado em relação às microcervejarias. Desde 1995, com a fundação da cervejaria Dado Bier, no estado do Rio Grande do Sul, que alega ser a primeira microcervejaria do Brasil [4], seguida pela cervejaria Colorado, do estado de São Paulo, fundada em 1996 [5], o movimento cervejeiro cresceu e, a partir do início da década de 2010 se popularizou, com estimativa de que existam atualmente cerca de 200 microcervejarias no país [3]. Esta busca por diferenciação do consumo é refletida também no aumento do coeficiente de penetração de importações neste setor, que indica a quantidade percentual de importações: este cresceu de 0,02% em 2005 para 0,21% em 2011 [3]. Cabe ressaltar que a indústria da cerveja no Brasil, como um todo, corresponde a 2% do PIB do país, representando um faturamento anual de R\$70 bilhões [6], portanto o mercado de artesanais, que representa cerca de 1% do total, corresponde a um faturamento de R\$700 milhões, com prognóstico de crescimento para 2% em até dez anos [7].

Além disto, existem sistemas de produção automática de cerveja que são comercializados em outros países, a exemplo do PicoBrew [8], dos Estados Unidos, cuja ideia é similar à de uma cafeteira automática e do Williams Warn [9], da Nova Zelândia, que é um projeto compacto mais parecido com o equipamento de um produtor caseiro, apesar de completamente automatizado.

¹Disponível em <https://www.brewersassociation.org/statistics/number-of-breweries/>

A motivação para este trabalho, portanto, surgiu a partir da constatação de que o mercado de cervejas artesanais é crescente e rentável, tanto no que diz respeito a fornecer equipamentos para produtores caseiros, quanto ao fornecimento de equipamentos e tecnologia para micro e pequenas cervejarias. Outro ponto que cabe ressaltar é que a maioria das indústrias nacionais fornecedoras de equipamentos para o mercado cervejeiro – dentre elas Mec Bier Microcervejarias, Bierking Equipamentos, Biermatik Comercial Ltda., Cervejando.com e Mybeer, dentre outros – oferece, na melhor das hipóteses, soluções semi-automatizadas para operação presencial, a despeito das soluções automáticas encontradas em outros países, o que motivou a automação mais complexa e remota deste trabalho. A motivação final deste projeto não é a criação de um produto, mas sim a compreensão e aplicação de tecnologias à área de automação do processo de produção de cerveja.

1.2 Objetivos

A descrição do objetivo do projeto é: um sistema de Linux embarcado que automatiza o processo de produção de cerveja desde o cozimento do mosto até seu resfriamento. Com relação ao controle e monitoramento, no sistema embarcado deve rodar um servidor web em conjunto com um algoritmo de controle automático da brassagem, que forneça ao operador do sistema uma interface web para gerenciamento de receitas e controle e monitoramento do sistema. O projeto da mecânica do sistema também deve ser contemplado, focando em uma configuração que permita a automação proposta.

Os objetivos gerais e específicos do projeto foram:

- Implantação de distribuição Debian do Linux no sistema BeagleBone Black
- Interfaceamento entre o sensor DS18B20 e o sistema BeagleBone Black
- Projeto, aquisição dos componentes e montagem do sistema mecânico
- Projeto do acionamento dos resistores de potência
- Projeto e implementação do programa de controle de acionamento de válvulas e bombas, e adição de ingredientes
- Desenvolvimento da interface web do sistema
- Teste e validação do funcionamento do sistema completo

1.3 Justificativas/relevância

Este trabalho objetivou a aplicação de conhecimentos e técnicas – além da busca pelo estado da arte – nas áreas de sistemas de controle, circuitos elétricos e eletrônicos, instrumentação, microcontroladores e sistemas digitais. Conhecimentos estes obtidos durante o curso de graduação em Engenharia Elétrica - Ênfase em Eletrônica.

Outro foco foi o aprendizado de novos conhecimentos, majoritariamente nas áreas de Linux embarcado, que compreende computadores em módulo e *single board computers* ou computadores de placa única (SBC), desenvolvimento de aplicações *web* e engenharia mecânica.

Este trabalho foi importante uma vez que se propôs a resolver problemas e criar métodos de automação para uma indústria representativa no mercado nacional, além de empregar tecnologias de ponta da área de sistemas embarcados a uma aplicação, fomentando o seu uso no Brasil. Do ponto de vista de criação de um produto, a contribuição deixada foi o incentivo da população ao **consumo consciente** de cervejas, por meio do fomento da cultura cervejeira — bandeira levantada pelas microcervejarias artesanais nacionais, com o lema "beba menos, beba melhor", e que é de interesse da saúde pública.

1.4 Organização do Trabalho

Este trabalho está distribuído em 6 capítulos, incluindo esta introdução, dispostos conforme a descrição que segue:

Capítulo 2: Descreve brevemente a teoria que foi consultada para o desenvolvimento deste projeto, desde o processo de produção de cerveja e o equipamento empregado com tal objetivo, passando pela teoria de *device tree* e programação em Node.js (Javascript) até uma breve introdução aos elementos semicondutores diodo e transistor, finalizando com o sensor de temperatura digital DS18B20.

Capítulo 3: Apresenta os materiais empregados no desenvolvimento deste trabalho de conclusão de curso, separados por categorias lógicas, como sistema mecânico e componentes eletrônicos. Uma seção dedicada à BeagleBone Black apresenta as suas características e configuração.

Capítulo 4: Discorre sobre os procedimentos, processos e métodos elaborados para atingir os objetivos propostos para este trabalho. Detalha o projeto das estruturas mecânica e de adição de lúpulos; descreve a organização dos diretórios da aplicação; explica o uso da IDE

Cloud9 e do sistema de controle de versão *Git* em conjunto com o *GitHub*; apresenta o desenvolvimento de script em Python e da aplicação *server-side* em Node.js; discorre sobre o planejamento e implementação da interface gráfica de usuário e; apresenta o projeto dos circuitos de interface do sistema embarcado com os elementos de potência do sistema.

Capítulo 5: Compila os resultados obtidos em todas as frentes deste trabalho: uso da ferramenta de controle de versão; resultados dos scripts desenvolvidos em Python; resultados e desempenho da aplicação *server-side* desenvolvida em Node.js; resultados de simulação e ensaios em bancada dos circuitos de interface e; resultados da montagem da estrutura mecânica, que foi colocado como último tópico dos resultados pois seus testes dependeram do sucesso das outras frentes de trabalho.

Capítulo 6: Conclui com um olhar crítico os diversos aspectos do projeto, com base nos resultados documentados no capítulo anterior. É dividido em várias seções para os tópicos dos resultados e também apresenta uma seção para considerações gerais e outra para trabalhos futuros.

Capítulo 2

Embasamento Teórico

Neste capítulo, será apresentado o embasamento teórico utilizado para o desenvolvimento deste Trabalho de Conclusão de Curso, abordando desde o processo de produção de cerveja e os equipamentos utilizados até os sistemas de controle e plataforma de implementação da interface de usuário, dentre outros tópicos.

2.1 Processo de Produção de Cerveja

O primeiro passo para o desenvolvimento do controle de automatização da produção de cerveja foi o entendimento dos seus processos e particularidades. Esta seção tem o objetivo de apresentar uma introdução à fabricação de cerveja, o que implica em significativas simplificações e resumos dos temas aqui percorridos. Para maiores informações, recomenda-se a consulta do material bibliográfico de referência.

A produção começa com a escolha e **moagem** dos maltes, produzidos a partir de cereais selecionados, sendo a cevada o cereal mais amplamente utilizado – embora outros grãos, como trigo, centeio e aveia também possam ser empregados [10]. O grão malteado é a fonte principal de açúcares fermentáveis utilizado na fabricação de cerveja e é produzido a partir da germinação parcial do cereal, sendo que diferentes processos e variedades de grãos resultam em diferentes qualidades de maltes. Estes, por sua vez, conferem características únicas aos diversos estilos de cerveja [1, 10]. O método de moagem do malte é escolhido com base nos métodos de brassagem e separação de mosto a serem empregados [10], e.g. quando a cama de grãos formada no processo serve como um filtro para separação do mosto, deve-se preservar a casca, utilizando uma moagem grossa e de maceração, evitando trituração.

Com os maltes moídos, segue-se para a **brassagem**, que consiste na mistura de deter-

minada quantidade de água quente aos maltes, possibilitando a extração e diluição dos seus açúcares fermentáveis. Enquanto um extrato em água fria tem rendimento na ordem de 15-22%, o *hot water extract* (HWE) – extrato em água quente, chega à ordem de 75-83% devido à atividade enzimática catalisadora [10].

O processo de brassagem pode ser realizado de várias maneiras, dentre elas [10]:

- (a) infusão simples, na qual os maltes são cozidos a uma temperatura fixa, quase isotérmica, utilizado tradicionalmente por cervejarias britânicas. É feita em uma tina de brassagem, na qual tanto o processo de extração dos açúcares quanto a separação do extrato do mosto são realizados. O cozimento se dá a uma temperatura na ordem de 63°C a 67°C, durante um período de 30 minutos a duas horas e meia. O líquido da panela (extrato) é recirculado para que as partículas sólidas em suspensão sejam separadas deste. Por isso, é importante que a moagem dos grãos seja grossa, já que estes assentam na tina e servem como filtro. Após a separação do extrato, a cama de grãos formada é lavada com um spray de água quente, processo denominado *sparging*;
- (b) decocção, na qual a moagem dos grãos é mais fina e os maltes utilizados são pouco modificados: devido à moagem fina, os grãos podem ser bombeados ou misturados. Este método usa três recipientes: um recipiente de mistura do mosto, um recipiente de decocção e um dispositivo de separação do extrato do mosto. Neste processo retira-se uma quantidade de líquido, que está a uma temperatura inicial de 35°C, e este é fervido e adicionado novamente ao restante. Após a mistura, a temperatura do mosto deve ser de cerca de 50°C e deve permanecer assim por um período de tempo. O processo é repetido outras duas vezes, para obter as temperaturas de 65°C e 76°C respectivamente;
- (c) dupla infusão; e
- (d) maceração escalonada, infusão por temperaturas programadas ou infusão por degraus de temperatura, que consiste de um processo que tem substituído os outros sistemas, em função de sua praticidade e economia de energia, da ordem de 30 a 50% se comparada a um programa de decocção similar. A extração de açúcares é feita de forma análoga ao processo de infusão simples, com a diferença de que as temperaturas do processo são controladas em rampas e patamares. A filtragem do extrato do mosto pode ser feita tanto utilizando a técnica de cama de grãos do processo de infusão simples quanto a filtragem direta da decocção. A figura 2.1 apresenta o exemplo de três programas de

temperatura de brassagem — de cima para baixo, os gráficos da figura se referem a brassagens por degraus, decocção simples e decocção dupla. No processo por degraus, o aumento de temperatura entre os patamares deve ser da ordem de $1^{\circ}\text{C}/\text{min}$.

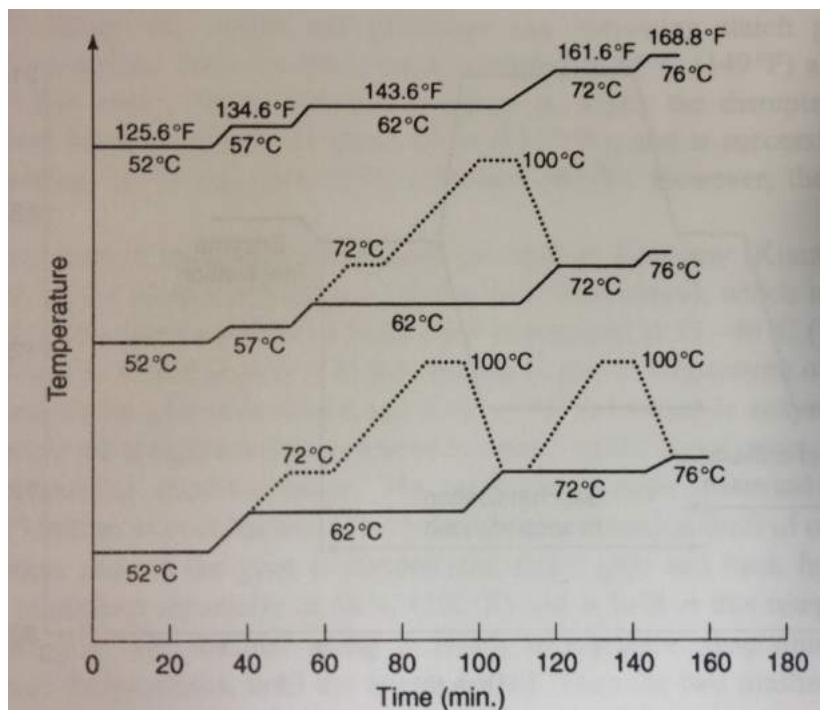


Figura 2.1: Programa de temperaturas típico de uma brassagem.
Fonte: BRIGGS (2011)

Após a **filtragem** do extrato do mosto, este é transferido para um caldeirão de **fervura**, processo no qual são adicionados os lúpulos e que leva cerca de uma hora, podendo se estender dependendo da receita [10], a exemplo das cervejas comerciais *90 Minute IPA* e *120 Minute IPA*, da cervejaria Dogfish Head, cujos tempos de fervura duram 90 minutos e 120 minutos, respectivamente [11, 12]. Lúpulo é uma flor em formato de cone e é utilizado para conferir amargor e aroma à cerveja, além de ser um ótimo conservante natural. Pode ser adicionado ao mosto em flores, *pellets* (pastilhas prensadas) ou extrato. Quando adicionado no início da fervura, contribui para o amargor da cerveja, por meio da isomerização de ácidos alfa que ocorre em função da alta temperatura. Em contrapartida, compostos aromáticos voláteis são perdidos neste processo e, por isso, também é adicionada uma quantidade de lúpulos ao final da fervura, para contribuir com o aroma [1, 10].

Além de conferir amargor à cerveja, por meio da adição de lúpulos, a fervura também é responsável pela coagulação de proteínas indesejáveis, assepsia do líquido, evaporação e consequente redução do seu volume, mudanças no sabor da cerveja e evaporação de compostos

voláteis indesejáveis [10].

Na sequência da fervura, o mosto deve ser **resfriado** rapidamente até 26°C para evitar oxidação, contaminação e criação de compostos orgânicos que introduzem sabores indesejados [1]. O processo de resfriamento é realizado com o emprego de um trocador de calor, denominado *chiller*. Também devem ser separadas e desprezadas as proteínas coaguladas e os restos de lúpulos decorrentes da fervura. Este processo geralmente é denominado *whirlpool*, em função de sua natureza, na qual o mosto fervido é centrifugado e os compostos indesejados se acumulam no centro e no fundo da panela de fervura [10]. O último passo é aerar ou até mesmo oxigenar o mosto, para que as leveduras possam se reproduzir corretamente no início da **fermentação** [10].

A levedura deve ser adicionada ao mosto resfriado e oxigenado o mais rápido possível para evitar contaminações, e o recipiente de fermentação comumente é selado, evitando a entrada de ar [10]. O processo de fermentação e os subsequentes processos de **maturação** e **envase** não serão detalhados, pois sua automação não é parte do escopo deste trabalho e o tema destes tópicos, em conjunto com a fermentação, envolve quase que exclusivamente outras áreas do conhecimento não relacionadas à engenharia elétrica.

Em resumo, é apresentada na figura 2.2 uma linha do tempo na qual estão listadas as partes essenciais do processo de produção de cerveja. Os momentos em que ocorrem cada uma destas partes são variáveis, dependendo de uma receita específica e, portanto, não foram especificados.

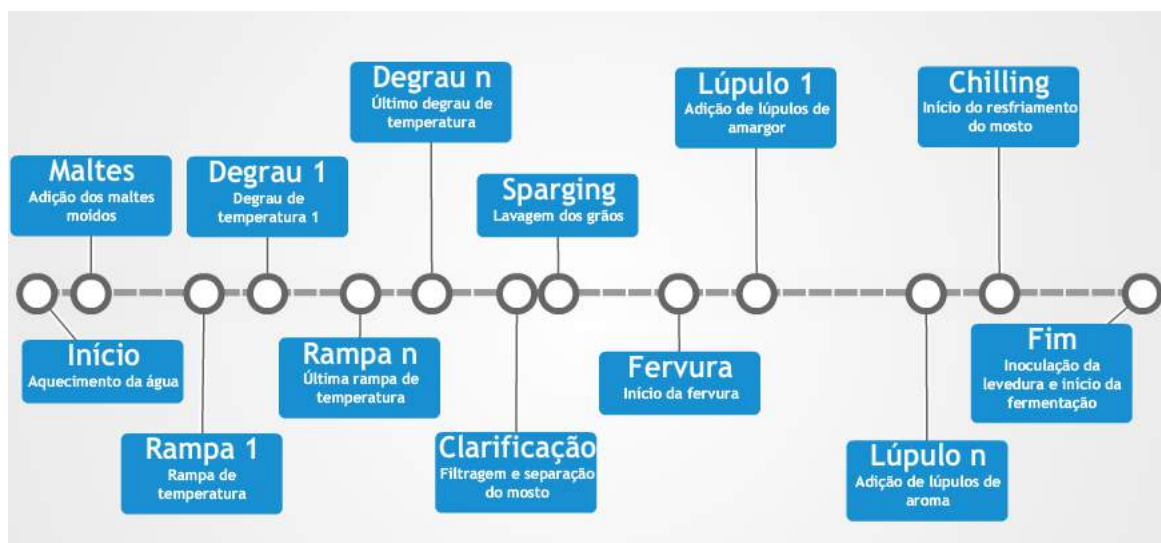


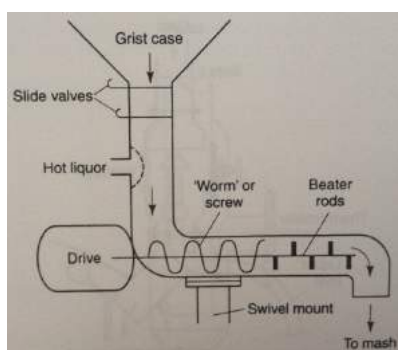
Figura 2.2: Linha do tempo de uma brassagem simples

2.2 Estruturas mecânicas de produção de cerveja

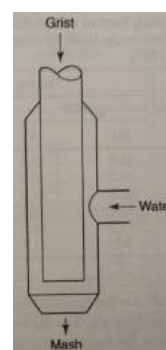
O processo de brassagem, que consiste no cozimento dos maltes para extração dos açúcares fermentáveis e filtração do líquido resultante, conhecido como mosto, é realizado em equipamentos específicos para estas tarefas. Em função das diferentes técnicas de brassagem e tecnologias desenvolvidas ao longo do tempo, diversos tipos e arranjos de equipamentos foram desenvolvidos [10].

Atualmente existe uma convergência nos métodos de fabricação de cervejas, motivada principalmente por questões econômicas, cujo objetivo do equipamento é maximizar a produção. Outros motivos para o desenvolvimento e aprimoramento de equipamentos são a necessidade de sempre produzir uma cerveja com as mesmas características, ou seja, padronizar a produção e, também, a preocupação crescente com redução do uso de energia e água, além da redução na produção de efluentes [10]. Ainda assim, equipamentos antigos continuam em uso, seja pela impossibilidade de reproduzir uma receita específica em equipamentos mais modernos ou pelo custo elevado da atualização das plantas de produção.

Na adição dos grãos à panela de mostura, o processo de mistura destes à água é importante para que a brassagem seja eficiente, uma vez que os grãos mal misturados podem formar aglutinados que impedem a extração dos açúcares. Para evitar este inconveniente dispositivos mecânicos foram desenvolvidos, conforme exposto na figura 2.3: em (a) a água e os grãos são misturados em um fluxo constante, determinado pela velocidade de giro de uma rosca sem fim e; em (b) a água é adicionada aos maltes em um ângulo tangente, de tal forma que um vórtex a mistura aos grãos. Com relação à água, a sua temperatura também deve ser controlada, para evitar que proteínas sejam inativadas, além de que seu volume inicial é predeterminado conforme a receita [10].



(a) Dispositivo com rosca sem fim



(b) Dispositivo de mistura por vórtex

Figura 2.3: Dispositivos de mistura de grãos à brassagem.

Fonte: BRIGGS (2011)

2.2.1 Painha de mostura

A painha de mostura, ou *mash tun* (MT), é o dispositivo mais simples para a preparação do mosto, uma vez que nela ocorre a extração dos açúcares, a filtração do extrato do mosto e a lavagem dos grãos [10]. A figura 2.4 apresenta uma configuração comum de MT, usada atualmente. As MTs possuem seção transversal circular e atualmente são feitas de aço inoxidável, revestidas com material isolante térmico. Embora originalmente fossem abertas, atualmente são cobertas para evitar perda de calor e espalhamento do vapor de água pelo ambiente da brassagem.

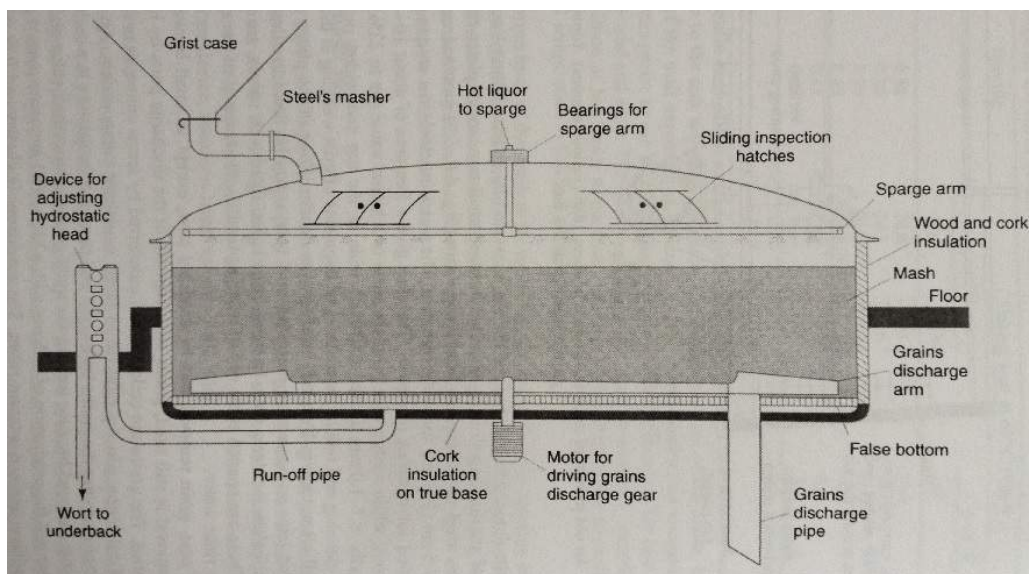


Figura 2.4: Esquema de painha de brassagem industrial moderna.

Fonte: BRIGGS (2011)

O fundo da MT é coberto por um fundo falso, que fica a alguns centímetros acima do fundo original e cuja função é atuar como filtro, em conjunto com a cama de grãos. Este fundo falso consiste de uma ou mais chapas de metal com rasgos da ordem de 0,7-1,0mm, que representam uma área livre para a drenagem de cerca de 12% da área total, ou uma trama com fios de metal cuja área livre chega a 22% [10].

O descarte dos grãos é feito através de um tubo, com a ajuda de uma pá que varre o fundo da painha ou um jato de ar comprimido. Métodos de remoção dos grãos descontinuados são a remoção manual (que ainda existe em pequenas instalações) e o enxágue dos grãos com bombeamento para um tanque anexo – este não mais usado em função da necessidade de mais água e posterior tratamento desta, que resulta em custos adicionais, além da perda do valor comercial dos grãos a serem descartados. Com relação à limpeza, as MTs são construídas

com suporte à limpeza limpeza no local (CIP), inclusive na região entre o fundo da panela e o fundo falso [10].

A água da lavagem dos grãos, ou *sparging*, é borrifada por um ou mais canos suspensos acima da cama de grãos, conforme ilustrado na figura 2.4. Estes canos são conhecidos como braços de *sparging* e geralmente giram no eixo em função da força da água, o que só é possível devido aos furos no tubo serem feitos na vertical, especialmente para este propósito. Outro aspecto da construção dos braços de *sparging* é que, à medida que se aproxima do centro da panela, os furos são feitos a uma distância maior entre si, possibilitando que a água seja igualmente distribuída sobre a cama de grãos [10]. O extrato do mosto é coletado por tubos no fundo da panela.

No caso da brassagem feita pelo processo de decocção, brassagem dupla ou infusão por temperatura controlada, diferentes recipientes de brassagem são utilizados. A figura 2.5 é um exemplo de tina de mistura (não confundir com mostura, embora a panela seja utilizada no processo de mostura) usada para brassagem de temperatura controlada. Nestes casos, a moagem dos maltes é mais fina e estes são constantemente misturados por uma pá no fundo da panela. Mesmo dentre estes sistemas os equipamentos diferem entre si e, diferente da MT, nestes sistemas é preciso utilizar outro recipiente para separar o extrato dos grãos. O separador pode ser uma tina de lavagem, conhecida também como *lauter tun* (LT), ou pode ser um filtro de mosto. Embora a LT e o filtro de mosto sejam amplamente utilizados na indústria, não serão detalhados neste trabalho, já que a abordagem prática adotada não inclui seu emprego.

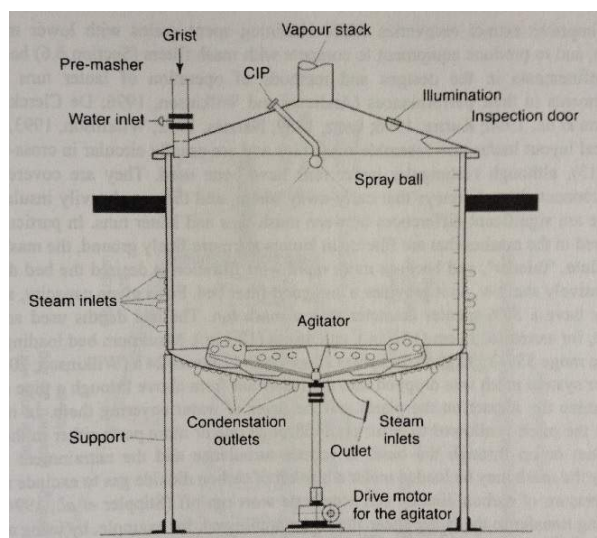


Figura 2.5: Esquema de panela de mistura do mosto.
Fonte: BRIGGS (2011)

2.2.2 Caldeirão de fervura

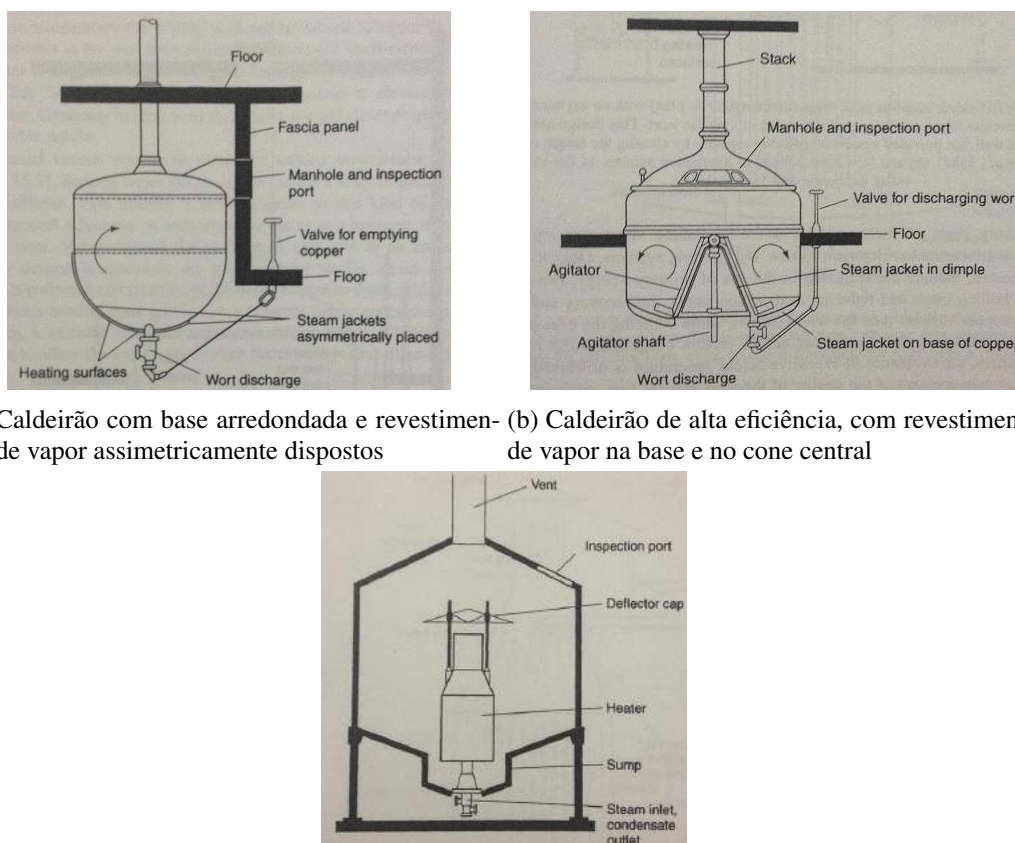
A fervura é o processo no qual o extrato do mosto é fervido com a adição de lúpulos em um caldeirão de fervura. Em inglês o termo utilizado para este caldeirão (*copper*, que significa cobre) lembra o fato de que inicialmente era comum o emprego de tinhas de cobre, dada a facilidade de moldagem que este metal permite, a alta condutividade térmica e a aparência atrativa dos caldeirões [10]. Este também é usualmente referido como *brewing kettle*, ou BK (caldeirão de fervura). Por muito tempo o estudo do processo de fervura foi negligenciado por ser considerado simples, mas à medida que a questão da economia de energia veio à pauta, estudos oriundos desta necessidade mostraram que o processo é mais complexo do que foi inicialmente considerado [10].

Como existe uma lista de objetivos que devem ser cumpridos durante a fervura, o projeto do equipamento é essencial para que estes sejam atingidos e, em um grau mais avançado, a maior economia de energia possa ser obtida. O primeiro objetivo – que não está necessariamente em ordem de importância – é a evaporação de água e consequente concentração do mosto, com taxas de evaporação inicialmente na faixa de 10% do volume por hora e que foram reduzidas com o desenvolvimento tecnológico das grandes indústrias, já que o custo de evaporação da água é caro em termos de demanda energética. O segundo objetivo importante da fervura é a esterilização do mosto, ou pelo menos a erradicação de formas vegetativas de micróbios – ainda que esporos possam sobreviver ao processo. A terceira função do processo é a evaporação de compostos voláteis indesejados, o que resultou em um desafio tecnológico para diminuir os tempos de fervura e quantidade de água evaporada sem que os compostos voláteis deixassem de ser eliminados efetivamente [10].

Dentre as mudanças que ocorrem no mosto durante a fervura, podem ser notadas criações, adições e transformações de substâncias químicas, como a dispersão de resinas e óleos do lúpulo no mosto e a isomerização de ácidos- α (transformação dos ácidos- α presentes no lúpulo em isômeros que conferem características de amargor à bebida), além da desnaturação e formação de coágulos de proteínas – processo que é favorecido por uma fervura rigorosa e prolongada – que virão a formar o chamado *trub*, resultado da decantação destas proteínas. Este processo de decantação pode ser acelerado por meio da adição de substâncias como gel de sílica ou musgo irlandês, conhecido como *irish moss* [10].

Historicamente o aquecimento dos BKs era feito de forma direta, com a queima de combustíveis sólidos. Embora estes não sejam empregados atualmente, ainda existem BKs que são aquecidos diretamente, por meio da queima de óleos ou gases. Ainda assim, atualmente

o maior elemento de aquecimento utilizado é o vapor d'água. Embora estes sistemas sejam mecanicamente mais complexos, o aquecimento a vapor reduz a quantidade de calor aplicada por unidade de área, o que evita a caramelização do mosto [10]. Na figura 2.6 são apresentadas três configurações de BKs, sendo (a) e (b) por meio de aplicação direta de vapor à superfície do recipiente e (c) utilizando um aquecedor interno, também a vapor.



(a) Caldeirão com base arredondada e revestimentos de vapor assimetricamente dispostos (b) Caldeirão de alta eficiência, com revestimentos de vapor na base e no cone central

(c) Caldeirão com aquecimento interno e mais profundo no centro, permitindo um aquecedor maior

Figura 2.6: Diferentes configurações de caldeirões de fervura.

Fonte: BRIGGS (2011)

Os projetos destes sistemas mecânicos são feitos com o objetivo de evitar que o mosto seja caramelizado ou descaracterizado de alguma forma e de modo a economizar energia. Com isso surgiram em função do tempo sistemas pressurizados e de aquecimento externo do líquido, dentre outras configurações diversas. Todos os sistemas modernos e eficientes de fervura são fechados e se aproveitam da pressurização de algum modo para reduzir o custo energético do processo de fervura, que se dá pela redução do tempo de fervura possibilitada pela elevação da temperatura do mosto. Contudo é preciso atentar-se ao fato de que temperaturas muito altas podem afetar negativamente a produção da cerveja e, portanto, valores

típicos de temperatura empregados pela indústria chegam a até 104°C, com exceções¹ [10].

A tabela 2.1 apresenta alguns sistemas de fervura utilizados na indústria e suas características com relação a temperatura, tempo de fervura e evaporação do mosto. Mesmo para um tipo específico de fervura, diversas configurações de equipamento podem ser adotadas.

Tabela 2.1: Características de diferentes sistemas de fervura.

Fonte: adaptado de BRIGGS(2011)

Sistema de aquecimento	Temperatura (°C)	Tempo de fervura (min.)	Evaporação (%)
Panela de "alta performance"	100	120-150	12-16
Aquecedores internos/externos, com contrapressão	102-103	60-80	8
Fervura de baixa pressão	103-104	55-65	6-7
Fervura dinâmica de baixa pressão	103-104	45-50	4,5-5
Fervura de alta temperatura / alta pressão	130-140	2,5-3	6-8
Aquecimento de película fina	100	35-40	4-4,7

Embora o objetivo principal deste trabalho não tenha sido a eficiência energética do sistema, neste parágrafo são apresentadas algumas considerações acerca do tema. Uma vez que o aquecimento de água une todas as partes do processo de produção de cerveja, não é realista acreditar que a economia de energia pode ser aplicada exclusivamente ao processo de fervura, embora este seja o mais custoso em termos energéticos, portanto automação de controles, equipe bem capacitada, boa conservação do edifício, *designs* eficientes, dentre outros fatores, são importantes para a economia de energia. No tocante à fervura em específico, geralmente são utilizados sistemas que recuperam energia de vapor e/ou água quente para reuso nas diversas etapas do processo de produção de cerveja [10]. Um exemplo é o sistema da figura 2.7, que utiliza energia do vapor de aquecimento utilizado na fervura para posterior aquecimento do mosto e pré-aquecimento de outra fervura. Para que isto seja possível, é essencial que a cervejaria realize consecutivas produções, uma vez que a energia é transferida para a produção seguinte.

¹O sistema de alta temperatura / alta pressão (140°C) é pouco utilizado e testes práticos mostram que os resultados em termos de qualidade da cerveja são questionáveis [10]

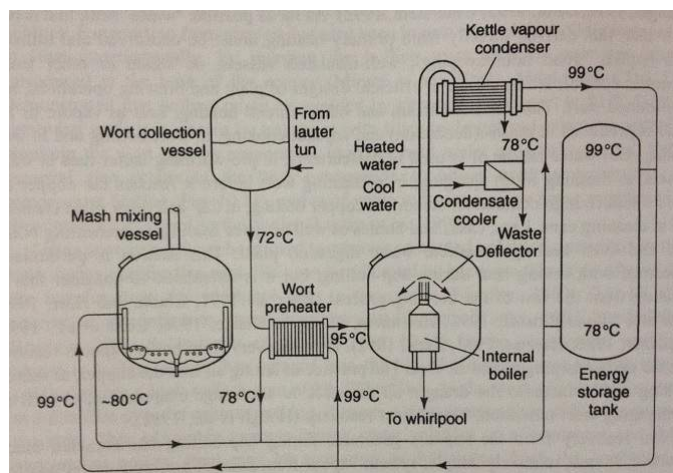


Figura 2.7: Arranjo no qual vapor do caldeirão de fervura é condensado e calor é recuperado e armazenado como água quente em um tanque de gradiente de temperatura.

Fonte: BRIGGS (2011)

2.2.3 Adição de lúpulos

É comum realizar a adição de lúpulos manualmente em pequenas cervejarias, porém deve-se observar que esta pode ser uma tarefa perigosa, pois o mosto pode subitamente vazar. Também, em adições tardias de lúpulo, ocorre a entrada de ar no BK, o que faz com que sistemas pressurizados não possam ser utilizados sem as devidas considerações e/ou modificações no equipamento [10].

A adição de lúpulos geralmente não é satisfatória quando são utilizadas flores, por isso pastilhas ou extratos são geralmente empregados. Em grandes cervejarias a maior parte do desafio de automação está no fato de que os lúpulos devem ser armazenados em ambientes com temperatura controlada, que geralmente ficam longe do BK, além de serem armazenados em caixas ou tanques. Além do sistema automático de transporte destes pacotes, em caso de BKs com pressurização, é preciso utilizar sistemas de câmaras de compressão para adicionar os lúpulos à panela [10].

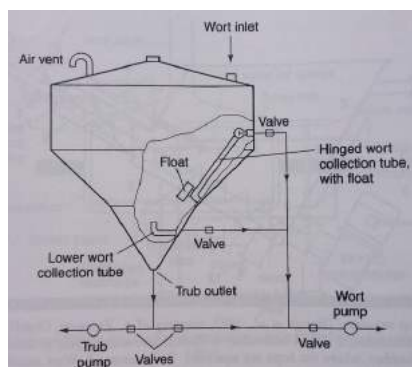
2.2.4 Clarificação e resfriamento do mosto

Quando o processo de fervura é finalizado, o mosto deverá estar claro, ou seja, com aparência brilhante. Ainda assim, é preciso remover o *trub*,² descrito na seção 2.2.2, e os restos de lúpulos utilizados no processo, já que o *trub* contribui com compostos sulfurosos e alcoóis indesejados, além de tornar a cerveja turva – o que pode ou não ser um problema [13]. Diver-

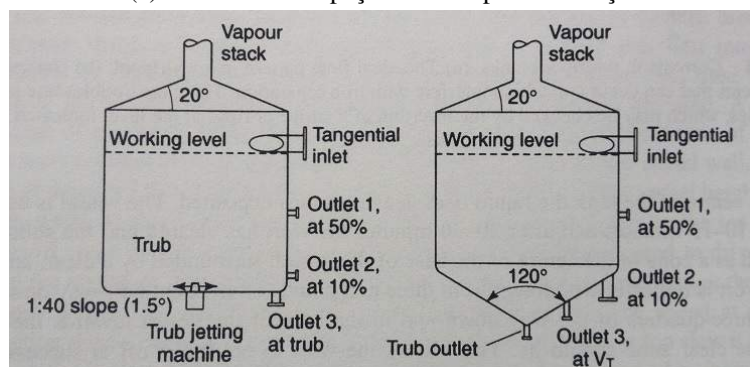
²O *trub* descrito neste capítulo é referente ao *hot break*, ou processo de fervura. Neste trabalho não será abordado o *trub* decorrente do *cold break*, ou resfriamento do mosto, que começa a ser formado em temperaturas próximas a 70°C

sof equipamentos foram criados com este propósito, seja para mostos com adição de lúpulos em flores, que facilitam a extração dos compostos indesejáveis, quanto para mostos feitos com pastilhas e extratos de lúpulo. Há sistemas chamados *hop back* que são empregados na filtragem de mostos com lúpulos em flor e que lembram MTs, nos quais o mosto é filtrado por um fundo falso, com elemento filtrador sendo a cama de lúpulos que decanta para o fundo da panela – inclusive há pequenas cervejarias que utilizam a MT para realizar este método de clarificação [10].

Embora existam outros sistemas de clarificação, o mais amplamente difundido na indústria atual é a técnica de *whirlpool* (redemoinho, em tradução livre) na qual mosto quente é injetado tangencialmente em um recipiente cilíndrico, a uma velocidade fixa, sendo a altura de injeção do mosto variável em função de diferentes projetos. Sistemas de gravidade geralmente não são suficientes e bombas ou sistemas projetados para utilizar o efeito de termossifão são utilizadas. As correntes de circulação do mosto decorrentes desta injeção tangencial fazem com que o material particulado fique concentrado no fundo e no centro do recipiente, possibilitando o escoamento controlado do líquido e a consequente separação do *trub* [10]. Na figura 2.8 são ilustrados sistemas de separação por decantação (a) e por *whirlpool* (b).



(a) Sistema de separação do *trub* por decantação



(b) Sistemas de separação do *trub* por *whirlpool*, com fundo plano e arredondado

Figura 2.8: Sistemas de clarificação.

Fonte: BRIGGS (2011)

Após o processo de clarificação deve começar a fermentação e, para que as leveduras possam ser inoculadas no mosto, este deve ser resfriado até a temperatura ideal de operação destes microorganismos, que está tipicamente na faixa 15-22°C para leveduras do tipo *ale* e 6-12°C para leveduras do tipo *lager*. O resfriamento rápido do mosto deve ocorrer, pois assim reações químicas decorrentes da fervura são interrompidas e a chance de contaminação do mosto é reduzida [10].

Para isto, um *cooler* (resfriador ou trocador de calor) é utilizado. Dentre os modelos mais largamente empregados encontra-se o *cooler* vertical, no qual o mosto passa por dentro de um arranjo de tubos finos, enquanto na parte externa destes água fria é circulada em contra-fluxo. Ainda assim, o modelo de *cooler* mais popular é do trocador de calor de placas, devido ao seu tamanho compacto, versatilidade e eficiência. As placas possuem padrões de desenho de tal forma que, quando são compactadas, elas formam canais pelos quais o mosto é passado em contra-fluxo a um agente resfriador (água fria, por exemplo). Estes trocadores de calor são projetados para que o escoamento seja turbulento e, conseqüentemente, a troca de calor seja mais eficiente. Quando um agente resfriador diferente da água é utilizado, a manutenção do sistema deve ser reforçada, já que vazamentos implicam na contaminação do mosto [10]. A figura 2.9 apresenta um esquema de *cooler* de placas.

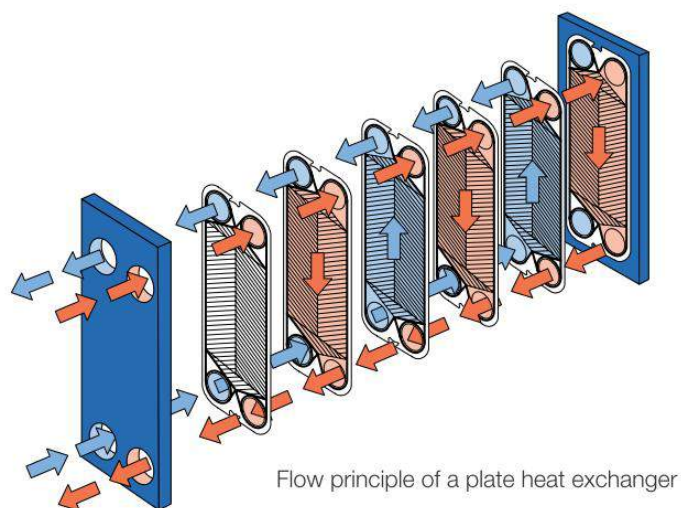


Figura 2.9: Esquema de funcionamento do trocador de calor de placas.

Fonte: Site da Offshore Energy Today³

³Disponível em <http://www.offshoreenergytoday.com/wp-content/uploads/2012/03/Alfa-Laval-to-Supply-Plate-Heat-Exchangers-for-Brazilian-Offshore-Plaforms.jpg>

Por fim, antes de inocular a levedura, é importante que o mosto seja suficientemente aerado ou oxigenado. Sem isso, a levedura não terá um ambiente adequado à sua reprodução e a atenuação do mosto – transformação de açúcares e oxigênio em álcool e gás carbônico – não será completada. Diferenças de solubilidade do mosto e dos tipos de levedura podem influenciar na quantidade de oxigênio necessária para um bom andamento da fermentação, sendo que para obter os níveis adequados, em escala industrial a oxigenação geralmente é forçada [10, 1]. Na produção de cerveja artesanal, a aeração geralmente é feita agitando o mosto resfriado ou com a ajuda de um sistema de aeração de aquário [1].

2.3 *Device-tree*

A *Device Tree* ou DT é basicamente uma estrutura de dados utilizada para descrever *hardware*, cujo objetivo é possibilitar que as características de *hardware* de um sistema específico sejam passadas para o SO durante o *boot*, ao invés de estas informações ficarem gravadas no SO [14], o que possibilita que um *driver* do Kernel seja usado em configurações diferentes de *hardware* sem a necessidade de modificações em seu código-fonte. Com relação aos sistemas equipados com processadores *advanced risc machine* (ARM), a necessidade de uso da DT surgiu devido ao grande número de novas placas, que tornou muito difícil manter uma versão do Kernel para cada sistema. Seu objetivo não é modificar o sistema em tempo de execução, porém no caso da BBB, foi criada uma solução chamada *Device Tree Overlay* (ou sobreposição) para executar esta função, facilitando a configuração de GPIO, dentre outras funções, e uso da DT pelo usuário final [15], fatores que são do interesse de aprendizes da área.

Uma DT é uma estrutura em árvore, composta de nós e propriedades. Uma abordagem para entender o seu funcionamento é por meio de um exemplo de sobreposição específico para a BBB. Na caixa de código-fonte 2.1 é apresentado um exemplo de sobreposição de DT específico da BBB que apresenta os principais conceitos para o uso desta ferramenta.

Neste código, na linha 14 é definida a plataforma à qual esta DT se refere. Com isso, ficam declaradas as funcionalidades de *hardware* do sistema. Na sequência, a identificação mostra quais sobreposições de DT estão carregadas e o nome do arquivo compilado deve ser igual ao nome nela atribuído; a versão deve ser 00A0 para a BBB. Entre as linhas 20 e 26 são listados os recursos de *hardware* utilizados por esta sobreposição, que impedem que outras sobreposições que usem os mesmos recursos sejam carregadas; no caso deste exemplo,

os pinos referentes à UART1, assim como o próprio dispositivo UART1 são utilizados. A seguir, os fragmentos descrevem qual dispositivo terá suas configurações sobrepostas. No fragmento 0, é o multiplexador dos pinos da BBB, compatível com o *driver pinctrl-single* - os dois valores hexadecimais utilizados para cada pino são o *offset* do pino com relação ao seu registrador e a configuração do pino. Já no fragmento 1 é configurada a interface UART1. Por enquanto, serão introduzidas somente as questões referentes à configuração do multiplexador, uma vez que seu conhecimento foi necessário para o uso dos pinos no modo GPIO.

Para determinar o valor hexadecimal de *offset* do pino ao qual se deseja aplicar uma configuração, primeiro é preciso obter seu nome a partir do seu número na barra de pinos, conforme ilustrado na figura C.3, consultando as tabelas ?? e ??, nas quais o nome do pino é referente ao **modo 0**, que também é o nome do pino no manual do SoC [16, 17]. A seguir, obtém-se o valor hexadecimal do registrador referente ao pino, na tabela II.1 presente no anexo II, adaptada do manual do SoC, e subtrai-se 0x800 de seu valor original.

```

1  /* Copyright (C) 2013 CircuitCo */
2  /dts-v1/;
3  /plugin/;
4
5  / {
6      compatible = "ti,beaglebone", "ti,beaglebone-black";
7
8      /* identification */
9      part-number = "BB-UART1";
10     version = "00A0";
11
12     /* state the resources this cape uses */
13     exclusive-use =
14         /* the pin header uses */
15         "P9.24",          /* uart1_txd */
16         "P9.26",          /* uart1_rxd */
17         /* the hardware ip uses */
18         "uart1";
19
20     fragment@0 {
21         target = <am33xx_pinmux>;
22         __overlay__ {
23             bb_uart1_pins: pinmux_bb_uart1_pins {
24                 pinctrl-single, pins = <
25                 /* P9.24 uart1_txd.uart1_txd MODE0 OUTPUT (TX) */
26                 0x184 0x20
27                 /* P9.26 uart1_rxd.uart1_rxd MODE0 INPUT (RX) */
28                 0x180 0x20
29                 >;
30             };
31         };
32     };
33

```

```

34 | fragment@1 {
35 |     target = <&uart2>; /* really uart1 */
36 |     __overlay__ {
37 |         status = "okay";
38 |         pinctrl-names = "default";
39 |         pinctrl-0 = <&bb_uart1_pins>;
40 |     };
41 | };
42 | };

```

Código-fonte 2.1: Sobreposição de *device tree*

Com relação à configuração do pino, este aceita os parâmetros de ajuste de *slew-rate*, ativação do *buffer* de entrada, ajuste e ativação do *pull-up/pull-down* interno e seleção do modo de operação do pino. Note-se que, com relação ao buffer de entrada, ao menos um blog da internet refere-se a este bit como sendo um bit de seleção para entrada **ou** saída [18], porém o manual indica que quando o bit está setado, o pino pode ser usado tanto como entrada quanto saída [17]. A tabela 2.2 apresenta os campos do registrador de controle de E/S, assim como a descrição de cada campo e os valores aceitos.

Tabela 2.2: Descrição dos campos do registrador de controle de *pads*.

Fonte: adaptado de TEXAS INSTRUMENTS(2014)

Bit	Campo	Valor	Descrição
31-7	Reservado		Reservado. Leitura retorna 0
6	SLEWCTRL	0 1	Seleciona entre <i>slew-rate</i> rápido ou lento 0 - rápido 1 - lento
5	RXACTIVE	0 1	Ativa modo de entrada para o pino 0 - somente saída 1 - Receptor ativado. Entrada ou saída.
4	PULLTYPESEL	0 1	Seleção de pull-up/pull-down 0 - pull-down 1 - pull-up
3	PULLUDEN	0 1	Habilita pull-up/pull-down 0 - habilitado 1 - desabilitado
2-0	MUXMODE	0-7	Seleção de funcionalidade do pino multiplexado

2.4 Programação *server-side*

Para programar o código da aplicação que roda no servidor, existem diversas linguagens de programação disponíveis, geralmente de alto nível, a exemplo de PHP, Python e Node.js. Neste trabalho, a linguagem PHP foi utilizada em pequena escala, porém foi o ambiente

interpretador de Javascript Node.js o escolhido para desenvolvimento da aplicação *server-side*.

2.4.1 Node.js

Node.js é um ambiente em tempo de execução recente, criado em 2009 [19], que permite a execução no servidor — neste caso, a BBB — de programas escritos em Javascript. Para tal, este ambiente utiliza um motor implementado pelo Google para seu navegador Chrome — o V8, que compila o Javascript em código de máquina e não *bytecode* ou interpretação *on-the-fly*, o que torna a sua execução extremamente rápida [20], embora seja importante lembrar que o *overhead* inerente de linguagens de tipagem dinâmica custa o tempo de processamento de resolução do tipo de variável utilizada [21].

Node, devido à natureza assíncrona de I/O do Javascript, implementa um suporte nativo à programação orientada a eventos, ainda que seja executado em uma única *thread*, ou seja, diversas porções de código são multiplexadas no tempo. Isto o torna um diferencial com relação às outras linguagens de programação mais populares, que exigem do desenvolvedor a implementação de processos de *multithreading* para atingir paralelismo de processamento [22]. Ainda, o estilo de programação em Node é orientado a *callbacks*, que nada mais é do que passar como argumento uma função que será executada assim que um evento acabar.

Historicamente, o desenvolvimento de *multithreading* surgiu da necessidade de serviços de rede que possibilitassem múltiplas comunicações em paralelo, o que é conhecido como múltiplas fontes de I/O e assim será referenciado neste trabalho. Como linguagens tradicionais de programação, mais antigas, focam na implementação de aplicações operadas por um único ser humano, com o desenvolvimento da internet esta realidade mudou drasticamente, exigindo processamento de I/O massivo [23]. Em processadores com diversos núcleos, isso significa processamento em paralelo, enquanto em processadores com um único núcleo é implementada uma multiplexação temporal que permite tal técnica. O fato mais notável das implementações de *multithreading* é a complexidade do código, que não é trivial e geralmente dá origem a códigos mal estruturados e repletos de variáveis globais, difíceis de manter e muitas vezes referenciados como código espaguete [22, 23].

Javascript, que nasceu como uma linguagem para execução no cliente, tornou-se a espinha dorsal de aplicações HTML modernas e, recentemente, ficou muito popular dentre aplicações no servidor. Especificamente em Node, o comportamento assíncrono é a regra geral, devido à orientação a eventos. Isto leva programadores acostumados a linguagens síncronas como

C a terem dificuldades iniciais de programação: o uso de *callbacks* de evento é fundamental para estruturar a ordem com que certas tarefas são executadas, sempre após o final do evento pai [23]. É importante salientar que este tipo de implementação ocorre de forma transparente ao desenvolvedor mas, na verdade, nada mais é do que um *loop* principal cujo objetivo é cuidar de todas as chamadas a funções. Outro ponto notável relacionado ao Node é que, para executar aplicações em múltiplos núcleos, é preciso executar múltiplas instâncias, o que pode ser gerenciado com bibliotecas de suporte [22, 23]. Note-se também que os termos *orientado a eventos* e *assíncrono* são equivalentes [23].

Para ilustrar o comportamento assíncrono do Node, no código da caixa 2.2 não há garantia alguma de que a função executada na segunda linha receba um parâmetro diferente de nulo (Null) [23]:

```
1 var botaoStatus = leBotao();
2 imprimeValorBotao(botaoStatus); //nao ha garantia de que
   leBotao ja acabou de ser executada
```

Código-fonte 2.2: Implementação incorreta de código em Node.js

A maneira correta para executar este tipo de código seria usando uma função de *callback*, como descrito no trecho de código da caixa 2.3, embora esta não seja a única maneira de fazê-lo. Observe-se que a função é declarada como um parâmetro de *leBotao* e não tem nome — é uma função anônima. Embora isto seja prático, do ponto de vista de manutenção de código e *debug* é uma prática que deve ser evitada, uma vez que pode originar uma situação de dificuldade de manutenção de código conhecida como *callback hell*, ou *inferno de callbacks* [24, 25].

```
1 leBotao(function(botaoStatus){
2   imprimeValorBotao(botaoStatus); //so acontece depois que o
   botao eh lido
3 });
```

Código-fonte 2.3: Implementação de código em Node.js orientada a callback

2.4.2 Node Package Manager - NPM

O *Node Package Manager*, ou simplesmente NPM, é não somente um gerenciador de pacotes como também é um repositório de pacotes de terceiros e um padrão para definição de dependências. Seu uso não é obrigatório, mas à medida que aplicações mais complexas são desenvolvidas se torna quase mandatório, pois assim é possível usar módulos prontos para executar diversas tarefas com facilidade. Uma vantagem do NPM é que os módulos podem ser instalados localmente, restringindo-os ao diretório do projeto e, portanto, garantindo certo

nível de segurança. [23]. Na caixa 2.4 é apresentado um exemplo no qual o pacote fictício *meuPacote* é instalado para a versão mais recente adicionada ao repositório do NPM:

```
1 npm install meuPacote@latest
```

Código-fonte 2.4: Instalação de pacote usando o NPM

Outro exemplo de uso do NPM é a definição de dependências de um projeto e posterior possibilidade de instalação de todas elas de uma só vez. Para isto, um arquivo no formato JSON descreve estas dependências e outras informações do projeto, conforme ilustrado no exemplo da caixa 2.5:

```
1 {
2   "name" : "meuPacote",
3   "version" : "1.0.0",
4   "dependencies" : {
5     "debug" : "0.3.x",
6     "nano" : "*",
7     "request" : ">0.2.0"
8   }
9 }
```

Código-fonte 2.5: Descrição e dependências de um projeto usando NPM

E a instalação das dependências é executada conforme descrito na caixa 2.6, assumindo que o presente diretório de trabalho é o mesmo do projeto:

```
1 npm install
```

Código-fonte 2.6: Instalação de dependências usando o NPM

2.4.3 *MEAN Stack*

MEAN é uma pilha de desenvolvimento (similar à amplamente difundida pilha LAMP) de código aberto, voltada para o desenvolvimento de aplicações web e que provê um conjunto de quatro ferramentas. Juntas, estas são a base necessária para a criação de aplicações tanto de servidor quanto do cliente. Cada uma das 4 letras deste acrônimo representa uma das ferramentas [26]:

- **MongoDB** - banco de dados orientado a objetos
- **Express.js** - framework para criação de servidor web e roteamento
- **Angular.js** - framework para aplicações web
- **Node.js** - base da aplicação do servidor

Note-se que toda a pilha é baseada na linguagem Javascript, o que permite ao desenvolvedor de uma solução completa uma curva de aprendizado mais rápida, pois menos tempo

é empregado aprendendo a sintaxe e o paradigma de programação e mais tempo dedicado à funcionalidade da aplicação em si [26]. Dois casos de aplicações web desenvolvidas em Node são o serviço de streaming de vídeo Netflix, que inclusive encontrou alguns problemas relacionados ao uso do framework Express [27], e a empresa de transporte Uber [28].

2.4.4 Servidor Express.js

Express é um framework de *middleware* para implementação de aplicações web no servidor, incluindo mas não limitado a roteamento e operações HTTP, a exemplo dos métodos GET e POST. Embora o Node apresente um módulo dedicado a HTTP, a proposta do Express é simplificar seu uso e evitar que o mesmo trabalho seja realizado múltiplas vezes e por várias pessoas diferentes [26].

Dentre as facilidades proporcionadas pelo Node está a fácil implementação de diferentes respostas para diferentes tipos de requisições baseadas no método HTTP e a renderização dinâmica de documentos HTML [23]. O tratamento de requisições baseadas no método HTTP, possibilitado pelo Express, também é conhecido como RESTful API, que é uma API baseada no REST (*Representational State Transfer*). Este por sua vez é um modelo para serviços web que implementa as operações HTTP POST, GET, PUT e DELETE, que são mapeadas como operações básicas de banco de dados: criar, ler, atualizar e deletar [26].

Capítulo 3

Materiais

Este capítulo lista os materiais empregados no desenvolvimento deste projeto e, para sua melhor organização, os diversos materiais estão divididos nos seguintes capítulos: **núcleo do projeto** — que compreende principalmente a BBB; **sistema mecânico** — onde são listadas as peças empregadas na construção tanto da estrutura de suporte do sistema quanto do sistema de painéis e tubulações; **componentes eletrônicos** — lista os componentes utilizados para a confecção dos circuitos de acionamentos de potência, dentre outros; **equipamentos, instrumentos, softwares e miscelânea** — apresenta os materiais de apoio ao projeto ou aqueles que não se encaixam em outras categorias, assim como os softwares utilizados para desenvolvimento.

Há um apêndice dedicado ao sensor DS18B20 (apêndice B) e outro à BBB e sua configuração (apêndice C).

3.1 Núcleo do projeto

Os materiais utilizados no núcleo do mesmo estão contidos na tabela 3.1:

Tabela 3.1: Lista de materiais que compõem o núcleo do projeto

Descrição	Quantidade
BeagleBone Black Rev. C	1
Fonte de tensão 5V/2A	1
Cartão µSD 4GB	1
Adaptador Wi-Fi USB Ralink RT5370	1
Sonda de aço inoxidável à prova d'água com sensor DS18B20	1

3.2 Sistema mecânico

Os materiais utilizados na construção da parte mecânica do sistema estão listados na tabela 3.2:

Tabela 3.2: Lista de materiais utilizados na confecção do subsistema mecânico

Descrição	Quantidade
Itens diversos	
Caldeirão de alumínio n.º36 (32 litros)	2
Resistor helicoidal com duas voltas, 110V/2200W	2
Fundo falso de aço inoxidável circular $\phi=36\text{cm}$	1
Bomba centrífuga magnética 12VDC Topsflo TL-B08H-12-1006	2
Trocador de calor de contra-fluxo, tubo de alumínio de 7m	1
Tecido mussalina x 1,5m (centímetros)	30
Tubulação	
Mangueira silicone 3/8"(metros)	1
Mangueira cristal 3/8"(metros)	2
Mangueira cristal 1/2"(metros)	7
Tubo schedule S40 1/2"aço inoxidável (metros)	3
Valvulas e acessórios	
Válvulas e acessórios	
Válvula de retenção portinhola 1/2"aço inoxidável	2
Válvula esfera latão, passagem plena 1/2"	2
Válvula solenóide 1/2"12V plástico uso geral	1
Válvula solenóide Danfoss 1/2"EV210BD 032U3620	3
Bobina Danfoss 15W 12VDC 042N7550	3
Conexões e acessórios	
Espigão de latão com redução 1/2- 3/8"	3
Bico de engate rápido de plástico 1/2"	1
Engate rápido de plástico 1/2"	1
Abraçadeira 3/8"	4
Abraçadeira 1/2"	2
Luva sextavada 1/2"latão	1
Arruela aço inoxidável 1/2"	4
Anel de vedação de silicone 1/2"	4
Contra porca aço inoxidável 1/2"	2
Cotovelo fêmea 90° 1/2"aço inoxidável	1
Cotovelo M/F 90° 1/2"aço inoxidável	1
Tee fêmea 90° 1/2"aço inoxidável	4
Niple duplo sextavado 1/2"aço inoxidável	9
Espigão sextavado 1/2"aço inoxidável	4
Estrutura metálica	
Cantoneira de ferro 2"x 1/8"(metros)	18

A figura 3.1 apresenta detalhes das conexões para a tubulação, trocador de calor, válvula solenóide Danfoss e do resistor de aquecimento.



(a) Válvula e solenóide Danfoss



(b) Resistor de aquecimento



(c) Conexões diversas



(d) Trocador de calor de contra-fluxo

Figura 3.1: Peças empregadas na construção da estrutura mecânica

3.3 Componentes eletrônicos

Os componentes eletrônicos utilizados para confecção dos circuitos de interface com a BBB, acionamentos de potência e circuitos diversos estão listados na tabela 3.3:

Tabela 3.3: Lista de componentes eletrônicos diversos empregados na confecção de PCBs e cabeamento do sistema

Componente	Modelo/valor/descrição	Quantidade
Placa de acionamentos de potência		
Capacitor cerâmico	100nF	1
Capacitor poliéster	330nF	1
Resistor	2,2k Ω 1/4W 5%	10
Resistor	22k Ω 1/4W 5%	10
Optoacoplador	4N25 DIP	7
Regulador de tensão	LM7805 TO220	1
Transistor MOSFET	IRF540N TO220	7
Diodo retificador	1N4007	6
Conector para alimentação	J4	2
Conector header	10 vias	1
Borne KRE	2 vias	6
Porta fusível	25mm 6A(MAX)	1
Fusível de vidro	20AG 5A 20mm	1
Cabeamento		
Conector latch	10 vias	1
Cabo flat (metros)	10 vias	2
Componentes diversos		
Fonte de tensão	12V/5A	1
Relé de estado sólido	Fotek SSR-25 DA (25A)	2
Cabo de cobre (metros)	4mm seção transversal	6
Tomada	20A	1
Adaptador T	padrão NEMA	1
Terminal anel	pré-isolado M6	4
Plug tomada	20A padrão NBR 14136	1
Servo Motor	TowerPro MG995	1

3.4 Equipamentos, instrumentos, *softwares* e miscelânea

Os equipamentos, instrumentos de medição e demais objetos e/ou *softwares* de suporte ao desenvolvimento do projeto estão listados na tabela 3.4.

Tabela 3.4: Lista de equipamentos, instrumentos e *softwares* de suporte ao desenvolvimento do projeto

Descrição	Detalhes
Instrumentos de medição	
Osciloscópio Agilent DSO-X 2002A	70MHz - 2 canais
Multímetro Minipa ET-2110	
Softwares	
FreeCAD	v. 0.15
MatterControl	v. 1.3.0 + plugin CNCBrasil
Cura	v. 15.06.03
LTSpice	
Proteus Schematic Capture	
MCU 8051 IDE	
Putty	
Diversos	
Impressora 3D	CNCBrasil Standard
Filamento para impressão 3D	ABS 1,75mm
Controle de versão	
GitHub	https://github.com/leograba/final_paper_tcc

Capítulo 4

Métodos

Este capítulo descreve todos os procedimentos de projeto e testes, dentre outros, empregados visando a obtenção do resultado final deste trabalho. Os procedimentos aqui empregados são embasados na teoria apresentada no capítulo 2. Cabe salientar que, embora este seja um projeto de integração entre *mecânica*, *hardware* e *software*, os esforços de desenvolvimento foram concentrados na área de desenvolvimento de *softwares*, desde o servidor web e o algoritmo de controle implementados em Node.js, até a aplicação de usuário, escrita em diversas linguagens, de programação ou não — a exemplo de HTML e CSS. A figura 4.1 apresenta uma visão geral do sistema, que compreende parte mecânica, *hardware* e *software*.

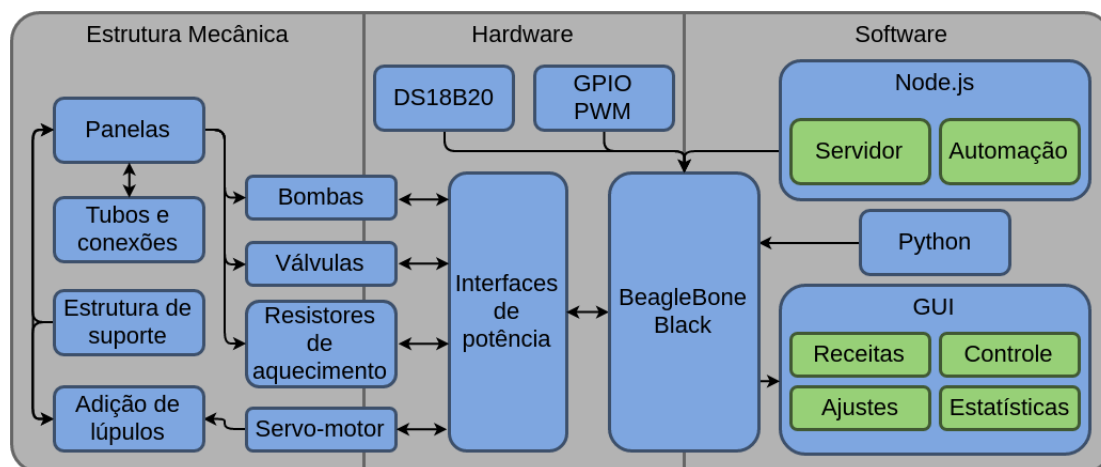


Figura 4.1: Diagrama em blocos do projeto

4.1 Estrutura mecânica

Para o projeto da parte mecânica, em primeiro lugar foi concebida uma configuração de sistema baseada tanto na literatura disponível quanto em projetos de sistemas comerciais de

produção de cerveja: um sistema de duas painéis análogas à MT e ao BK, sendo que durante o processo de cozimento do mosto, o BK pudesse ser utilizado como HLT. Para atender a esta especificação, fez-se necessário o projeto de um sistema de recirculação entre as duas painéis. Observa-se que esta proposta representa um híbrido entre o sistema de 3 painéis tradicional - que consiste de uma panela para mosturação, uma para lavagem dos grãos, ou *sparging*, e uma para fervura; e o sistema popularizado pela Braumeister, composto de 1 panela, com recirculação. Esta abordagem permitiu a economia financeira e simplificação mecânica do sistema de três painéis com o benefício do *sparging* inexistente no sistema de uma panela.

4.1.1 Funcionamento da estrutura

Na figura 4.2 é apresentada uma representação esquemática da parte funcional.

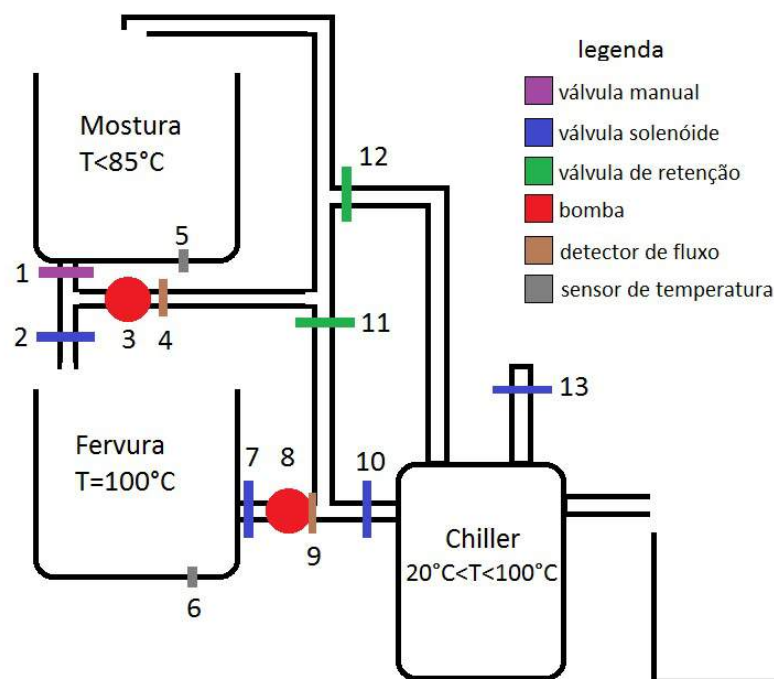


Figura 4.2: Representação esquemática da estrutura mecânica

Na panela superior ou MT, rotulada como *Mostura*, os grãos são adicionados após o aquecimento inicial da água:

- A válvula 1 é uma válvula manual do tipo esfera, com passagem plena. Durante a operação do equipamento ela deve ficar sempre aberta, portanto seu uso é realizado somente em caso de emergências.

- Após a adição dos grãos, a bomba 3 é ligada para que o líquido da panela seja recirculado. As válvulas de retenção 11 e 12 não permitem que o líquido passe por elas, portanto há somente um caminho possível para este, que é ascender pela tubulação até entrar novamente na MT. Note-se que não há válvula automatizada na saída da MT, uma vez que o mosto fica em recirculação constante ao longo do processo de cozimento.
- Após a mostura a bomba 3 é desligada e a válvula solenóide 2 é aberta, escoando o líquido para a panela de fervura BK. Simultaneamente, a válvula solenóide 7 e a bomba 8 são ativadas, fazendo com que a água de lavagem presente na BK/HLT flua pela tubulação até a MT, iniciando o processo de *sparging*. Durante um tempo predefinido, o mosto e a água de lavagem recirculam pelas duas panelas e se misturam.
- Durante a fervura, as válvulas 2 e 7 são fechadas. Neste período, os grãos drenados que estão na MT devem ser retirados manualmente, já que ela será usada posteriormente para armazenamento da água de esfriamento do mosto — água usada para limpeza do sistema e que reduz a produção de efluentes do mesmo.
- Após a fervura, as válvulas solenóides 7 e 10 são ativadas, permitindo o escoamento do líquido para o trocador de calor. Simultaneamente a válvula solenóide 13 é aberta e água fria circula em contra-fluxo para resfriar o mosto. Esta água sai quente do trocador de calor e passa pela válvula de retenção 12, preenchendo a parte superior da tubulação e sendo armazenada na MT para posterior limpeza do sistema.
- O mosto que sai resfriado do *chiller* cai no balde de fermentação. Neste processo o líquido entra em contato com o ar, o que é não somente desejável como essencial para o sucesso da fermentação, portanto esta é a última parte automatizada que tem relação com a cerveja. Uma solução futura e que permite automação desta transição é o uso de um aerador em conjunto com um tanque de fermentação selado.
- Por fim, a água armazenada na MT é aquecida e recirculada pelo sistema, promovendo uma limpeza preliminar deste.

Para o bom funcionamento do sistema proposto, algumas condições devem ser atendidas:

- Se o volume total de líquido nas duas panelas ao final da mostura exceder o volume da BK, ela transbordará. Por este motivo é importante que exista um sistema de detecção de transbordo. Ainda assim, deve-se requisitar que o usuário do sistema calcule corretamente as proporções da receita, pois mesmo que seja aplicado um sistema automático anti-transbordo, ocorrerá perda de insumos devido ao acúmulo de mosto inutilizável na MT em caso de erro.
- São necessários filtros de material particulado na saída das panelas para o bom funcionamento das válvulas e bombas [29].
- Os dispositivos mecânicos (válvulas, bombas, tubulação, dentre outros) e eletrônicos (sensores e atuadores) em contato com o sistema mecânico devem ser apropriados às altas temperaturas impostas pela natureza do processo [29, 30].
- As panelas, tendo como referência seu fundo, não devem ser alinhadas concêntricas, já que isto torna a adição de lúpulos e o processo de manutenção do equipamento desajeitados.
- A automação da técnica de *whirlpool* não é contemplada nesta configuração de sistema. Se o usuário a considera necessária, ele deve se encarregar de executá-la manualmente.
- A única válvula com pressão suficiente para ser servo-operada é a 13, assumindo-se que a pressão incidente sobre ela seja maior do que 0,5bar, portanto as outras devem ser de operação direta [29, 30].
- A limpeza automatizada, ou CIP, não é contemplada em sua totalidade nesta configuração do sistema, devido à alta complexidade [31], conforme indica a figura 4.3, e ao custo de implementação proibitivo para o presente trabalho. Não obstante, é recomendado ao usuário do sistema que faça a limpeza do mesmo tão rápido quanto possível após o final da produção.

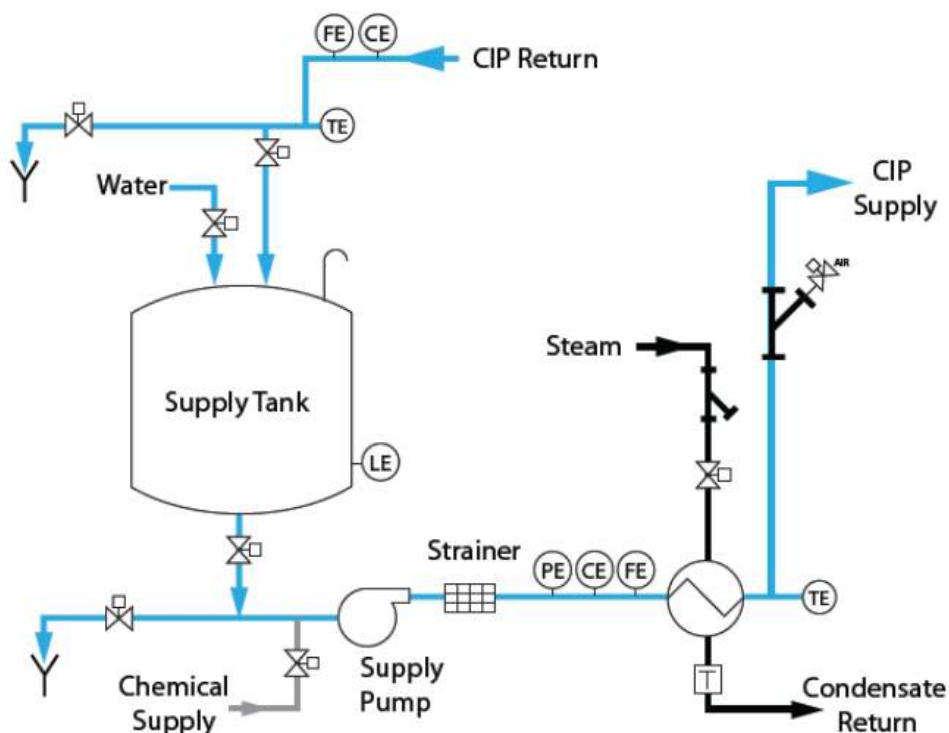


Figura 4.3: Sistema CIP básico.
Fonte: ROSE e MONTGOMERY (2010)

4.1.2 Dimensionamento da parte funcional

Com a filosofia de operação do sistema mecânico definida, foi possível realizar o dimensionamento deste sistema. A primeira consideração a ser feita é que os materiais e métodos aqui empregados não seguem nenhuma norma que permita o uso deste equipamento para fabricação de cerveja visando sua comercialização — tal escolha foi feita em função do alto custo de um sistema completamente dentro das normas e também pelo fato de este ser um trabalho cujo foco é a automação e o acesso remoto do sistema, e não a comercialização de cerveja.

Não obstante, o material da tubulação escolhido foi o aço inoxidável AISI304. Já que o volume de líquido a ser trabalhado é pequeno, menor do que 40 litros, e a pressão de trabalho não é maior do que a pressão das bombas escolhidas posteriormente, foi decidido o uso de tubulação de diâmetro de 1/2" (21,34mm de diâmetro externo) e espessura da parede no padrão Schedule 40 (2,77mm de espessura). Embora esta espessura seja superdimensionada para a presente aplicação, o mecânico responsável pela montagem do sistema a requisitou para que fosse possível fazer rosca sem danificar a integridade dos tubos. As conexões, seguindo a escolha dos tubos, também foram de aço inoxidável de 1/2".

As ligações entre tubos, conexões e outros componentes do sistema são ligações rosqueadas. Este é um meio de ligação antigo, porém de baixo custo, fácil execução e usualmente empregadas em tubulações de diâmetro nominal pequeno, menor do que 4" [32]. Para vedação foi escolhido o Teflon, uma vez que é um material de baixo custo e alta disponibilidade e o padrão de rosca adotado neste projeto é o BSP, baseado na norma ISO. Cabe salientar que, embora as ligações rosqueadas sejam permitidas sob certas circunstâncias na prática, elas são relegadas a instalações de baixa responsabilidade [32].

As painelas foram escolhidas no material de alumínio, em função do custo proibitivo do aço inoxidável. São caldeirões padrão utilizados no ramo de hotelaria e restaurantes e disponíveis em vários volumes. As duas painelas utilizadas neste trabalho tem capacidade para 32 litros e diâmetro do fundo de 36cm, portanto sua especificação no mercado é *caldeirão de alumínio n. 36*. Tal volume, 60% superior ao da produção máxima aconselhada para este sistema, é necessário devido ao volume dos grãos, à quantidade da água de lavagem e às perdas por evaporação que exigem uso de um volume de água inicial superior ao nominal.

O parâmetro inicial escolhido para seleção das válvulas solenóide foi a temperatura máxima de operação, cuja fonte de dados de operação é fornecida pelos fabricantes. Em seguida, foi escolhida uma vedação adequada: os dois tipos mais comuns de vedação com suporte a temperatura de pelo menos 100°C são o EPDM e FKM (etileno-propileno-dieno e fluoreto de vinilideno, respectivamente) [29]. Estes materiais possuem uma tabela de compatibilidade de fluidos cuja classificação pode ser satisfatória, boa, duvidosa, insatisfatória e desconhecida — tanto para cerveja quanto para o mosto, ambas as vedações são classificadas como satisfatórias, embora para água a classificação do FKM seja somente boa [33]. Por fim, um parâmetro que foi verificado antes da escolha final das válvulas é a posição de operação, que varia conforme os modelos do fabricante [29]. Com base nos parâmetros supracitados, foi decidido usar:

- Válvula solenóide 1/2" 12V plástico uso geral para água de resfriamento do trocador de calor.
- Válvula solenóide Danfoss 1/2" EV210BD 032U3620 para demais válvulas.
- Solenóide Danfoss 15W 12VDC 042N7550 para acionamento das válvulas Danfoss.

Informações técnicas sobre a válvula EV210BD 032U3620 estão no anexo III e sobre a bobina 042N7550 no anexo IV.

As bombas de recirculação foram escolhidas com base na temperatura de operação e grau alimentício. Posteriormente, tanto a vazão quanto a perda de carga do sistema foram

estimadas para verificar se a bomba escolhida era adequada: o modelo Topsflo B08H-12-1006 tem capacidade máxima de vazão de 10l/min e carga expressa em coluna de água de 6m. Como a capacidade máxima do sistema é de 20l e idealmente a temperatura deve subir à taxa de 1°C/min, além de que a altura do sistema é menor do que 2m e as perdas de carga não decorrentes da gravidade foram consideradas desprezíveis para este caso, foi decidido que a bomba escolhida é adequada ao projeto. No anexo V encontra-se a folha de dados técnicos do dispositivo.

Quanto às conexões, a figura 4.4 apresenta um diagrama conceitual do sistema, contendo as peças necessárias para a montagem do mesmo:

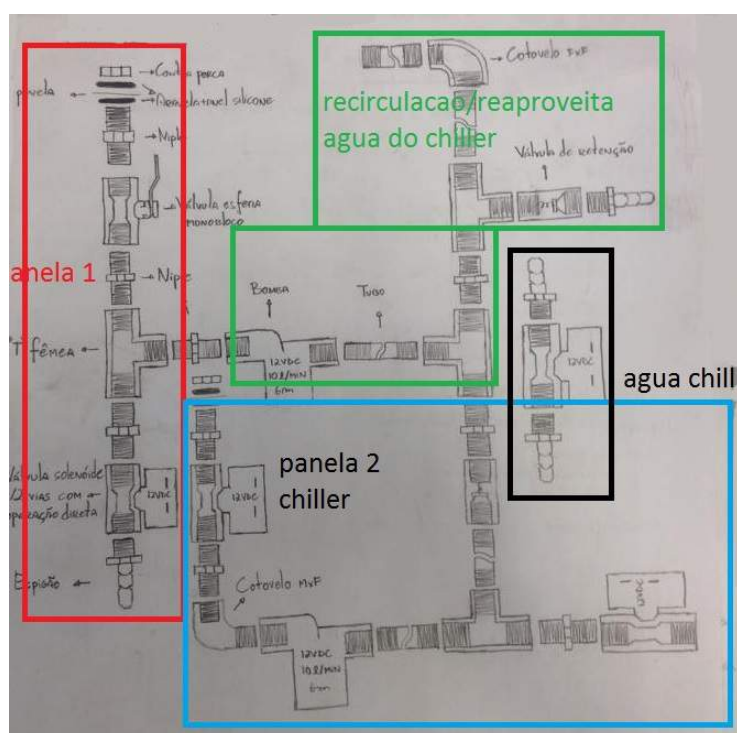


Figura 4.4: Diagrama conceitual das conexões mecânicas

Na figura 4.5 é apresentado um diagrama da MT construído no software FreeCAD, à qual estão conectados o resistor de potência e um niple com vedação. Também estão presentes na figura a válvula esfera manual, um *tee* fêmea, uma válvula Danfoss EV210B e três niples de conexão entre estes componentes. O modelo da válvula foi obtido a partir do website da Danfoss e importado, enquanto a modelagem dos outros componentes foi desenvolvida na plataforma FreeCAD. Na figura 4.6 são apresentadas em detalhe as conexões do resistor e do niple à panela e, por isto, a estrutura da mesma foi omitida. Os itens em amarelo são contra-porcas, em azul arruelas e, em vermelho anéis de vedação (*O-rings*).

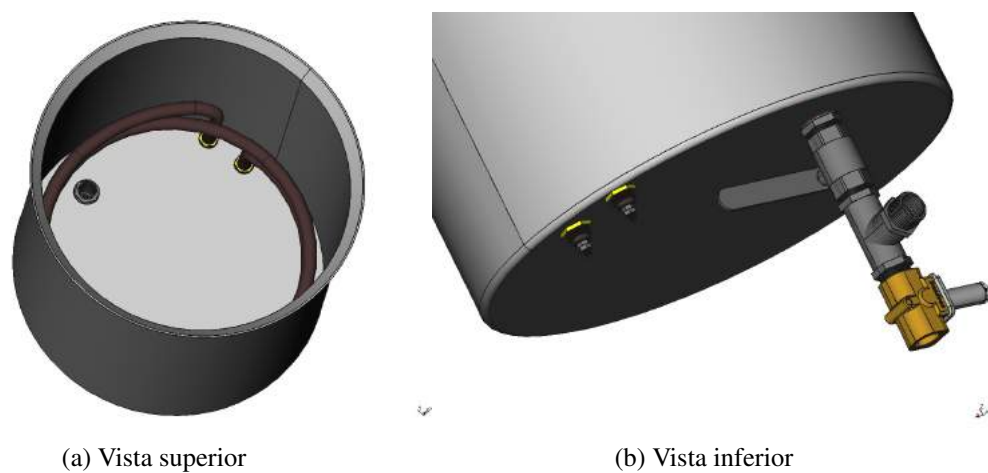


Figura 4.5: Diagrama da panela de mostura e conexões

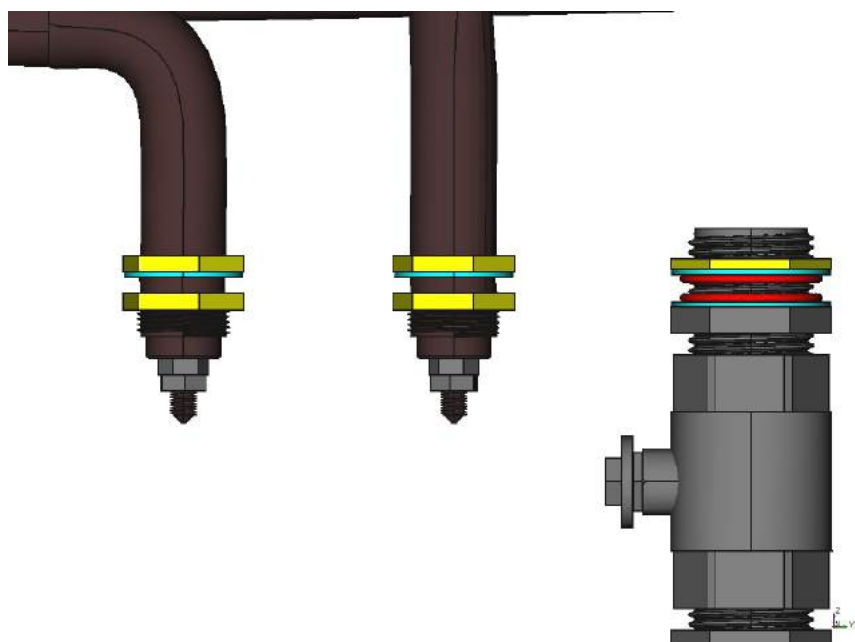


Figura 4.6: Detalhes da vedação da panela de mostura

4.1.3 Estrutura metálica de suporte

Parte importante do projeto é o dimensionamento da estrutura metálica de suporte às painéis, já que esta é responsável não somente pelo alojamento do sistema mecânico como também pela facilidade de manutenção. Para definição da posição das painéis, foram medidas primeiramente as suas alturas e da estrutura de adição de lúpulos (que será detalhada à parte na seção 4.2) e do *chiller*. Na sequência, foi estimado um espaço de manobra entre as duas painéis e uma altura mínima do chão, com base na figura 4.2. Todas as medidas citadas estão dispostas na tabela 4.1.

Tabela 4.1: Dimensões elementos mecânicos relevantes para o projeto da altura da estrutura metálica

Descrição	Altura (cm)
Painel BK	30,0
Painel MT	30,0
<i>Chiller</i>	30,0
Estrutura dos lúpulos aberta	20,0
Vão livre entre painéis	20,0
Vão do <i>chiller</i> ao BK	10,0
Espaço entre MT e topo	10,0

Somando-se todos os valores da tabela 4.1, obteve-se que a estrutura metálica deve ter 1,5m de altura. A largura deve ser de 36,5cm para acomodar as painéis com uma pequena folga e o comprimento escolhido é de 70cm, o que permite alinhar as painéis de maneira não concêntrica, conforme abordado na seção 4.1 — este valor é suficiente, já que o diâmetro somado das duas painéis é de 72cm e é preciso que exista uma intersecção para permitir o escoamento por gravidade da MT ao BK. Na figura 4.7 é apresentado o projeto da estrutura já com as prateleiras para acomodar as painéis. Note-se que foram escolhidas cantoneiras de 2"x 1/8" para a construção deste equipamento.

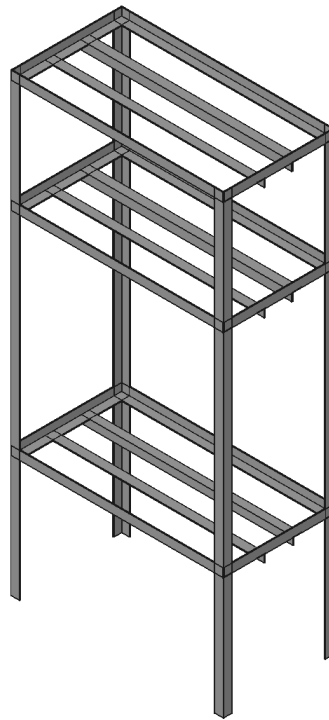


Figura 4.7: Projeto da estrutura metálica

4.2 Estrutura de adição de lúpulos

Devido à necessidade de adicionar lúpulos à receita em proporções e tempos predefinidos, foi necessário o projeto de uma estrutura automática de adição de lúpulos. Em função das inúmeras possibilidades de combinações de lúpulos e temporizações, a escolha natural do projeto foi uma estrutura que permita esta flexibilidade — uma caixa com oito compartimentos separados, que permite a adição sequencial destes insumos. Com base em um pacote de lúpulos em *pellets* de 50g, estimou-se que seu volume é de 160cm^3 e, sabendo-se que uma receita de 20l dificilmente utiliza uma quantidade maior do que 400g de lúpulos, a decisão tomada foi a de que cada um dos 8 compartimentos deveria ter os 160cm^3 necessários para acomodar 50g. Para confirmar que 400g é uma quantidade razoável, foi seguido o cálculo de IBU de [1], detalhado no apêndice J.

Fixando a altura do compartimento em 10,0cm foram escolhidos valores de comprimento e largura de 16,0cm e 8,0cm respectivamente, assumindo inicialmente que as paredes das divisórias tem espessura desprezível. A equação para volume de um prisma e a escolha das dimensões são demonstradas em 4.1:

$$V = l \cdot w \cdot h = V_{50g} \cdot 8 \quad (4.1a)$$

$$= l \cdot w \cdot 10 = 160 \cdot 8$$

$$l \cdot w = 128, \text{fixando } l = 16\text{cm} \quad \therefore$$

$$w = 128/16 = 8\text{cm} \quad e \quad (4.1b)$$

$$w_{\text{compartimento}} = l/8 = 16/8 = 2\text{cm} \quad (4.1c)$$

Posteriormente aos cálculos de 4.1, durante o projeto no software FreeCAD, observou-se que a assumpção de espessura desprezível é falsa e um retrabalho nas medidas do compartimento levou às dimensões externas finais de 18,0cm x 8,4cm x 10,0cm; as paredes externas tendo espessura de 3,0mm e as internas de 2,0mm. A figura 4.8 esboça as dimensões externas da caixa e a figura 4.9 apresenta o corpo da estrutura.

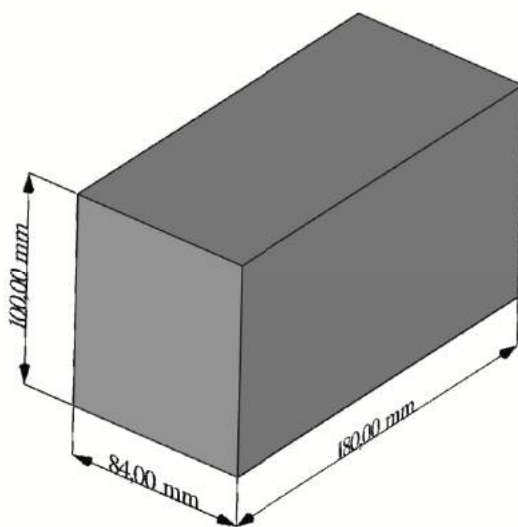


Figura 4.8: Dimensões da estrutura de adição de lúpulos

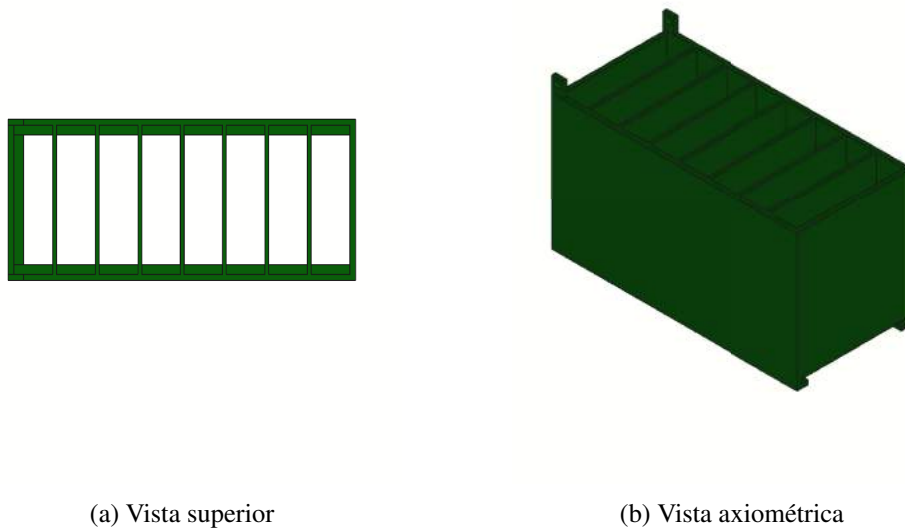


Figura 4.9: Corpo da estrutura de adição de lúpulos

Foram projetadas duas portinholas: uma na face superior da caixa, para reabastecimento manual dos lúpulos, com uma alça e fixada na caixa por dois pinos, de modo que a abertura desta tampa é um movimento de rotação e; uma na face inferior para adição automática, de correr, de modo que a abertura é um movimento de translação; ambas estão ilustradas na figura 4.10.

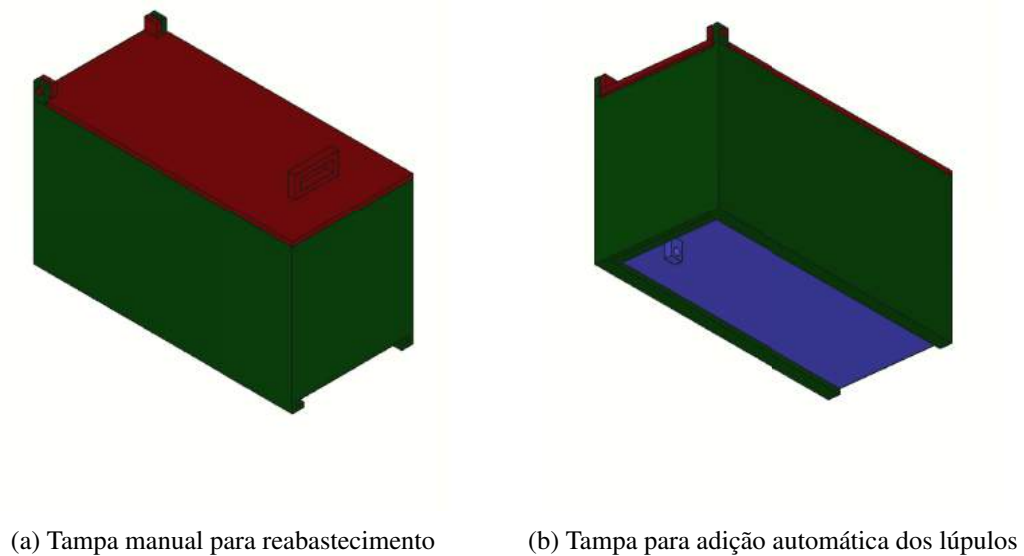


Figura 4.10: Tampas da estrutura de adição de lúpulos

Duas varetas de comprimento l_1 e l_2 são ligadas ao encaixa da tampa de correr, represen-

tada em azul na figura 4.10 (b), e a elas é acoplado um servo-motor cujo ângulo de operação θ define a abertura — ou deslocamento — da tampa, definido por d . A figura 4.11 esquematiza as ligações.

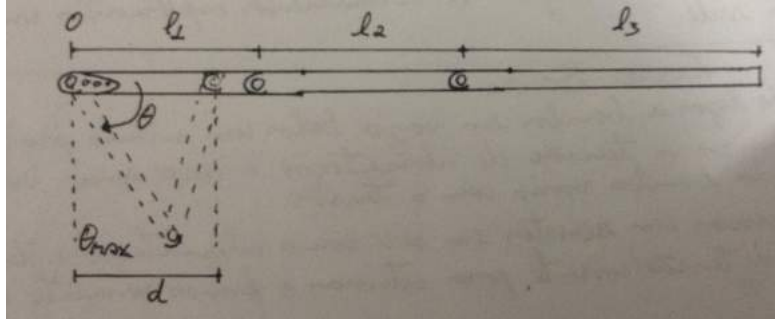


Figura 4.11: Esquema de ligação das varetas à tampa de correr

Sendo o comprimento da tampa igual a l_3 , a condição na equação 4.2 deve ser satisfeita para que a excursão máxima de abertura da caixa possa ser atingida. É possível demonstrar com o auxílio da Lei dos Cossenos, em 4.3 que $l_1 = l_2 = l$ para que a mínima abertura da caixa possa ser atingida.

$$l_1 + l_2 \geq l_3 \quad (4.2)$$

$$l_2^2 = d^2 + l_1^2 - 2dl_1 \cdot \cos(\theta) \quad (4.3)$$

$$\text{se } \theta = 90^\circ \rightarrow d = 0 \therefore$$

$$l_2^2 = l_1^2 \rightarrow l_2 = l_1 = l \quad (4.4)$$

A taxa máxima de deslocamento em função do ângulo do servo motor é expressa em 4.5:

$$\frac{dl_3}{d\theta} = \frac{2l}{90^\circ} \quad (4.5)$$

Entretanto, quanto maior o valor desta taxa, menor é a resolução de abertura da caixa, portanto obtém-se que o menor valor possível de l , a partir da equação 4.2 é $l = l_3/2$. Assim sendo, a taxa de deslocamento calculada a partir da equação 4.5 é de $2mm/^\circ$.

Embora o valor teórico de θ seja 90° , na prática, devido ao encaixe entre tampa e vareta não ser na extremidade, este valor é reduzido. Tal fato não deve ser negligenciado, uma vez que isto pode forçar a operação do servo-motor ou quebrar alguma das peças do sistema. Sabe-se pelo projeto executado no FreeCAD que a distância da borda ao encaixe da tampa é de 2,3cm e portanto, com o auxílio da Lei dos Cossenos, obtém-se que $\theta_{max} = 82^\circ$. Quanto ao valor mínimo não há preocupação, já que a abertura da caixa terá uma excursão reduzida em 2,3cm com relação ao seu comprimento. As varetas não foram calculadas novamente para corrigir esta excursão pois a redução foi considerada uma boa prática, que evita que a tampa se desacople da caixa. A figura 4.12 apresenta o esquema completo de construção da estrutura de adição de lúpulos.

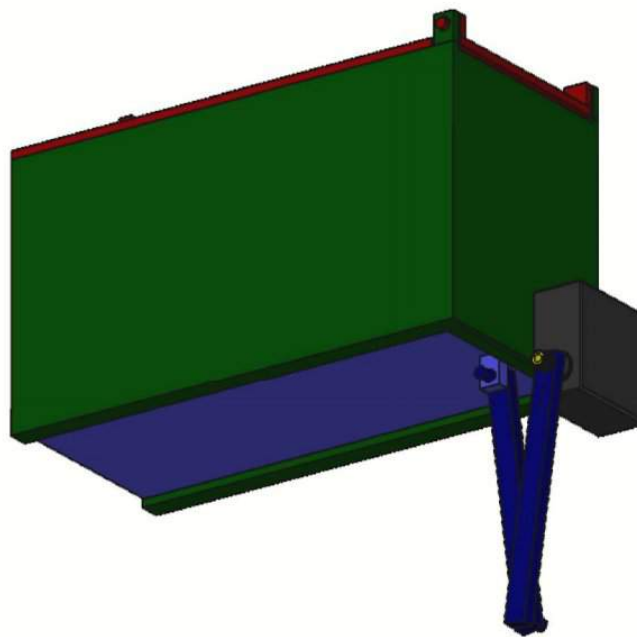


Figura 4.12: Estrutura de adição de lúpulos

4.2.1 Controle do servo-motor

Uma vez que a caracterização do funcionamento do servo-motor foi realizada com o auxílio do microcontrolador AT89S52, esta será descrita no apêndice I.

4.3 Organização do diretório da aplicação

Dado o nível de complexidade a que chegou o desenvolvimento da aplicação de controle do sistema, foi necessário adotar um sistema de divisão de arquivos por função. Dentre os tipos de arquivos há principalmente códigos-fonte, imagens e registros. O objetivo desta seção é explicar a organização adotada.

Cabe primeiramente observar que há duas grandes frentes de projeto com relação a software: a aplicação do servidor e a aplicação do cliente. Uma vez que há momentos em que ambas estão entrelaçadas, foi decidido por não separar os arquivos em pastas *cliente* e *servidor*, mas somente classificar cada arquivo como pertencente principalmente a uma destas duas frentes.

A figura 4.13 apresenta a estrutura reduzida de diretórios do projeto, excluídos diretórios de arquivos temporários e bibliotecas auxiliares, enquanto a tabela 4.2 lista todos os diretórios e arquivos relevantes ao projeto, assim como os descreve brevemente e os distingue entre *cliente* ou *servidor*.

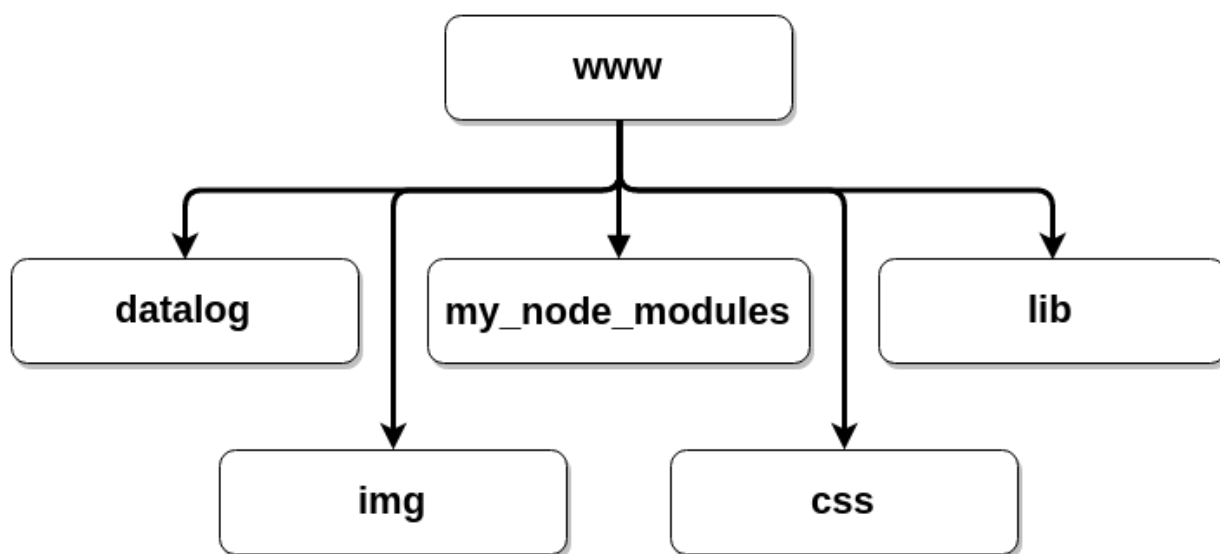


Figura 4.13: Estrutura de diretórios da aplicação da BBB

Tabela 4.2: Lista de diretórios e arquivos da aplicação da BBB

Diretório/arquivo	Descrição	Frente
www	diretório raiz	
about.php	página de descrição do projeto	cliente
control.php	GUI de controle do sistema	cliente
controle.js	arquivo raiz da aplicação do servidor	servidor
dyn_graph2.html	gráfico dinâmico de linha/área	cliente
index.html	página inicial - contém somente uma imagem clicável	cliente
listreceipe.php	gerenciador de receitas - permite criar, editar e excluir	cliente
newreceipe.php	GUI de edição de receitas	cliente
settings.php	GUI de configurações não referentes à brassagem	cliente
startreceipe.php	escolha da receita e início da brassagem	cliente
stats.php	estatísticas diversas de produções anteriores	cliente
www/css	configurações de estilo da GUI	
buttons.css	estiliza alguns botões da GUI	cliente
config.css	configurações gerais de estilo da GUI	cliente
form.css	estilo do formulário de edição de receitas	cliente
www/datalog	registros e backup de dados em memória não volátil	
backup.log	registro detalhado da última brassagem	servidor
default.csv	registro de temperaturas e tempos do DS18B20	servidor
instant.csv	última leitura do DS18B20 da MT	servidor
instant_bk.csv	última leitura do DS18B20 do BK	servidor
lockfile	arquivo que define se há uma brassagem em execução	servidor
www/img	imagens geradas para a GUI	
beer2.png	favicon	cliente
beer_compiler.png	imagem da página inicial da GUI	cliente
tplot.png	gráfico estático da temperatura em função do tempo	cliente
www/my_node_modules	módulos em Node.js desenvolvidos neste projeto	
ctrl.js	implementação do algoritmo de controle da brassagem	servidor
gpio_cfg.js	configuração e controle de GPIO e PWM	servidor
log_check_misc.js	verificação de integridade de receita. Leitura e registros de informações referentes à brassagem	servidor
routes.js	rotas do servidor Express.js	servidor
www/lib	módulos requisitados por códigos-fonte do cliente	
deleterecipe.php	deleta uma receita do servidor	servidor
header.js	auxiliar de criação do cabeçalho da GUI	cliente
header.php	criação do cabeçalho da GUI	cliente
listreceipe.js	lê conteúdo ou deleta receita do servidor	servidor
newreceipe.js	organiza e salva automaticamente a GUI de edição de receitas	cliente / servidor
newreceipe.php	salva ou carrega receita para a GUI de edição de receitas	servidor
previewreceipe.php	lê conteúdo essencial da receita e devolve para a GUI	cliente / servidor
log	registros de erros de aplicações da BBB	
recipes	receitas de cerveja	
node_modules	módulos do Node.js instalados por meio do NPM	
d3-3.4.10	biblioteca Javascript de visualização de dados	
rickshaw	API de geração de gráficos baseada na biblioteca D3.js	
tmp	diretório para testes e arquivos temporários	

4.4 IDE e sistema de controle de versão

Para o desenvolvimento de software deste projeto, foram empregadas as ferramentas *Cloud9* e *GIT*, que são uma IDE e um sistema de controle de versão, respectivamente. Informações sobre estas ferramentas e seu uso no projeto estão contidas no apêndice D.

4.5 Geração de gráfico e registro de temperatura em Python

Para o armazenamento da temperatura do sensor DS18B20 em função do tempo, assim como a geração de um gráfico com o histórico de temperaturas para o usuário do sistema, foram escritos dois *scripts* em Python, sendo a saída do *script* de *log* a entrada do *script* de geração do gráfico — embora ambos possam ser executados independentemente, a geração do gráfico precisa de um arquivo contendo o histórico de temperaturas.

4.5.1 Registro de temperatura

Para o arquivo do *log*, foram primeiramente escritas algumas funções:

- **tread()** — lê o arquivo referente ao sensor de temperatura e retorna deste somente a temperatura formatada em graus celsius.
- **tprint_all(tcelsius)** — recebe temperatura em °C e imprime com formatação HTML em °C, °F, K e °R; pode ser usada em conjunto com a função *tread()*
- **tprint_all_terminal(tcelsius)** — idêntica à função *tprint_all()*, porém imprime com formatação para o terminal do Linux
- **tlog(file = "/var/www/default.csv")** — cria/adiciona a um arquivo de log no formato CSV a temperatura medida e a data/hora correspondente no formato *Unix epoch* — que é definido como o número de segundos passados desde 1 de janeiro de 1970 não considerando segundos bissextos — em intervalo de amostragem definido na função, até que o script seja interrompido de alguma maneira. Em função de erros de leitura esporádicos do sensor, se a temperatura lida for menor ou igual a zero, esta é descartada. Imprime a temperatura lida a cada amostragem. O tempo de amostragem é definido pela soma do tempo de leitura do arquivo (inerente ao sistema) com o tempo ocioso definido nesta função: para que o tempo de amostragem seja o mais próximo possível de 1s, foi

realizado um estudo de caso, descrito no apêndice K, e com isso obteve-se o valor de 0,2202s ocioso definido nesta função. Além disto, nesta função é chamada *tlog_instant* que registra em um arquivo separado a última leitura válida de temperatura e tempo.

- **tlog_instant(temperature, epoch, file = "/var/www/datalog/instant.csv")** — registra a última leitura válida de temperatura e tempo em um arquivo, criando ou sobrescrevendo o mesmo.
- **tlog_test(file = "/var/www/datalog/default.csv")** — função que gera um arquivo de *log* com rampas e degraus de temperatura, para uso em simulações posteriores do funcionamento do sistema, descritas na seção 4.6.5.

Sendo a função *tlog* a mais relevante, esta pode ser obtida na caixa de código-fonte 4.1. O script completo está registrado no código-fonte E.1 do apêndice E. Ainda assim, as funções por si só não são executadas sozinhas, portanto foi criado um script auxiliar em Python que chama a função *tlog*, descrito no código-fonte 4.2.

```

1 def tlog(file = "/var/www/datalog/default.csv"):
2     """salva temperatura e Unix Time em .csv """
3     tsample = 0.2202 #amostra a cada ~1 segundo
4     buff_temp = tread() #guarda o último valor lido
5     exist = os.path.isfile(file)
6     with open(file, 'a', 1) as log:
7         if exist is False: #se vai criar o arquivo agora
8             log.write("temperatura,data\n")
9         while True: #loop infinito
10             temp_celsius = tread()
11             epoch = time.time() #lê a data/hora do sistema
12             if temp_celsius >= 0: #evita leitura errada
13                 log.write("%f,%f\n" % (temp_celsius, epoch))
14                 tlog_instant(temp_celsius, epoch)
15                 time.sleep(tsample) #espera n segundos
16                 print "registrando temperatura!", temp_celsius

```

Código-fonte 4.1: Função de *log* da temperatura em Python

```

1 import temp
2 temp.tlog()
3 # para executar no terminal, basta executar:
4 # sudo python log.py

```

Código-fonte 4.2: Script para gravação das leituras de temperatura em Python

4.5.2 Gráfico de temperatura

Com o script que gera um registro da temperatura medida pelo sensor DS18B20 em função do tempo, o próximo passo é a análise dos dados coletados, sendo uma das ferramentas de análise a plotagem de um gráfico. Para fazê-lo em Python, foi decidido usar as bibliotecas *Matplotlib* e *NumPy*, integrantes do ecossistema *SciPy*.

Matplotlib é uma biblioteca desenvolvida especificamente para a plotagem 2D de figuras estáticas em scripts Python, no Python shell (similar ao ambiente interpretador de *softwares* proprietários como MATLAB e Mathematica), em aplicações de servidores e *toolkits* com GUI [34]. Para instalar esta biblioteca, bastou executar o comando descrito na caixa de código-fonte 4.3.

```
1 sudo apt-get install python-matplotlib
```

Código-fonte 4.3: Instalação da biblioteca Matplotlib

NumPy é um pacote em Python que fornece suporte a arranjos e matrizes multidimensionais, além de diversas funções matemáticas de suporte a estes formatos de dados. É amplamente empregado nas áreas de computação científica, engenharia e matemática [35]. Este pacote é parte integrante do ecossistema *SciPy*, cuja instalação é apresentada na caixa de código-fonte 4.4.

```
1 sudo apt-get install python-scipy
```

Código-fonte 4.4: Instalação do ecossistema SciPy

Desenvolvimento do gráfico

Com todas as ferramentas necessárias instaladas, pôde ser iniciado o desenvolvimento do script que gera o gráfico da temperatura em função do tempo. A abordagem inicial foi criar um script que gera um arquivo PDF do gráfico a partir do exemplo encontrado na página http://matplotlib.org/examples/pylab_examples/multipage_pdf.html, modificando este script para as necessidades do projeto.

Como o desenvolvimento foi realizado sem uma interface gráfica, referida como *backend*, foi preciso explicitar a saída como um renderizador, conforme indicado no código-fonte 4.5, que documenta a importação das bibliotecas/funções utilizadas no script. Note-se que a mudança da saída ocorreu necessariamente antes da importação da biblioteca *numpy* para que não ocorresse erro.

```
1 import time
2 import matplotlib
3 matplotlib.use('AGG') #muda a output para um renderer ao inves
  de backend
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from scipy import interpolate
```

Código-fonte 4.5: Importação das bibliotecas para geração de gráfico em Python

Na única função do script, *graph_gen(file = '/var/www/default.csv', graph = '/var/www/plot.png')*, foi explicitado que a entrada é um arquivo CSV e a saída um arquivo PNG. Nesta

função o arquivo CSV foi lido linha por linha em um *loop*, sendo a primeira linha do arquivo ignorada em função de ela não conter dados, conforme ilustrado na figura 5.1. Ao longo das iterações deste *loop*, a data/hora do primeiro e último registro de temperatura foram usadas para gerar um título para o gráfico e, a cada iteração os dados são decodificados, o que se resume basicamente a separar o que há antes e depois da vírgula presente na linha lida do arquivo CSV.

Na parte da função referente à plotagem do gráfico, além de configurações triviais como cor de fundo, tamanho de fontes e legendas, dentre outras, é notável o ajuste da escala do tempo, para o qual foi pensado um registro de temperatura contínuo e não somente o registro de uma produção de cerveja: para até duas horas de registro a escala é impressa em minutos; entre duas horas e um dia e meio de registros, em horas; entre um dia e meio e três meses de registros, em dias; e acima disto em meses. Após isto, o gráfico é plotado, salvo no formato PNG com densidade de pontos por polegada de 150dpi, e fechado para liberar a memória do sistema.

O script que chama a função *graph_gen* está presente no mesmo arquivo em que está a função. Note-se que o gráfico é gerado periodicamente a cada 300 segundos e o tempo necessário para executar tal tarefa é computado e impresso na saída padrão do Python. O arquivo completo que contém a função *graph_gen* está documentado no código-fonte E.2 do apêndice E.

4.6 Aplicação *server-side* em Node.js

Nesta seção será detalhado o uso da linguagem de programação Javascript no ambiente de interpretação Node.js. O escopo da aplicação desta linguagem no projeto se aplica, mas não é limitado a: acesso ao *hardware*, incluindo barramentos de propósito geral (GPIO) e periféricos como PWM; criação de um servidor web minimalista; e controle do processo de produção de cerveja. Node.js já vem instalado na distribuição padrão do Debian utilizada neste projeto, porém é possível instalar ou atualizar o ambiente por meio do gerenciador de pacotes *apt-get*, conforme descrito na caixa de código-fonte 4.6:

```
1 sudo apt-get install nodejs
```

Código-fonte 4.6: Instalação e/ou atualização do Node.js

Com isto, o gerenciador de pacotes do Node chamado NPM (Node Package Manager) também é instalado no sistema. Neste projeto, o desenvolvimento em Node foi legado do

desenvolvimento prévio em PHP e, por isto, tudo foi realizado no diretório `/var/www`, cujo dono é o usuário `www-data`. Em função disto, foi criada manualmente a pasta `/var/www/node_modules` e o dono foi trocado para o usuário `debian` (enquanto sob condições normais este processo seria automático). Para evitar este problema, os módulos poderiam ser instalados globalmente, porém por questões de segurança, esta foi a abordagem realizada, uma vez que os módulos do Node ficam autocontidos no diretório do projeto. O processo de criação do diretório `/var/www/node_modules` é descrito no trecho de código 4.7:

```
1 cd /var/www
2 sudo mkdir node_modules
3 sudo chown debian:debian node_modules
```

Código-fonte 4.7: Criação do diretório `/var/www/node_modules`

Além disto, foi instalado o módulo *Debug* do Node, que ajuda no desenvolvimento da aplicação da seguinte maneira: ele encapsula a função nativa do Javascript `console.log`, análoga ao `printf` da linguagem C, de tal maneira que as mensagens só são impressas na tela se a variável `DEBUG` estiver configurada no terminal durante a execução da aplicação. Além disto, diferentes grupos de mensagens de *debug* podem ser configurados, permitindo flexibilidade. Para diferenciar estes grupos, o módulo adota um esquema de cores que facilita a compreensão das mensagens de *debug*.

Para ilustrar seu uso, na caixa 4.8 é descrita a instalação do módulo; o trecho de código-fonte 4.9 descreve como usar o módulo em uma aplicação de Node e a figura 4.14 apresenta o resultado do exemplo, para as seguintes situações: não imprime mensagens de debug; imprime grupos de mensagens de debug e; imprime todas as mensagens de debug disponíveis.

```
1 npm install debug
```

Código-fonte 4.8: Instalação do módulo *Debug*

```
1 "use strict";
2
3 var dbg_gpio = require('debug')('gpio');
4 var dbg_pwm = require('debug')('pwm');
5
6 console.log("Inicio exemplo:");
7 for(var i = 0; i < 3; i++){
8     switch(i){
9         case 0:
10             dbg_gpio("Debug situacao gpio");
11             break;
12         case 1:
13             dbg_pwm("Debug situacao pwm");
14             break;
15         case 2:
16             dbg_gpio("Debug ambos gpio");
17             dbg_pwm("Debug ambos pwm");
```

```

18     break;
19 }
20 }

```

Código-fonte 4.9: Exemplo de uso do módulo *Debug*

```

leonardo@grabaDev:~/final_paper_tcc/tmp$ node testenode.js
Início exemplo:
leonardo@grabaDev:~/final_paper_tcc/tmp$ DEBUG=gpio node testenode.js
Início exemplo:
  gpio Debug situacao gpio +0ms
  gpio Debug ambos gpio +2ms
leonardo@grabaDev:~/final_paper_tcc/tmp$ DEBUG=pwm node testenode.js
Início exemplo:
  pwm Debug situacao pwm +0ms
  pwm Debug ambos pwm +2ms
leonardo@grabaDev:~/final_paper_tcc/tmp$ DEBUG=gpio,pwm node testenode.js
Início exemplo:
  gpio Debug situacao gpio +0ms
  pwm Debug situacao pwm +3ms
  gpio Debug ambos gpio +1ms
  pwm Debug ambos pwm +0ms
leonardo@grabaDev:~/final_paper_tcc/tmp$ DEBUG=* node testenode.js
Início exemplo:
  gpio Debug situacao gpio +0ms
  pwm Debug situacao pwm +3ms
  gpio Debug ambos gpio +2ms
  pwm Debug ambos pwm +0ms

```

Figura 4.14: Exemplo de uso do módulo *Debug*

Neste capítulo deve-se assumir que todas as operações realizadas e arquivos referidos estão no diretório `/var/www`, salvo indicações em contrário. Note-se também que, no início dos códigos-fonte completos, presentes no apêndice F, é usada a declaração `"use-strict"`, que evita o uso de potenciais armadilhas do Javascript, e.g. usar inadvertidamente uma variável global sem declará-la.

4.6.1 Acesso a GPIO e PWM

O acesso às características de *hardware* da BBB necessárias para o controle das válvulas, bombas, resistores de potência e servo-motor do sistema é feito por meio do módulo de código-aberto *Octalbonescript*, que consiste de um *fork* (ou derivação) do módulo oficial da BBB de acesso a hardware, chamado *Bonescript*. Sua escolha se deve ao fato de que esta biblioteca era mais completa e estável durante o desenvolvimento deste projeto do que aquela que lhe deu origem. Sua instalação está descrita na caixa 4.10. A documentação da API deste módulo foi obtida em <https://github.com/theoctal/octalbonescript/wiki>.

```
1 npm install octalbonescript
```

Código-fonte 4.10: Instalação do Octalbonescript

Uma vez que este módulo compila uma sobreposição de *device tree* a partir de um template fornecido junto com a instalação, foi necessário modificá-lo para acomodar a descrição de hardware do sensor DS18B20. Para tal, foi investigado que o comportamento da biblioteca *Octalbonescript* com relação à *device tree* é o seguinte: se a sobreposição já está carregada não é feita nova compilação do fragmento nem tentativa de carregá-la novamente, portanto foi necessário resetar a BBB toda vez que novos testes eram realizados.

O template editado fica no diretório *node_modules/octalbonescript/dts/* e o nome do arquivo modificado é *OBS_UNIV_template.dts*. As modificações realizadas, assim como as linhas do código modificadas estão descritas na caixa de código-fonte 4.11. Linhas que comecem com "X" foram apagadas, as outras foram adicionadas.

```
1 [389]          /* P9_11 (THE ONE-WIRE DS18B20) */
2 [390]          bb_wl_pins: pinmux_bb_wl_pins {
3 [391]              pinctrl-single,pins = <0x070 0x37>; };
4          /* Mode 7, Pull-Up, RxActive */
5 X[392]          P9_11_gpio_pin: pinmux_P9_11_gpio_pin {
6 X[393]              pinctrl-single,pins = <0x070 0x2F>; };
7          /* Mode 7, RxActive */
8 X[394]          P9_11_gpio_pu_pin: pinmux_P9_11_gpio_pu_pin {
9 X[395]              pinctrl-single,pins = <0x070 0x37>; };
10          /* Mode 7, Pull-Up, RxActive */
11 X[396]          P9_11_gpio_pd_pin: pinmux_P9_11_gpio_pd_pin {
12 X[397]              pinctrl-single,pins = <0x070 0x27>; };
13          /* Mode 7, Pull-Down, RxActive */
14 X[398]          P9_11_uart_pin: pinmux_P9_11_uart_pin {
15 X[399]              pinctrl-single,pins = <0x070 0x36>; };
16          /* Mode 6, Pull-Up, RxActive */
17 [879]          onewire@0 {
18 [880]              compatible = "w1-gpio";
19 [881]              status = "okay";
20 [882]              pinctrl-names = "default";
21 [883]              pinctrl-0 = <&bb_wl_pins>;
22 X[884]              pinctrl-1 = <&P9_11_gpio_pin>;      gpios = <&
23          gpio1 30 0>;
24 X[885]              pinctrl-2 = <&P9_11_gpio_pu_pin>;
25 X[886]              pinctrl-3 = <&P9_11_gpio_pd_pin>;
26 X[887]              pinctrl-4 = <&P9_11_uart_pin>;
27 [888]          };
28          P9_11 {
29 X[1232]              gpio-name = "P9_11";
30 X[1233]              gpio = <&gpio1 30 0>;
31 X[1234]              input;
32 X[1235]              dir-changeable;
33 X[1236]          };
34 X[1237]
```

Código-fonte 4.11: Modificações feitas ao template de *device tree* do Octalbonescript

Com esta configuração da *device tree*, foi preciso parar de carregar o fragmento descrito na seção C.0.5, pois ao tentar usar o *Octalbonescript*, passou a ocorrer um erro do tipo **EEXIST: file already exists**, que indica em um âmbito geral que um arquivo que deveria ser criado já existe.

A partir deste ponto, foi desenvolvido um módulo em Node nomeado *gpio_config* responsável pelo gerenciamento de GPIO e PWM, baseado no *Octalbonescript*. Foram criados objetos para cada elemento do sistema, usados para guardar a identificação do pino de GPIO de cada objeto, o estado e a configuração de cada pino. Para isto, uma função auxiliar foi criada.

Também foi criado um objeto que reúne todos os objetos dos pinos, para facilitar a configuração inicial e também manter controle sobre o status do sistema durante seu funcionamento. Uma vez que em Node os objetos são todos ponteiros, ao atribuir um objeto a outro, o que se está fazendo é apontar os dois objetos para o mesmo endereço de memória. A descrição completa das declarações de objetos pode ser encontrada no código-fonte F.2 no apêndice F.

Além disto, foi criado um objeto que guarda o status do sistema: o número total de pinos a serem configurados, o número de pinos configurados corretamente, o número de pinos como GPIO, PWM, analógicos e interrupções. Quanto às funções do módulo, estas são:

- **changeStatusIO(pin, val)** — recebe o nome do pino, e.g. *mash_heat*, e o valor (*true* ou *false* caso GPIO; numérico caso PWM). Ajusta o valor do pino conforme passado para a função, assim como trata de atualizar as variáveis de controle dos pinos.
- **getSystemStatus()** — retorna o valor/estado de todos os pinos do sistema, sendo 0 ou 1 para GPIO e numérico para PWM.
- **pinsConfiguration()** — configura todos os pinos declarados como entrada, saída ou PWM. Em caso de erro de configuração de algum pino é impressa uma mensagem com o auxílio do módulo *debug*, portanto cabe ao administrador do sistema testar se a configuração está funcionando no momento da instalação do equipamento ou quando julgar necessário (o programa não pára de funcionar em caso de erro do programador).
- **ioTest()** — função de teste. Implementa um algoritmo que chaveia os pinos de GPIO do sistema de tal forma sequencial, ou seja, chamar esta função periodicamente em um *loop* faz com que um efeito cascata seja produzido, caso os pinos estejam ligados a LEDs.

Na função *pinsConfiguration*, é preciso notar que dentro do laço de repetição *for* é cha-

mada uma função anônima para a configuração de cada pino. Isto é necessário pois, se todas as operações fossem realizadas diretamente dentro do laço, o comportamento assíncrono do Node geraria inconsistências na execução do código. Assim sendo, o objetivo da função anônima é criar um escopo que permite a execução assíncrona do código sem que a referência passada para esta função anônima se degenere.

Uma vez que esta função precisa receber o valor e não o ponteiro da variável de iteração do laço, tanto o pino a ser configurado quanto o seu índice (variável de iteração) no laço de repetição são passados com o auxílio do método *call()*, que neste caso copia o pino como o elemento *this* do Javascript e o índice como o primeiro argumento da função, criando um novo escopo por meio de uma clausura. Sem isso, antes que a primeira invocação da função anônima acabasse, o valor de *i*, por exemplo, já seria *all_io_objects.length* e não zero, que seria o valor esperado.

Para tornar a compreensão desta necessidade mais clara, a caixa de código-fonte 4.12 apresenta um exemplo no qual duas funções com um laço *for* são chamadas: a primeira delas imprime o valor da variável de iteração para todas as chamadas do laço imediatamente e com um atraso de 100ms, mas sem precauções com relação ao comportamento assíncrono do Node; já a segunda função utiliza o método descrito no parágrafo anterior. Observa-se na figura 4.15 que sem tomar precauções, na chamada com atraso, é impresso o último valor da variável de iteração do laço em todas as iterações. Isto deixa claro a necessidade de criar um novo escopo, assim como demonstra que esta é uma fonte de erros para programadores inexperientes e/ou descuidados.

```

1  'use-strict';
2  var debug1 = require('debug')('debug1');
3  var debug2 = require('debug')('debug2');
4
5  function pinConfiguration() {
6      for(var i = 0; i < 3; i++){
7          setTimeout(function() { debug1('atraso: ' + i
8              ); }, 100);
9          debug1('imediato: ' + i);
10     }
11 }
12
13 function pinConfiguration2() {
14     for(var i = 0; i < 3; i++){
15         (function () {
16             var i2 = this;
17             setTimeout(function() { debug2('
18                 atraso: ' + i2);}, 100);
19             }).call(i);
20         debug2('imediato: ' + i);
21     }
22 }

```

```

20 }
21
22 pinConfiguration2 ();
23 pinConfiguration ();

```

Código-fonte 4.12: Uso de clausura para criação de escopo

```

leonardo@grabaDev:~/final_paper_tcc/tmp$ DEBUG=* node testeclosure.js
debug2 imediato: 0 +0ms
debug2 imediato: 1 +3ms
debug2 imediato: 2 +2ms
debug1 imediato: 0 +0ms
debug1 imediato: 1 +1ms
debug1 imediato: 2 +0ms
debug2 atraso: 0 +96ms
debug2 atraso: 1 +2ms
debug2 atraso: 2 +0ms
debug1 atraso: 3 +2ms
debug1 atraso: 3 +0ms
debug1 atraso: 3 +0ms

```

Figura 4.15: Resultado de chamada de função sem criação de escopo x com criação de escopo

4.6.2 Servidor Express.js

Mesmo com o servidor Apache configurado por padrão na BBB, quando foi decidido o uso de Node.js como linguagem do servidor o Apache foi substituído pelo *framework* Express, dados os benefícios apontados na seção 2.4. Para isto, a consulta à documentação da API deste *framework* (<http://expressjs.com/pt-br/4x/api.html>) foi a base de desenvolvimento do servidor.

Outros dois módulos auxiliares foram empregados em conjunto com o Express: o módulo *Body-parser*, cuja função neste projeto é a interpretação de dados codificados em JSON pelo cliente; e o módulo *PHP-express*, que permite a interpretação de arquivos PHP pelo servidor. Este último módulo foi empregado devido a algumas funcionalidades desenvolvidas em PHP em uma fase inicial do projeto, portanto suportando a aplicação legada. Reescrever os códigos PHP em Javascript é uma opção, porém em função do tempo necessário para tal tarefa, a solução adotada foi executar os códigos legados. Na caixa 4.13 é documentada a instalação dos módulos Express, *Body-parser* e *PHP-express*.

```

1 npm install express
2 npm install body-parser
3 npm install php-express

```

Código-fonte 4.13: Instalação dos módulos necessários para implementar o servidor web em Node.js

O código-fonte no qual é configurado o servidor é o */var/www/controle.js*, que é também o arquivo raiz da aplicação em Node, e está disponível no apêndice F.1. É interessante observar

a configuração do PHP-express, descrita no trecho de código-fonte 4.14, no qual observa-se que logo ao adicionar o módulo ao código é passado o caminho do PHP, que deve estar previamente instalado no sistema. Também é notável o fato de que os arquivos PHP são identificados por meio de uma expressão regular, cuja função à qual ela é aplicada direciona estes arquivos para o *engine* do PHP.

```

1 var phpExpress = require('php-express')({ // assumes php is
    in your PATH
2   binPath: 'php'
3 });
4
5 // set view engine to php-express
6 app.set('views', './');
7 app.engine('php', phpExpress.engine);
8 app.set('view engine', 'php');
9
10 // routing all .php file to php-express
11 app.all(/.+\.php$/, phpExpress.router);

```

Código-fonte 4.14: Configuração do PHP-express

Com relação à configuração do Express, três pontos devem ser notados: o diretório no qual é executada a aplicação é configurado como o diretório raiz do servidor, seja para servir arquivos HTML e CSS ou arquivos de mídia e apoio da aplicação; o endereço de IP e porta são especificados de modo a permitir acesso à BBB pela internet e; são configuradas rotas para diferentes categorias de requisições do cliente. O fragmento de código-fonte 4.15 apresenta os três pontos discutidos neste parágrafo.

```

1 var express = require("express");
2 var app = express();
3 var routes = require('./my_node_modules/routes.js');
4
5 app.use(express.static(__dirname)); //add the directory where
    HTML and CSS files are
6 var server = app.listen(8587, "192.168.1.155", function () {
    //listen at the port and address
7   var host = server.address().address;
8   var port = server.address().port;
9   var family = server.address().family;
10  debug('Express server listening at http://%s:%s %s', host,
    port, family);
11 });
12
13 app.route('/controle') //used to unite all the request types
    for the same route
14 .post(routes.controleRoute);
15
16 app.route('/startrecipe') //used to unite all the request types
    for the same route
17 .post(routes.startrecipeRoute);
18
19 app.route('/config') //used to unite all the request types for

```



```

    the same route
20 .post(routes.configRoute);
21
22 app.route('/clientrequest') //used to unite all the request
    types for the same route
23 .post(routes.clientrequestRoute);

```

Código-fonte 4.15: Configuração do servidor web com o *framework* Express.js

Note-se que é adicionado o módulo *routes*, que foi escrito separadamente, somente para facilitar a compreensão do código, embora seja ele que define o comportamento das rotas. Quanto a ele, seu código-fonte é apresentado na caixa de código-fonte do apêndice F.3. Ele consiste de quatro funções — uma para cada rota:

- **controleRoute(req, res)** — realiza o controle direto de GPIO e PWM. Faz uso do módulo *gpio_cfg*, descrito na seção 4.6.1.
 - Muda o valor de um terminal de GPIO ou PWM, conforme especificado pelo cliente.
 - Retorna para o cliente o status de todos os pinos da aplicação, independente da sua configuração.
- **startRecipeRoute(req, res)** — executa diversas funções relacionadas à produção de cerveja, inclusive iniciar o processo de controle da mesma, conforme a requisição enviada pelo cliente.
 - Retorna a lista de receitas cadastradas no servidor.
 - Verifica integridade de uma receita, ou seja, se ela contém informações suficientes para que uma produção de cerveja seja iniciada.
 - Realiza uma série de verificações e começa uma brassagem, se aprovado.
 - Verifica se já existe alguma receita em andamento.
 - Retorna erro para o cliente se nenhuma requisição é válida.
- **configRoute(req, res)** — ajusta ou responde com a data e hora da BBB, conforme a solicitação do cliente. Só permite que este ajuste seja realizado se o serviço de ajuste automático de data/hora não estiver funcionando por algum motivo, e.g. problema de conexão com a internet.
- **clientrequestRoute(req, res)** — função auxiliar do processo de brassagem, usada para sinalizar que uma nova fase do processo deve começar, após autorização do operador/usuário.

Dentre as particularidades de implementação deste módulo, observa-se o uso dos argumentos *req* e *res* em todas as funções, uma vez que estes são padronizado pela API do Express para requisições do tipo POST feitas pelo cliente. O argumento *req* contém todos os dados enviados pelo cliente no padrão HTTP POST, que é parte da transferência de estado representacional, conhecida como REST. Poderia ser o padrão HTTP GET, por exemplo, porém neste projeto o POST foi escolhido para que o usuário não possa adicionar parâmetros por meio da URL.

Uma vez que o módulo Express está em conformidade com as restrições REST, ele é considerado um serviço *RESTful*. Quanto ao argumento *res*, este contém o objeto responsável por enviar a resposta do servidor para o cliente após o processamento da requisição, que é conseguida por meio método *send*. É importante observar que, neste projeto, todas as requisições do cliente e respostas do servidor foram feitas usando a codificação de dados JSON.

O exemplo expresso na caixa de código-fonte 4.16 demonstra o uso dos argumentos *req* e *res* descritos nesta seção. Foi assumido que são recebidos os dados *comando* e *valor*. Se o comando for "inverter", é chamada a função que inverte um botão, cuja resposta pode ser *errol* ou *sucesso*. Ainda, se o comando for diferente de "inverter", a resposta é *erro2*.

```

1 function processaRequisicaoPost(req, res) {
2   var comando = req.body.comando;
3   var botao = req.body.valor;
4
5   if(comando == "inverter"){
6     inverteBotao(botao, function(err){
7       if (err) res.send({'resposta':"errol"}); return;
8       res.send({'resposta':"sucesso"});
9     });
10  }
11  else res.send({'resposta':"erro2"});
12 };

```

Código-fonte 4.16: Exemplo de uso dos argumentos *req* e *res* definidos pelo Express.js

Também é importante entender o uso dos módulos *fs* e *child_process* incluídos por padrão no Node.js e cuja documentação da API pode ser obtida no site do Node.js (<https://nodejs.org/dist/latest-v4.x/docs/api/>). O módulo *fs* implementa o acesso ao sistema de arquivos, com funções de leitura e escrita a arquivos, mudança de dono ou modo de execução de arquivos e diretórios, criação e leitura de diretórios, dentre outras. Quanto à leitura e escrita de arquivos, esta pode ser realizada de forma assíncrona ou síncrona — usuários que ainda estão no processo de aprendizado de orientação a eventos tendem a usar a versão síncrona, porém isto deve ser evitado, uma vez que faz o bloqueio da execução da

aplicação enquanto o arquivo não acabar de ser acessado.

Já o módulo *child_process* permite a criação de processos filhos do Node, que são executados independentemente deste: basicamente é possível executar uma linha de comando ou script do *shell* independente do Node. Neste módulo, foi utilizada a função *exec* para ajustar a data e hora da BBB após receber uma requisição do cliente, por meio dos comandos **date -s** e **hwclock -w**, que ajusta a data e hora do sistema e sincroniza com o RTC, respectivamente. A implementação desta funcionalidade pode ser verificada na função *configRoute* do código-fonte F.3, encontrado no apêndice F.

4.6.3 Registros e verificações em geral

Para as funções de registro, ou *log*, e de verificações, além de eventuais funções de miscelânea, foi escrito um módulo separado, cujo código-fonte está documentado na caixa F.4 do apêndice F. As funções e suas respectivas descrições são apresentadas:

- **startTemperatureLogging()** — inicia o script Python descrito na seção 4.5.1, por meio de um processo filho do Node.
 - Tenta deletar o arquivo que contém a última leitura de temperatura (*/var/www/datalog/instant.csv*) antes de iniciar o script em Python.
 - Define o comportamento caso o script pare de ser executado: imprime uma mensagem de debug.
- **stopTemperatureLogging()** — aborta a execução do script Python iniciado por *startTemperatureLogging*.
- **getTemperatureReading(logFilePath, logReadHandler, callback)** — lê a mais recente amostragem de temperatura, que está salva no arquivo especificado pelo argumento *logFilePath*. Além da descrição aqui apresentada, o funcionamento desta função está descrito no algoritmo da figura 4.16
 - Recebe como argumento uma função de callback padrão, ou seja, cujo primeiro argumento é o possível erro e o segundo argumento é o valor de temperatura obtido em caso de sucesso.
 - O argumento *logReadHandler* é um objeto externo à função e que deve guardar o valor da última leitura de temperatura, as últimas 5 *timestamps* (registros de data/hora) e a contagem de erros de leitura de temperatura referentes a um sensor específico.

- Se há algum problema na leitura do arquivo que deve conter o registro da última temperatura lida, a variável de erro é incrementada e, se a contagem exceder 180 erros, a mensagem de erro da função de *callback* indica que muitos erros foram detectados. Note-se que 180 é um valor quase que arbitrário que pode representar 3 minutos de leituras consecutivas erradas ou 180 erros esporádicos de leitura do arquivo e cuja escolha foi realizada pois representa 5% das amostras contidas em uma hora. Esta por sua vez, é aceita como o tempo mínimo de uma brassagem ou fervura. Testes práticos com relação a este aspecto não foram realizados, portanto há espaço para a realização de ensaios para determinação de um valor mais adequado.
 - Em caso de leitura do arquivo com sucesso, se a formatação do valor lido estiver errada, imprime erro, incrementa a variável de erro e, no caso de a contagem exceder 180, seta a mensagem de erro da função de *callback* indicando que muitos erros foram detectados.
 - Se o valor lido for consistente, atualiza o objeto *logReadHandler*. Compara as últimas 5 *timestamps* obtidas e incrementa a variável de erro se todas elas forem coincidentes — isto indica primariamente que o sensor de temperatura está mal conectado, ou mesmo desconectado em caso de erros consecutivos. Por isto a mensagem de erro da *callback* retorna a possibilidade de problemas com o sensor. No caso de a contagem de erros exceder 180, a mensagem de erro da função de *callback* indica que muitos erros foram detectados, ao invés de avisar sobre o sensor.
 - Por fim, se a leitura do registro de temperatura passar por todas as checagens, é retornado nulo na mensagem de erro e o valor de temperatura como segundo argumento.
- **checkRecipeIntegrity(recipe, path, res)** — Verifica a integridade da receita selecionada pelo usuário antes de permitir o início de uma brassagem. Os argumentos recebidos pela função são o nome da receita, o diretório de receitas a ser verificado e o objeto que contém a resposta da requisição AJAX feita pelo cliente, respectivamente.
 - Se não consegue ler o arquivo da receita, responde com erro ao cliente.
 - Caso contrário, lê o arquivo *lockfile* para verificar se já existe uma brassagem em andamento. Se não consegue ler o *lockfile* ou ele indica que há uma brassagem em andamento, também responde com erro.

- Verifica a consistência da receita com o seguinte critério: há campos estritamente necessários para a produção e que impedem o início da receita se estiverem em branco, campos que geram avisos mas permitem o início de uma receita após o usuário se declarar ciente e campos indiferentes para o sistema. São os campos estritamente necessários: água de brassagem, tempo da fervura, temperatura inicial de brassagem, primeiro degrau de temperatura, tempo do primeiro degrau, malte 1, quantidade do malte 1, lúpulo 1, quantidade do lúpulo 1 e tempo de adição do lúpulo 1. Os campos que geram avisos são: nome da receita, estilo, levedura, água de sparging e temperatura de sparging. Todos os outros campos são indiferentes à verificação de consistência.
 - Se tudo deu certo, uma variável global é setada para registro e uso posterior.
- **logToFile(message, code, notableTimestamp)** — função que realiza registro do status da brassagem sempre que invocada. Basicamente, ela salva a variável global *environmentVariables* em um arquivo de texto.
- O argumento *message* pode conter uma string com uma mensagem explicativa contendo, por exemplo, a situação ou estágio da brassagem em que ocorreu o registro, mas pode também ser deixado em branco. O argumento *code* tem basicamente o mesmo propósito, mas é um número predefinido para cada situação possível da brassagem, que permite que o arquivo de registro possa ser interpretado com maior facilidade; por este motivo, recomenda-se veementemente que não seja deixado em branco. A tabela 4.3 apresenta a correspondência entre os valores do argumento *code* e seus significados.
 - O argumento *notableTimestamp* é usado para indicar quando uma situação notável ocorreu, como por exemplo o início da brassagem, o final de uma rampa ou degrau de temperatura, o início da fervura, etc. Para todas as outras situações deve ser deixado em branco, sendo que as situações notáveis são predefinidas.
 - O objeto *environmentVariables*, declarado no código-fonte raiz da aplicação em Node, que foi abordado na seção 4.6.2 e pode ser obtido no apêndice F, também é descrito na caixa 4.17. Ele é salvo no arquivo de *log* no formato JSON e para isto é empregada a função *JSON.stringify* conforme descrito na linha de código-fonte exibida na caixa 4.18.

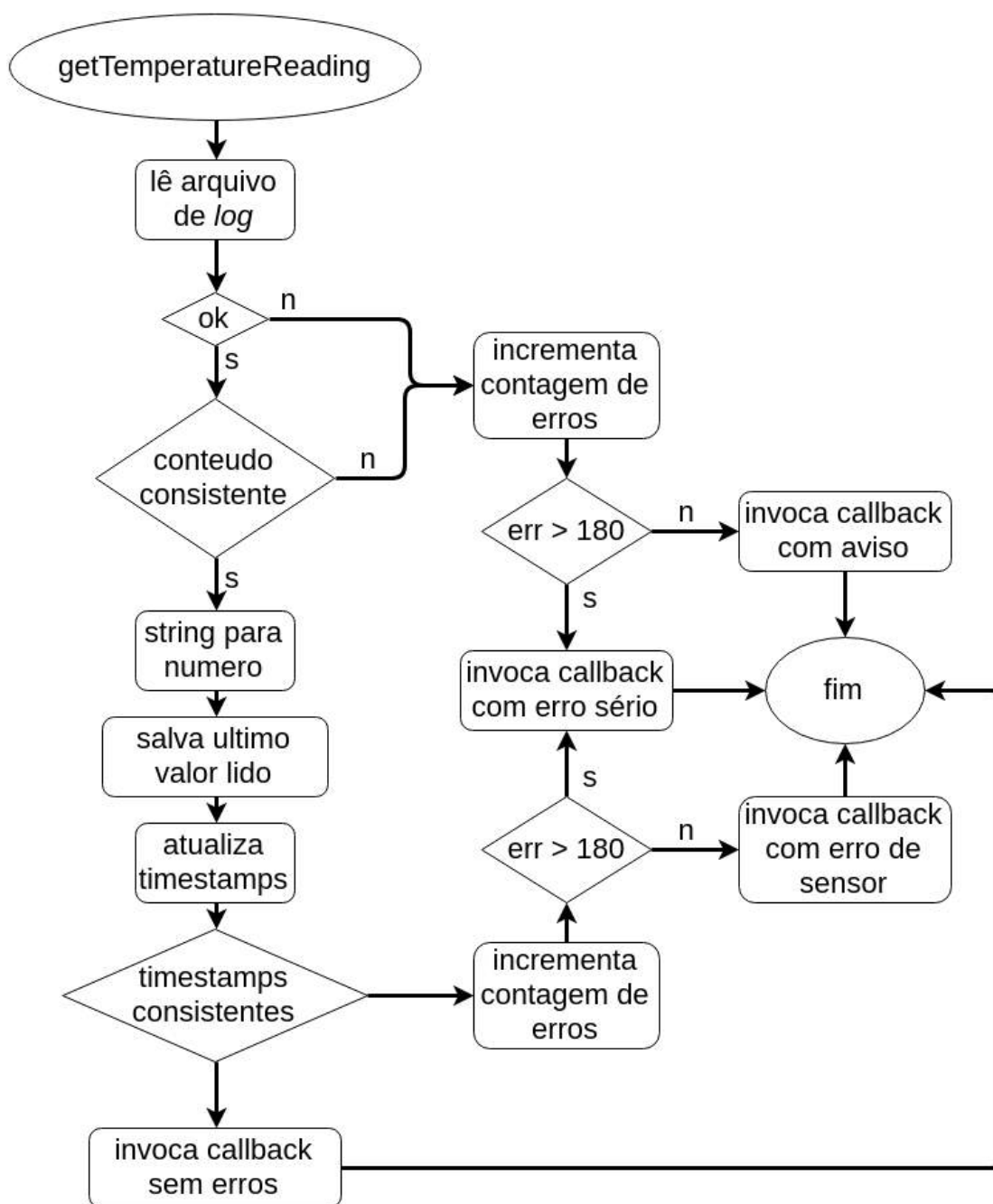


Figura 4.16: Representação em fluxograma do algoritmo de implementação da função `getTemperatureReading`

- **`logToFile(message, code, notableTimestamp)`** — função que realiza registro do status da brassagem sempre que invocada. Basicamente, ela salva a variável global `environmentVariables` em um arquivo de texto.
 - O argumento `message` pode conter uma string com uma mensagem explicativa contendo, por exemplo, a situação ou estágio da brassagem em que ocorreu o registro, mas pode também ser deixado em branco. O argumento `code` tem ba-

sicamente o mesmo propósito, mas é um número predefinido para cada situação possível da brassagem, que permite que o arquivo de registro possa ser interpretado com maior facilidade; por este motivo, recomenda-se veementemente que não seja deixado em branco. A tabela 4.3 apresenta a correspondência entre os valores do argumento *code* e seus significados.

- O argumento *notableTimestamp* é usado para indicar quando uma situação notável ocorreu, como por exemplo o início da brassagem, o final de uma rampa ou degrau de temperatura, o início da fervura, etc. Para todas as outras situações deve ser deixado em branco, sendo que as situações notáveis são predefinidas.
 - O objeto *environmentVariables*, declarado no código-fonte raiz da aplicação em Node, que foi abordado na seção 4.6.2 e pode ser obtido no apêndice F, também é descrito na caixa 4.17. Ele é salvo no arquivo de *log* no formato JSON e para isto é empregada a função *JSON.stringify* conforme descrito na linha de código-fonte exibida na caixa 4.18.
- **sendRecipeNames(path, res)** — responde para o cliente usando o objeto *res* com todas as receitas disponíveis no diretório passado como argumento *path* para a função.
 - É preciso notar que, quando uma receita é deletada, a única modificação realizada é que a extensão do arquivo *.recipe* é modificada para *.recipe.del*, assim é fácil recuperar a receita caso o usuário a tenha deletado por engano. Por isso nesta função, além de ler os nomes dos arquivos presentes no diretório e excluir a extensão, foi preciso ignorar os arquivos deletados.

```

1 //global variables that should also be saved to a backup file
  periodically
2 global.environmentVariables = {
3   warn: "", //if not empty holds some warning message
4   code: "", //tells the same as msg, but as an index, easier
      to check programatically
5   tmpMT: "", //mash tun temperature
6   tmpMTsetp: "", //mash tun current setpoint
7   tmpBK: "", //brewing kettle temperature, also the "hot
      liquor tank" for sparging
8   tmpBKsetp: "", //brewing kettle/hot liquor tank current
      setpoint
9   timeLeft: "", //helping variable to tell the client the time
      left for the step rests or the boil, etc
10  readyForNextStep: false, //set whenever the system is ready
      for the next step
11  auto: true, //whether the process control is running
      automatically or there is human intervention
12  processFail: false, //flag is set if the process fails
      irreversibly

```

```

13 msg: "", //holds some explanatory message
14 timestamps:{//notable timestamps
15     curr: "", //epoch time of the current variables state
16     start: "", //epoch time of the first request to start a
        recipe
17     startHeating: "", //epoch time of the start to heat the
        mash water
18     finishHeating: "", //epoch for the finish of the heating
        of mash water
19     //start of the ramps
20     sRamp0: "", sRamp1: "", sRamp2: "", sRamp3: "", sRamp4: "
        ", sRamp5: "", sRamp6: "", sRamp7: "",
21     //finish of the ramps
22     fRamp0: "", fRamp1: "", fRamp2: "", fRamp3: "", fRamp4: "
        ", fRamp5: "", fRamp6: "", fRamp7: "",
23     startDrain: "", //start of the sparging process, when the
        mash is partially drained to the BK
24     startSparge: "", //here the recirculation pump starts
        working
25     heatingBoil: "", //started to heat the wort after sparging
26     boilStart: "", //time when temperature is near enough
        boiling (>96°C)
27     //hop additions
28     hAdd0: "", hAdd1: "", hAdd2: "", hAdd3: "", hAdd4: "",
        hAdd5: "", hAdd6: "", hAdd7: "",
29     boilFinishScheduled: "", //time when the boil is scheduled
        to finish
30     chillStart: "", //time when the the boil really finishes
        and the chilling starts
31     end: "", //epoch time of the end of the brewing (there is
        cleaning after it)
32     cleaningStart: "", //when the cleaning recirculation
        process starts
33 },
34 ioStatus: gpioCfg.all_io, //also records the IO status
35 okToStart: false, //true if a recipe is ok enough to start
        a production
36 recipe: "" //recipe name
37 };

```

Código-fonte 4.17: Declaração do objeto *environmentVariables*

```

1 dataToSave = JSON.stringify(global.environmentVariables) + "\n";

```

Código-fonte 4.18: Uso da função *JSON.stringify* para codificar um objeto em string JSON

Tabela 4.3: Correspondência entre o valor numérico e o significado dos códigos usados para registro da brassagem

Valor	Significado
0	requisição para início da brassagem
1	produção iniciada
2	esquentando água da mostura
3	esperando adição dos grãos
4	rampa da mostura em execução
5	degrau da mostura em execução
6	<i>sparging</i> em execução
7	transbordamento da MT
8	esquentando mosto para a fervura
9	fervendo o mosto
10	lúpulo adicionado
11	resfriando o mosto
12	fim do resfriamento, esperando OK do operador para limpeza
13	recirculação de água para limpeza

4.6.4 Controle do processo de brassagem

O módulo responsável pelo controle do processo de brassagem é o *ctrl.js*, cujo código-fonte está disponível no apêndice F, na caixa de código-fonte F.5. É importante ressaltar que, embora o comportamento do Node.js seja assíncrono o processo de brassagem é sequencial, o que implica no uso de diversas funções de *callback* aninhadas. No intuito de evitar o *inferno de callbacks*, apresentado na seção 2.4, este processo sequencial foi quebrado em algumas funções que implementam a funcionalidade do módulo e outras auxiliares. Seu funcionamento e aspectos chave são discutidos na presente seção.

Início do processo

A primeira função do processo de controle é a *startMashingProcess(recipe, res, lockFile, recipesPath, callback)*, invocada a partir de uma requisição do cliente e, portanto, sua chamada está presente no módulo referente às rotas do Express, discutido na seção 4.6.2. Ela recebe como parâmetros o nome da receita a ser iniciada, o objeto de resposta da requisição AJAX, o caminho para o arquivo *lockfile*, o caminho para o diretório das receitas e uma função de *callback*. Em primeiro lugar é verificada a flag global *global.environmentVariables.okToStart* que indica se a receita está pronta para ser executada — esta flag é setada previamente pela função *checkRecipeIntegrity*, descrita na seção 4.6.3 e chamada durante o processo de in-

teração pré-brassagem do usuário com o sistema. Depois disto, também é verificado se o nome da receita passado como argumento é idêntico ao da variável global de controle *global.environmentVariables.recipe*.

Caso as verificações iniciais sejam positivas, é escrito o valor "1" no arquivo *lockfile*, indicando daqui para a frente que há uma brassagem em andamento. O conteúdo da receita é lido em uma variável e o *log* de temperatura é iniciado, ou seja, o script Python é invocado como um processo filho do Node. Também é feito o *log* inicial do processo de brassagem. Se tudo ocorreu sem erros, o servidor envia uma resposta para o cliente indicando sucesso, chama a função de *callback* com a variável *erro* nula e invoca a função *heatMashWater*, que dá prosseguimento ao controle do processo. Em caso de erro em alguma das situações descritas, é enviada uma resposta de erro para o cliente e a função de *callback* é invocada com a variável *erro* indicando o erro ocorrido. Note-se que a função de *callback* não tem importância no que diz respeito ao controle, mas serve somente para retornar o status da tentativa de início da produção.

Aquecimento da água da mostura

A função *heatMashWater(recipeContents)* recém chamada, dá prosseguimento ao controle da brassagem. Como o seu nome sugere, ela é responsável pelo aquecimento da água da brassagem até que seja atingida a temperatura inicial, mantendo este *setpoint* até que o usuário adicione os maltes à MT. O único argumento que a função recebe é o conteúdo da receita, a partir do qual é obtido o *setpoint*. Inicialmente a bomba de recirculação é ligada e é feito um *log* da brassagem que anota a timestamp notável de início de aquecimento.

São iniciados dois laços de repetição: um que faz um *log* da situação da brassagem a cada 5 segundos e outro que é repetido a cada 1 segundo e que lê a temperatura da água e chama a função auxiliar *updateHeatingConditions* para controlar o elemento aquecedor da MT. No *callback* da *updateHeatingConditions* fica definido que, se esta é a primeira vez que a temperatura desejada é atingida, o laço de repetição do *log* é atualizado para refletir a mudança de situação de "esquentado água da brassagem" para "esperando adição dos maltes". Se não é a primeira vez que é atingido o *setpoint*, a *callback* não faz nada. Em resumo, após a temperatura atingir o valor desejado, ela é mantida até que o usuário adicione os maltes.

Concomitantemente, o laço de controle de temperatura da função *heatMashWater* verifica a *flag* global *global.environmentVariables.readyForNextStep* que só é setada após o usuário indicar, por meio da GUI, que ele adicionou os maltes. Quando isto acontece, a *flag* é zerada

e os laços de *log* e de controle são interrompidos. Por fim é chamada a função de controle de rampas e degraus de temperatura, mas antes de entrar nos seus detalhes de funcionamento, será explicada a função auxiliar *updateHeatingConditions(vessel, setp, temperature, callback)*.

Função de controle de temperatura

Os argumentos que ela recebe possuem uma interpretação bem direta: *vessel* é o recipiente a ser controlado, ou seja, pode ser a MT ou o BK; *setp* e *temperature* são respectivamente a temperatura desejada e a atual e; *callback* é a função executada sempre que a temperatura é maior ou igual ao valor do *setpoint*, e cujo argumento passado para ela é nulo. O controle do elemento aquecedor é implementado da seguinte forma: quando a temperatura está abaixo de 70% do *setpoint*, o aquecimento fica ligado 100% do tempo; quando a temperatura está entre 70% e 90% do *setpoint*, o aquecimento fica ligado 66% do tempo; quando a temperatura está entre 90% e 100% do *setpoint*, o aquecimento fica ligado 33% do tempo e; acima disto o aquecimento fica desligado.

Cabe ressaltar que a vantagem desta função auxiliar não é o sistema de controle de aquecimento, mas sim o fato de que este é auto-contido, ou seja, para quaisquer implementações de controle de temperatura (PID, histerese, direto), basta modificar esta função, sem alteração do resto do código. Outra condição que deve ser atendida é que esta função precisa ser chamada periodicamente dentro de um laço de repetição, uma vez que ela somente atualiza o valor da saída (elemento aquecedor) baseada nas entradas (temperatura e *setpoint*).

Controle de rampas e degraus de temperatura

De volta ao fluxo de controle da produção, após o usuário adicionar os maltes, a função *rampControl(recipeContents)* é chamada. Ela recebe somente o conteúdo da receita, a partir do qual são obtidas as temperaturas e tempos dos degraus de repouso da mostura, assim como a temperatura da água de *sparging*. Se a temperatura de *sparging* não estiver definida, é assumido que esta parte do processo deve ser omitida. Depois desta verificação é feito um *log* indicando início de rampa de temperatura e é iniciado um *loop* no qual é registrada a situação da brassagem a cada 5 segundos.

Em paralelo, outro laço de repetição implementa a parte relativa ao controle: se ainda há a possibilidade de mais degraus de temperatura, esta é verificada:

- Em caso positivo, os valores do degrau e temperatura são obtidos, o loop de registro é atualizado e o controle do próximo degrau é iniciado
- Em caso negativo isto significa que esta parte do processo acabou e é sinalizado que a fervura/sparging deve começar.

Se o processo de fervura/sparging deve começar, os resistores de aquecimento da MT e do BK são desligados, assim como a bomba de recirculação da MT. Os laços de registro e controle são interrompidos e a fervura ou a lavagem é iniciada, dependendo da verificação feita anteriormente quanto ao *setpoint* da variável de sparging.

Se a etapa de rampas e degraus ainda não acabou, a temperatura da MT é lida e o laço de controle de temperatura é executado, usando a função *updateHeatingConditions*. Ainda, caso haja água de lavagem para esquentar, o resistor de aquecimento do BK é ligado sempre que o resistor de aquecimento da MT estiver desligado. Ambos os resistores de aquecimento são impedidos de ligar ao mesmo tempo pois o sistema elétrico foi dimensionado para operar com corrente máxima referente a um resistor em operação.

Lavagem dos grãos

A função de controle da lavagem dos grãos é fortemente baseada em temporização: o fluxo de vazão da MT para o BK e o fluxo de bombeamento de líquido do BK para a MT devem ser iguais para que não ocorra transbordamento de nenhum dos recipientes, mas como na prática estes fluxos são muito diferentes e variam conforme a receita, o ideal seria o projeto de dispositivos de verificação de transbordamento da panela.

É iniciado um laço de registro da situação da brassagem a cada 5 segundos e, em paralelo, outro laço executado a cada 0,5 segundos verifica se há transbordamento da MT. Em caso afirmativo, a bomba do BK é desligada e um tempo predefinido é esperado até que o bombeamento seja retomado. Em paralelo aos dois laços de repetição, a válvula de drenagem da MT é inicialmente aberta por um tempo determinado em função do volume inicial da água de brassagem, definido como $0,5\text{segundos} \cdot \text{volume}(l)$. Somente quando este tempo acaba é que é a válvula de drenagem e a bomba de recirculação do BK são ativadas, para iniciar a lavagem dos grãos; também é neste momento que é iniciado um terceiro *loop*.

Neste, executado a cada 0,5 segundos, nada acontece em caso de transbordamento. Se tudo estiver ocorrendo conforme planejado, este laço espera um tempo baseado no volume da água de lavagem, definido como $3\text{segundos} \cdot \text{volume}(l)$. Depois que este tempo acaba, é feita a drenagem completa da MT, todos os laços de repetição são interrompidos e é dado início ao processo de fervura.

Fervura

A função *theBoil(recipeContents)* recebe somente o conteúdo da receita. É iniciado um laço de registro da brassagem a cada 5 segundos e em paralelo, após ligar o resistor de aquecimento do BK, é iniciado um laço supervisor da fervura executado a cada 1 segundo e no qual são controladas as adições de lúpulos. Neste laço, só é iniciada a contagem do tempo de fervura depois que a temperatura do mosto sobe além dos 96°C, momento no qual também é atualizado o laço de registro e são agendadas para execução as adições dos lúpulos por meio da função *setTimeout*. Quando a fervura acaba, é desligado o resistor de aquecimento do BK, fechado o compartimento de adição de lúpulos e interrompidos os laços de registro e supervisão da fervura.

4.6.5 Simulação do controle do processo de brassagem

Para testar o código de controle, foi utilizada a função *tlog_test* do script Python de log de temperaturas, apresentada na seção 4.5.1. Com isto, foi possível usar a interface web para executar o processo de controle de brassagem como operador do sistema, usando uma receita de cerveja especificamente criada para o propósito de *debug*, já que uma receita de verdade tem duração de mais de duas horas, tempo elevado para teste de código. As características da receita são apresentadas na tabela 4.4. Parâmetros porventura omitidos significam que estes foram deixados em branco ao preencher a receita na interface web.

Tabela 4.4: Parâmetros de receita de cerveja utilizados para *debug*

Parâmetro	Valor
Nome	Quick Response Debug
Estilo	no style
Água mosturação	30 litros
Água <i>sparging</i>	20 litros
Temperatura <i>sparging</i>	50°C
Fervura do mosto	5 minutos
Malte 1	no malts
Quantidade malte 1	1 kg
Lúpulo 1	hop 1
Quantidade lúpulo 1	1 g
Tempo de adição 1	1 minuto
Lúpulo 2	hop 2
Quantidade lúpulo 2	1 g
Tempo de adição 1	4 minutos
Lúpulo 3	hop 3
Quantidade lúpulo 3	1 g
Tempo de adição 3	2 minutos
Temperatura inicial	30
Temperatura 1	35°C
Tempo 1	1 minuto
Temperatura 2	40°C
Tempo 2	0 minutos

4.7 Interface de usuário

A interface de usuário foi desenvolvida para acesso via navegador da internet, em um computador com tela de resolução WXGA (1366 x 768 pixels). As linguagens utilizadas para a implementação da aplicação foram: a linguagem de marcação HTML, a linguagem de folhas de estilo CSS, a linguagem de programação *client-side* interpretada Javascript em conjunto com a biblioteca *crossbrowser* jQuery e o método AJAX, a linguagem de programação *server-side* interpretada PHP e, a linguagem de programação *server-side* Javascript interpretada pela aplicação Node.js.

4.7.1 Aspectos gerais da GUI

A estrutura da aplicação foi desenvolvida de forma que o operador do sistema conseguisse tanto gerenciar suas receitas de cerveja e analisar dados estatísticos de maneira fácil, quanto para promover a comunicação eficiente entre o cliente e o servidor, implementado em Node.js e discutido na seção 4.6. A estrutura da aplicação do cliente é apresentada:

- Página inicial
- Sobre
- Configurações
- Estatísticas
- Iniciar brassagem
 - Controle de brassagem
- Gerenciador de receitas
 - Editor de receitas

Um ponto em comum para esta estrutura, com exceção da página inicial, foi o desenvolvimento de uma barra de navegação em PHP, cujo código-fonte está documentado na caixa G.1 do apêndice G, de tal modo que o mesmo código-fonte pudesse ser reaproveitado para todas as páginas web. A implementação consistiu na criação de uma estrutura de *divs* e *links* em HTML, sendo que a função detecta a página atual e realça este link, para dar ao usuário um efeito visual de que certo botão da barra de navegação está pressionado.

É usada uma função em javascript, descrita na caixa de código-fonte 4.19 e também no apêndice G, que faz a ponte entre PHP e javascript, já que o servidor Express não foi configurado para suportar as variáveis *http_host* e *request_uri* — assim sendo, a função passa a URL da página atual para o script PHP por meio de um POST HTTP em AJAX.

```

1 function headerPHP (pathToHeader) { //create the header of the
   page
2   $.post (pathToHeader, {url:document.URL} , function (data,
   status) { //send the page URL to PHP
3     if (status == "success") { //if POST is successfull
4       $('body').prepend($(data)); //add the received
       header to the top of the page
5       $('body').show(); //and displays the hidden page
       afterwards
6     }
7   });
8 }

```

Código-fonte 4.19: Função que passa a URL da página atual para um módulo em PHP

Com relação ao estilo do template desenvolvido para toda a GUI, este é documentado nos códigos-fonte CSS G.14, G.15 e G.16 do apêndice G. O processo de criação do template foi iterativo e dependente das funcionalidades a serem implementadas, mas não seguiu uma lista de tarefas ou algo similar; do contrário, seu desenvolvimento foi realizado em conjunto com o resto da interface web.

4.7.2 Boas vindas e informações do projeto

A *página inicial* contém somente uma mensagem de boas vindas clicável, que leva para a *página sobre*. Esta por sua vez, contém informações sobre o projeto: objetivo, local, autor, professor orientador e contato. Ambos os códigos-fonte estão inclusos respectivamente nas caixas G.3 e G.4 do apêndice G.

4.7.3 Configurações gerais da BBB

A *página configurações* apresenta a data/hora da BBB com taxa de atualização de 1 segundo e permite o seu ajuste, para que o usuário tenha uma alternativa caso a configuração de data/hora automática falhe ou não exista acesso à internet naquele momento. A obtenção de um valor de data/hora introduzido pelo usuário do sistema utiliza o campo de formulário *datetime-local* do padrão HTML5, que implementa automaticamente um calendário para escolha da data e devolve a entrada do usuário de acordo com o padrão ISO 8601, cujo formato é apresentado na tabela 4.5. A letra T da string delimita a separação entre data e hora e a letra Z indica que o fuso horário é o UTC 0, também conhecido como GMT.

Tabela 4.5: Padrão de data/hora no formato ISO 8601

AAAA-MM-DDTHH:MM:SSZ
2016-05-07T18:31:53Z

Tanto a obtenção da data da BBB quanto a requisição para sua mudança foram implementadas por meio de requisições AJAX para a rota *config* do servidor. Cabe salientar que só é permitido ao usuário mudar a configuração de data/hora do sistema se o servidor identificar que o serviço de ajuste automático *ntp* não estiver funcionando.

4.7.4 Estatísticas e gráfico dinâmico

A página *estatísticas* apresenta a temperatura instantânea da MT, um gráfico dinâmico da temperatura da MT e do BK que é atualizado automaticamente e um gráfico do histórico de temperatura, plotado usando o script Python desenvolvido na seção 4.5.2. O gráfico dinâmico foi desenvolvido em uma página separada e posteriormente adicionado como um *iframe*, ou seja, uma página HTML dentro de outra página HTML. Seu código-fonte está disponível na caixa G.5 do anexo G.

O gráfico dinâmico foi baseado na biblioteca D3.js, escrita em Javascript e voltada para a manipulação de documentos baseados em dados, em conjunto com o *framework* Rickshaw para plotagem de gráficos. A instalação da biblioteca D3.js consistiu no download do arquivo compactado e posterior extração, enquanto para o Rickshaw bastou clonar o repositório do Github. Os passos executados estão descritos na caixa 4.20.

```

1 cd /var/www
2 https://github.com/mbostock/d3/archive/v3.5.10.zip
3 sudo tar -zxvf v3.5.10.tar.gz
4 sudo rm v3.5.10.tar.gz
5 sudo git clone https://github.com/shutterstock/rickshaw.git

```

Código-fonte 4.20: Instalação da biblioteca D3.js e do framework Rickshaw

O gráfico implementado é baseado em um exemplo disponível em <https://github.com/shutterstock/rickshaw/blob/master/examples/extensions.html> e o código-fonte empregado neste projeto está disponível na caixa G.6 do anexo G. Suas possibilidades de configuração pelo usuário são:

- Possibilidade a seleção de quais conjuntos de dados plotar
- Opção de plotagem em área empilhada/sobreposta ou em linha
- Ligação dos pontos de maneira interpolada, que suaviza transições bruscas e descon- tinuidades; linear, que liga os pontos diretamente ou; em degrau, que traça uma linha horizontal e outra vertical entre dois pontos
- Possibilidade de aplicar um filtro de suavização com intensidade ajustável (filtro de médias móveis);
- Escolha do número máximo de pontos plotados na tela — 300, 1800 ou 3600 pontos
- Barra de zoom

As características de plotagem dos gráficos consistem na escala e numeração automática dos eixos, incluindo escala baseada no tempo Epoch, e na possibilidade de obter informações

detalhadas sobre um ponto do gráfico ao posicionar o cursor sobre ele: valor do ponto e sua respectiva data/hora.

Exceptuando-se o controle do número máximo de pontos, que foi implementado para limitar a 300, 1800 e 3600 pontos, e a atualização constante do gráfico, implementados separadamente para este projeto, todas as outras funcionalidades foram baseadas nas funções do Rickshaw. Um aspecto importante da implementação do gráfico foi que, ao carregar a página web, é feita a leitura do arquivo CSV com o registro do histórico de temperatura, porém quando é feita a atualização deste histórico, é lido o arquivo CSV que contém somente a última leitura de temperatura, com o objetivo de não sobrecarregar tanto o cliente quanto o servidor com a troca e processamento de dados redundantes. Com relação à temperatura instantânea, é este gráfico dinâmico que a fornece para a página mãe *estatísticas* a cada 1 segundo, que por sua vez repassa para o seu elemento HTML correspondente.

4.7.5 Gerenciador de receitas

A interface do gerenciador de receitas foi programada para apresentar as seguintes opções e funcionalidades: criar uma nova receita; editar ou apagar uma receita já cadastrada e; obter uma prévia de determinada receita ao repousar o mouse sobre seu nome.

Para listar as receitas do diretório *www/recipes* foi usada uma função em PHP escrita diretamente no código-fonte da página web, que está disponível na caixa G.7 do apêndice G. Deve-se notar que esta função, além de excluir a extensão *.recipe* do nome dos arquivos de receitas, substitui todos os caracteres *underline* por espaços, uma vez que ao salvar a receita o processo inverso é realizado. Ela adiciona as receitas à lista apresentada ao usuário no formato de um link para cada receita, sendo que estes, por sua vez, redirecionam o usuário para a página de edição de receitas quando clicados. Ao lado de cada link, também é dinamicamente adicionado um botão para deletar a receita correspondente.

Em javascript, foram implementadas as funcionalidades de prévia de determinada receita a partir das funções *mouseover* e *mouseout* da biblioteca jQuery. Além disto, também foram implementadas as funções para deletar e criar receitas. Note-se que as funções que deletam receitas e geram prévia foram implementadas em um arquivo separado ao do código HTML, e que está apresentado na caixa de código-fonte G.8 do apêndice G.

Ambas as funções do módulo adicional usam a requisição HTTP POST, implementada por meio do método AJAX, para comunicação com o servidor. Especificamente para a função que deleta receitas, sua implementação possibilitou ao usuário desfazer a ação em caso de

clique acidental. Mesmo quando o clique não é acidental, a implementação foi realizada de tal forma que a receita não é realmente deletada da BBB, mas somente adicionada a extensão *.del* ao nome do arquivo.

4.7.6 Editor de receitas

O editor de receitas foi uma das primeiras páginas acrescentadas à GUI, portanto grande parte do gerenciamento do formulário HTML que foi implementado é realizado em PHP. Dentre as funcionalidades, estão a adição de novos campos de maltes, lúpulos e temperaturas à medida que o usuário preenche a receita; a verificação dos dados inseridos para que o usuário não adicione entradas com caracteres especiais ou diferentes do esperado (e.g. usuário escrever letras em um campo no qual são esperados números); a verificação e aviso ao usuário de que há campos importantes não preenchidos; a reordenação dos campos quando o usuário apaga um malte, lúpulo ou temperatura e; o salvamento automático da receita na BBB.

O editor de receitas está dividido em três códigos-fonte disponíveis no apêndice G: o código HTML, que só implementa o esqueleto da página e chama as funções dos outros arquivos, apresentado na caixa G.9; um módulo com funções implementadas em javascript, apresentado na caixa G.11; e um módulo com funções implementadas em PHP, apresentado na caixa G.10.

No código HTML, ao carregar a página web, foi incluído o módulo PHP e chamada uma função que verifica se foi recebida uma receita pela URL, por meio do método GET. Somente em caso afirmativo é que uma função lê o conteúdo da receita em um arranjo e a retorna para a página HTML, que por sua vez carrega os dados no formulário. Após isto, foi implementado um trecho de código javascript que organiza os campos da receita caso existam buracos. Por exemplo: se o malte 1 e o malte 3 estão cadastrados mas não há malte 2, o código javascript trata de reorganizar o malte 3 para malte 2. Em seguida foi obtido o nome da receita a partir da URL, e por fim chamada uma função toda vez que algum campo do formulário perde o foco, que implementa o rearranjo dos campos se necessário e faz o *autosave* da receita.

No módulo escrito em javascript, é interessante o funcionamento da função recursiva que reorganiza os campos do formulário, cuja assinatura é *rearrange(catg, line)*, sendo o argumento *catg* o indicador de quais campos devem ser reorganizados (maltes, lúpulos ou degraus de temperatura) e *line* o argumento que indica a partir de qual linha deve ser feita a ordenação. O algoritmo de implementação está descrito no fluxograma da figura 4.17.

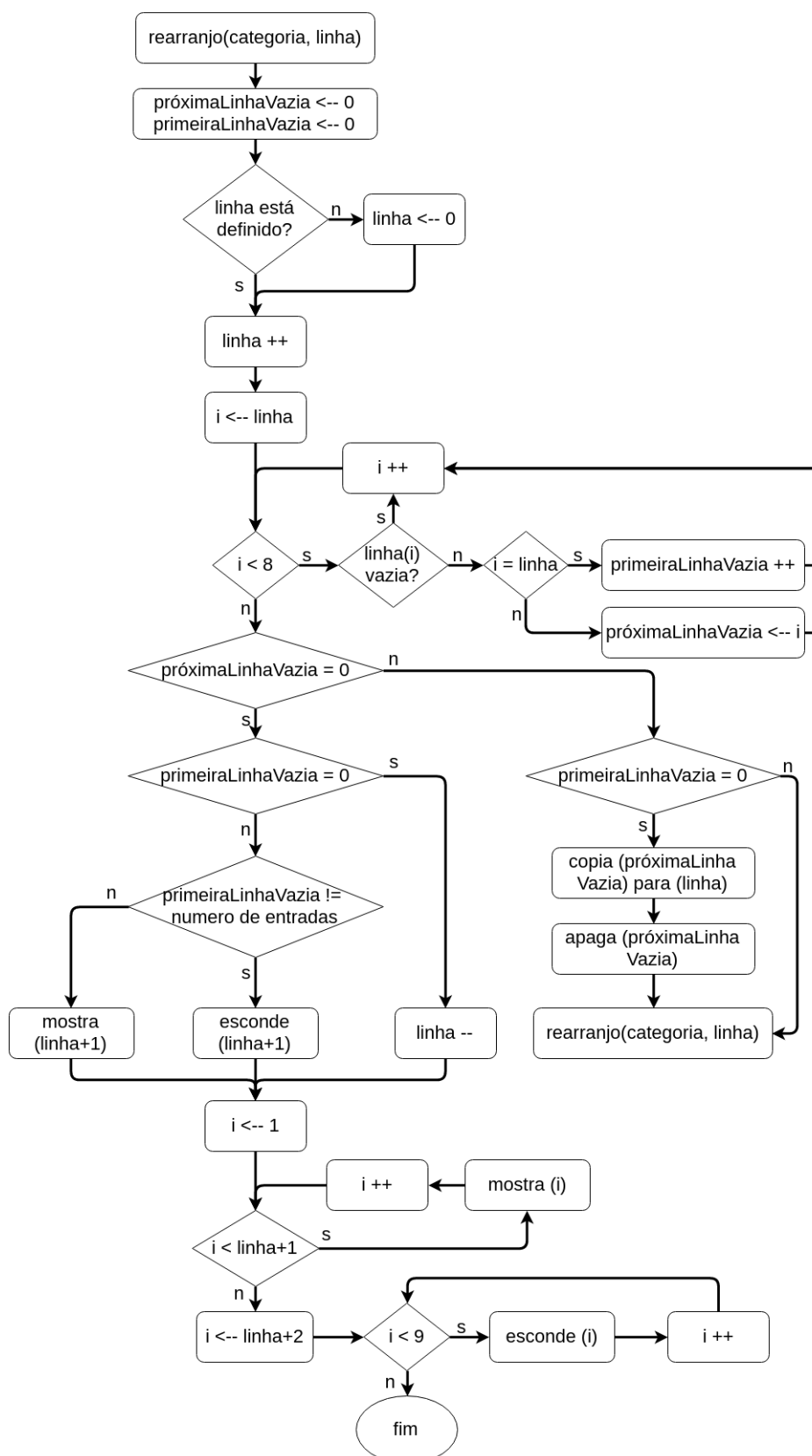


Figura 4.17: Representação em fluxograma do algoritmo de implementação da função *rearrange*

4.7.7 Gerenciador de brassagem

A interface que controla o andamento de uma produção, cujo código-fonte está descrito na caixa G.12 do anexo G apresentou uma interface inicial que permitiu ao usuário do sistema selecionar a receita alvo da produção e iniciá-la. É preciso notar, porém, que a primeira função executada faz uma requisição para o servidor perguntando se há uma receita já em andamento: em caso positivo, a página é redirecionada para o painel de controle das receitas, descrito na seção 4.7.8, impedindo portanto o usuário de iniciar o controle de duas receitas concorrentemente.

Se não houver uma receita em andamento, ao carregar a página é feita uma requisição para o servidor pedindo a lista de receitas disponíveis e, quando uma receita é selecionada para produção, é gerada uma prévia desta usando os mesmos recursos descritos na seção 4.7.5. Quando o usuário clica no botão de iniciar uma produção, é feita uma requisição para o servidor para que a receita possa ser iniciada e, após a resposta, somente se tudo estiver certo a produção é iniciada. Se existirem pontos de atenção para o usuário, ele deverá responder a uma mensagem indicando estar ciente das possíveis situações de risco e, se algo estiver errado, o usuário é avisado que correções na receita devem ser realizadas antes que esta possa ser iniciada. O fluxograma da função de checagem de erros é apresentado na figura 4.18.

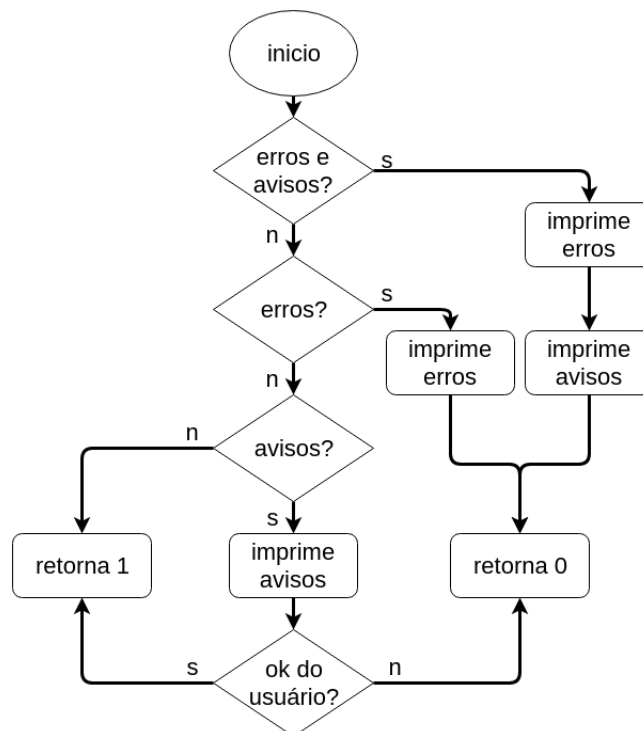


Figura 4.18: Representação em fluxograma do algoritmo de implementação da função *errorWarningHandler*

4.7.8 Painel de controle de brassagem

O painel de controle de brassagem é uma interface composta de botões para o controle de válvulas, bombas e aquecedores, além de uma barra deslizante para o controle do servo-motor da estrutura de adição de lúpulos. Também há um botão que indica se o modo automático está ativado, o que impede o usuário de fazer modificações. Também é impressa uma mensagem que indica o status atual do sistema tanto quando ele está ocioso quanto durante as várias fases do processo de brassagem. O código-fonte está disponível na caixa G.13 do anexo G.

Esta página específica da GUI está intrinsecamente relacionada com o código do servidor, abordado na seção 4.6. A função *refreshSystemStatus*, cujo objetivo foi requisitar ao servidor o status de GPIO e PWM, atualiza a página web a cada 0,5 segundos. Já a função *updateStatusMessage*, baseada na resposta do servidor, atualizou a mensagem de status conforme a etapa da produção de cerveja:

- Se o sistema estava ocioso, era impressa a mensagem "Sistema parado".
- Se detectado um erro irreversível, era impresso "Erro no sistema. Algo impediu que esta receita continue".
- No caso da brassagem, havia diversas mensagens para cada situação. As mensagens estão disponíveis no código-fonte G.13.

4.8 Circuitos de interface entre a BBB e sensores/atuadores

A presente seção discorre acerca do projeto dos circuitos de interface entre a BBB, que representa o núcleo do sistema proposto, e as interações com o sistema mecânico, por meio de sensores e atuadores, dentre outros circuitos auxiliares.

4.8.1 Acionamentos de potência

O circuito de acionamentos de potência foi projetado com o objetivo de controlar os atuadores do sistema: bombas de recirculação de líquido, solenóides das válvulas eletromecânicas, resistores de aquecimento e servo-motor. Com exceção do acionamento do servo-motor, os outros foram projetados de tal maneira que qualquer unidade de acionamento pode ser aplicada a qualquer atuador, já que os requisitos de potência atendem a todas as especificações. Cabe ressaltar aqui que os optoacopladores empregados nos circuitos (4N25) são optoacopladores antigos e de uso geral, que possuem substitutos mais adequados no mercado, mas

que foram empregados pela facilidade com que podem ser obtidos em baixa escala e foram suficientemente adequados para os circuitos desenvolvidos.

As bombas de recirculação escolhidas para este projeto possuem tensão nominal de 12Vdc e corrente nominal de 1,6A, enquanto as válvulas solenóides da Danfoss consomem 12Vdc e 1,25A e a válvula solenóide genérica é especificada para 12Vdc e 300mA. O relé de estado sólido consome 7,5mA para uma alimentação de 12Vdc. Tendo em vista estes parâmetros, foi projetado o circuito da figura 4.19, que representa um módulo de acionamento de potência. *v2-ctr* representa o ponto de ligação com a BBB; *GND_S* é a referência da BBB; e os pinos *Vcc* e *v2* são a saída do circuito, à qual deve ser ligado o elemento atuador.

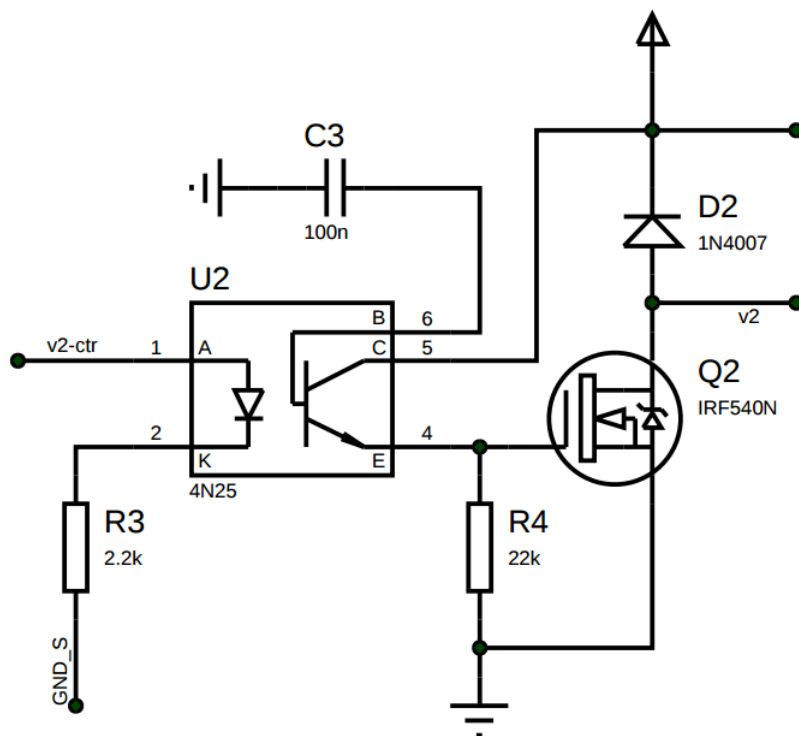


Figura 4.19: Módulo do circuito de acionamentos de potência

Note-se que no módulo é empregado um transistor IRF540N do tipo MOSFET, cujas especificações podem ser consultadas no anexo VI. A escolha deste transistor foi motivada pelo seu baixo custo e alta disponibilidade, mas devido às suas especificações este não pôde ser acionado diretamente com 3,3V a partir de um pino da BBB, por isto o emprego da fonte de alimentação de +12V. Ainda assim, isto não foi um prejuízo ao projeto, tendo em vista que, para proteger a BBB de transientes que a pudessem danificar, foi empregado um optoacoplador como elemento isolador entre o hardware de controle e o hardware de potência.

O valor do resistor R3 foi escolhido de modo que a corrente máxima no terminal da BBB seja de 1,5mA. O diodo D2 foi colocado em caráter de proteção do circuito, uma vez que o chaveamento de cargas indutivas gera uma tensão reversa que pode danificar os componentes — com o diodo presente, a corrente reversa tem um caminho de dissipação. O acionamento dos atuadores é realizado em nível alto, já que este faz com que o transistor do optoacoplador conduza, ativando o MOSFET.

4.8.2 Acionamentos de potência

Com relação ao circuito de acionamento do servo-motor, há duas diferenças que devem ser levadas em consideração. Primeiramente, este dispositivo atuador tem faixa de alimentação de 4,8Vdc a 6Vdc, portanto se faz necessário o uso de um regulador de tensão. A segunda diferença é o fato de que há um terminal para controle de posição do dispositivo, fazendo com que este não possa ser ligado em série com o MOSFET. Para adequar a esta situação, foi utilizado um resistor de *pull-down* em série com o MOSFET para compor a saída do circuito. O diagrama esquemático do módulo de acionamento do servo-motor é apresentado na figura 4.20.

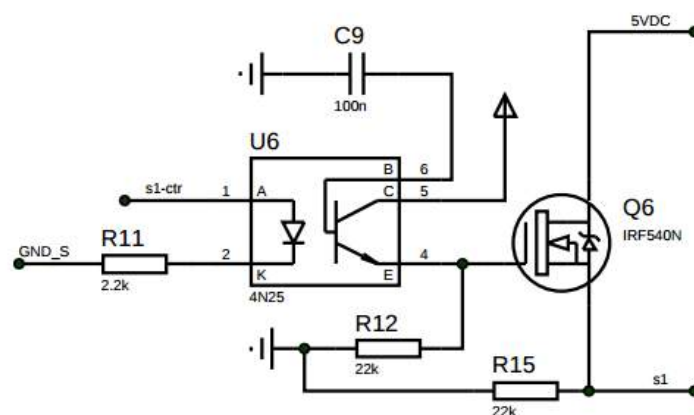


Figura 4.20: Módulo do circuito de acionamento do servo-motor

O diagrama esquemático completo do circuito de acionamentos de potência está disponível no apêndice L. Os resultados das simulações e ensaios em bancada são apresentados e discutidos nas seções 5.4.1 e 5.4.2 do capítulo 5, para o módulo de acionamentos gerais e o módulo de acionamento do PWM, respectivamente.

4.8.3 Detector de *zero-crossing*

O circuito detector de *zero-crossing*, ou seja, detector do momento em que o valor da tensão da senóide da rede passa por zero, foi projetado após a constatação de que a instabilidade da frequência da rede faz com que a fase entre ela e o sinal de PWM usado para o controle dos SSR variem, impossibilitando que seja controlada a quantidade de semi-ciclos que o SSR conduz. Cabe salientar aqui que existem diversas configurações de detectores de passagem por zero, porém o projeto de um detector simples foi realizado no intuito de comprovar conceitos adquiridos ao longo da graduação.

Com este detector, que gera um pulso sempre que a senóide passa por zero, é possível sincronizar a ativação do SSR com o pulso e, portanto, realizar o controle da potência dos resistores de aquecimento do sistema com maior exatidão. Para a detecção, foi projetado um circuito que retifica e atenua a senóide da rede. Este sinal é usado para controlar um transistor que atua como chave: quando a senóide retificada em onda completa e atenuada tem valor menor do que aproximadamente 0,7V, o transistor para de conduzir, gerando então o pulso que indica a passagem por zero. Na figura 4.21 é apresentado o diagrama esquemático do circuito.

Os resultados das simulações e ensaios em bancada são apresentados e discutidos na seção 5.4.3 do capítulo 5.

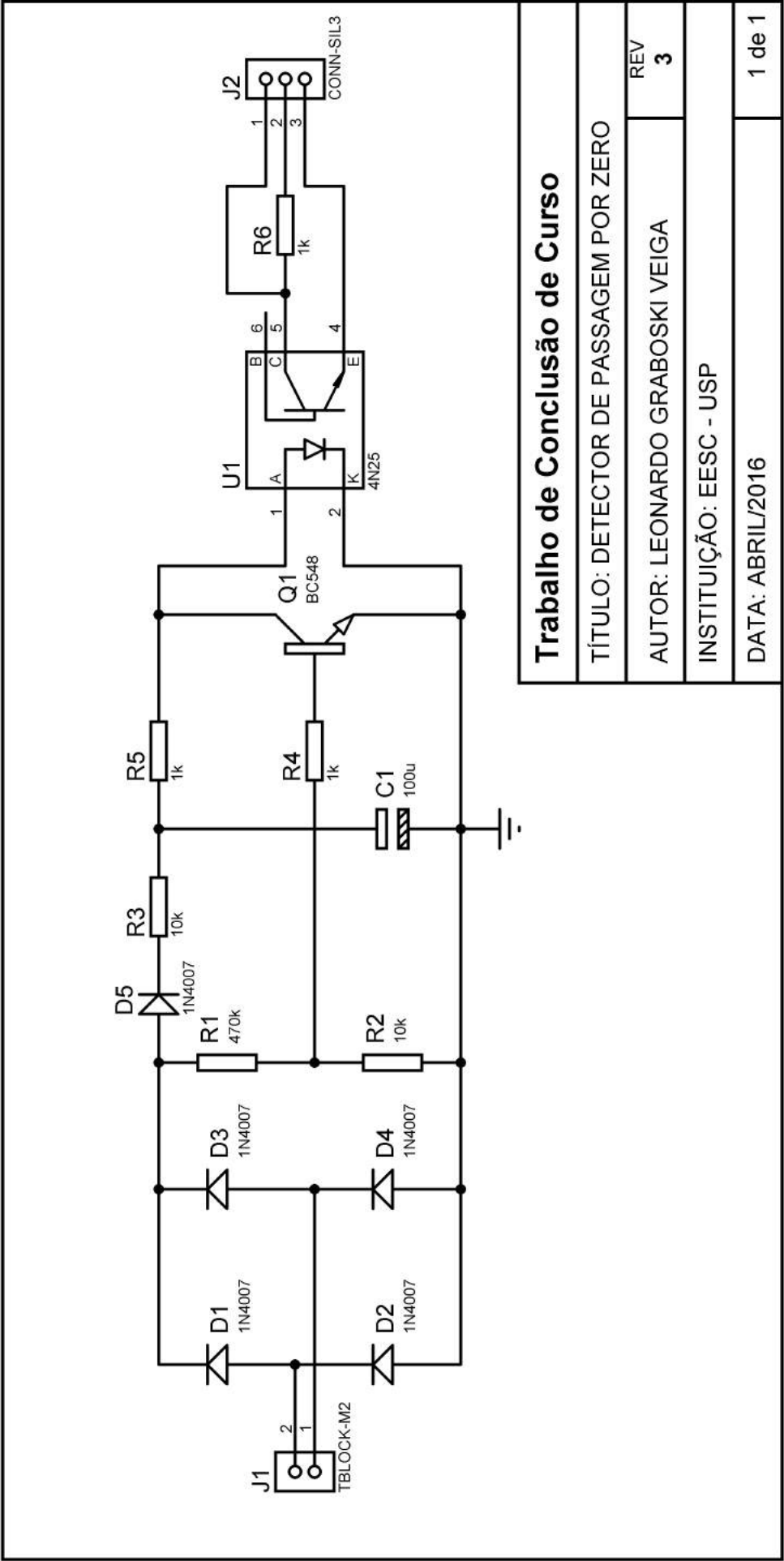


Figura 4.21: Circuito detector de passagem por zero da senóide da rede

Capítulo 5

Resultados e Discussões

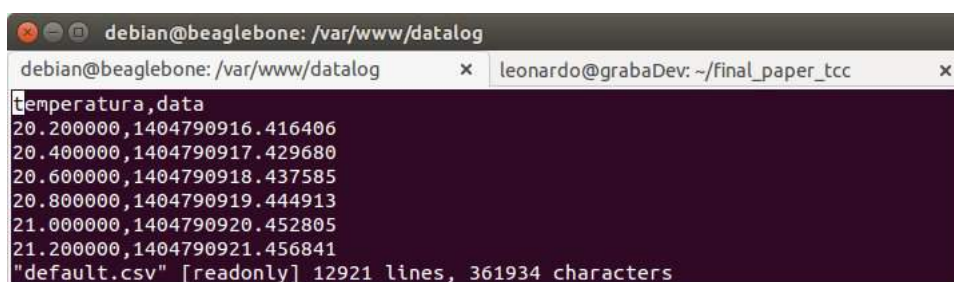
Neste capítulo são apresentados e discutidos os resultados obtidos a partir dos métodos descritos na seção 4.

5.1 Geração de gráfico e registro de temperatura em Python

Nesta seção são apresentados os resultados e discussões referentes ao registro de temperatura em arquivo CSV e ao gráfico gerado em Python.

5.1.1 Registro de temperatura

Na figura 5.1 é apresentada uma amostra de arquivo CSV gerado pelo script 4.2, e que valida o funcionamento da aplicação.



```
debian@beaglebone: /var/www/datalog
debian@beaglebone: /var/www/datalog x leonardo@grabaDev: ~/final_paper_tcc x
temperatura,data
20.200000,1404790916.416406
20.400000,1404790917.429680
20.600000,1404790918.437585
20.800000,1404790919.444913
21.000000,1404790920.452805
21.200000,1404790921.456841
"default.csv" [readonly] 12921 lines, 361934 characters
```

Figura 5.1: Arquivo CSV com registro de temperaturas gerado em Python

Observa-se a partir deste que o tempo de amostragem, mesmo após o estudo do apêndice K, tem média cujo valor foi de 1,008s, diferente dos 1s desejados. Ainda assim, é preciso ressaltar que este valor está sujeito a mudanças em função da carga do sistema, o que significa que a diferença de 8ms do tempo médio não seria eliminada durante a execução da aplicação de controle do processo de produção de cerveja. Caso seja estritamente necessá-

rio para uma aplicação que o tempo de amostragem não sofra variações, deve ser cogitado o uso de um sistema operacional de tempo real, do subsistema PRU-ICSS da BBB ou de um microcontrolador auxiliar que se comunique com a BBB por meio de um barramento de comunicação.

Para confirmar o funcionamento da função que faz o registro da última temperatura lida em um arquivo separado, o comando *tail* do Linux foi empregado. Na figura 5.2 é apresentado o resultado prático desta abordagem.

<pre> debian@beaglebone:~/brewing\$ sudo python log.py registrando temperatura! 25.0 registrando temperatura! 25.0 registrando temperatura! 25.0 registrando temperatura! 25.0 registrando temperatura! 25.0 registrando temperatura! 25.0 registrando temperatura! 25.0 </pre> (a) Script python em execução	<pre> debian@beaglebone:/var/www/datalog\$ tail -f instant.csv temperatura,data 60.000000,1464046530.528117 tail: instant.csv: file truncated temperatura,data 25.000000,1464373062.864578 tail: instant.csv: file truncated temperatura,data 25.000000,1464373063.913387 tail: instant.csv: file truncated temperatura,data 25.000000,1464373064.949328 tail: instant.csv: file truncated temperatura,data 25.000000,1464373065.981745 tail: instant.csv: file truncated temperatura,data 25.000000,1464373067.019056 tail: instant.csv: file truncated temperatura,data 25.000000,1464373068.062740 tail: instant.csv: file truncated temperatura,data 25.000000,1464373069.094996 tail: instant.csv: file truncated </pre> (b) Arquivo de registro da última temperatura
--	--

Figura 5.2: Conformidade da atualização do arquivo de registro da última temperatura capturada pelo script Python

5.1.2 Gráfico de temperatura

Na figura 5.3 é apresentado um gráfico que foi gerado na BBB a partir de 7200 pontos amostrados, ou seja, aproximadamente 120 minutos. Este tempo foi escolhido uma vez que dificilmente o cozimento do mosto ou a fervura dura mais do que isso, portanto provando que a implementação é factível para o uso a que se destina. A tabela 5.1 apresenta informações estatísticas sobre o tempo de computação necessário para a geração do gráfico. A figura 5.4 apresenta o processo de verificação do número de pontos registrados e o posterior processo de coleta dos dados que deram origem à tabela 5.1.

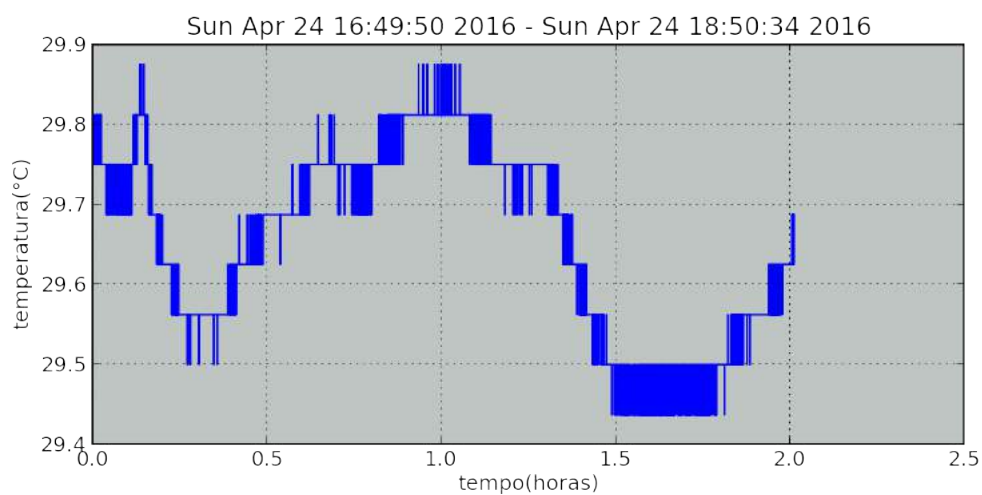


Figura 5.3: Gráfico com 7200 pontos gerado em Python

Tabela 5.1: Estatísticas referentes ao tempo de geração de gráfico na BBB, para 7200 pontos em Python

Descrição	Valor
N.º de amostras	30
Média (s)	2,106
Máximo (s)	2,068
Mínimo (s)	2,149
Desvio padrão (s)	0,023

```

debian@beaglebone:~/brewing$ wc -l /var/www/datalog/default.csv
7201 /var/www/datalog/default.csv
debian@beaglebone:~/brewing$ sudo python grafico.py
tempo para gerar grafico (s) = 2.127
tempo para gerar grafico (s) = 2.068
tempo para gerar grafico (s) = 2.136
tempo para gerar grafico (s) = 2.077
tempo para gerar grafico (s) = 2.116
tempo para gerar grafico (s) = 2.083
tempo para gerar grafico (s) = 2.085
tempo para gerar grafico (s) = 2.126
tempo para gerar grafico (s) = 2.118
tempo para gerar grafico (s) = 2.117
tempo para gerar grafico (s) = 2.085
tempo para gerar grafico (s) = 2.099
tempo para gerar grafico (s) = 2.113
tempo para gerar grafico (s) = 2.087
tempo para gerar grafico (s) = 2.115
tempo para gerar grafico (s) = 2.090
tempo para gerar grafico (s) = 2.138
tempo para gerar grafico (s) = 2.091
tempo para gerar grafico (s) = 2.094
tempo para gerar grafico (s) = 2.126
tempo para gerar grafico (s) = 2.095
tempo para gerar grafico (s) = 2.139
tempo para gerar grafico (s) = 2.080
tempo para gerar grafico (s) = 2.149
tempo para gerar grafico (s) = 2.089
tempo para gerar grafico (s) = 2.093
tempo para gerar grafico (s) = 2.145
tempo para gerar grafico (s) = 2.095
tempo para gerar grafico (s) = 2.120
tempo para gerar grafico (s) = 2.082

```

Figura 5.4: Geração de gráfico em Python múltiplas vezes para obter o tempo médio

Também foi realizado um registro de temperatura durante 38 dias ininterruptos, para comprovar a robustez da aplicação de registro de temperatura, a capacidade de geração de gráfico para um número de pontos muito superior ao de uma produção de cerveja e também para comprovação do funcionamento da escala ajustável em função da logevidade do arquivo de registros. O gráfico gerado a partir deste registro é apresentado na figura 5.5 e indica a temperatura da saída de ventilação de um notebook com problemas de resfriamento.

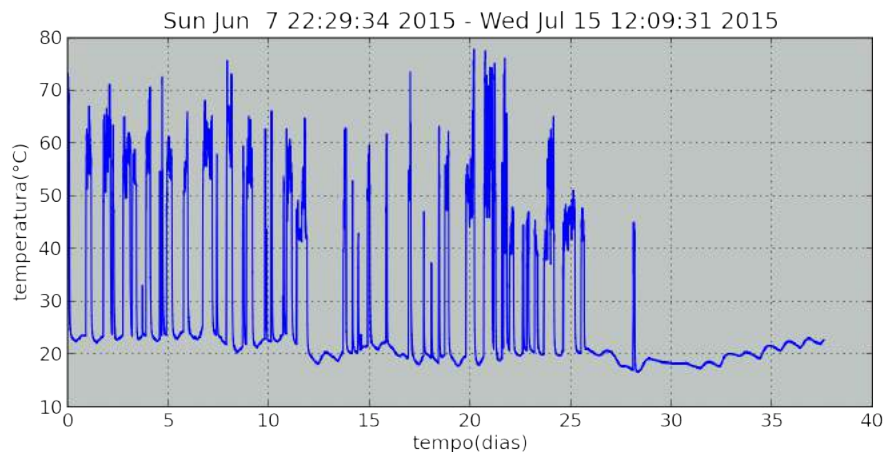


Figura 5.5: Geração de gráfico em Python a partir de um volume elevado de dados

É importante notar que as duas aplicações escritas em Python para este projeto foram desenvolvidas em uma fase inicial, ou seja, ao longo do primeiro mês e meio deste. Isto significa que, naquele momento, o desenvolvedor não possuía conhecimentos práticos de Linux embarcado e nem mesmo de linguagens de programação de alto nível, com excessão de MATLAB. Isto foi traduzido em dois pontos que foram constatados somente muito tempo depois da escrita dos códigos Python: a curva de aprendizado de todo um sistema novo custou uma parcela significativa de tempo do projeto e; o fato de as tarefas necessárias para cumprir o projeto estarem mal definidas naquele momento inicial, levaram o desenvolvedor a usar mais tempo do que seria razoável destinar a esta tarefa.

A análise do tempo de leitura do sensor DS18B20, descrita no apêndice K comprova o resultado que se espera sabendo que o Linux não é um sistema operacional de tempo-real: o tempo de amostragem não é constante. Este é um fato já conhecido e, portanto, a análise conduzida não significa necessariamente uma contribuição inovadora. Por outro lado, esta análise levou o desenvolvedor à ideia de que talvez seja possível criar dentro do próprio script uma maneira de ajustar o tempo de amostragem a partir de valores anteriores de tal modo que o tempo médio seja o desejado, o que requer um estudo de caso e análise de aplicação: vislumbrando o fato de que o tempo de amostragem do DS18B20 é limitado pelo driver do Kernel a 750ms e que a resposta de um sistema térmico é muito lenta se comparada com um sistema eletrônico, é interessante estudar a possibilidade e os efeitos da implementação de um controlador PID ou suas variações sob as condições propostas.

Quanto ao gráfico de temperatura, seu uso ficou restrito ao início do projeto e foi relegado ao segundo plano depois da implementação do gráfico dinâmico usando D3.js e Rickshaw. O conhecimento adquirido a partir da necessidade de gerar este gráfico mostrou que a linguagem de programação Python é aliada de pesquisadores, com as bibliotecas NumPy e SciPy, dentre outras, e pode ser uma alternativa de baixo custo a soluções proprietárias como o MATLAB; por outro lado, a curva de aprendizado e, possivelmente a facilidade de consulta a documentação sejam pontos nos quais as soluções proprietárias se sobressaem. Também foi surpreendente o fato de a BBB plotar em 2,1 segundos um gráfico com 7200 pontos — em parte a surpresa se deu pelo fato de o desenvolvedor não conhecer as limitações da plataforma que estava usando, quando foi desenvolvido o script que plota o gráfico.

O registro de temperatura de 38 dias ininterruptos mostra que a linguagem Python, na plataforma BBB, suportou criar e lidar com um arquivo de registros CSV de 3.283.200 entradas, o que representa um arquivo de 95,2MB, sabendo-se que cada entrada ocupou 29 caracteres

no arquivo. Esta reflexão também levou à consideração de que deveria ter sido implementado um banco de dados, ao menos para testes.

Por fim, estes códigos ficaram legados no projeto, porém poderiam facilmente ser substituídos por módulos em Node.js, o que padrozinaría a aplicação e a tornaria mais coerente, do ponto de vista que não seria necessário executar um comando do shell a partir do Node.js para executar o script Python. Além disto, o comportamento assíncrono do Node.js possibilitaria a leitura do DS18B20, que demora 750ms, em paralelo com outras atividades nativamente, o que é um benefício não implementado no script Python.

5.2 Aplicação *server-side* em Node.js

Nesta seção serão apresentados os resultados referentes à implementação *server-side* em Node.js. Serão apresentados desde resultados simples, como a verificação da instalação do Node, até o resultado da simulação do processo de controle automático da brassagem.

O primeiro teste realizado, descrito na figura 5.6, demonstra que o Node.js foi corretamente instalado.

```
debian@beaglebone:/var/www$ node  
> console.log("Olá mundo!");  
Olá mundo!  
undefined  
> process.exit();
```

Figura 5.6: Teste de funcionamento do Node.js

5.2.1 Acesso a GPIO e PWM

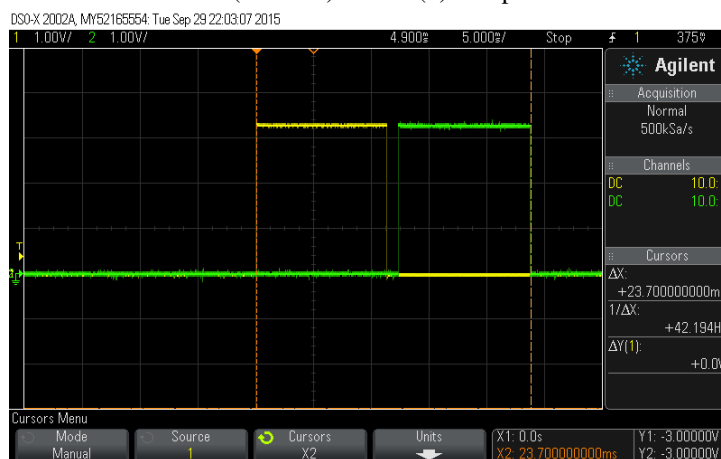
O teste de funcionamento do acesso a GPIO foi realizado por meio de chamadas periódicas à função *testIO* descrita na seção 4.6.1. Uma vez que esta função chaveia 9 pinos, sendo que somente um deles fica em nível alto por vez, as observações em bancada foram realizadas com dois pinos consecutivos, dada a limitação de 2 canais do osciloscópio.

O resultado de um teste preliminar, resultante da observação em bancada para o chaveamento destes pinos, é apresentado na figura 5.7. Neste teste preliminar, foram coletados dados relativos somente a uma amostra.



(a) Tempo em nível alto do canal 1 (amarelo)

(b) Tempo em nível alto do canal 2 (verde)



(c) Tempo total de chaveamento

Figura 5.7: Observação do chaveamento de dois pinos de GPIO consecutivos com tempo em nível alto esperado de 10ms

A partir desta coleta de dados, obtém-se os valores de tempo médio em nível alto, tempo inativo entre chaveamentos e erro de tempo em nível alto, todos para somente um ciclo de amostragem. Os valores estão descritos na tabela 5.2.

Tabela 5.2: Estatísticas referentes ao tempo de um único chaveamento de GPIO usando o módulo *gpio_config* em Node.js

Descrição	Valor (ms)
Tempo médio esperado em nível alto	10,0
Tempo médio medido em nível alto	11,3
Tempo inativo entre chaveamentos	1,1
Erro de tempo em nível alto (%)	13,0

Foi notado também que a carga de processamento exigida da CPU é de 20%, valor não desprezível. Este resultado está documentado na figura 5.8.

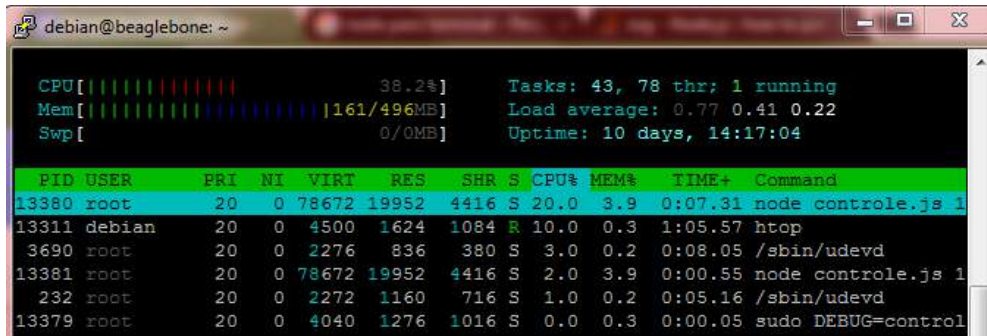


Figura 5.8: Carga da CPU dedicada ao chaveamento de GPIO em Node.js com tempo de chaveamento de 10ms

Observados os valores do teste preliminar, nos quais tanto o tempo de chaveamento, que idealmente deveria ser nulo, quanto o tempo em nível alto, que apresentou erro de 13% não foram desprezíveis, foi decidido realizar um estudo mais completo do acesso a GPIO, medindo não somente temporização quanto uso de CPU pelo processo. Para isto foram definidos tempos em nível alto desde 0,1ms até 500ms e foram coletadas 30 amostras para cada situação. Os dados consolidados são apresentados na tabela 5.3.

Tabela 5.3: Estatísticas referentes ao chaveamento de GPIO usando o módulo *gpio_config* em Node.js

Tempo esperado em nível alto(ms)	Tempo médio em nível alto (ms)	Tempo mínimo em nível alto	Tempo máximo em nível alto	Desvio padrão (ms)	Erro (%)	Carga da CPU (%)
0,1	1,89	1,69	3,83	0,40	1793,21	56
0,2	1,81	1,57	2,73	0,22	806,71	56
0,5	1,84	1,68	3,62	0,34	268,09	56
1,0	1,85	0,45	3,79	0,51	85,10	56
1,5	2,69	2,16	4,69	0,49	79,40	60
2,0	2,77	1,88	3,81	0,26	38,45	38
2,5	3,51	2,92	3,85	0,17	40,34	46
3,0	4,29	3,50	6,00	0,44	43,08	43
4,9	5,80	4,58	6,80	0,50	18,28	43
5,0	6,37	5,53	8,13	0,44	27,36	31
5,1	6,68	5,60	8,33	0,63	30,94	33
10,0	11,46	10,55	12,51	0,37	14,58	21
20,0	21,71	21,34	23,07	0,38	8,57	12
50,0	51,74	50,62	54,68	0,71	3,47	10
100,0	102,08	100,76	104,04	0,57	2,08	10
500,0	502,41	501,40	503,40	0,37	0,50	10

A partir destes dados, foi observado que a técnica de chaveamento de GPIO em Node.js apresentou erro acima de 5% até para valores entre 20ms e 50ms, o que desqualifica o uso desta técnica em situações nas quais temporização é crítica. Para valores abaixo de 10ms, o

erro superior a 30% torna esta técnica inviável mesmo para aplicações nas quais a tolerância é alta. Observou-se que o limite médio inferior de chaveamento ficou na ordem de 1,8ms e foi alcançado somente para tempos esperados iguais ou menores do que 1,0ms.

Outra informação obtida a partir da tabela 5.3 é o fato de que o tempo de chaveamento não foi limitado pelo uso da CPU. Da figura 5.8 nota-se que 18% da carga da CPU estava sendo requisitada pelas outras tarefas da BBB e, portanto, mesmo para o pior caso de chaveamento observado, com uso de 60%, ainda assim, havia poder de processamento ocioso.

Quanto ao PWM, foi verificado o seu funcionamento para a frequência de $f=50\text{Hz}$, ou seja, a mesma frequência necessária para operação do servo-motor, e ciclo de trabalho de 50%. O resultado está exposto na figura 5.9. Resultados complementares a este são apresentados na seção 5.4, que aborda especificamente os resultados do servo-motor, mas que está intrinsecamente ligada ao bom funcionamento do PWM.

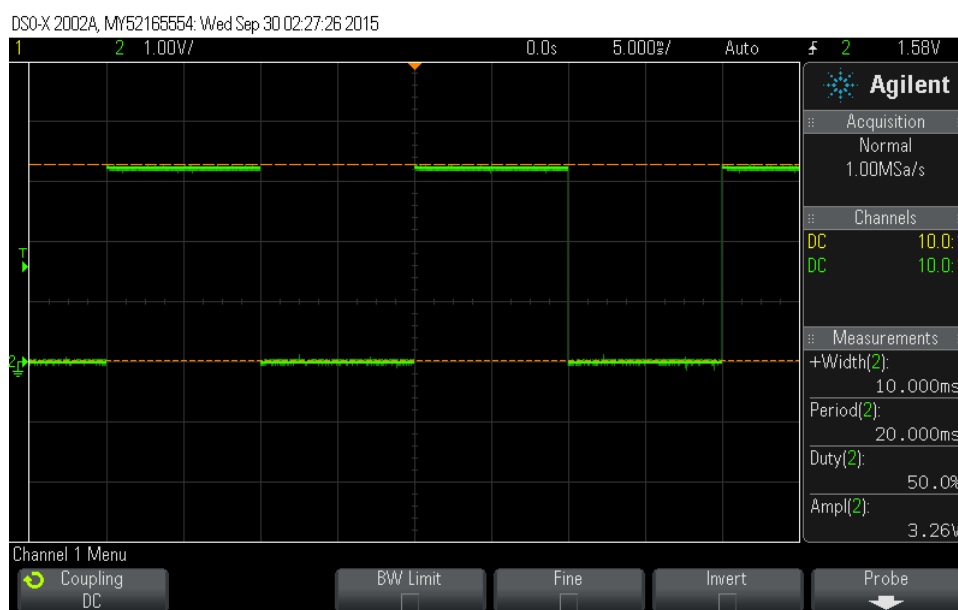


Figura 5.9: Verificação de funcionamento do PWM configurado por meio do Node.js, com $f=50\text{Hz}$ e $\text{duty-cycle}=50\%$

5.2.2 Servidor Express

Como o objetivo do servidor implementado neste trabalho foi exclusivamente o de servir conteúdo via web e proporcionar a base para a interface de usuário, o capítulo de resultados 5.3, que apresenta os resultados referentes à interface gráfica de usuário, é um resultado direto do sucesso da implementação utilizando o *framework* Express.

Não obstante, foi observado que a execução da aplicação completa em Node.js, quando o

sistema fica ocioso, ocupa 8,1% dos recursos de memória da BBB, ou seja 40Mb de um total de 496Mb. Quanto aos recursos de processamento, o *software* de monitoramento de recursos *htop* oscila entre 0% e 1% de uso da CPU para esta tarefa.

Para observar a carga do sistema e comprovar o embasamento teórico exposto no capítulo 2.4, que indica que o uso de Node.js é apropriado para operações de I/O intensivo, foi observado o consumo de recursos da aplicação com um número variado de abas de internet abertas ao mesmo tempo na interface de controle manual do sistema. A escolha desta página como *benchmark* deriva-se do fato de que ela se comunica com o servidor a cada 0,5s. Os resultados do teste se encontram na tabela 5.4.

Tabela 5.4: Consumo de recursos de memória e CPU da aplicação em função do número de clientes conectados à BBB

<i>Nº de abas</i>	Memória RAM (%)	Faixa de consumo da CPU (%)
1	8,7	2 - 6
2	8,8	4 - 7
5	9,0	6 - 12
10	9,9	11 - 32
25	10,1	21 - 54
50	10,1	34 - 68
75	10,1	52 - 93
100	10,2	38 - 91
125	10,3	58 - 91
200	10,3	45 - 91

É importante notar que, após fechar todas as abas, o consumo de recursos do sistema voltou às condições iniciais, além de que o desempenho do computador do cliente foi monitorado e, em nenhum momento, o consumo de recursos dos núcleos de CPU ficou travado em 100%. Outro teste realizado quando as 200 abas foram abertas, foi usar a interface de usuário em diversas abas aleatórias para comprovar que esta continuava funcional, mesmo com muitas conexões concorrentes: foi notada lentidão em comparação à operação com somente uma aba aberta, porém a interface continuou funcional. A figura 5.10 apresenta o gerenciador de recursos do navegador Google Chrome e, por conseguinte, a quantidade de recursos de CPU, memória RAM e rede demandados do computador do cliente com 200 abas da aplicação abertas.

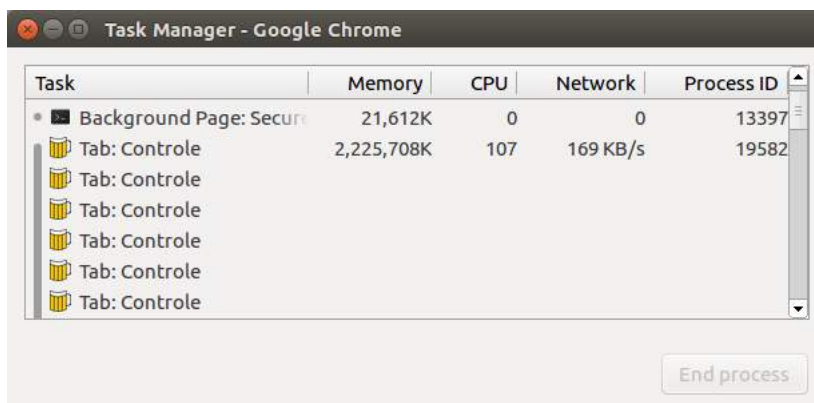


Figura 5.10: Consumo de recursos do computador do cliente para 200 abas abertas no navegador Google Chrome

Quanto aos resultados expressos na tabela 5.4, é importante salientar que, mesmo que o consumo mínimo da CPU não tenha crescido proporcionalmente com o aumento do número de abas, ainda assim à medida que este (número de abas) crescia, era notável que a CPU passava cada vez mais tempo em níveis mais elevados de processamento.

É importante salientar que testes no âmbito de segurança não foram conduzidos, embora este seja um tópico de grande interesse na área de internet das coisas (IoT). Nota-se também que nenhuma restrição de acesso à interface gráfica de usuário foi implementada neste projeto.

5.2.3 Registros e verificações em geral

O módulo de registros e verificações em geral, conforme descrito na seção 4.6.3, tem funções para controlar o registro de temperatura, ler o último valor registrado de temperatura, verificar a integridade de uma receita, responder para o cliente o nome de todas as receitas cadastradas no servidor e registrar o processo de brassagem. Esta seção é dedicada a apresentar somente resultados parciais do controle do registro de brassagem e resultados do registro do processo de brassagem, uma vez que a grande maioria destas funções tem seus resultados descritos nas seções 5.1.1 (registro de temperatura em Python) e 5.3 (interface gráfica de usuário).

Com relação ao controle do registro de temperaturas, a única observação aqui apresentada é que o arquivo de registros foi corretamente criado tanto durante as simulações do controle da brassagem quanto durante os testes de automação da parte mecânica. Isto implica que as funções que iniciam e param o script Python a partir do Node cumprem o seu objetivo.

Quanto ao registro do processo de brassagem, guardado no arquivo *backup.log*, após a simulação da receita descrita na seção 4.6.5, verificou-se que foi gerado um arquivo com 249

registros e que neste todos os códigos de brassagem da tabela 4.3 foram encontrados ao menos em uma linha de registro. A tabela 5.5 apresenta os campos *código*, *msg* e *timestamps:curr*) dos registros de situações notáveis, conforme descrito na seção 4.6.3, e importantes da brassagem.

Tabela 5.5: Apresentação dos campos *código*, *msg* e *timestamps:curr*) (timestamp no momento do log) convertida de *epoch time* para data/hora, das situações notáveis e importantes da brassagem

Linha	Código	Mensagem	Timestamp (UTC-3)	Notável
1	0	request to start production	23-05-2016 20:21:26	sim
2	1	production starting	23-05-2016 20:21:32	não
3	2	heating mash water	23-05-2016 20:21:32	sim
9	3	waiting for grains	23-05-2016 20:22:01	sim
12	4	mash ramp in progress	23-05-2016 20:22:13	sim
19	5	mash step[1] in progress	23-05-2016 20:22:44	sim
32	4	mash ramp in progress	23-05-2016 20:23:46	sim
60	5	mash step[2] in progress	23-05-2016 20:26:03	sim
74	6	sparging in progress	23-05-2016 20:27:11	sim
78	7	mash tun overflow	23-05-2016 20:27:31	não
84	6	sparging in progress	23-05-2016 20:28:01	não
126	8	heating wort to boil	23-05-2016 20:31:29	sim
127	9	boiling wort	23-05-2016 20:31:30	sim
139	10	added hop [0]	23-05-2016 20:32:30	sim
152	10	added hop [2]	23-05-2016 20:33:30	sim
165	10	added hop [1]	23-05-2016 20:34:30	sim
179	11	chilling the wort	23-05-2016 20:35:31	sim
203	12	chill finished - brewing finished	23-05-2016 20:37:32	sim
207	13	cleaning recirculation	23-05-2016 20:37:47	sim
254	13	cleaning recirculation	23-05-2016 20:41:42	não

5.2.4 Simulação do controle do processo de brassagem

Para simular o controle do processo de brassagem, foi iniciada a receita de *debug* descrita na seção 4.6.5 a partir da GUI e observado o comportamento tanto desta quanto das mensagens de *debug* impressas pela aplicação no terminal. A depuração de todos os módulos escritos para esta aplicação foi ativada e, inicialmente, nenhum *hardware* foi ligado à BBB. As figuras 5.11 e 5.12 apresentam excertos relevantes do conjunto de mensagens impresso no terminal.

```

routes Command: inProgress +16s
routes Command: getRecipes +134ms
log deleted recipes indexes: 12 +44ms
routes Command: startRequest +9s
log ready to rock! +75ms
log start is declared. Logging notable timestamp +51ms
log request to start production - code: 0 - auto: false +37ms
log { resp: 'success', warn: 'levedura;', err: '' } +84ms
log backup log started! +185ms
routes Command: startRecipe +2s
log production starting - code: 1 - auto: true +100ms
log backup log updated! +100ms
gpio Pin mash_pump turned HIGH +145ms
log startHeating is declared. Logging notable timestamp +21ms
log heating mash water - code: 2 - auto: true +55ms
routes Started to heat mash water +53ms
log Temperature logging started! +72ms
log backup log updated! +124ms
ctrl Executing temperature control loop[1] +801ms
ctrl Heating power variable: 2 +36ms
ctrl mash_heat: 66% heating power +14ms
gpio Pin mash_heat turned HIGH +11ms
ctrl Executing temperature control loop[1] +963ms
ctrl Heating power variable: 1 +42ms
ctrl mash_heat: 66% heating power +13ms
gpio Pin mash_heat turned HIGH +28ms
ctrl Executing temperature control loop[1] +937ms
ctrl Heating power variable: 0 +28ms
ctrl mash_heat: 66% heating power +10ms
gpio Pin mash_heat turned LOW +24ms
ctrl Executing temperature control loop[1] +950ms
ctrl Heating power variable: 2 +25ms
ctrl mash_heat: 66% heating power +11ms
gpio Pin mash_heat turned HIGH +27ms
log heating mash water - code: 2 - auto: true +877ms
log backup log updated! +37ms

```

(a) Acesso à GUI e início da brassagem

```

routes Client request: ready +120ms
ctrl Executing temperature control loop[1] +798ms
ctrl Ramp control started! +55ms
log sRamp0 is declared. Logging notable timestamp +13ms
log mash_ramp in progress - code: 4 - auto: true +20ms
log backup log updated! +57ms
ctrl Heating power variable: 1 +28ms
ctrl mash_heat: 0% heating power +12ms
gpio Pin mash_heat turned LOW +23ms
ctrl Executing temperature control loop[2] +915ms
ctrl Heating power variable: 1 +53ms
ctrl mash_heat: 66% heating power +15ms
gpio Pin mash_heat turned HIGH +15ms
ctrl Executing temperature control loop[2] +938ms
ctrl Heating power variable: 0 +38ms
ctrl mash_heat: 66% heating power +11ms
gpio Pin mash_heat turned LOW +19ms
ctrl Executing temperature control loop[2] +953ms
ctrl Heating power variable: 2 +19ms
ctrl mash_heat: 66% heating power +13ms
gpio Pin mash_heat turned HIGH +36ms
ctrl Executing temperature control loop[2] +941ms
log Log not updating, check temperature probe connection! +21ms
ctrl Heating power variable: 1 +14ms
ctrl mash_heat: 100% heating power +14ms
gpio Pin mash_heat turned HIGH +13ms
log mash_ramp[1] in progress - code: 4 - auto: true +883ms
log backup log updated! +46ms
ctrl Executing temperature control loop[2] +21ms
log Log not updating, check temperature probe connection! +176ms
ctrl Heating power variable: 1 +23ms
ctrl mash_heat: 100% heating power +14ms
gpio Pin mash_heat turned HIGH +25ms
ctrl Executing temperature control loop[2] +782ms

```

(b) Início do controle de temperatura

```

ctrl Step finished, starting next ramp or going to sparging. +19ms
ctrl Current temperature = 60
Reading errors = 9 +15ms
ctrl Executing temperature control loop[2] +911ms
ctrl setpoint pointer: 2 +4ms
ctrl End of the mash ramps/steps process 2 +5ms
ctrl Executing temperature control loop[2] +1s
ctrl setpoint pointer: 3 +10ms
ctrl End of the mash ramps/steps process 3 +8ms
ctrl Executing temperature control loop[2] +1s
ctrl setpoint pointer: 4 +3ms
ctrl End of the mash ramps/steps process 4 +17ms
ctrl Executing temperature control loop[2] +1s
ctrl setpoint pointer: 5 +19ms
ctrl End of the mash ramps/steps process 5 +19ms
log mash step[6] in progress - code: 5 - auto: true +509ms
log backup log updated! +29ms
ctrl Executing temperature control loop[2] +587ms
ctrl setpoint pointer: 6 +12ms
ctrl End of the mash ramps/steps process 6 +12ms
ctrl Executing temperature control loop[2] +1s
ctrl setpoint pointer: 7 +8ms
ctrl End of the mash ramps/steps process 7 +5ms
ctrl Executing temperature control loop[2] +1s
gpio Pin mash_heat turned LOW +17ms
gpio Pin boil_heat turned LOW +25ms
gpio Pin mash_pump turned LOW +32ms

```

(c) Final do controle de temperatura

```

ctrl Starting the sparging process +59ms
log startDrain is declared. Logging notable timestamp +17ms
log sparging in progress - code: 6 - auto: true +26ms
gpio Pin mash_valve turned HIGH +41ms
log backup log updated! +86ms
log startDrain is declared. Logging notable timestamp +5s
log sparging in progress - code: 6 - auto: true +22ms
log backup log updated! +38ms
log sparging in progress - code: 6 - auto: true +5s
log backup log updated! +39ms
ctrl Finished the first drain +5s
gpio Pin boil_valve turned HIGH +31ms
gpio Pin boil_pump turned HIGH +25ms
log startSparge is declared. Logging notable timestamp +31ms
log sparging in progress - code: 6 - auto: true +30ms
log backup log updated! +39ms
gpio Pin boil_pump turned HIGH +431ms
ctrl Tempo de sparging: 500 +9ms
ctrl Fake overflow ON for testing purposes : 500 +18ms
gpio Pin led turned HIGH +28ms
ctrl Overflow detected, pumping on hold +74ms
gpio Pin boil_pump turned LOW +9ms
log mash tun overflow - code: 7 - auto: true +4s
log backup log updated! +35ms
log mash tun overflow - code: 7 - auto: true +5s
log backup log updated! +34ms
log mash tun overflow - code: 7 - auto: true +5s
log backup log updated! +29ms
log mash tun overflow - code: 7 - auto: true +5s
log backup log updated! +28ms
log mash tun overflow - code: 7 - auto: true +5s
log backup log updated! +31ms
log mash tun overflow - code: 7 - auto: true +5s
log backup log updated! +29ms
ctrl Fake overflow OFF for testing purposes +484ms
gpio Pin led turned LOW +23ms
gpio Pin boil_pump turned HIGH +22ms
ctrl Tempo de sparging: 1000 +13ms

```

(d) Início da lavagem dos grãos (*sparging*)

Figura 5.11: Excertos relevantes das mensagens de depuração do processo de simulação da brassagem (parte 1)

```

ctrl mash drained, sparging process finished +2s
gpio Pin mash_valve turned LOW +27ms
gpio Pin boil_valve turned LOW +21ms
gpio Pin boil_pump turned LOW +19ms
ctrl Heating wort until it boils +44ms
log heatingBoil is declared. Logging notable timestamp +11ms
log heating wort to boil - code: 8 - auto: true +21ms
gpio Pin boil_heat turned HIGH +26ms
log backup log updated! +35ms
ctrl Boil time left: NaN +983ms
ctrl boiling started! Burn, baby, burn! +45ms
log boilStart is declared. Logging notable timestamp +12ms
log boiling wort - code: 9 - auto: true +48ms
ctrl Hop hop first scheduled to be added in 60000ms +25ms
ctrl Hop hop third scheduled to be added in 180000ms +14ms
ctrl Hop hop second scheduled to be added in 120000ms +13ms
log backup log updated! +38ms
ctrl Boil time left: 240000 +824ms
ctrl Boil time left: 240000 +1s
ctrl Boil time left: 240000 +1s
ctrl Boil time left: 240000 +1s
ctrl Boil time left: 240000 +1s
log boiling wort - code: 9 - auto: true +89ms
log backup log updated! +35ms
ctrl Boil time left: 234933 +883ms
ctrl Boil time left: 234933 +1s
ctrl Boil time left: 234933 +1s
ctrl Boil time left: 234933 +1s
ctrl Boil time left: 234933 +1s

```

(a) Início da fervura do mosto

```

ctrl Hop pointer closure: 0 +490ms
ctrl Adding hop hop first +14ms
log hAdd0 is declared. Logging notable timestamp +49ms
log added hop [0] - code: 10 - auto: true +28ms
log backup log updated! +45ms
gpio Servo angle set to: 11.25 +105ms
log boiling wort - code: 9 - auto: true +31ms
log backup log updated! +36ms
ctrl Boil time left: 179652 +212ms
ctrl Boil time left: 179652 +1s
ctrl Boil time left: 179652 +1s
ctrl Boil time left: 179652 +1s
ctrl Hop hop first added +481ms
gpio Servo angle set to: 0 +66ms
log boiling wort - code: 9 - auto: true +88ms
log backup log updated! +42ms

```

(b) Adição de lúpulo à panela de fervura

```

ctrl end of the boil +640ms
gpio Pin boil_heat turned LOW +22ms
log Stopping the temperature logging! +22ms
log chillStart is declared. Logging notable timestamp +48ms
log chilling the wort - code: 11 - auto: true +23ms
gpio Pin boil_valve turned HIGH +25ms
gpio Pin chill_valve turned HIGH +18ms
gpio Pin water_valve turned HIGH +42ms
ctrl Starting to chill the wort. Estimated time: 120000 +16ms
log backup log updated! +229ms
log Temperature logging stopped! +28ms
gpio Servo angle set to: 0 +18ms
log chilling the wort - code: 11 - auto: true +5s
log backup log updated! +37ms
log chilling the wort - code: 11 - auto: true +5s
log backup log updated! +42ms
log chilling the wort - code: 11 - auto: true +5s
log backup log updated! +39ms

```

(c) Início do resfriamento do mosto

```

ctrl Finished chilling the wort. +5s
gpio Pin boil_valve turned LOW +25ms
gpio Pin chill_valve turned LOW +16ms
gpio Pin water_valve turned LOW +20ms
log end is declared. Logging notable timestamp +16ms
log chill finished - brewing finished - code: 12 - auto: true +22ms
log backup log updated! +46ms
log waiting for the user ok before the cleaning - code: 12 - auto: true +5s
log backup log updated! +36ms
routes Client request: ready +4s
ctrl Start recirculation of water to clean the equipment +243ms
log cleaningStart is declared. Logging notable timestamp +21ms
log cleaning recirculation - code: 13 - auto: true +32ms
ctrl Mash tun Loop +32ms
gpio Pin mash_pump turned HIGH +26ms
log backup log updated! +147ms
log cleaning recirculation - code: 13 - auto: true +5s
log backup log updated! +36ms
log cleaning recirculation - code: 13 - auto: true +5s

```

(d) Recirculação de água para limpeza preliminar

Figura 5.12: Excertos relevantes das mensagens de depuração do processo de simulação da brassagem (parte 2)

Em seguida, após a certificação de que o processo de automação estava coerente com o processo de produção de cerveja e com as especificações do equipamento, por meio de ajustes incrementais nos códigos-fonte do projeto, foram ligados à BBB: interfaces de potência aos pinos de GPIO, com LEDs ligados às suas saídas para inspeção visual; relés de estado sólido ligados diretamente aos pinos de GPIO da BBB respectivos ao controle dos resistores de potência — à saída destes nada foi adicionado, uma vez que possuem LED indicador do estado da saída; e interface de potência ligada ao PWM, com o servo-motor acoplado à sua saída. O resultado desta simulação é apresentado na figura 5.13, na qual o esforço de equiparar as fases documentadas com as das figuras 5.11 e 5.12 foi parcialmente efetivo: em função do fato de que algumas partes de transição são rápidas, não foi prático documentar o equivalente ao *log* do terminal em imagens estáticas.



(a) Acesso à GUI e início da brassagem



(b) Início do controle de temperatura



(c) Final do controle de temperatura

(d) Início da lavagem dos grãos (*sparging*)

(e) Início da fervura do mosto



(f) Início do resfriamento do mosto



(g) Recirculação de água para limpeza preliminar

Figura 5.13: Momentos relevantes da simulação observados na GUI e nos circuitos de interface de potência

Quanto aos recursos de hardware consumidos durante o processo de brassagem, foi gravado um vídeo da tela do computador usando o software de captura de tela *SimpleScreenRecorder* e, posteriormente, o consumo de memória e CPU do sistema foi sintetizado na tabela 5.6. É importante notar que o uso de recursos do comando *htop*, usado como monitor de recursos para este caso, é da ordem de 10% e portanto não é desprezível, e também os dados da tabela 5.6 são valores instantâneos e não representam nenhum tipo de inferência estatística.

Também foi usado o comando *sar* do pacote *sysstat* para gerar um arquivo de *log* de performance da CPU, memória RAM e volume de tráfego de rede IPv4, a partir do qual foram gerados os gráficos apresentados nas figuras 5.14 e 5.15, que apresentam o uso de CPU e memória RAM; e recursos de rede utilizados em função do tempo, respectivamente.

A partir dos dados obtidos, constatou-se que os recursos de processamento (CPU) ficaram abaixo de 40%, salvo em um único ponto do gráfico da figura 5.14, indicando que a BBB trabalhou muito abaixo do seu limite de processamento. Com relação à memória RAM, foi observado que seu uso está entre 55% e 60%. É notável a partir da observação do gráfico que, após o fim do processo de fervura, a quantidade de processamento demandada da BBB foi reduzida pela metade.

Tabela 5.6: Consumo de recursos de memória e processamento da BBB ao longo do processo de brassagem

Fase da brassagem	Memória RAM (Mb)	Consumo da CPU (%)
Seleção da receita	138	10
Autorização do início de produção	138	22
Aquecimento da água da brassagem	140	28
Transição de início do controle de temperatura	140	34
Controle de rampa	140	20
Controle de degrau	140	35
Fim do controle de temperatura	141	28
Deteção de transbordamento	141	24
Lavagem dos grãos	141	29
Fim da lavagem	142	34
Fervura	143	31
Resfriamento do mosto	139	14
Começo da recirculação de limpeza	139	27
Fim do processo — ocioso	139	15

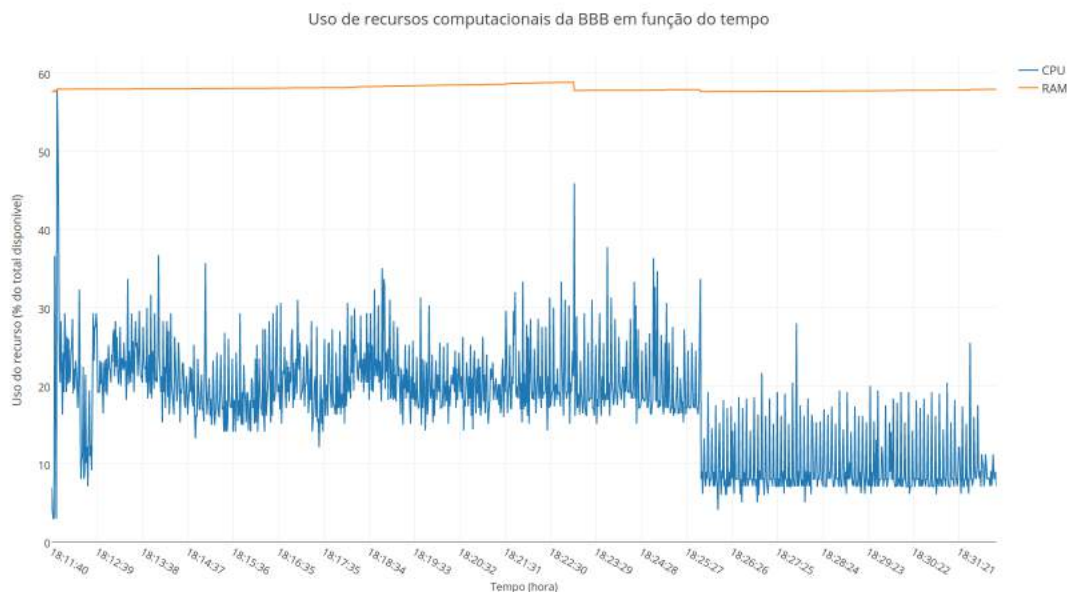


Figura 5.14: Gráfico de consumo de recursos de CPU e memória RAM da BBB em função do tempo, durante uma simulação de brassagem

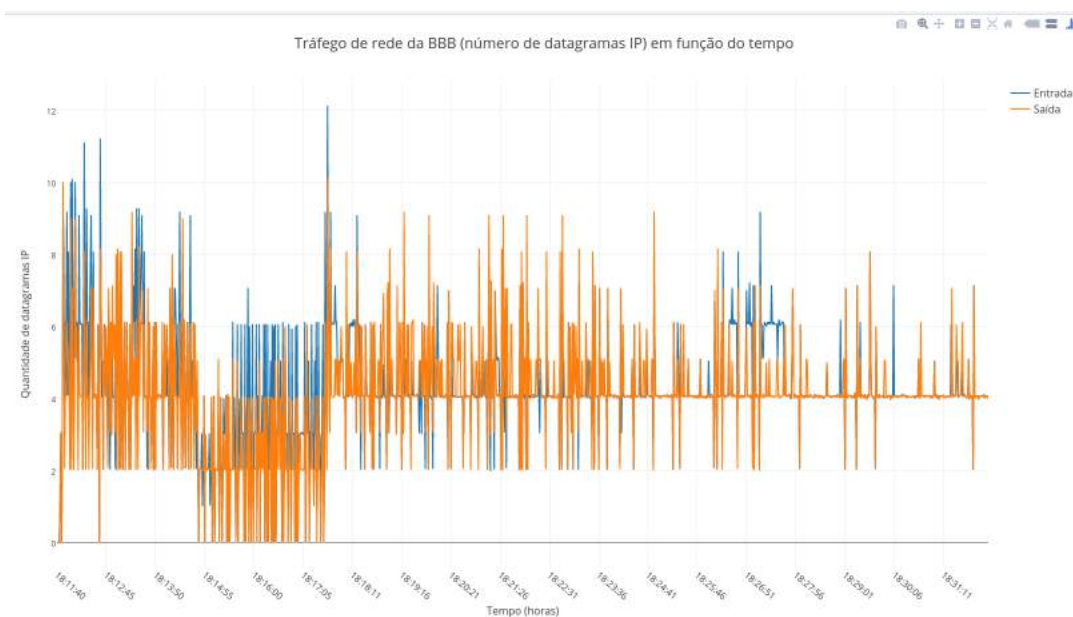


Figura 5.15: Gráfico de consumo de recursos de rede da BBB em função do tempo, durante uma simulação de brassagem

Node.js foi a linguagem de programação (interpretador da linguagem Javascript) adotada *de facto* como a linguagem de servidor para o presente projeto. Seu uso foi motivado pela existência da biblioteca de acesso a hardware *Octalbonescript* e da IDE Cloud9, e se tornou a escolha do desenvolvedor, uma vez que foi considerada de fácil aprendizado, reduziu a complexidade do projeto uma vez que reuniu todas as particularidades deste em torno de si — desde a leitura dos sensores de temperatura até a comunicação com o cliente. Cabe ressaltar

que, embora a atualização dos programas instalados na BBB não tenha sido parte do escopo deste projeto, neste momento o interpretador Node.js está na versão 6.0, enquanto a empregada na BBB é a versão 0.10 — o que pode ter impactado negativamente na performance do sistema em algum momento.

A partir do estudo de acesso a GPIO, cujos resultados estão documentados na seção 5.2, foi observado que o tempo mínimo de chaveamento foi limitado por algum fator que não a carga da CPU, o que reforçou a hipótese de a versão antiga do Node estar limitando o seu desempenho — testes mais aprofundados devem ser realizados para comprovar ou refutar esta hipótese.

O servidor web implementado por meio do *framework* Express se mostrou estável e ocupou poucos recursos da BBB, além de passar por um teste de estresse que comprovou sua virtude para processamento de I/O intensivo. Embora seja notável que nenhum equipamento de produção de cerveja deve ter 200 operadores conectados a ele, por questões logísticas e de segurança, o teste de estresse mostrou a possibilidade de implementar um servidor em Node para fazer comunicação com diversas cervejarias de consumidores finais e promover não somente serviços de *backup* para os usuários do produto mas também a possibilidade de desenvolver aplicações de internet das coisas, no sentido de reunir grandes conjuntos de dados e aplicar algoritmos de aprendizado de máquina para atuar na performance dos equipamentos, detectar falhas antes mesmo que elas aconteçam, levantar padrões de preferências de consumidor para elaborar receitas de cerveja mais saborosas, dentre outras inúmeras possibilidades.

No âmbito de internet das coisas, a questão da segurança é um tópico fortemente discutido e trabalhado, mas que não foi testado neste trabalho. Seria preciso verificar quais os recursos de proteção disponíveis no *framework* Express ou mesmo em outros módulos.

Ainda, no módulo de registros, está implementada a função que verifica se uma receita pode ser iniciada ou não. Talvez fosse interessante fazer esta verificação no momento em que a receita é salva, porém com o recurso de salvamento automático, este pode não ser o melhor caminho. Também poderia ser interessante modificar o paradigma de chamadas à função de *log* de tal maneira que fosse evitado o processo de iniciar e parar o timer do laço *setInterval* que a invoca no intuito de modificar os seus parâmetros de chamada.

A simulação do controle do processo de brassagem se mostrou essencial para garantir a implementação coerente da automação do sistema. Ainda assim, não somente a aplicação deveria ter sido arquitetada desde o começo favorecendo o processo de *debug* como ferramentas

auxiliares de prevenção a erros e testes deveriam ter sido adotadas. Uma ferramenta simples de prevenção a erros e boas práticas é o *jshint* — uma ferramenta de análise estática de código inicialmente desenvolvida por Douglas Crockford, o criador e disseminador da codificação JSON. As mensagens de *debug* impressas no terminal também poderiam ter sido separadas de maneira mais eficiente e não somente por módulos.

Quanto aos recursos computacionais demandados pelo processo de produção de cerveja, quando em modo automático, foi importante notar que não houve sobrecarga do sistema, uma vez que o usuário não é impedido de acessar nenhuma função da GUI neste período. Foi interessante notar que, após a fervura do mosto, o consumo de recursos de processamento caiu pela metade e este fator foi atribuído a princípio à parada de execução do script Python no final da função *theBoil*. Cabe ressaltar que esta foi somente uma hipótese e precisa ser testada antes de ser tomada como verdade absoluta. No quesito de recursos de rede, o tráfego foi baixo mesmo com o *overhead* gerado pelo envio de dados desnecessários do servidor para o cliente — os mesmos dados discutidos com relação ao arquivo de registro da brassagem.

5.3 Interface de usuário

Nesta seção são apresentados os resultados obtidos a partir do teste prático da interface de usuário, por meio de um computador com tela de resolução WXGA (1766 x 768 pixels).

5.3.1 Página inicial

A página inicial foi composta de uma imagem clicável com uma mensagem de boas vindas parodiando o uso do compilador Gcc como se estivesse sendo compilada a receita de uma cerveja. O clique nesta imagem leva à página de apresentação. O resultado visual de seu desenvolvimento pode ser verificado na figura 5.16.



Figura 5.16: Página inicial da GUI

5.3.2 Página de apresentação

A página de apresentação contém uma breve descrição do projeto, cujo nome fantasia adotado foi *Cervejaria do futuro*, em alusão à automação e ao acesso pela internet. Foi adicionado um aviso indicando que a interface pode apresentar problemas de design em resoluções diferentes da WXGA, e também uma seção com vídeos de testes preliminares do sistema. O resultado visual do desenvolvimento desta página pode ser verificado na figura 5.17.

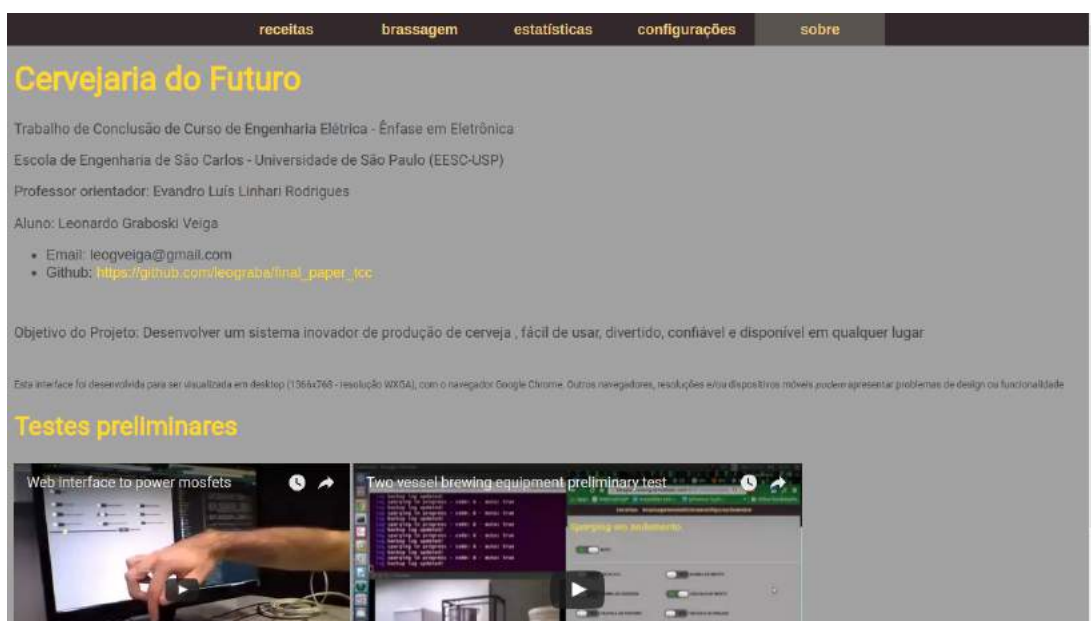


Figura 5.17: Página de apresentação do trabalho

Também é importante notar, na parte superior da imagem, a implementação da barra de navegação documentada na seção 4.7.1 e o funcionamento adequado da detecção da presente página, com o intuito de ressaltar o link (ou botão) desta — que no caso é o link *sobre*.

5.3.3 Interface de configurações

Para mudar para a interface de configurações, o botão correspondente da barra de navegação foi clicado e a página resultante é apresentada na figura 5.18. Para verificar a consistência da configuração de data e hora do sistema embarcado, foi aberta uma aba do navegador na página web <http://ntp.br>, comentada na seção C.0.4. Simultaneamente, foi realizada uma tentativa de reconfiguração da data/hora por meio da GUI, sendo os resultados apresentados na figura 5.19. Posteriormente, o serviço do *NTP* foi desligado na BBB e nova tentativa de reconfiguração da data/hora foi realizada, cujo resultado está expresso na figura 5.20.

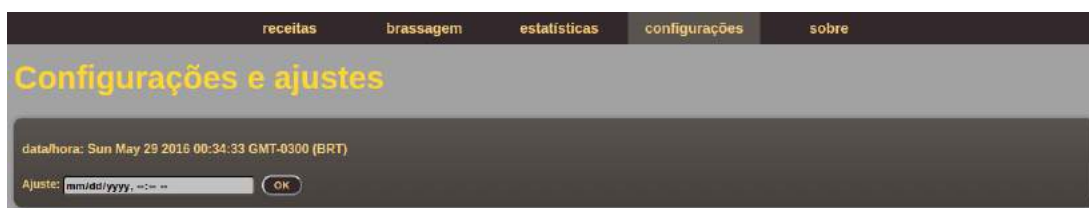


Figura 5.18: Página de configuração de data/hora



Figura 5.19: Tentativa de configuração de data/hora negada

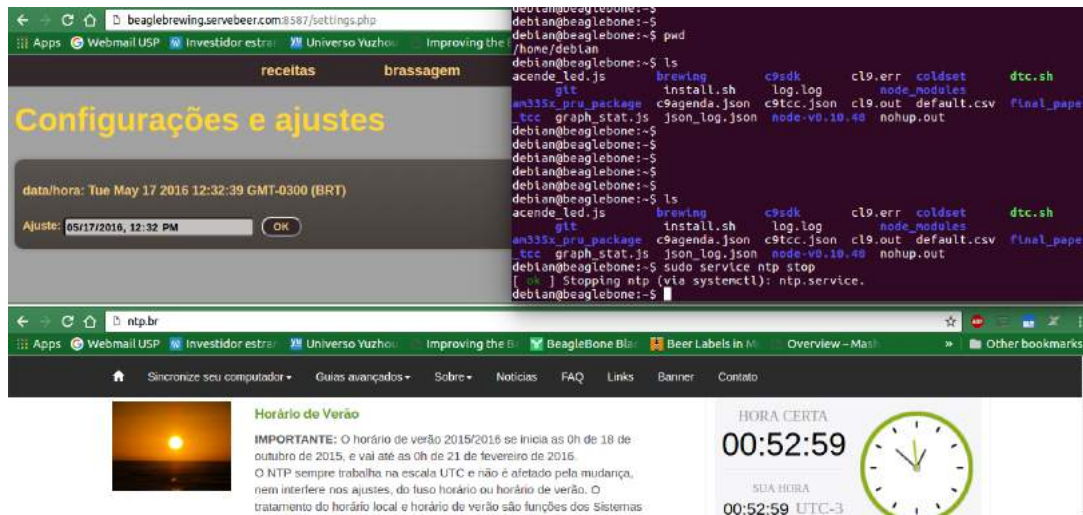


Figura 5.20: Tentativa de configuração de data/hora aceita

5.3.4 Página de estatísticas

Apesar do nome sugestivo, conforme abordado no capítulo 4, a página apresenta o gráfico de temperatura em função do tempo gerado a partir do script em Python e também gera e atualiza o gráfico dinâmico e interativo por meio do *framework* Rickshaw. O gráfico gerado em Python e apresentado nesta página é o mesmo da figura 5.5 e, como seus resultados estão comentados na seção 5.1.2 estes serão omitidos aqui. Já com relação ao gráfico dinâmico, este é apresentado na figura 5.21 sob as seguintes condições:

- Registro de temperatura do DS18B20 gerado a partir do script em Python.
- Aquecimento de um volume aleatório de água e retirada da sonda deste volume por um período de tempo aleatório.
- Ajuste dos pontos inicial e final da plotagem, por meio da barra deslizante sob o gráfico.
- Informação detalhada de um ponto do gráfico a partir do posicionamento do cursor sobre ele.
- Seleção de somente um conjunto de dados para plotagem em área e com ligação de pontos interpolada.

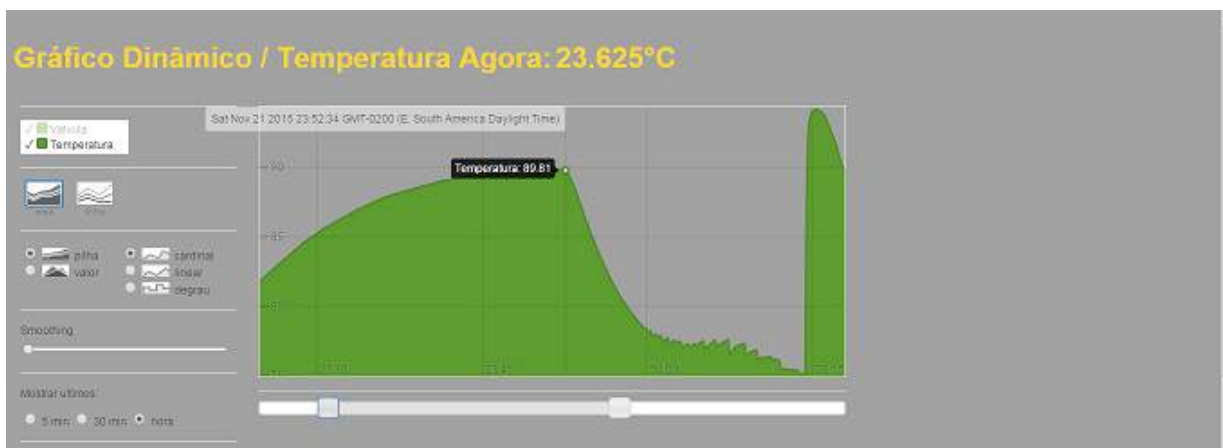


Figura 5.21: Gráfico dinâmico de temperatura em função do tempo

A figura 5.22 documenta o uso do filtro de médias móveis para quatro situações distintas: sem aplicação de filtro — 373 pontos plotados; 40 pontos plotados; 16 pontos plotados e; 4 pontos plotados.

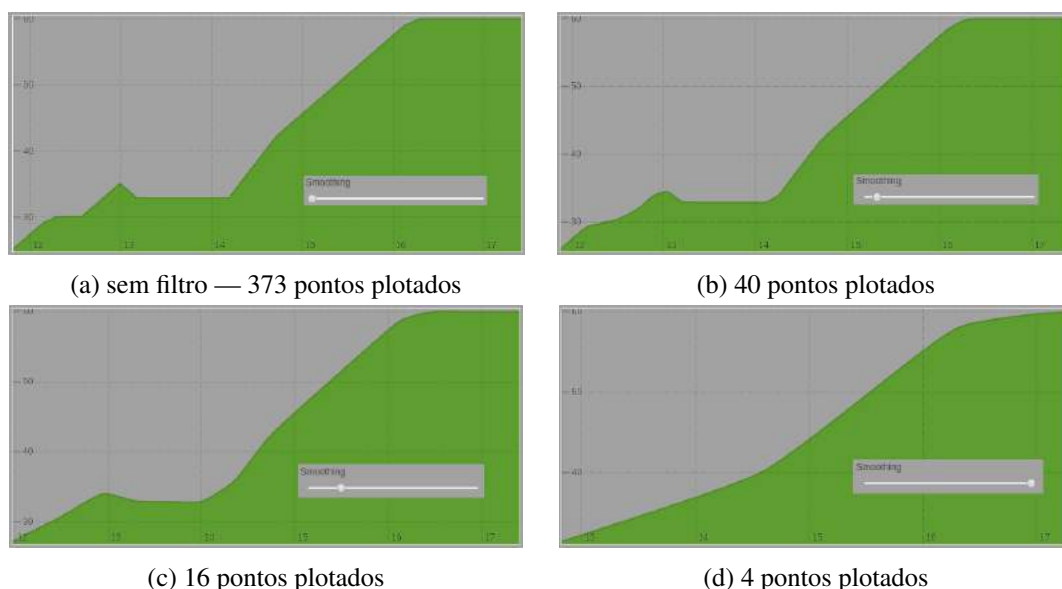


Figura 5.22: Aplicação do filtro de médias móveis com diferentes intensidades

Por *pontos plotados*, entende-se que após os 373 pontos do registro de temperatura passarem pelo filtro de médias móveis sob determinada configuração de intensidade, estes são reduzidos aos números citados — 40, 16 e 4 pontos. Como o ajuste do filtro é feito por meio de uma barra sem graduação, não há informações quantitativas sobre este senão a realimentação visual da barra provida na figura 5.22.

A partir da realimentação visual promovida pela figura 5.22, é possível notar que é melhor o usuário não aplicar o filtro, uma vez que este promove distorção da curva original se

configurado com intensidade inadequada, como observado nas situações (b), (c) e (d).

5.3.5 Interface de brassagem

A interface de brassagem possibilitou ao operador do sistema atuar neste, fosse manualmente quando não havia produção em andamento, ou automaticamente ao iniciar uma brassagem. Embora todas as situações possíveis desta interface, assim como a comprovação da sua efetividade, estejam documentadas na seção 5.2.4, em especial na figura 5.13, ainda assim são apresentadas duas amostras em melhor resolução, obtidas por meio de captura de tela e não fotos. Na figura 5.23 é apresentada a interface que permite o início de uma receita ou ir para o controle manual do sistema; na figura 5.24 é apresentado o painel de controle do sistema, que fica travado durante a produção automática de cerveja, servindo somente para propósitos de monitoramento neste caso.



Figura 5.23: Interface gráfica para configuração de início da brassagem



Figura 5.24: Interface gráfica do painel de controle do sistema mecânico

Com o sistema parado, o arquivo *lockfile* foi editado para simular que havia uma receita em produção. Isto foi feito com o objetivo de verificar que não é possível acessar a interface da figura 5.23 nesta situação, e o resultado foi positivo: ao tentar o acesso, a página foi redirecionada para o painel de controle da figura 5.24.

Observações do painel de controle decorrentes da simulação em bancada documentada na seção 5.2.4, mostraram que por vezes o estado do sistema na GUI fica defasado com relação ao estado do sistema na BBB e no hardware de interface. No modo manual, *glitches* na GUI foram observados com frequência ao mudar o estado dos pinos de GPIO ou do PWM. Ambas as situações se devem ao fato de que a taxa de atualização do painel de controle é de 0,5 segundos — comprovada quando a taxa de atualização foi modificada temporariamente para 0,1 segundos e os atrasos e *glitches* cessaram. Outro ponto notável do painel de controle do sistema é que não foi implementado nenhum tipo de restrição com relação a quem pode acessar este recurso — com exceção do fato de que, se o serviço de *no-ip* fosse desativado, somente usuários conectados na rede LAN poderiam acessar a GUI.

5.3.6 Gerenciador e editor de receitas

A figura 5.25 apresenta como resultado a interface gráfica do gerenciador de receitas, para o caso da receita *Exemplo Kolsch* deletada, porém com possibilidade de recuperação, e também com a prévia da receita *TheMightyMightyIPA* carregada, devido ao fato de o cursor estar posicionado sobre o seu nome na lista. Ao recarregar a página, foi notado que a receita *Exemplo Kolsch* não foi listada, comprovando a funcionalidade de deletar receita.



Figura 5.25: Gerenciador de receitas da UI
Cursor sobre a receita *TheMightyMightyIPA* e receita *Exemplo Kolsch* excluída

Foi observado que, ao correr o cursor pela lista de receitas, diversas prévias destas receitas eram carregadas sequencialmente e posteriormente, quando o cursor se fixava sobre uma receita específica, havia um atraso até que a sua prévia fosse carregada. Tal comportamento levou à observação de recursos de CPU e memória RAM da BBB na seguinte condição de estresse: foi movido aleatoriamente o cursor sobre a lista de receitas por 60 segundos e, após isto, ele foi fixado sobre uma receita específica e esperou-se que a prévia se estabilizasse nesta receita. Os resultados estão contidos na figura 5.26, cujos instantes notáveis estão descritos na tabela 5.7. Para relacionar corretamente os momentos das ações executadas, foi gravado um vídeo de captura de tela enquanto o procedimento era realizado, com um relógio aberto para garantir os instantes de transição.

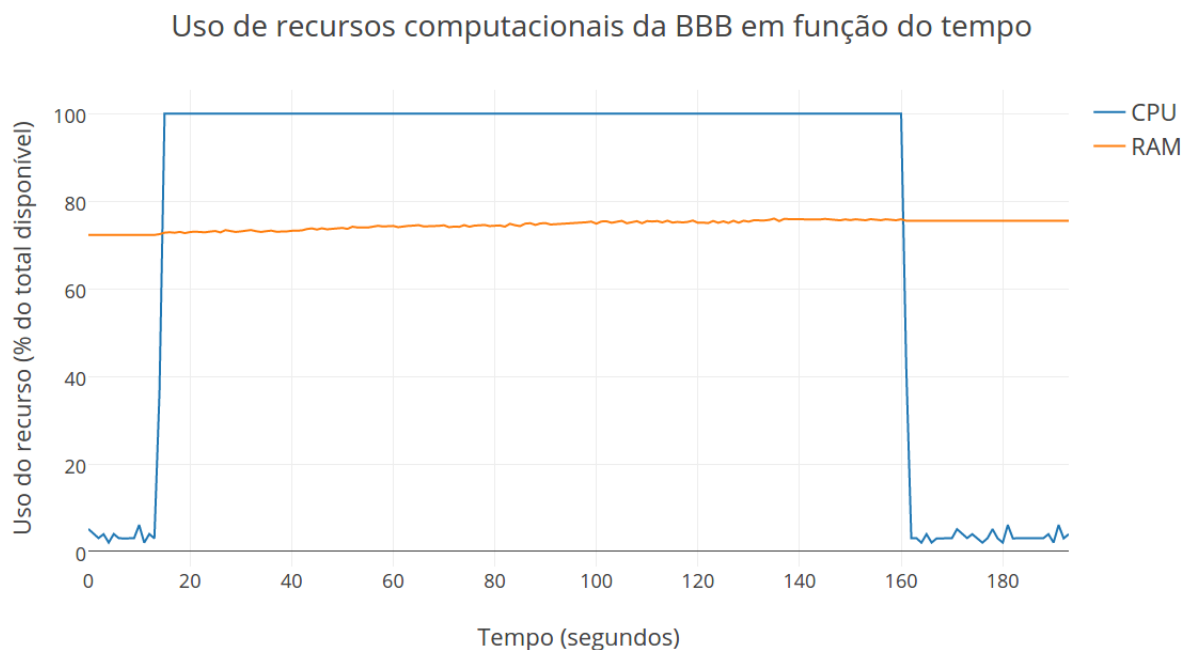


Figura 5.26: Gráfico de consumo de recursos de CPU e memória RAM da BBB em função do tempo, durante estresse do gerenciador de receitas

Tabela 5.7: Instantes notáveis do estresse do gerenciador de receitas

Descrição	Tempo (s)
Início do monitoramento dos recursos da BBB	0
Início da movimentação do cursor (estresse)	14
Final da movimentação do cursor (estresse)	75
Estabilização da prévia	161
Fim do monitoramento dos recursos da BBB	193

A partir dos resultados apresentados, observa-se que mesmo com o gerenciador de receitas cumprindo sua função, a função de prévia da GUI consumiu 100% dos recursos de CPU

da BBB desde o instante em que foi iniciado o estresse do gerenciador até 1 minuto e 26 segundos após este. Também é possível observar a partir do gráfico da figura 5.26 que houve um aumento da demanda do sistema por memória RAM durante o período de estresse — esta demanda se estabilizou, porém durante os 22 segundos nos quais a demanda por processamento esteve na faixa de grandeza das condições iniciais, a demanda por memória RAM não foi reduzida.

Para testar a criação de uma nova receita, foi escolhido o nome "*Nov@? Re\$\$eita#*", que é um nome inválido dado o uso de caracteres especiais. O resultado é apresentado na figura 5.27. A partir destes resultados, percebe-se que o esforço de proteção aplicado ao servidor e descrito na seção 4.7.6, que impediu esta receita de ser salva, não foi correspondido do lado do cliente, permitindo a este realizar uma operação ilegal. Nota-se também na figura 5.27 (d) que caracteres especiais foram passados para a URL do editor de receitas.



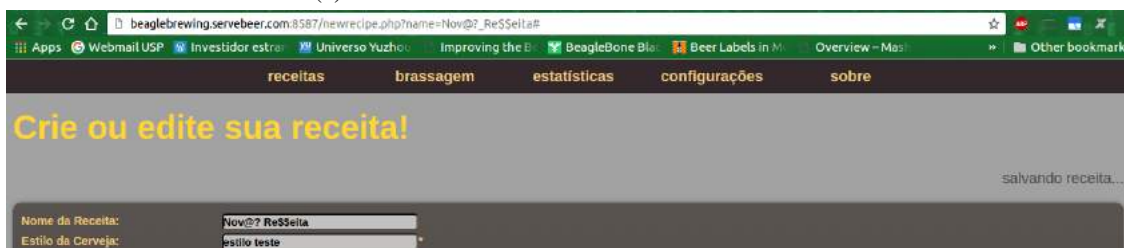
(a) Botão para criar nova receita



(b) Caixa de entrada auto-explicativa para o usuário



(c) Tentativa de criar receita com nome inválido



(d) Nome inválido aceito

Figura 5.27: Teste de criação de receita com nome inválido

Após a verificação de que foi possível a edição de uma receita com nome inválido, mas sem a possibilidade de salvá-la, foram executados testes para uma receita cujo nome escolhido foi "*Exemplo American Stout*" e cujo conteúdo inserido apresentou parâmetros compatíveis com uma receita real. Do ponto de vista do cliente, foi comprovado o funcionamento

da função recursiva que adiciona campos novos à medida que a receita é preenchida, conforme explicado na seção 4.7.6 e ilustrado na figura 5.28, que compara uma receita vazia à "Exemplo American Stout".



Figura 5.28: Editor de receitas

Uma vez que a função recursiva é executada na máquina do cliente, a partir das ferramentas de desenvolvedor do navegador Google Chrome foi observado o comportamento da execução do código javascript durante o evento de perda de foco de um campo da seção maltes do formulário. Esta seção foi escolhida devido ao fato de ela ter 7 linhas de ingredientes preenchidas no momento do teste e, portanto, apresentar a condição que demanda o maior número de chamadas possível da função recursiva *rearrange*. O resultado é apresentado na figura 5.29. A máquina que foi utilizada para este teste possui 4 núcleos Intel Core i5 M 580 @ 2,67GHz e dois pentes de memória RAM SODIMM DDR3 4GB @ 1066MHz.

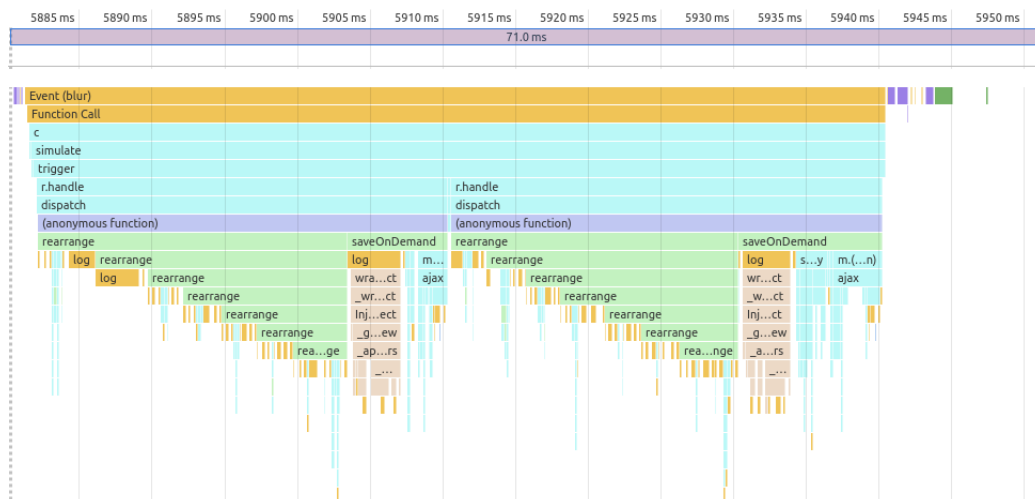


Figura 5.29: Análise de execução do código Javascript da interface de edição de receitas

Este teste foi repetido para obter um número de 10 amostras dos tempos de execução total do frame, de cada iteração da função *rearrange* e do tempo necessário para salvar a receita,

por meio da função *saveOnDemand*. Os resultados são apresentados na tabela 5.8.

Tabela 5.8: Tempos relativos a processamento do editor de receitas, no cliente

Descrição	Média (ms)	Máximo (ms)	Mínimo (ms)	Desvio padrão (ms)	Coef. de variação (%)
Tempo do frame	64,00	78,30	47,10	9,46	14,78
Tempo total <i>rearrange</i> 1ª iteração	18,78	25,09	14,15	3,78	20,13
Tempo de processamento <i>rearrange</i> 1ª iteração	0,64	1,52	0,16	0,41	64,06
Tempo total <i>rearrange</i> 2ª iteração	14,95	19,91	10,54	3,30	22,07
Tempo de processamento <i>rearrange</i> 2ª iteração	0,57	1,1	0,17	0,30	52,63
Tempo total <i>rearrange</i> 3ª iteração	11,85	15,22	8,41	2,10	17,72
Tempo de processamento <i>rearrange</i> 3ª iteração	0,56	0,76	0,18	0,20	35,71
Tempo total <i>rearrange</i> 4ª iteração	9,35	12,05	7,17	1,61	17,22
Tempo de processamento <i>rearrange</i> 4ª iteração	0,58	1,00	0,16	0,24	41,38
Tempo total <i>rearrange</i> 5ª iteração	7,44	9,48	5,59	1,23	16,53
Tempo de processamento <i>rearrange</i> 5ª iteração	0,65	0,98	0,33	0,24	36,92
Tempo total <i>rearrange</i> 6ª iteração	5,39	6,85	4,03	0,88	16,33
Tempo de processamento <i>rearrange</i> 6ª iteração	0,59	0,98	0,29	0,21	35,59
Tempo total <i>rearrange</i> 7ª iteração	3,12	3,96	2,20	0,61	19,55
Tempo de processamento <i>rearrange</i> 7ª iteração	0,76	1,95	0,32	0,45	59,21
Tempo total <i>saveOnDemand</i>	5,01	7,16	3,85	1,11	22,16
Tempo de processamento <i>saveOnDemand</i>	0,23	0,54	0,14	0,13	56,52

A partir dos resultados apresentados, foi notado que tanto a função *rearrange* quanto *saveOnDemand* estavam sendo executadas duas vezes, impondo uma redundância não prevista à interface de edição de receitas e, por conseguinte, consumindo recursos em dobro desta — além de isto significar o dobro de operações de escrita na memória eMMC da BBB.

Quanto ao tempo médio total de execução da função *rearrange*, foi notado que este é diminuído para cada iteração: tal resultado era esperado, já que o tempo total da n-ésima iteração é o somatório do tempo de execução desta e de todas as iterações subsequentes. Já

o tempo de processamento para cada iteração não diminuiu, o que também é esperado, uma vez que a função *rearrange* não tem laços de repetição internos que dependam do valor n da n -ésima iteração. Ainda assim, para todos os valores médios da tabela 5.8, percebeu-se que estes apresentam baixa confiabilidade estatística, dados os valores dos coeficientes de variação não desprezíveis.

A interface de usuário foi projetada para apresentação em um display de resolução específico, muito embora testes em outras resoluções tenham mostrado que ela se ajustou sem perda de funcionalidades e, em certos casos, com poucas distorções de design. Este tópico é extremamente relevante quando se fala em controle via internet, já que a popularização dos *smartphones* promoveu o acesso da população à internet por meio de displays dos mais variados tamanhos. Certamente seria de grande utilidade o desenvolvimento de aplicativos para celulares.

No que diz respeito à barra de navegação, esta não foi inicialmente planejada, porém promoveu a navegabilidade da GUI, que antes era cansativa em função de uma estrutura com diversos botões de voltar e um menu que concentrava todas as funcionalidades. Neste ponto, a barra de navegação foi um dos itens de sucesso da parte do projeto voltada ao usuário. A página de configurações trouxe conhecimento aplicado acerca do uso do protocolo NTP, porém a sensação que ela transmite é de que faltam opções de configuração e que, talvez esta página em específico nem devesse existir e a função de ajuste de hora deveria ser realocada.

A página de estatísticas apresentou uma implementação interessante de gráfico dinâmico e configurável em diversos aspectos, o que trouxe à reflexão: o usuário final realmente precisa de todos estes recursos de configuração? Ele quer estes recursos à sua disposição? Ele se sente seguro diante de uma gama de recursos que não irão efetivamente melhorar sua experiência com o produto? Mesmo que a resposta para todas estas questões seja sim, ainda assim ficou claro, após a observação dos resultados, que ao menos o filtro de médias móveis deveria ser omitido, pois além de não trazer vantagem, pode distorcer o gráfico de temperatura.

Quanto à interface de brassagem, sua implementação ficou limpa e funcional, ótima para controle tanto por meio da interface *desktop* à qual ela foi projetada quanto por meio de um dispositivo móvel. Seu único problema — a questão do *glitch* — foi solucionado ao mesmo tempo em que foi detectado. Obviamente a solução impacta em maior consumo de recursos computacionais da BBB, exigindo um novo estudo, porém é inegável sua eficácia.

A exemplo dos *glitches* da interface de brassagem, a interface de receitas também apresentou alguns problemas, porém a partir do momento que estes foram notados, sua solução

é simples do ponto de vista do desenvolvedor desta GUI. A interface cumpriu seu propósito de maneira fluída e despreocupante do ponto de vista de usuário, sem impactar significativamente no consumo de recursos da máquina do cliente.

5.4 Circuitos de interface entre a BBB e sensores/atuadores

Esta seção apresenta os resultados de simulação e de testes de bancada referentes aos circuitos de acionamento, assim como considerações acerca destes resultados. A figura 5.30 apresenta a implementação em matriz de conexões elétricas (*protoboard*) dos circuitos ensaiados nesta seção.

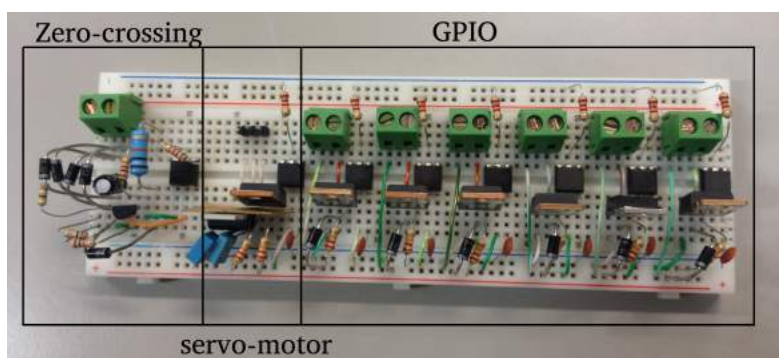


Figura 5.30: Implementação em *protoboard* dos circuitos de interface de potência, servo-motor e detector de passagem por zero

5.4.1 Acionamentos de potência

O módulo de acionamentos de potência foi simulado no software LTSpice. Para simular uma bomba como carga, foi utilizado um resistor de $7,5\Omega$ em série com um indutor de 30mH. O valor do resistor foi calculado com base nos parâmetros de tensão e corrente do atuador e o valor do indutor foi escolhido para que a carga não fosse puramente resistiva — a impossibilidade de determinar a indutância na prática se deve ao fato de que a bomba já possui algum circuito de controle, o que significa que o modelo de simulação empregado foi genérico. O sinal de entrada foi uma onda quadrada de período 100ms, valor alto o suficiente para excluir transitórios e válido uma vez que o chaveamento na prática é esporádico. A figura 5.31 apresenta o resultado da simulação.

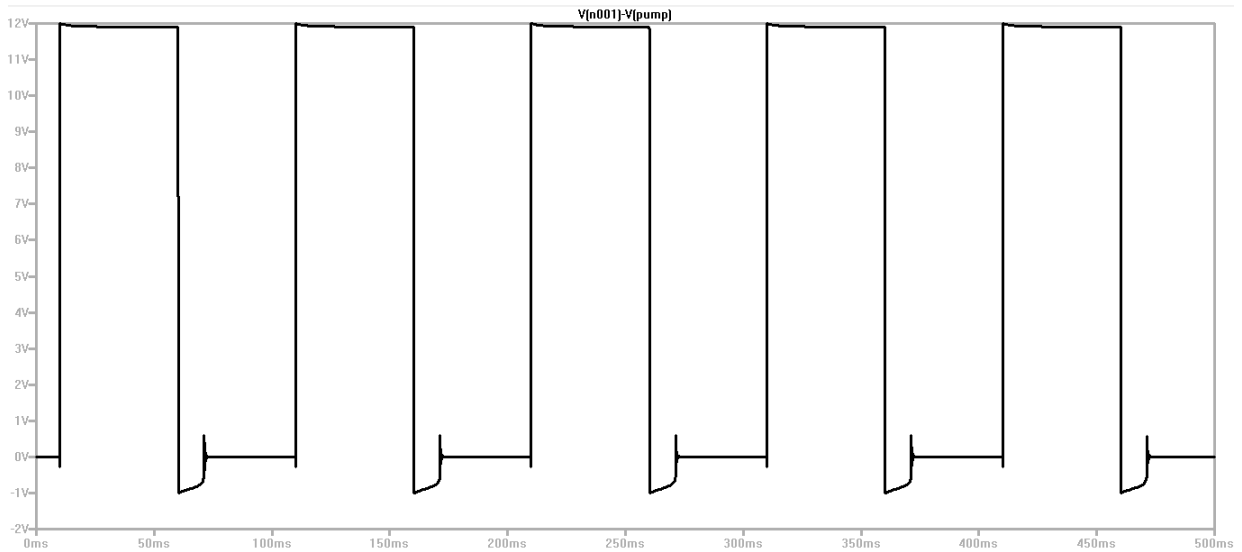


Figura 5.31: Resultado da simulação do módulo de acionamentos de potência com carga simulando a bomba de recirculação

A simulação da válvula utilizou o valor calculado de $9,6\Omega$ para o resistor e 300mH para o indutor. Neste caso, o valor do indutor também foi assumido para que a carga simulada não fosse puramente resistiva. O sinal de entrada foi uma onda quadrada de período 100ms, valor alto o suficiente para excluir transitórios e válido uma vez que o chaveamento na prática terá um período muito mais alto. A figura 5.32 apresenta o resultado da simulação.

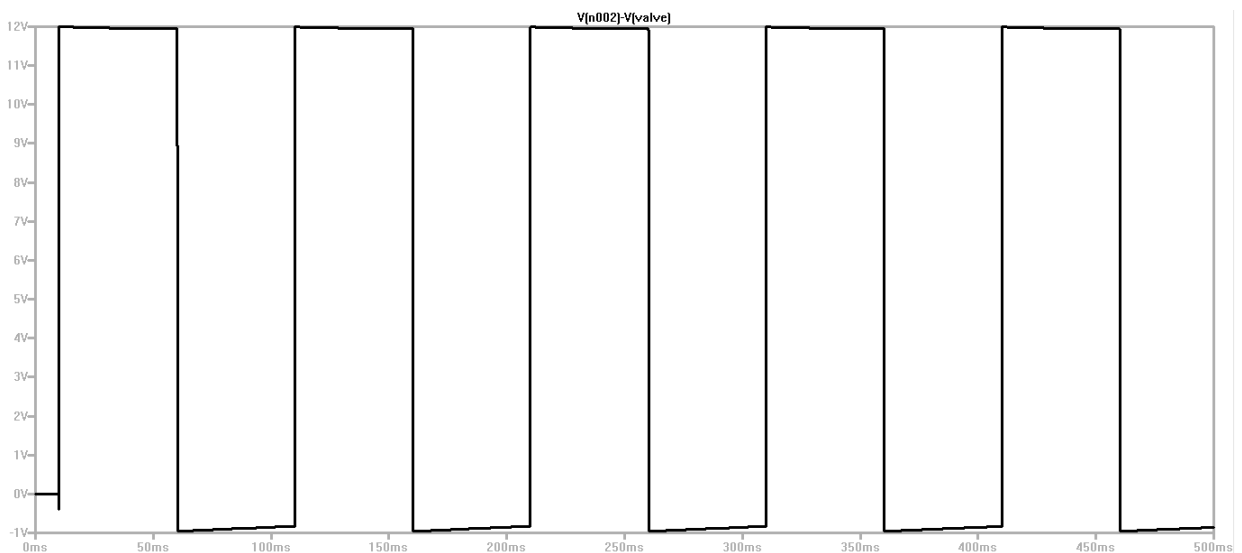


Figura 5.32: Resultado da simulação do módulo de acionamentos de potência com carga simulando uma válvula solenóide

O SSR foi simulado como uma carga puramente resistiva de 1600Ω e o sinal de entrada foi uma onda quadrada de 20ms de período, suficiente para excluir transientes de chaveamento. A figura 5.33 apresenta o resultado da simulação.

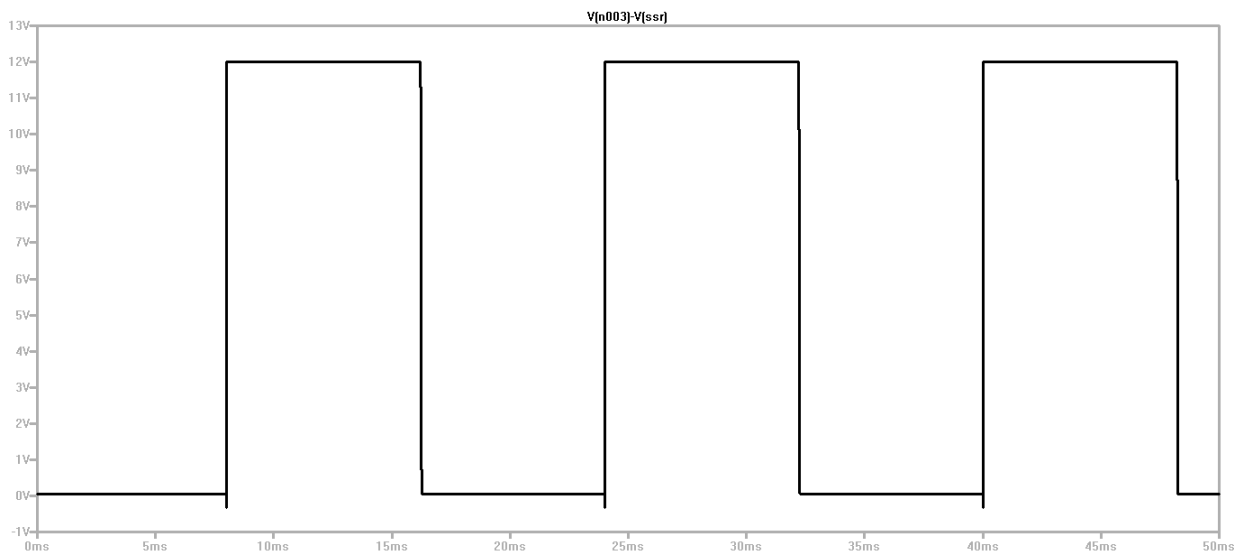


Figura 5.33: Resultado da simulação do módulo de acionamentos de potência com carga simulando o SSR

Com relação aos ensaios de bancada do módulo de acionamento das bombas, SSR e válvulas, o primeiro ensaio foi realizado com a saída em aberto e para valores de resistor de porta do MOSFET de $10\text{k}\Omega$, cujo resultado é apresentado na figura 5.34, e $22\text{k}\Omega$, cujo resultado é apresentado na figura 5.35. Foram registradas com o osciloscópio as formas de onda no pino da BBB, em verde, e na porta do MOSFET, em amarelo.

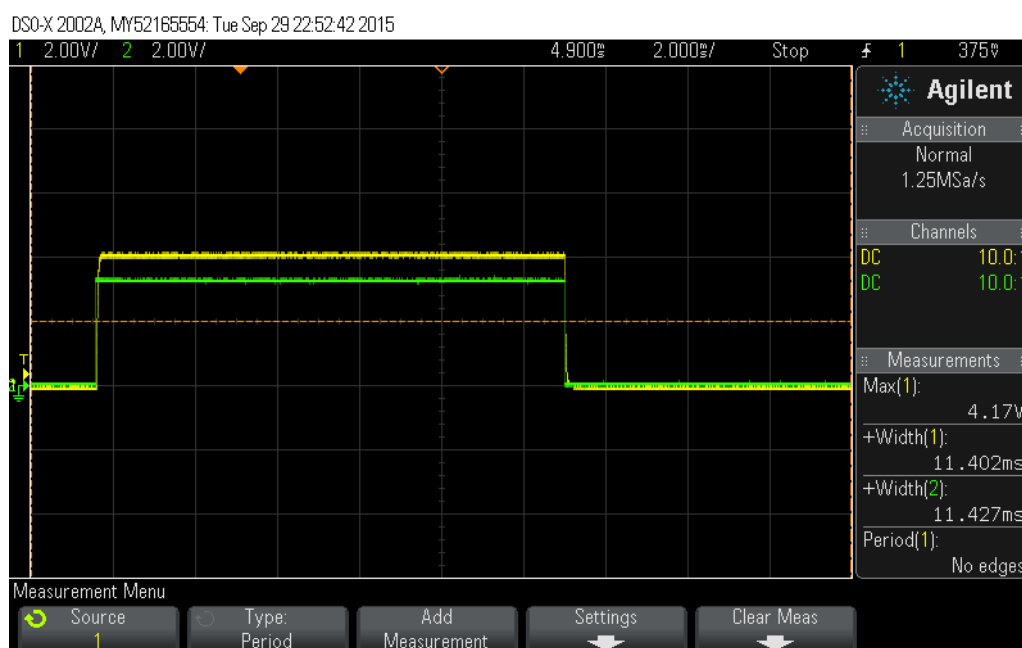


Figura 5.34: Ensaio do módulo de acionamentos de potência com $R_{\text{porta}} = 10\text{k}\Omega$

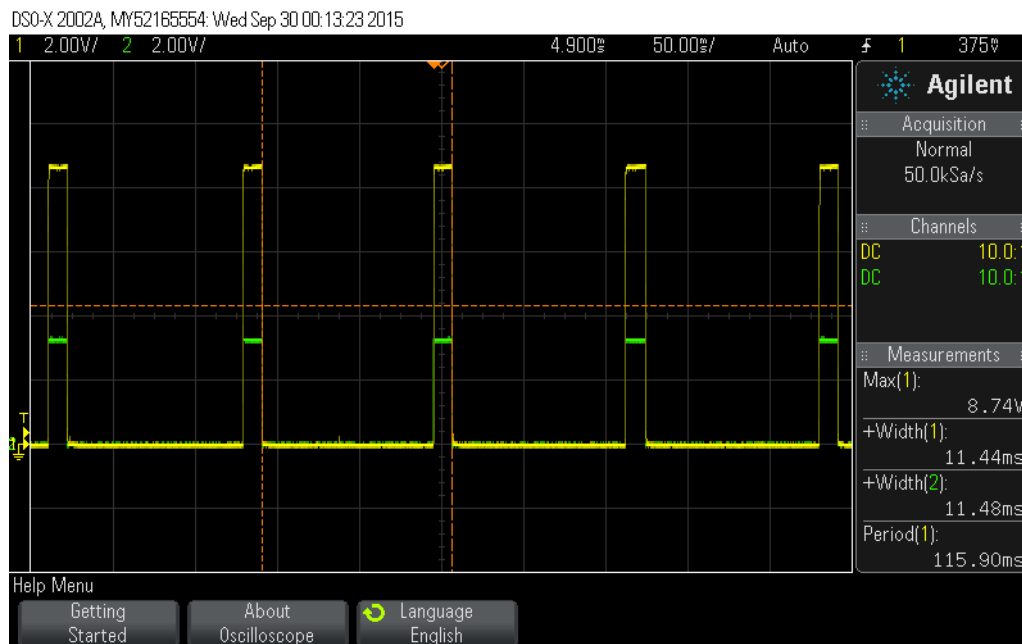


Figura 5.35: Ensaio do módulo de acionamentos de potência com $R_{porta} = 22k\Omega$

Em seguida, foram ligados como carga o SSR e posteriormente a válvula solenóide, cujos resultados estão descritos nas figuras 5.36 e 5.37. Foram registradas as formas de onda na porta (em verde) e no dreno (em amarelo) do MOSFET.

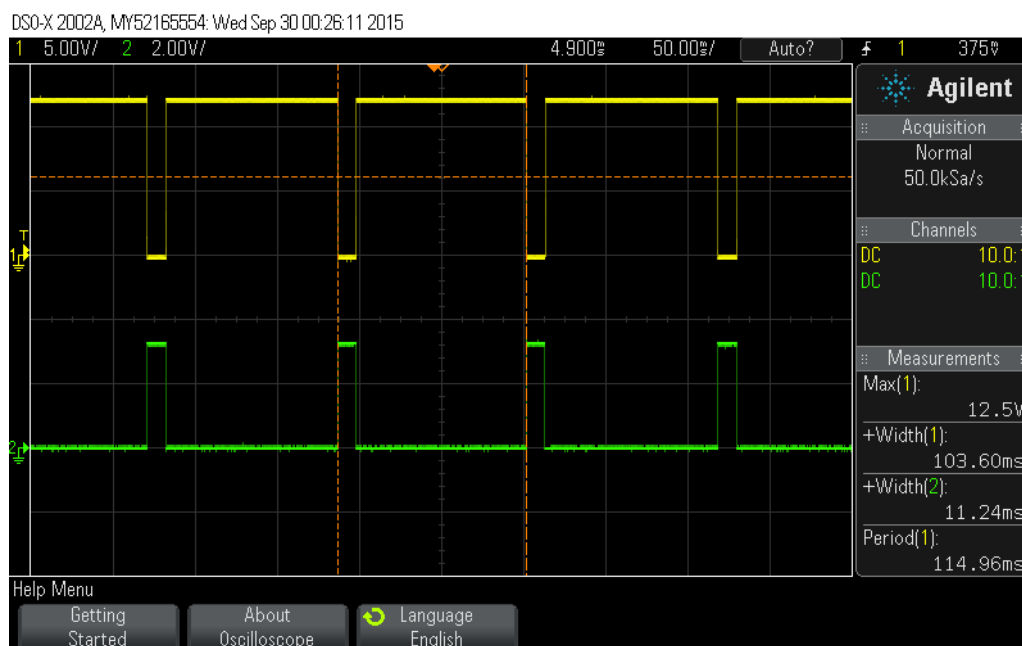


Figura 5.36: Ensaio do módulo de acionamentos de potência com SSR ligado à saída

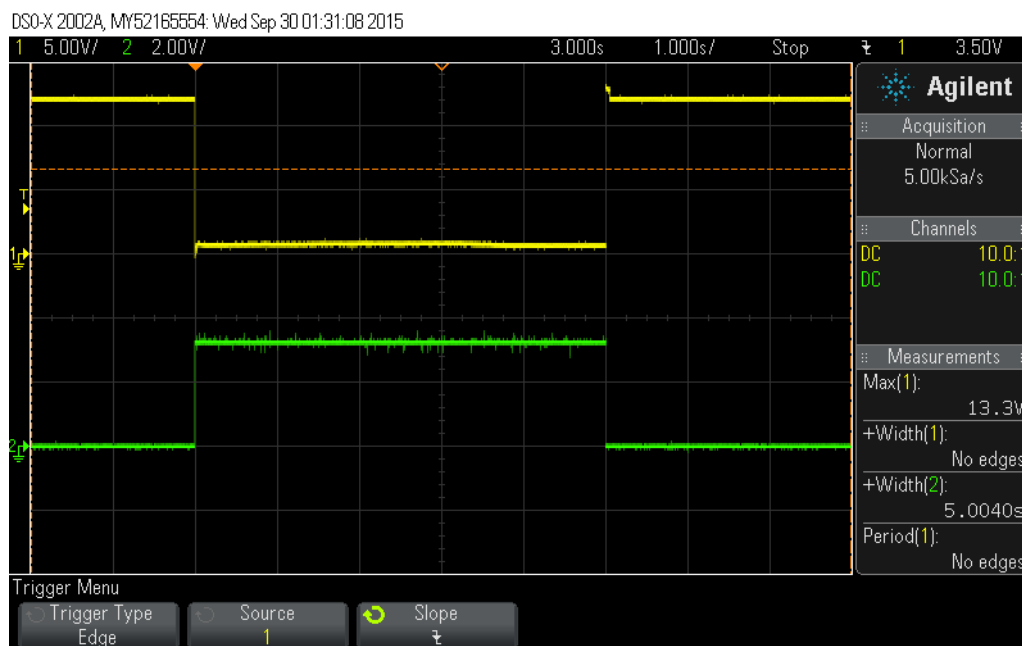


Figura 5.37: Ensaio do módulo de acionamentos de potência com válvula solenóide ligada à saída

Com relação ao solenóide, ao variar a tensão aplicada sobre os seus terminais foram determinados os valores de tensão mínima de acionamento e máxima antes do desligamento da válvula, apresentados na tabela 5.9.

Tabela 5.9: Tensões de operação da válvula solenóide

Acionamento (V)	Desligamento (V)
$\geq 7,0$	$\leq 1,5$

Outra medição realizada foi a resposta de tensão da válvula ao ser energizada, descrita nas figuras 5.40 e 5.39. Para realizar tais medições, foi colocado um resistor de $1,17\Omega$ em série com a válvula e medida a tensão sobre esta (em verde) e também foi monitorada a tensão da fonte de alimentação (em amarelo). O circuito montado em placa de protótipo está ilustrado na figura 5.38.

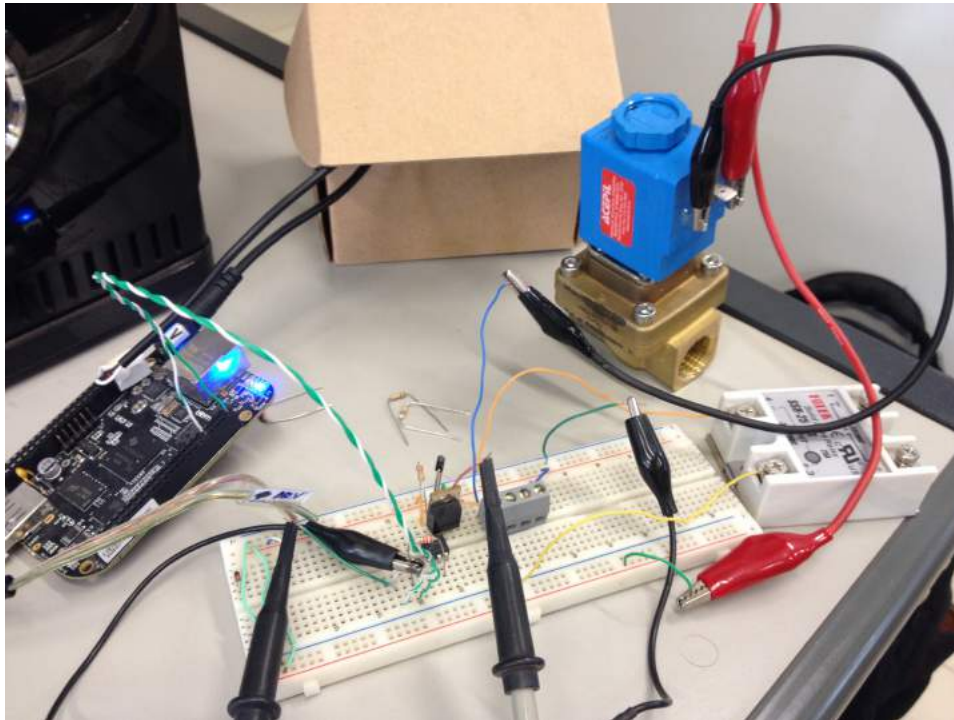


Figura 5.38: Implementação do módulo de acionamentos de potência em placa de protótipo

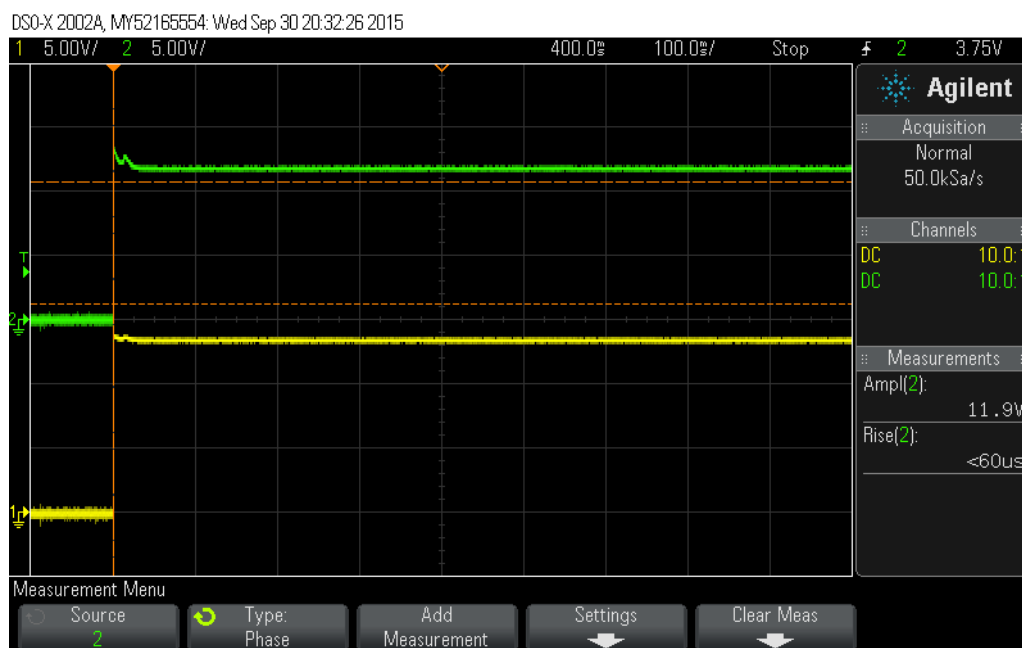


Figura 5.39: Resposta ao degrau do acionamento da válvula solenóide

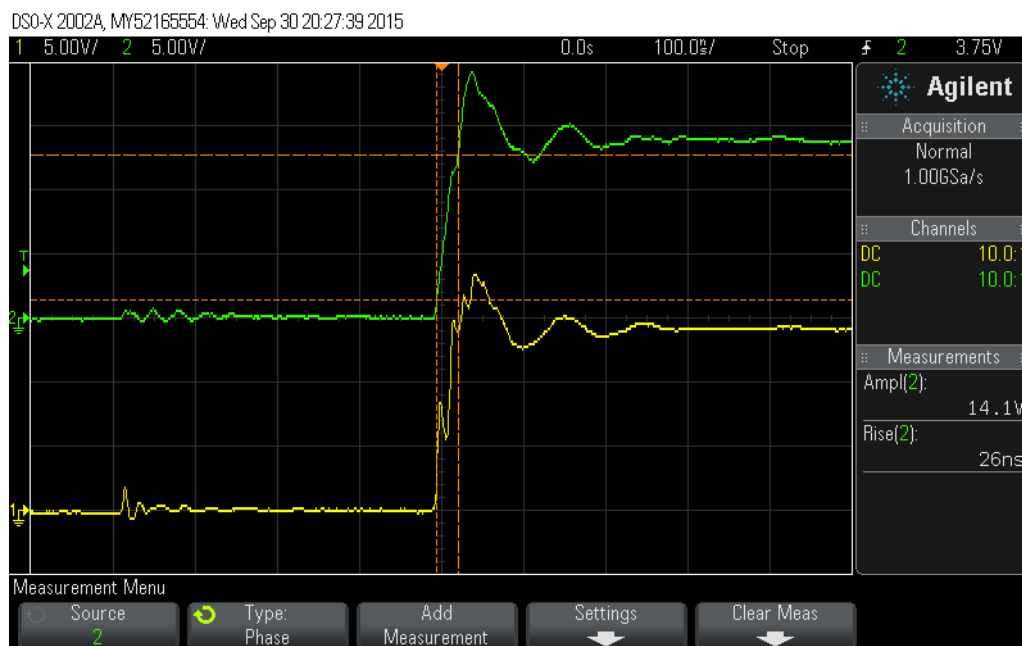


Figura 5.40: Resposta ao degrau do acionamento da válvula solenóide, em close

No que diz respeito ao funcionamento do SSR, foi adicionada uma carga puramente resistiva de $470\text{k}\Omega$ à sua saída e ligado um sinal gerado pelo PWM da BBB, com frequência de 60Hz e período em alta de 1ms, para comprovar seu funcionamento. Observando as forma de onda do PWM e da senóide da rede aplicada sobre o SSR em série com o resistor em função do tempo, foi constatado que a frequência da rede não é estável em 60Hz. Este comportamento está ilustrado na figura 5.41, na qual a forma de onda amarela representa o PWM e a verde é a senóide da rede; observa-se que em instantes de tempo distintos, a fase entre o PWM e a senóide é diferente.

Observa-se que no estudo da estabilidade da frequência da rede, em nenhum momento a saída do SSR esteve ligada, apesar dos pulsos gerados pelo PWM na entrada. Isto se deve ao fato de este modelo específico de SSR possuir um controle interno que permite somente o acionamento durante a passagem da senóide por zero, também conhecido como *zero-crossing*. Isto fez com que a variação da frequência da rede se tornasse um problema para o acionamento do SSR. A figura 5.42 ilustra o acionamento do SSR em diversos instantes da senóide da rede.

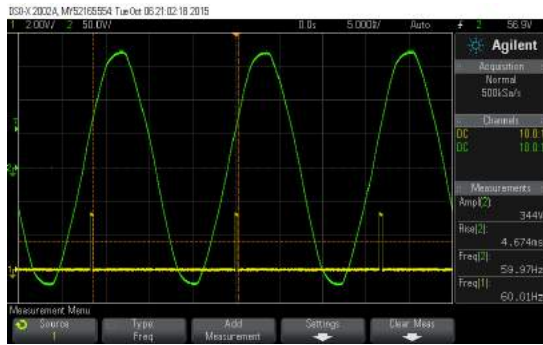
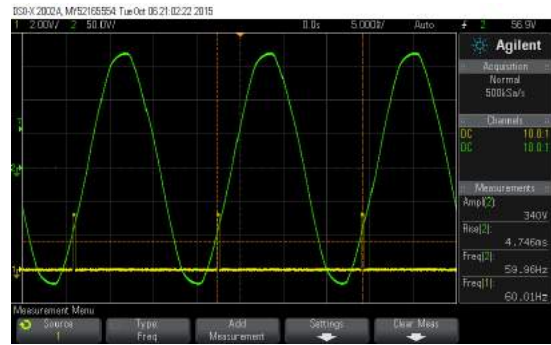
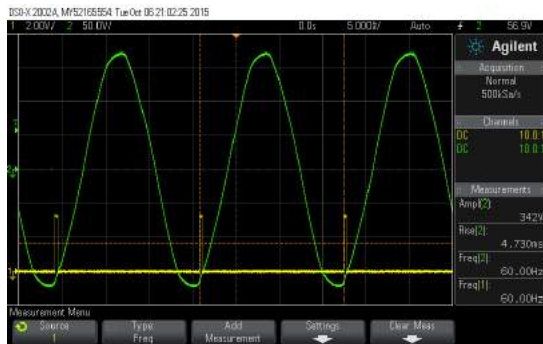
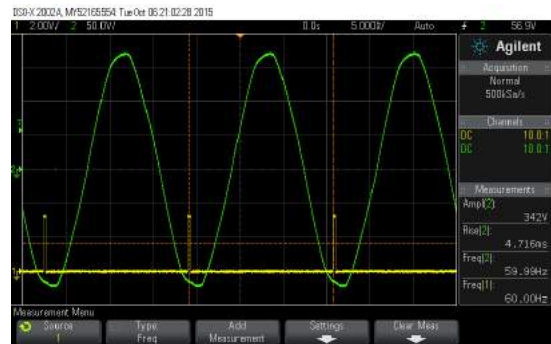
(a) $t = t_0$ (b) $t = t_1$ (c) $t = t_2$ (d) $t = t_3$

Figura 5.41: Instabilidade da frequência da rede em função do tempo

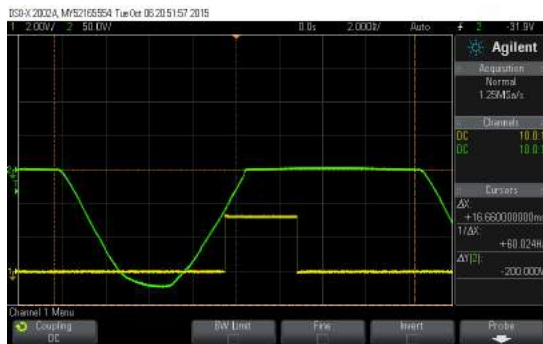
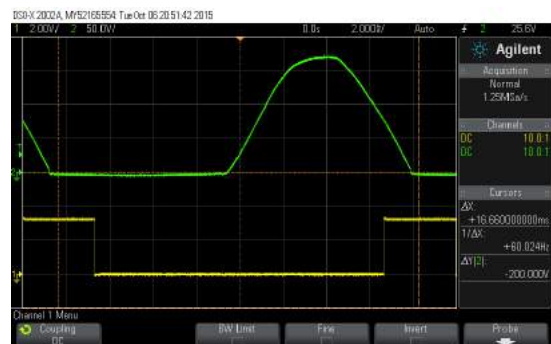
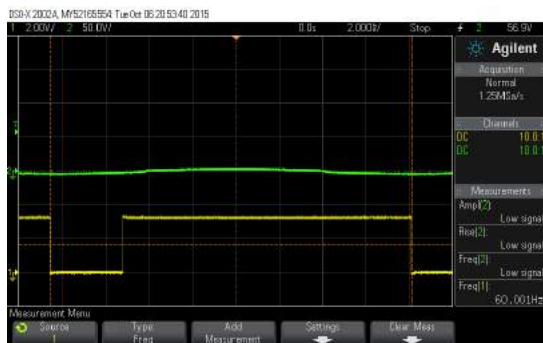
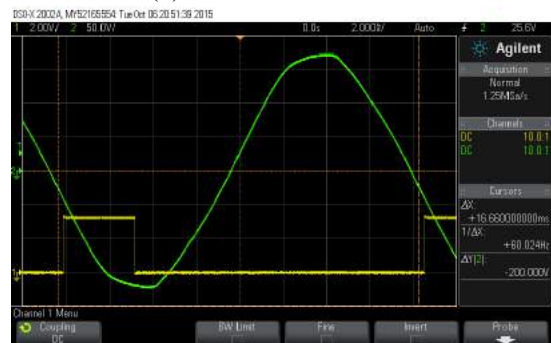
(a) Acionamento em $t=0$ (b) Acionamento em $t=\pi$ (c) Acionamento em $t=0$ e $t=\pi$ (d) Acionamento em $t \neq 0$ e $t \neq \pi$

Figura 5.42: Acionamento do SSR em diferentes instantes da senóide da rede

5.4.2 Servo-motor

A simulação do circuito de acionamento do servo-motor foi realizada com a saída em aberto ou seja, sem carga, tendo em vista que na prática foi ligado somente o terminal de controle do atuador, que deve apresentar alta impedância. O sinal de entrada empregado foi uma onda retangular com período de 20ms e *duty-cycle* de 10%. A figura 5.43 apresenta o resultado da simulação.

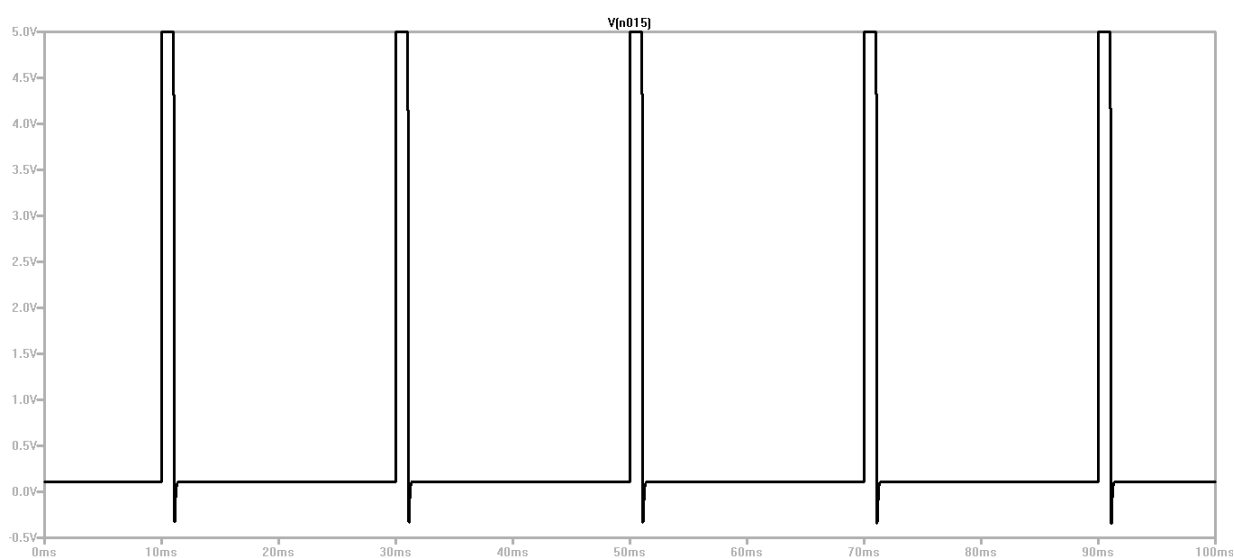
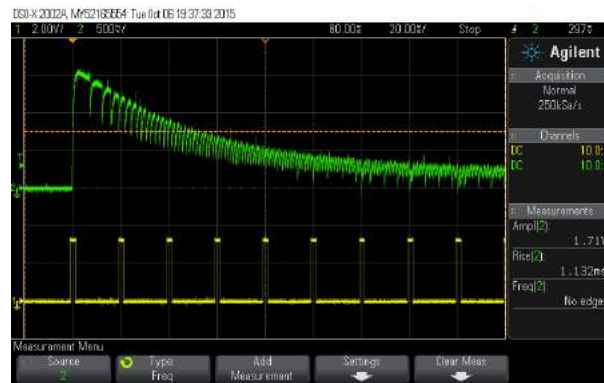
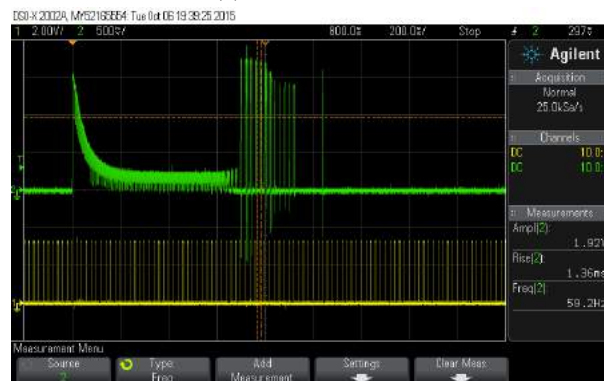


Figura 5.43: Resultado da simulação do módulo de acionamentos de potência com carga simulando o servo-motor

Quanto aos ensaios de bancada, foi ligado um resistor de $1,17\Omega$ em série com a alimentação do servo-motor e foi observado indiretamente o comportamento da corrente de alimentação deste por meio do uso de um osciloscópio, medindo a tensão sobre o resistor. Em todas as medições, foi registrado o comportamento do servo no período de transição em que o ângulo deste era variado de 90° . Na figura 5.44 é apresentado o resultado sem carga e na figura 5.45 é apresentado o comportamento com uma carga. O esquema para ensaios com carga está ilustrado na figura 5.46.



(a) Visão em close



(b) Visão ampla

Figura 5.44: Comportamento da corrente de alimentação do servo-motor durante operação sem carga

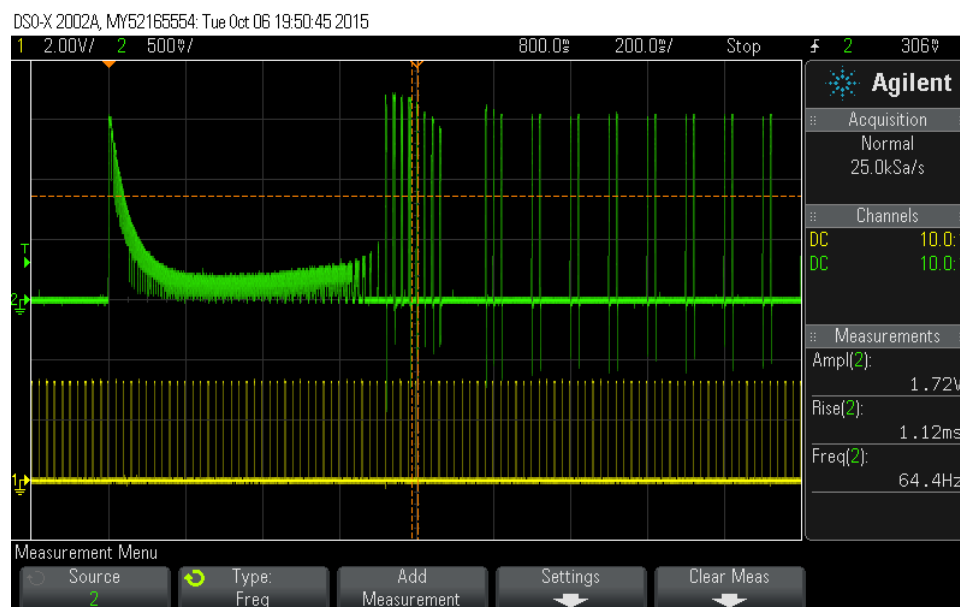


Figura 5.45: Comportamento da corrente de alimentação do servo-motor durante operação com carga

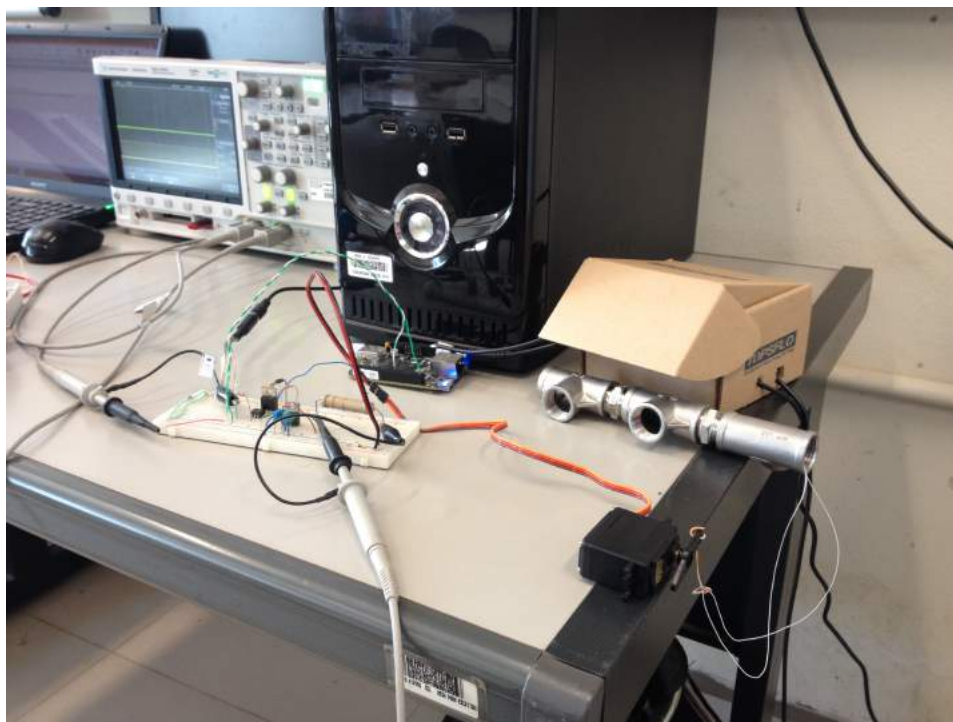


Figura 5.46: Teste do servo-motor com carga

Também foi testado o comportamento da alimentação do servo quando ele está fixo em uma posição e é aplicado um torque mecânico ao seu eixo, conforme o esquema ilustrado na figura 5.46. O resultado é apresentado na figura 5.47.

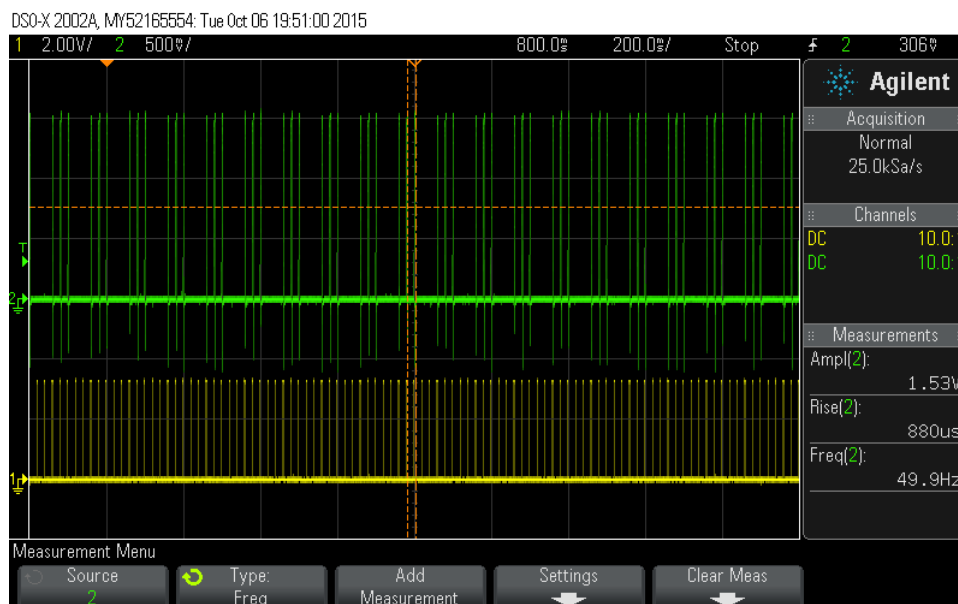


Figura 5.47: Comportamento da corrente de alimentação do servo-motor ao aplicar um torque ao seu eixo

Com base nos resultados encontrados, enfatiza-se o uso de uma fonte de tensão separada

para a fonte do servo-motor, no lugar da fonte da BBB. Isto se deve aos pulsos de corrente do servo, que podem exigir mais corrente do que a fonte da BBB pode fornecer e, consequentemente, reiniciar a BBB durante seu funcionamento.

5.4.3 Detector de zero-crossing

A primeira simulação do circuito da figura 4.21 foi realizada com o objetivo de comprovar o seu funcionamento para dois valores de R_2 : $10k\Omega$ e $22k\Omega$. O circuito simulado é descrito na figura 5.48 e o resultado está expresso na figura 5.49, na qual a forma de onda em preto representa a saída do circuito; em vermelho a forma de onda sobre o coletor do transistor e; em azul a forma de onda aplicada à base do transistor.

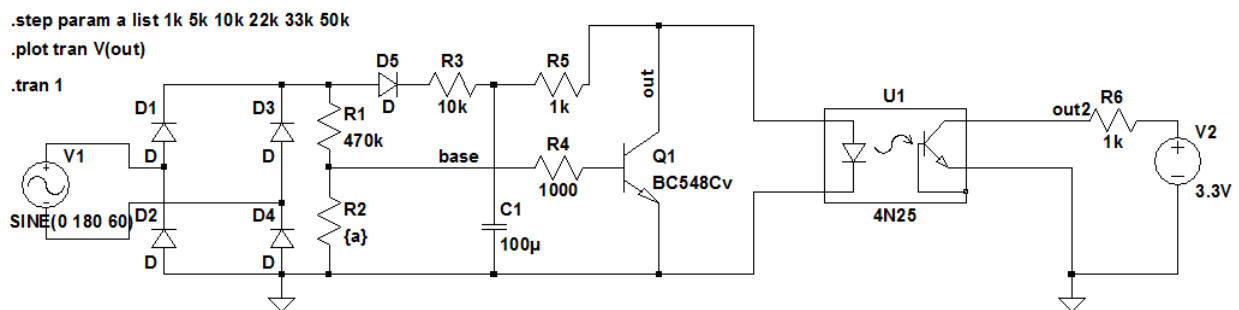


Figura 5.48: Descrição da simulação do circuito detector de zero-crossing

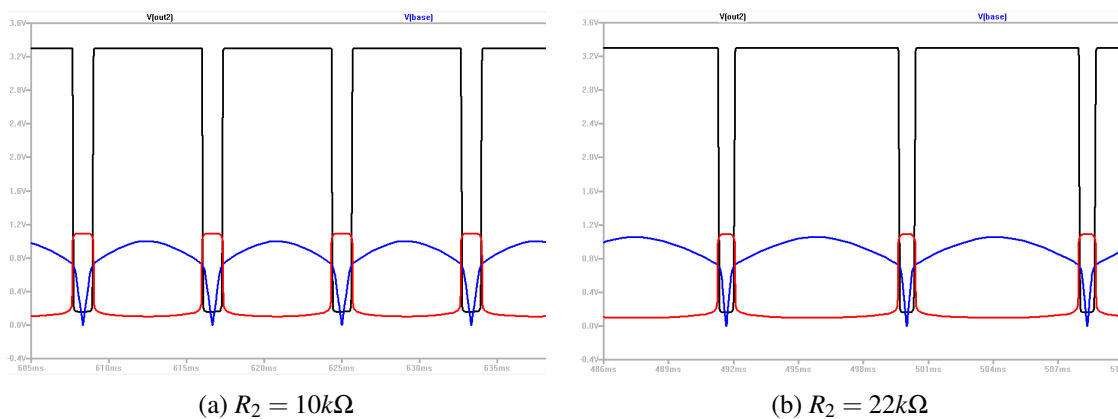


Figura 5.49: Comportamento do circuito detector de zero-crossing para diferentes valores de atenuação da senóide retificada

Para ambos os valores do resistor, foi comprovado o funcionamento do circuito, porém observou-se que à medida que seu valor aumenta, a largura do pulso de saída se estreita. Isto se deve ao fato de que, quanto mais atenuado o sinal retificado, maior a sua porção que fica abaixo de 0,7V e, por conseguinte, maior o tempo que o transistor permanece em corte. Este

comportamento observado foi comprovado com a simulação do circuito para diversos valores de R_2 , desde $1\text{k}\Omega$ até $50\text{k}\Omega$. O resultado da simulação está descrito nas figuras 5.50 (a) e (b), que mostra a forma de onda sobre o coletor e aplicada à base do transistor. A tabela 5.10 apresenta a largura do pulso que indica passagem por zero em função dos valores de R_2 simulados.

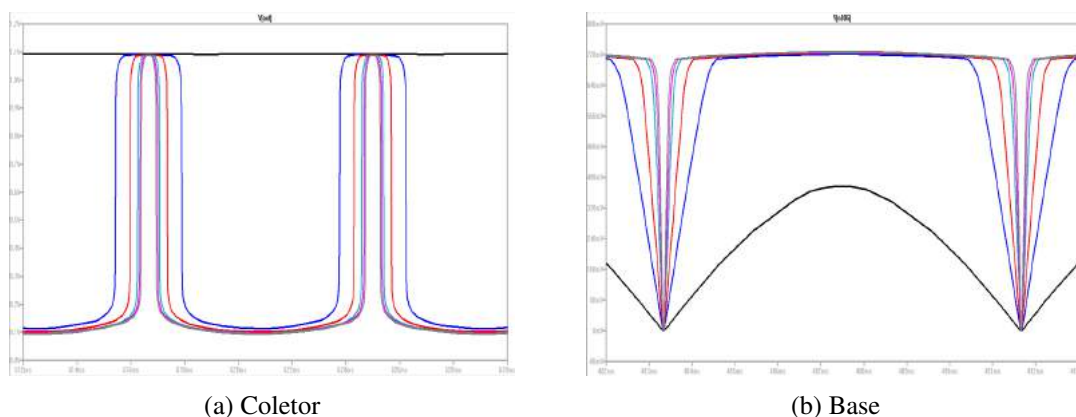


Figura 5.50: Verificação do comportamento do circuito detector de *zero-crossing* para diferentes valores de R_2

Tabela 5.10: Largura do pulso que indica passagem por zero em função do valor de R_2

R_2 ($\text{k}\Omega$)	largura do pulso (ms)
1	-
5	2,47
10	1,38
22	0,81
33	0,66
50	0,56

É possível observar que o circuito não funciona corretamente para $R_2=1\text{k}\Omega$ (forma de onda preta), uma vez que o potencial V_{be} não é suficiente para que o transistor passe a conduzir. Por outro lado, observa-se que quanto maior o valor de R_2 , menor é a largura do pulso que indica a passagem por zero. Também é interessante notar que a senóide da figura (b) é ceifada quando o transistor entra em condução, pois a junção base-emissor tem o comportamento de um diodo.

Foi decidido o uso de $R_2=10\text{k}\Omega$ por ser um valor que traz uma largura de pulso suficiente para a aplicação proposta.

Quanto à saída do optoacoplador, foi notado que o circuito simulado não fornece corrente suficiente para que a incidência luminosa sobre a base do fototransistor seja suficientemente

intensa para permitir a condução plena deste dispositivo, conforme indicado na figura 5.51. Isto não é um problema para a fonte de alimentação de 3,3V que será utilizada no projeto, conforme indicado em (a), já que a condução do circuito foi suficiente para fornecer um nível lógico baixo na saída, porém observou-se o comportamento mais acentuado quando a fonte foi substituída por uma de 12V, como ilustrado em (b).

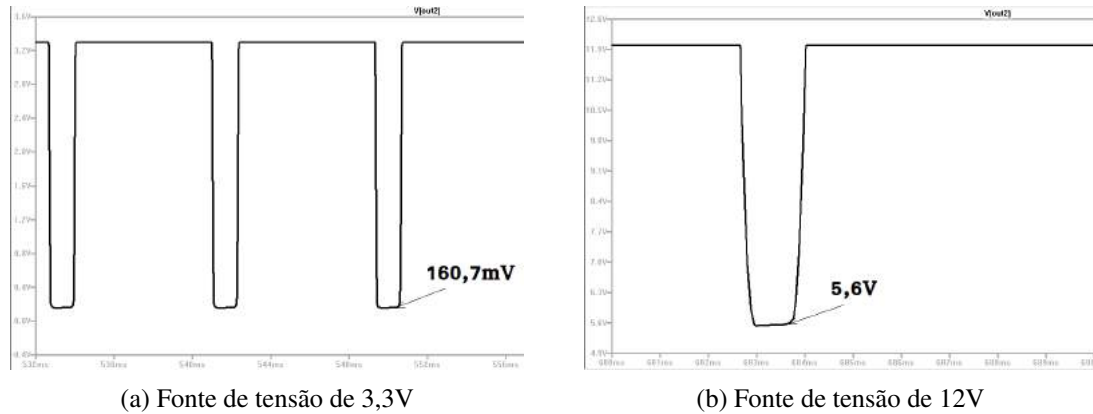


Figura 5.51: Simulação da saída do circuito detector de *zero-crossing* para diferentes valores de tensão de alimentação

Outra preocupação com relação ao circuito foi o fato de que a queda de tensão sobre o resistor R3 é elevada e, portanto, a potência que este deve suportar não é desprezível. Seu valor é de 1,32W e foi obtido a partir da simulação realizada no LTSpice. A figura 5.52 apresenta os valores de tensão (em preto), corrente (em azul) e potência (em vermelho) sobre o resistor, em função do tempo.

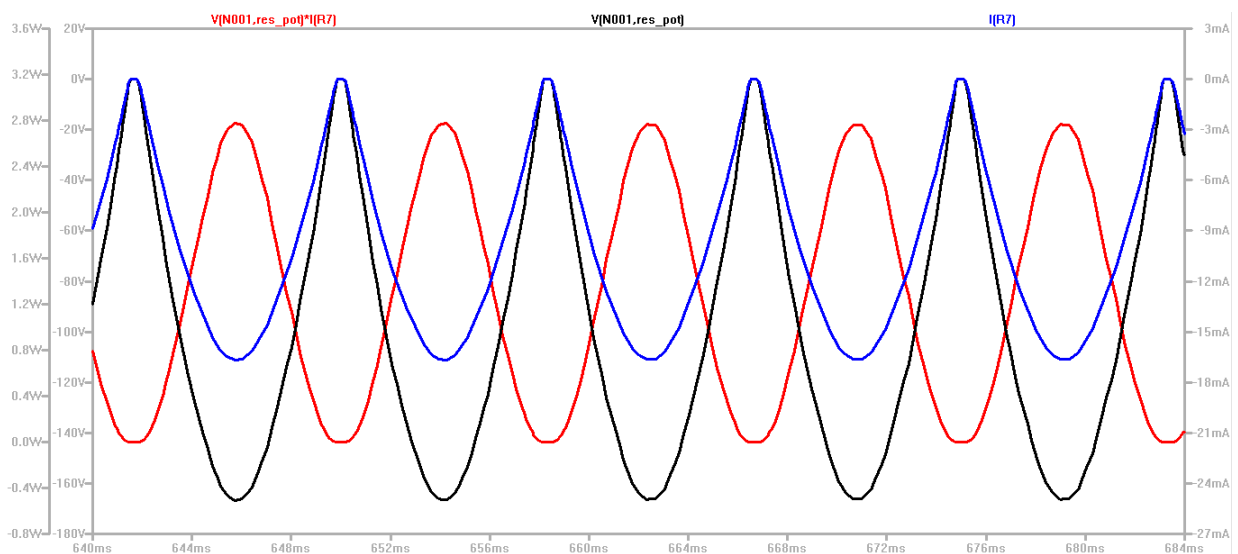


Figura 5.52: Curvas de tensão, corrente e potência sobre o resistor de potência do circuito detector de *zero-crossing*

Na implementação em bancada, foi utilizado o valor de $R_2=10k\Omega$, conforme documentado no capítulo 4. A figura 5.53 apresenta as formas de onda sobre o coletor e a base do transistor, em amarelo e verde, respectivamente. Observou-se comportamento idêntico ao da simulação e tempo de condução de 1,54ms, correspondente a um erro de 11,6% com relação ao valor da simulação.

Também foram verificadas as formas de onda na entrada e saída do optoacoplador, em amarelo e verde respectivamente, para valores de alimentação de 5V e 12V. Os resultados são apresentados na figura 5.54. Observou-se que o comportamento da simulação, exposto na figura 5.51, foi comprovada. Também observou-se a redução da largura do pulso de saída e mudança de amplitude da forma de onda no coletor do transistor de detecção de passagem por zero, se comparado à figura 5.53.

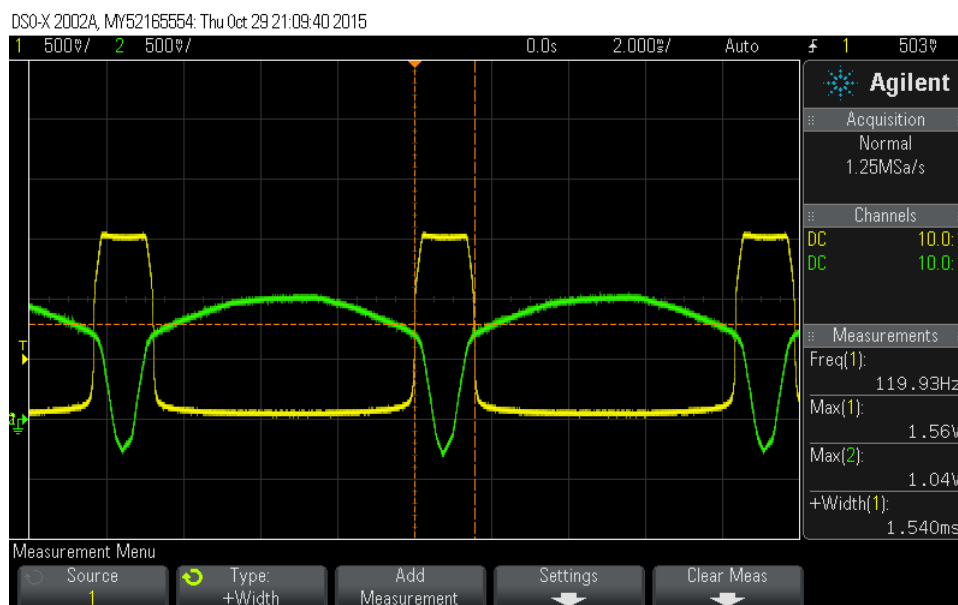


Figura 5.53: Formas de onda de tensão sobre a base (verde) e o coletor (amarelo) do transistor de detecção de passagem por zero

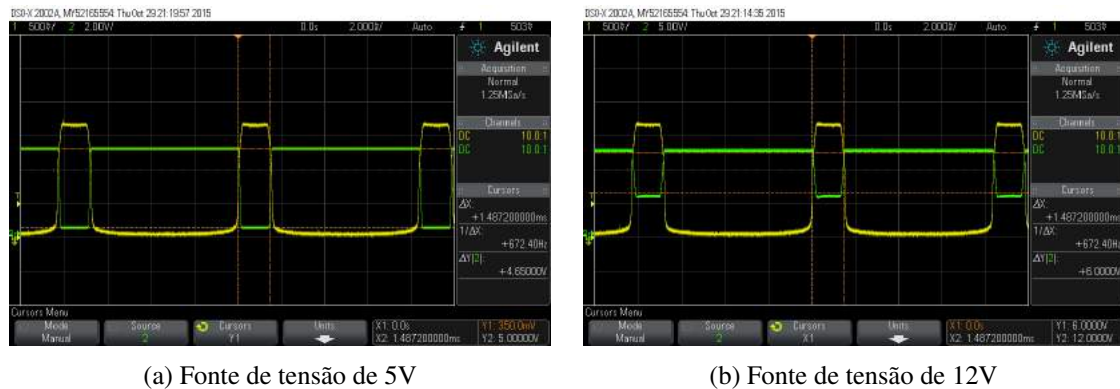


Figura 5.54: Entrada e saída do circuito detector de *zero-crossing* para diferentes valores de tensão de alimentação

Por fim, foram capturadas as formas de onda da tensão sobre o resistor de potência e da saída do optoacoplador, que estão documentadas na figura 5.55. Com isto, foi observado que a tensão sobre este resistor é a tensão da rede retificada e também que o circuito funciona conforme o esperado, por meio da constatação de que as duas formas de onda registradas estavam sincronizadas.

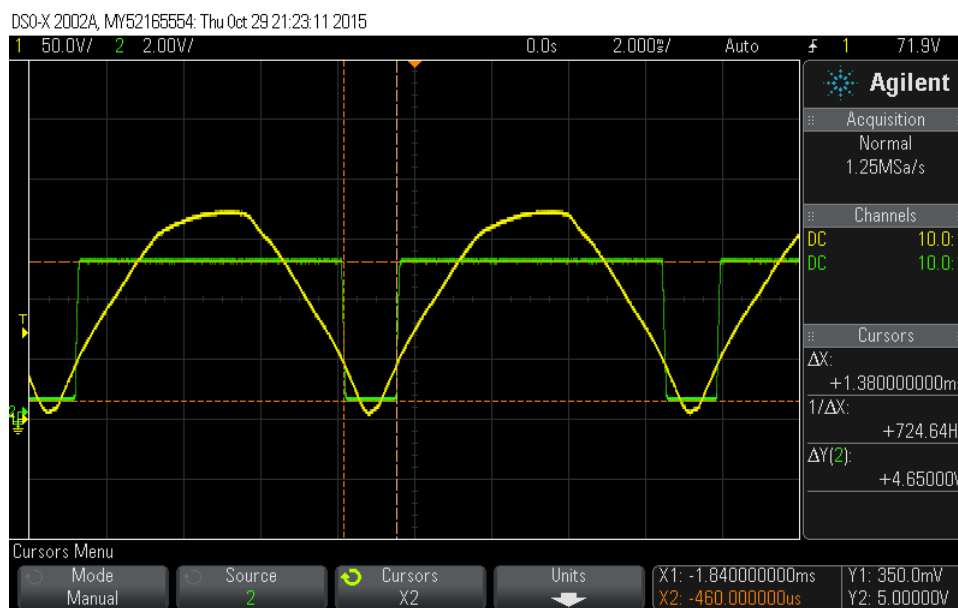


Figura 5.55: Forma de onda de tensão sobre o resistor de potência e a saída do optoacoplador do circuito detector de *zero-crossing*

Com relação aos circuitos projetados, o que se observou foi que nos testes de bancada estes corresponderam ao comportamento obtido por meio de simulações e, portanto, suas funções foram atingidas com êxito. Em função do tempo impeditivo, não foi possível realizar uma implementação em PCB, porém este é um passo essencial no longo prazo, já que não é possível produzir um produto usando uma matriz de contatos.

Foi interessante constatar que a frequência da rede é variável em torno 60Hz e também que o SSR só pode ser disparado quando a tensão passa por zero: ambos os fatores combinados foram os responsáveis pelo projeto do detector de passagem por zero, que permitiria o controle do número de semiciclos de senóide nos quais os resistores de potência estariam ativos e isto, por sua vez, poderia permitir um controle mais preciso da potência destes do que uma janela de tempo por si só. Mais uma vez, estas hipóteses só podem ser confirmadas com uma nova implementação e testes de comparação de janela de tempo versus controle do número de ciclos. O comportamento da alimentação do servo-motor também foi interessante do ponto de vista de conhecimento, mas não impactou diretamente no projeto proposto — a propósito, seu funcionamento foi o único resultado relacionado à estrutura de adição de lúpulos, que não foi construída.

Quanto ao detector de *zero-crossing*, a única observação prática é o fato de que a saída do circuito começa a indicar passagem por zero meio período de duração do pulso antes da passagem efetiva por zero, portanto em usos posteriores este comportamento deve ser levado em conta.

5.5 Estrutura mecânica

Esta seção documenta a montagem da parte mecânica — que compreende o sistema funcional e a estrutura de suporte — além do cabeamento entre o sistema embarcado e os elementos de potência, e os testes de integração entre GUI, sistema embarcado, algoritmo de automação e sistema mecânico realizados.

5.5.1 Montagem da mecânica e interface com o sistema embarcado

A montagem do equipamento foi realizada a partir do conteúdo descrito na seção 4.1, com a ajuda de um mecânico qualificado. A figura 5.56 apresenta os tubos e conexões durante o seu processo de rosqueamento e vedação e a figura 5.57 apresenta as painéis conectadas à tubulação e encaixadas na estrutura de suporte.



Figura 5.56: Processo de rosqueamento e vedação da tubulação e elementos relacionados



Figura 5.57: Painéis e tubos conectados repousando sobre a estrutura de suporte

A tubulação foi montada separada das painéis pois foi constatado durante o processo que, quando foi tentado montá-la a partir das painéis, se tornou impossível rosquear um dos tubos: ao girá-lo no sentido horário, uma das extremidades era afrouxada e, ao girá-lo no sentido anti-horário, o mesmo acontecia com a outra extremidade. Como a conexão entre as painéis e a tubulação foi feita empregando contra-porcass, a abordagem aqui descrita foi possível. A figura 5.58 apresenta em detalhe o resultado da conexão entre as painéis e encanamento. Note-se que para o BK, a conexão foi feita na lateral da painél, de forma similar ao descrito na figura 2.8 (b), enquanto a MT foi conectada por um furo no fundo da painél para aproveitamento do maior volume de mosto possível.



(a) Tina de mostura



(b) Caldeirão de fervura

Figura 5.58: Detalhe de conexão das painéis com a tubulação

O próximo passo foi acomodar a BBB e a *protoboard* na estrutura de suporte, ao lado da MT. Também foram feitos furos na estrutura para a fixação dos dois relés de estado sólido. A figura 5.59 apresenta a acomodação e cabeamento dos circuitos, e fixação de um dos relés à estrutura de suporte. Como pode ser observado em (b), o SSR fixado à estrutura, ao qual é ligada a tensão da rede elétrica, ficou desprotegido no que diz respeito a contato incidental com o operador da máquina, o que é um risco de segurança do equipamento que poderia ter sido facilmente solucionado, mas que não foi notado durante o processo de cabeamento.

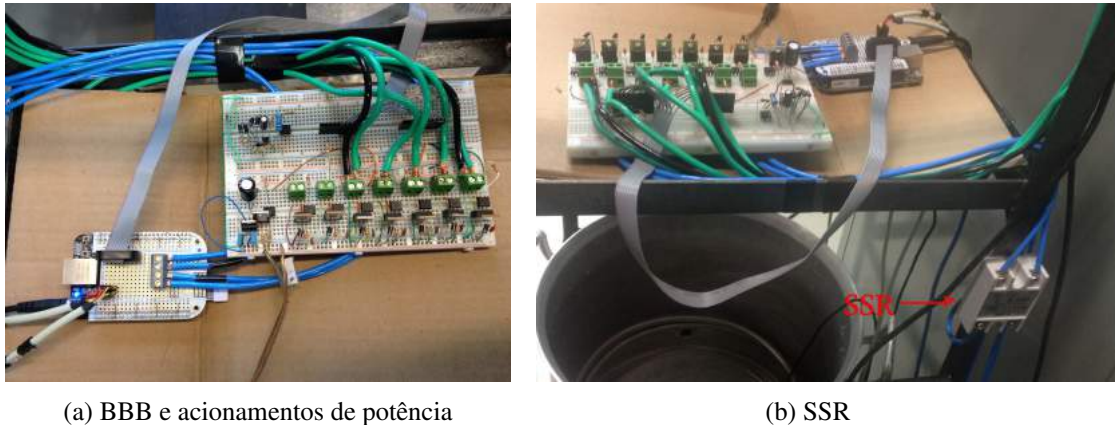


Figura 5.59: Acomodação do sistema embarcado e acionamentos de potência à estrutura de suporte

Por fim, pode ser observado nas figuras 5.60 e 5.61, respectivamente, os detalhes e a visão geral da estrutura cabeada e finalizada. Na figura que apresenta os detalhes, é possível perceber que as conexões elétricas foram instaladas por meio de torção dos cabos aos terminais das bombas e válvulas. Isto pode ser adequado para testes, mas é inaceitável do ponto de vista de operação não supervisionada do produto por questões de segurança. Nota-se também, na figura que expõe a visão geral, que o cabeamento em si fica preso à estrutura mas ainda assim exposto a possíveis puxões inadvertidos e derramamento de líquido quente.

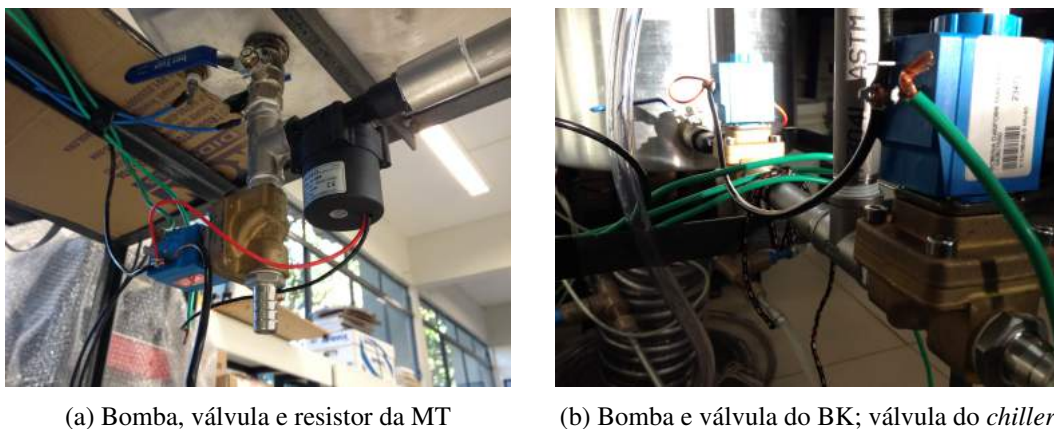


Figura 5.60: Detalhes de ligações elétricas dos elementos de potência



Figura 5.61: Estrutura mecânica finalizada e pronta para testes

5.5.2 Integração entre GUI, sistema embarcado, algortimo de automação e sistema mecânico

Para testar os elementos do sistema mecânico, foi adicionado um volume de 20 litros de água na MT e 10 litros no BK. Após isto os elementos do sistema foram testados manualmente, ou seja, controlados por meio da GUI, em diferentes combinações:

1. Bomba do mosto
2. Válvula do mosto
3. Bomba e válvula da fervura
4. Bombas do mosto e de fervura e válvula da fervura
5. Válvulas da fervura e do mosto e bomba da fervura
6. Válvulas da fervura e do chiller
7. Válvulas da fervura e do chiller e bomba da fervura

Para todos os casos foi constatado por meio de comprovação visual que o sistema funcionou conforme o esperado. As figuras 5.62 (a), (b), (c) e (d) apresentam os resultados para as situações 1, 2, 4 e 5 respectivamente. Foi observado a partir destes resultados que a vazão de água para dentro da MT é menos intensa na situação 3 e mais intensa na situação 4, porém esta última merece uma observação: à medida que o volume de água na MT se aproximou do volume de água no BK, este laço de recirculação se estabilizou e o somatório dos fluxos de entrada e saída da tubulação foi nulo no longo prazo.



(a) Bomba do mosto



(b) Válvula do mosto



(c) Válvulas da fervura e do mosto e bomba da fervura



(d) Bombas do mosto e de fervura e válvula da fervura

Figura 5.62: Testes do sistema mecânico com controle pela GUI

Outro teste de equipamento realizado foi a simulação descrita na seção 4.6.5. O sistema mecânico se comportou de acordo com a simulação, porém foi observado que as temporizações atribuídas no código, durante a lavagem dos grãos, não são adequadas para uma receita real, o que levou à cronometragem dos tempos necessários para drenar a MT e para bombear o conteúdo para ela novamente. Estes resultados são apresentados na tabela 5.11.

Para este teste foi gravado um vídeo no qual não só o equipamento foi monitorado por uma câmera, mas também foi gravado o comportamento da GUI e as mensagens impressas pela aplicação no terminal. O vídeo está disponível em <https://www.youtube.com/watch?v=>

Liz8gi9YoFU — a figura 5.63 documenta alguns momentos deste vídeo.

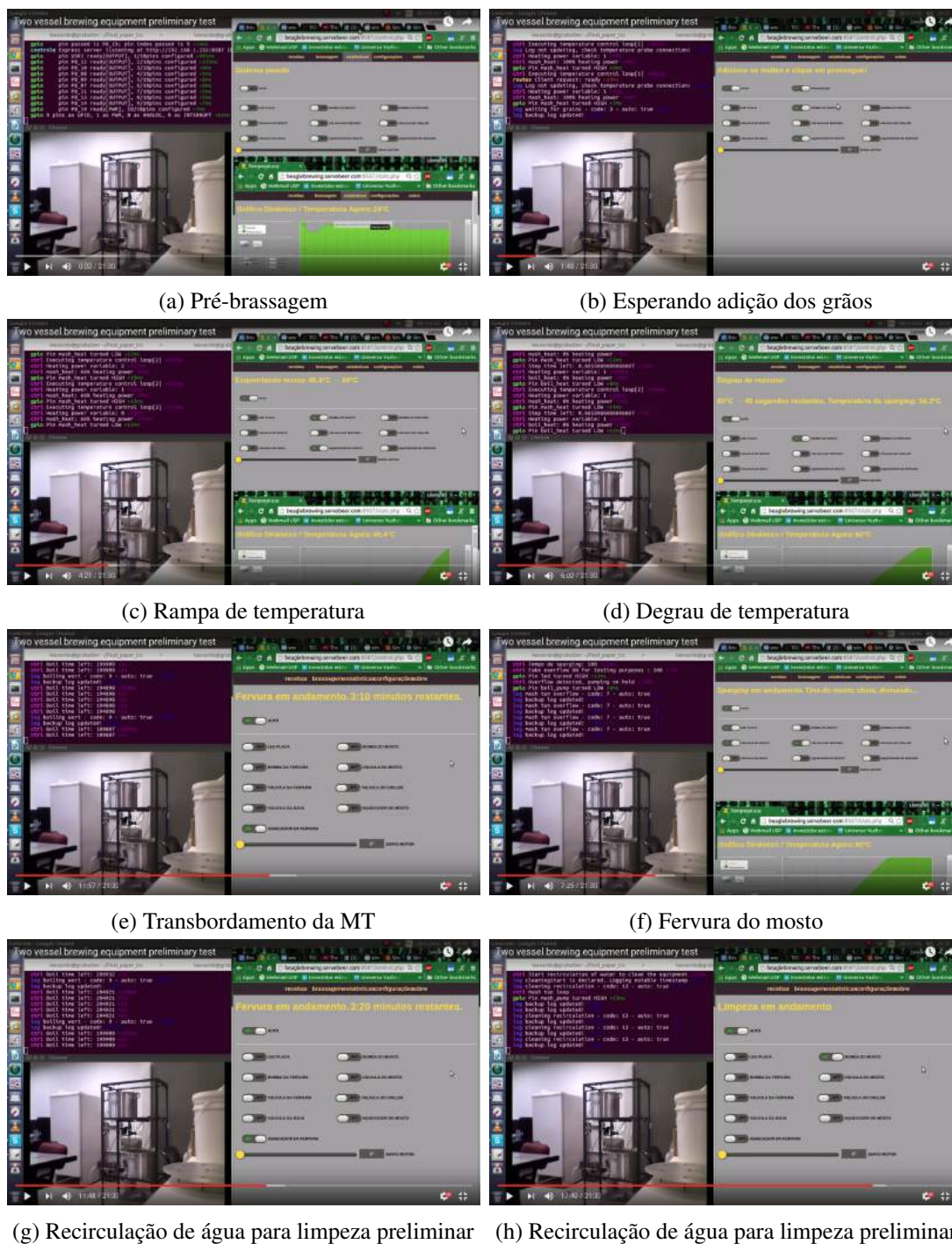


Figura 5.63: Simulação de produção para teste da parte mecânica

Tabela 5.11: Tempos de de drenagem e bombeamento do sistema mecânico para 20 litros de água

Descrição	Tempo (s)
Drenagem da MT para o BK	145
Bombeamento do BK para a MT	190

A implementação da estrutura mecânica foi o ponto ao qual mais foram despendidos recursos, sejam eles financeiros ou de tempo. Não somente a complexidade deste projeto, mas também o pouco conhecimento de mecânica do projetista foram fatores que influenciaram neste alto custo. Por falta de experiência o equipamento projetado se mostrou desajeitado para manutenção e limpeza, portanto cabe ao usuário do sistema ser disciplinado ao utilizá-lo e realizar a limpeza o quanto antes possível, mesmo que seja uma tarefa complexa. Cabe notar que, na idealização do projeto, este seria equipado com um sistema CIP completo, o que não foi possível devido a custo de implementação e complexidade elevada que seria adicionada à proposta do projeto.

Capítulo 6

Conclusões

Este capítulo é dedicado às conclusões levantadas a partir dos resultados obtidos para este trabalho de conclusão de curso. As primeiras considerações observadas da realização deste projeto são que o funcionamento da simulação comprovou a capacidade do sistema de produzir uma receita de cerveja, porém a produção de uma receita em si requer que mais testes sejam realizados, para fazer ajustes finos do equipamento, das temporizações, adicionar os detectores de transbordamento das panelas, dentre outros tópicos que surgirem ao longo dos futuros testes. Após os testes preliminares descritos no capítulo de resultados, também é preciso modificar alguns aspectos da aplicação do servidor, de modo a impedir o usuário de ligar as válvulas e bombas em configurações que possam danificar ou sobrecarregar o sistema.

O uso do sistema embarcado BeagleBone Black possibilitou o amplo aprendizado acerca do Linux embarcado, desde conceitos relacionados à implementação deste no sistema operacional, como por exemplo a manipulação e uso da *device tree* até a observação de que há um amplo ecossistema de soluções disponíveis no mercado, e não somente a plataforma escolhida para este projeto. Foram absorvidos os conceitos de que é possível programar um sistema destes de maneira similar a um computador tradicional, porém sempre tomando cuidados relacionados a performance e consumo, além de outras características como robustez. O fato de um sistema destes possibilitar o uso de linguagens de programação de alto nível representou uma economia de tempo desenvolvendo software que possibilitou o foco em outras áreas do projeto.

Foi interessante observar que, no âmbito de desenvolvimento de software, mesmo com todo o planejamento de uma solução antes de sua implementação, o aprendizado de novas técnicas de programação e de depuração da aplicação foram constantes, o que significou que, a cada módulo implementado, era possível reescrevê-lo em uma estrutura completamente

diferente e cada vez mais elegante, não fosse o tempo escasso que impossibilitou esta abordagem. Ainda assim, os códigos escritos atenderam aos objetivos propostos. De maneira análoga, a interface de usuário passou por diversas modificações até a versão apresentada, decorrentes de um processo de aprendizado e percepção, no qual sempre havia uma melhoria a ser implementada com foco na experiência de usuário.

6.1 Considerações gerais

Por fim, conclui-se que os pré-requisitos para o funcionamento do sistema automatizado de produção de cerveja com controle e monitoramento remoto foram implementados com sucesso. Ainda assim, os resultados das implementações mostraram diversos pontos de melhoria até que um produto seja viável. Além disto, notou-se que a integração entre os diversos aspectos do projeto — parte mecânica, GUI, acionamentos de potência e algoritmo de automação — foi descomplicada e não causou transtornos. Atribui-se a estas observações o detalhamento com que cada ponto foi desenvolvido separadamente.

6.2 Trabalhos futuros

Como trabalhos futuros, ficam todas as correções expostas no capítulo de resultados e discussões. Além disto, é interessante implementar um sistema de controle de temperatura por meio de um controlador PID ou derivado deste, possivelmente mais adequado do que o método empregado neste trabalho — para tal, pode-se fazer o uso do subsistema PRU presente na BBB e cujo apêndice A apresenta um estudo teórico.

Outros aspectos do trabalho que devem ser melhorados em futuras ocasiões são o retrabalho do projeto do sistema mecânico com foco em ergonomia, simplificação da estrutura e certificação para que a produção de cerveja possa ser comercializada, além do emprego de métodos de projeto mais elaborados; um projeto mais detalhado do cabeamento que faz a interface entre o circuito de acionamentos de potência e os elementos de potência e; um bastidor para alojar o sistema embarcado de maneira protegida contra derramamento de líquidos. Também entra para a lista de trabalhos futuros a construção e teste da estrutura de adição de lúpulos e o projeto de uma estrutura de adição de grãos.

Referências Bibliográficas

- [1] J.J. PALMER. *How to Brew*. Brewers Publications. Terceira edição, 2006.
- [2] BREWERS ASSOCIATION. *Number of Breweries*. Disponível em: <https://www.brewersassociation.org/statistics/number-of-breweries/>, Acesso em: 12 de novembro de 2015.
- [3] O. CERVIERI JÚNIOR et al. *O Setor de Bebidas no Brasil*. BNDES Setorial, Rio de Janeiro, n. 40, p. [93]-129, set. 2015.
- [4] REVISTA DA CERVEJA. n.1, mai. 2012.
- [5] REVISTA DA CERVEJA. n.2, jul. 2012.
- [6] ASSOCIAÇÃO BRASILEIRA DA INDÚSTRIA DA CERVEJA. *Anuário 2014*. Disponível em: <http://www.cervbrasil.org.br/arquivos/anuariofinal2014.pdf>, Acesso em: 13 de novembro de 2015.
- [7] ASSOCIAÇÃO BRASILEIRA DE BEBIDAS. *Categorias*. Disponível em: <http://www.abrabe.org.br/categorias/>, Acesso em: 18 de novembro de 2015.
- [8] PICOBREW INC. *Pico Brew*. Disponível em: <https://www.picobrew.com/>, Acesso em: 18 de novembro de 2015.
- [9] WILLIAMSWARN NZ LTD. *The WilliamsWarn Personal Brewery*. Disponível em: <http://www.williamswarn.com/The-WilliamsWarn#.VkyEqnarTIU>, Acesso em: 18 de novembro de 2015.
- [10] D.E. BRIGGS et al. *Brewing - Science and Practice*. Woodhead Publishing Limited. Terceira edição, 2011.
- [11] A. TIERNEY-JONES. *1001 Cervejas para Beber Antes de Morrer*. Sextante, 2011.

- [12] DOGFISH HEAD. *120 Minute IPA*. Disponível em: <http://www.dogfish.com/ brews-spirits/the-brews/occassional-rarities/120-minute-ipa.htm>, Acesso em: 23 de novembro de 2015.
- [13] A. FODOR. *How to Remove Trub*. Brew Your Own, v. 3, n. 12, dez. 1997. Disponível em: <http://byo.com/stories/issue/item/890-how-to-remove-trub>, Acesso em: 14 de dezembro de 2015.
- [14] Devicetree.org. *Device Tree Usage*. Disponível em: http://www.devicetree.org/ Device_Tree_Usage, Acesso em: 21 de dezembro de 2015.
- [15] J. COOPER. *Introduction to the BeagleBone Black Device Tree*. Disponível em: <https://learn.adafruit.com/introduction-to-the-beaglebone-black-device-tree?view=all>, Acesso em: 21 de dezembro de 2015.
- [16] G. COLEY. *BeagleBone Black System Reference Manual*. Revision C1, mai. 2014. Disponível em: https://github.com/CircuitCo/BeagleBone-Black/blob/master/BBB_SRM.pdf?raw=true, Acesso em: 02 de dezembro de 2015.
- [17] TEXAS INSTRUMENTS. *AM335x Sitara™ Processors Technical Reference Manual*. Revision fev. 2015. Disponível em: <http://www.ti.com/lit/ug/spruh731/spruh731.pdf>, Acesso em: 02 de dezembro de 2015.
- [18] D. MOLLOY. *Beaglebone: GPIO Programming on ARM Embedded Linux*. mai. 2012. Disponível em: https://www.youtube.com/watch?v=wui_wU1AeQc, Acesso em: 28 de dezembro de 2015.
- [19] S. TILKOV, S.; VINOSKI. *Node.js: Using JavaScript to Build High-Performance Network Programs*. 2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), Lincoln/NE, p. 86-89, nov. 2015. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7426643>, Acesso em: 19 de abril de 2016.
- [20] ZÁKOVÁ K. BOSÁK, T. *Node.js Based Remote Control of Thermo-optical Plant*. 2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV), Bangkok - Tailândia, p. 209-213, fev. 2015. Disponível em: <http://>

[//ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7087293](http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7087293), Acesso em: 19 de abril de 2016.

- [21] T. OGASAWARA. *Workload Characterization of Server-Side JavaScript*. 2014 IEEE International Symposium on Workload Characterization (IISWC), Raleigh/NC, p. 13-21, out. 2014. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6983035>, Acesso em: 19 de abril de 2016.
- [22] S. TILKOV, S.; VINOSKI. *Node.js: Using JavaScript to Build High-Performance Network Programs*. IEEE Internet Computing, p. 80-83, nov. 2010. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5617064>, Acesso em: 19 de abril de 2016.
- [23] P. TEIXEIRA. *Professional Node.js - Building Javascript Based Scalable Software*. John Wiley Sons, Inc., 2013.
- [24] callbackhell.com. *Callback Hell - A Guide to Writing Asynchronous JavaScript Programs*. Disponível em: <http://callbackhell.com/>, Acesso em: 19 de abril de 2016.
- [25] S. ROBINSON. *Avoiding Callback Hell in Node.js*. Stack Abuse, jul. 2015. Disponível em: <http://stackabuse.com/avoiding-callback-hell-in-node-js/>, Acesso em: 19 de abril de 2016.
- [26] T. OGASAWARA. *Workload Characterization of Server-Side JavaScript*. 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Milão - Itália, p. 280-285, dez. 2015. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7389066>, Acesso em: 20 de abril de 2016.
- [27] Y. XIAO. *Node.js in Flames*. The Netflix Tech Blog, nov. 2014. Disponível em: <http://techblog.netflix.com/2014/11/nodejs-in-flames.html>, Acesso em: 20 de maio de 2016.
- [28] Nodejs Foundation. *How Uber Uses Node.js to Scale Their Business*. Disponível em: <https://nodejs.org/static/documents/casestudies/Nodejs-at-Uber.pdf>, Acesso em: 20 de maio de 2016.
- [29] DANFOSS. *How to Use Solenoid Valves*. Disponível em: http://www.danfoss.com/NR/rdonlyres/CDF180FA-4AFE-46AF-99A1-4E20A4FC1418/0/ICPS600A402_1_oms_solenoid_valves_how_to_use.pdf, Acesso em: 22 de abril de 2016.

- [30] JEFFERSON AUTOMAÇÃO. *Válvulas Solenóides - Informação de Engenharia*. Disponível em: <http://www.jefferson.ind.br/imagens/download/Valvula-solenoides.pdf>, Acesso em: 22 de abril de 2016.
- [31] E. ROSE, T.; MONTGOMERY. *Brewery CIP Automation Systems*. MBAA District NW Fall Meeting, Portland/OR, out. 2010. Disponível em: http://www.mbaa.com/districts/northwest/events/documents/2010_10_08brewerycipautomation.pdf, Acesso em: 22 de abril de 2016.
- [32] F. FERRAZ. *Meios de Ligação de Tubos - Conexões de Tubulação - Válvulas Industriais*. IFBA, abr. 2009. Disponível em: http://docente.ifb.edu.br/paulobaltazar/lib/exe/fetch.php?media=apostila_tubulacao_ifba.pdf, Acesso em: 22 de abril de 2016.
- [33] IPEX. *EPDM FKM Chemical Resistance Guide*. IPEX Incorporated Canadá. Primeira Edição, 2009.
- [34] J.D. HUNTER. *Matplotlib A 2D graphics environment*. Computing In Science Engineering, v. 9, n. 3, p. 90-95, IEEE COMPUTER SOC, 2007.
- [35] scipy.org. *NumPy*. Disponível em: <http://www.numpy.org/>, Acesso em: 24 de abril de 2016.
- [36] J. FEDRIZZI, M.; SORIA. *Application of a single-board computer as a low cost pulse generator*. Measurement Science and Technology, v. 26, n. 9, set. 2015. Disponível em: <http://arxiv.org/abs/1504.07024>, Acesso em: 02 de dezembro de 2015.
- [37] TEXAS INSTRUMENTS. *PRU assembly language tools v2.1*. Disponível em: <http://www.ti.com/lit/ug/spruhv6a/spruhv6a.pdf>, Acesso em: 29 de março de 2016.
- [38] TEXAS INSTRUMENTS. *PRU optimizing C/C++ compiler v2.1*. Disponível em: <http://www.ti.com/lit/ug/spruhv7a/spruhv7a.pdf>, Acesso em: 29 de março de 2016.
- [39] MAXIM INTEGRATED. *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*. 2008. Disponível em: <https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>, Acesso em: 02 de dezembro de 2015.

- [40] B. LINKE. *Overview of I-Wire Technology and Its Use*. jun. 2008. Disponível em: <http://pdfserv.maximintegrated.com/en/an/AN1796.pdf>, Acesso em: 02 de dezembro de 2015.
- [41] MAXIM INTEGRATED. *Guidelines for Reliable Long Line I-Wire Networks*. set. 2008. Disponível em: <http://pdfserv.maximintegrated.com/en/an/AN148.pdf>, Acesso em: 02 de dezembro de 2015.
- [42] THE LINUX KERNEL ARCHIVES. *Index of /doc/Documentation/w1*. Disponível em: <https://www.kernel.org/doc/Documentation/w1/>, Acesso em: 03 de dezembro de 2015.
- [43] H.W.; WOLTMAN B.D. HE, N.; HUANG. *The Use of BeagleBone Black Board in Engineering Design and Development*. The 2014 ASEE North Midwest Section Conference, Iowa City, IA, out. 2014.
- [44] J. LUMME. *BeagleBone Home Automation*. Packt Publishing, 2013.
- [45] THE BEAGLEBOARD.ORG FOUNDATION. *Debian*. Disponível em: <http://beagleboard.org/project/debian/>, Acesso em: 03 de dezembro de 2015.
- [46] THE BEAGLEBOARD.ORG FOUNDATION. *BeagleBoard.org Latest Firmware Images*. Disponível em: <http://beagleboard.org/latest-images>, Acesso em: 03 de dezembro de 2015.
- [47] putty.org. *Download Putty*. Disponível em: <http://www.putty.org>, Acesso em: 03 de dezembro de 2015.
- [48] NTP.BR. *Saiba mais sobre o NTP*. Disponível em: <http://ntp.br/>, Acesso em: 21 de dezembro de 2015.
- [49] NIC.BR. *Núcleo de Informação e Coordenação do Ponto BR*. Disponível em: <http://nic.br/>, Acesso em: 21 de dezembro de 2015.
- [50] Divisão Serviço da Hora. *Divisão Serviço da Hora - DSHO*. Disponível em: <http://pcdsh01.on.br/>, Acesso em: 21 de dezembro de 2015.
- [51] Cloud9 IDE Inc. *License for the SDK and Packages*. Disponível em: <https://cloud9-sdk.readme.io/docs/the-licenses-for-cloud9-sdk-and-packages>, Acesso em: 25 de abril de 2016.

- [52] STRAUB B. CHACON, S. *Pro Git - Everything You Need to Know About Git*. Apress Media LLC, Segunda Edição, 2014. Disponível em: <https://git-scm.com/book/en/v2>, Acesso em: 25 de abril de 2016.
- [53] Brewer's Friend. *Beer Styles - IBU Chart Graph*. Disponível em: <http://www.brewersfriend.com/2009/01/24/beer-styles-ibu-chart-graph-bitterness-range/>, Acesso em: 23 de abril de 2016.

Apêndice A

Programmable Real-time Unit – PRU-ICSS

A PRU-ICSS – *Programmable Real-Time Unit Subsystem and Industrial Communications Subsystem* (Unidade Programável de Tempo-Real e Subsistema de Comunicação Industrial, em tradução livre) é um subsistema do processador AM3358 que equipa a BBB, fabricado pela *Texas Instruments*, embora não possua suporte oficial da mesma [16]. Esta unidade consiste de dois núcleos RISC de 32 bits, uma memória compartilhada de 12kB, uma memória de dados e uma memória de programa para cada núcleo, ambas com 8kB de capacidade, e periféricos internos – além da possibilidade de acessar todos os eventos, pinos e recursos do SoC AM3358 por meio de uma interface OCP, que confere a este subsistema grande flexibilidade. Cada núcleo da PRU pode operar independentemente ou de maneira sincronizada, dependendo de como o *firmware* para eles é escrito [17]. Na figura é ilustrado o diagrama de blocos deste sistema.

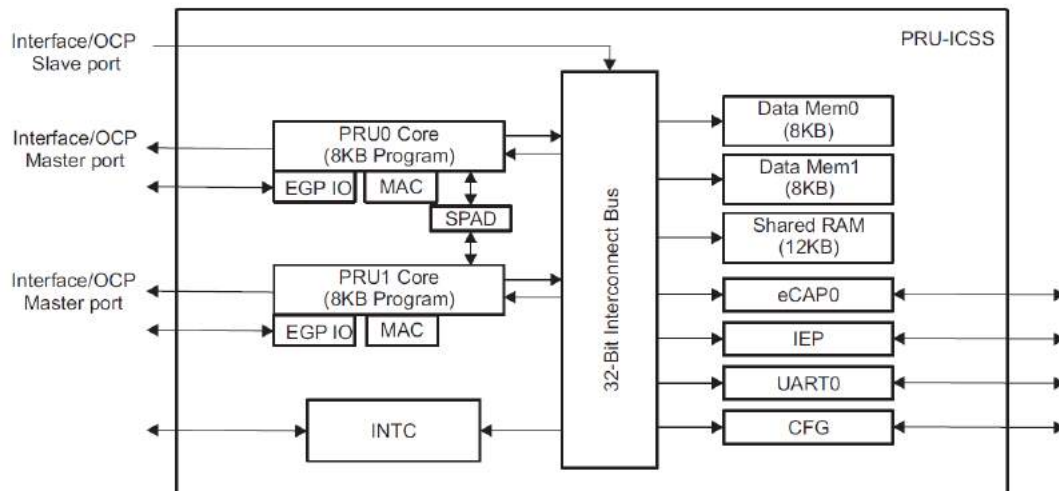


Figura A.1: Diagrama de blocos do subsistema PRU-ICSS.

Fonte: TEXAS INSTRUMENTS (2015)

Os núcleos da PRU foram otimizados para realizar tarefas com requisitos severos de tempo-real e, portanto, tem um *clock* de 200MHz e um sistema de instruções RISC no qual todas elas são executadas em um ciclo de *clock*, propositalmente sem suporte a *pipelining*. Cada núcleo tem 31 registradores, cuja utilização vai de propósito geral a indexação e controle de GPIO, acessíveis a nível de bit, byte, *halfword* (16 bits), *word* (32 bits) ou por ponteiro [17]. Uma vez que cada instrução pode ser executada em 5ns e o acesso a GPIO é feito por meio de uma única instrução de acesso a registradores, isto significa que é possível obter uma resolução de chaveamento determinística de no máximo 5ns, com dados obtidos na prática, por [36], de *jitter* RMS de 290ps e estabilidade de frequência na ordem de 10 ppm.

Uma vez que as diversas funcionalidades do módulo PRU são acessadas por meio de mapeamento da memória de dados, é preciso saber a faixa de endereço destas. Aqui deve ser feita a distinção entre acesso à memória local da PRU, que compreende os endereços de 0x0000_0000 a 0x0007_FFFF, e o acesso à memória global, cujos endereços começam em 0x0008_0000. Também é importante notar que as memórias internas dos módulos podem ser acessados com o endereço global de memória, mas isso reduz significativamente o tempo de acesso, já que o sinal é roteado para fora da PRU (por meio do OCP) antes de ser recebido [17]. Na tabela A.1 é apresentado o mapa de memória local e na tabela A.2 é apresentado o mapa de memória global. Também há uma tabela de constantes gravada em hardware, com valores de uso recorrente, para economizar instruções de programação e registradores, nos quais estes endereços precisariam ser carregados se não estivessem disponíveis em hardware [17].

Embora cada PRU suporte 30 canais de entrada e 32 canais de saída no modo direto [17], a BBB roteia somente 25 pinos do SoC ao conector de expansão. Destes, 10 pinos são exclusivos da PRU0 (8 saídas ou 9 entradas) e 15 pinos para a PRU1 (13 saídas ou 14 entradas). Eventualmente, alguns pinos pré-configurados para outras funções devem ser reconfigurados na *device tree* antes que possam ser utilizados [16]. Os pinos da BBB acessíveis pela PRU podem ser obtidos na figura A.2.

P9				P8			
1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8
9	10	11	12	9	10	11	12
13	14	15	16	13	14	15	16
17	18	19	20	17	18	19	20
21	22	23	24	21	22	23	24
25	26	27	28	25	26	27	28
29	30	31	32	29	30	31	32
33	34	35	36	33	34	35	36
37	38	39	40	37	38	39	40
41	42	43	44	41	42	43	44
45	46			45	46		

Figura A.2: Pinos da BeagleBone Black acessíveis pela PRU-ICSS.

Fonte: Site oficial da Fundação BeagleBoard.org¹

¹Disponível em <http://beagleboard.org/Support/bone101>

Tabela A.1: Mapa de Memória Local da PRU.
 Fonte: adaptado de TEXAS INSTRUMENTS(2014)

Endereço Inicial	PRU0	PRU1
0x0000_0000	Mem. dados da PRU0	Mem. dados da PRU1
0x0000_2000	Mem. dados da PRU1	Mem. dados da PRU0
0x0001_0000	Mem. dados compartilhada	Mem. dados compartilhada
0x0002_0000	INTC	INTC
0x0002_2000	Controle da PRU0	Controle da PRU0
0x0002_2400	Reservado	Reservado
0x0002_4000	Controle da PRU1	Controle da PRU1
0x0002_4400	Reservado	Reservado
0x0002_6000	CFG	CFG
0x0002_8000	UART0	UART0
0x0002_A000	Reservado	Reservado
0x0002_C000	Reservado	Reservado
0x0002_E000	IEP	IEP
0x0003_0000	eCAP0	eCAP0
0x0003_2000	Reservado	Reservado
0x0003_2400	Reservado	Reservado
0x0003_4000	Reservado	Reservado
0x0003_8000	Reservado	Reservado
0x0004_0000	Reservado	Reservado
0x0008_0000	System OCP_HP0	System OCP_HP1

As interfaces de interrupção e de GPIO estão contidas nos registradores R30 e R31: o registrador R31 é utilizado tanto para interface dos pinos configurados como entrada quanto para leitura e/ou geração de eventos de interrupção, enquanto o registrador R30 retorna status ou muda o valor dos pinos configurados como saída, para leitura e escrita, respectivamente. Embora a entrada possa ser configurada nos modos de captura paralela de 16 bits e trem de pulsos de 28 bits, estes modos de operação não serão detalhados, já que seu uso não está previsto no escopo deste projeto. O mesmo é válido para a saída, que além do modo direto, pode ser configurada para transmitir um trem de pulsos [17].

Outra característica da PRU é a existência de um IEP - *Industrial Ethernet Peripheral* (Periférico Ethernet Industrial, em tradução livre), composto de um *timer* para Ethernet industrial com 8 eventos de comparação e uma porta digital de E/S. O *timer* Ethernet industrial é simplesmente um temporizador de 32 bits e, portanto, pode ser utilizado para uso geral. É um contador positivo, cujo valor dos incrementos pode ser programado na faixa de 1-16, com compensador de uso opcional. Os 8 registradores comparadores permitem a criação de eventos cada vez que o valor do comparador corresponde ao do temporizador [17].

Tabela A.2: Mapa de Memória Global da PRU.
 Fonte: adaptado de TEXAS INSTRUMENTS(2014)

Endereço de Offset	PRU-ICSS
0x0000_0000	Mem. dados da PRU0
0x0000_2000	Mem. dados da PRU1
0x0001_0000	Mem. dados compartilhada
0x0002_0000	INTC
0x0002_2000	Controle da PRU0
0x0002_2400	Debug da PRU0
0x0002_4000	Controle da PRU1
0x0002_4400	Debug da PRU1
0x0002_6000	CFG
0x0002_8000	UART0
0x0002_A000	Reservado
0x0002_C000	Reservado
0x0002_E000	IEP
0x0003_0000	eCAP0
0x0003_2000	Reservado
0x0003_2400	Reservado
0x0003_4000	IRAM da PRU0
0x0003_8000	IRAM da PRU1
0x0004_0000	Reservado

No que diz respeito ao desenvolvimento de *firmware* para a PRU, há um conjunto de instruções RISC em assembly, suportado por um conjunto de ferramentas que ajudam a gerar o arquivo binário a ser carregado na memória de programa. Alguns exemplos são o *assembler*, *archiver* e o *linker* [37]. Outra ferramenta de desenvolvimento proporcionada pela Texas Instruments é um compilador C/C++, que facilita a programação em níveis de abstração elevados [38]. A figura A.3 apresenta um diagrama de blocos que ilustra os passos a serem executados desde compilar o código C/C++ até carregar o arquivo executável na memória de programa da PRU.

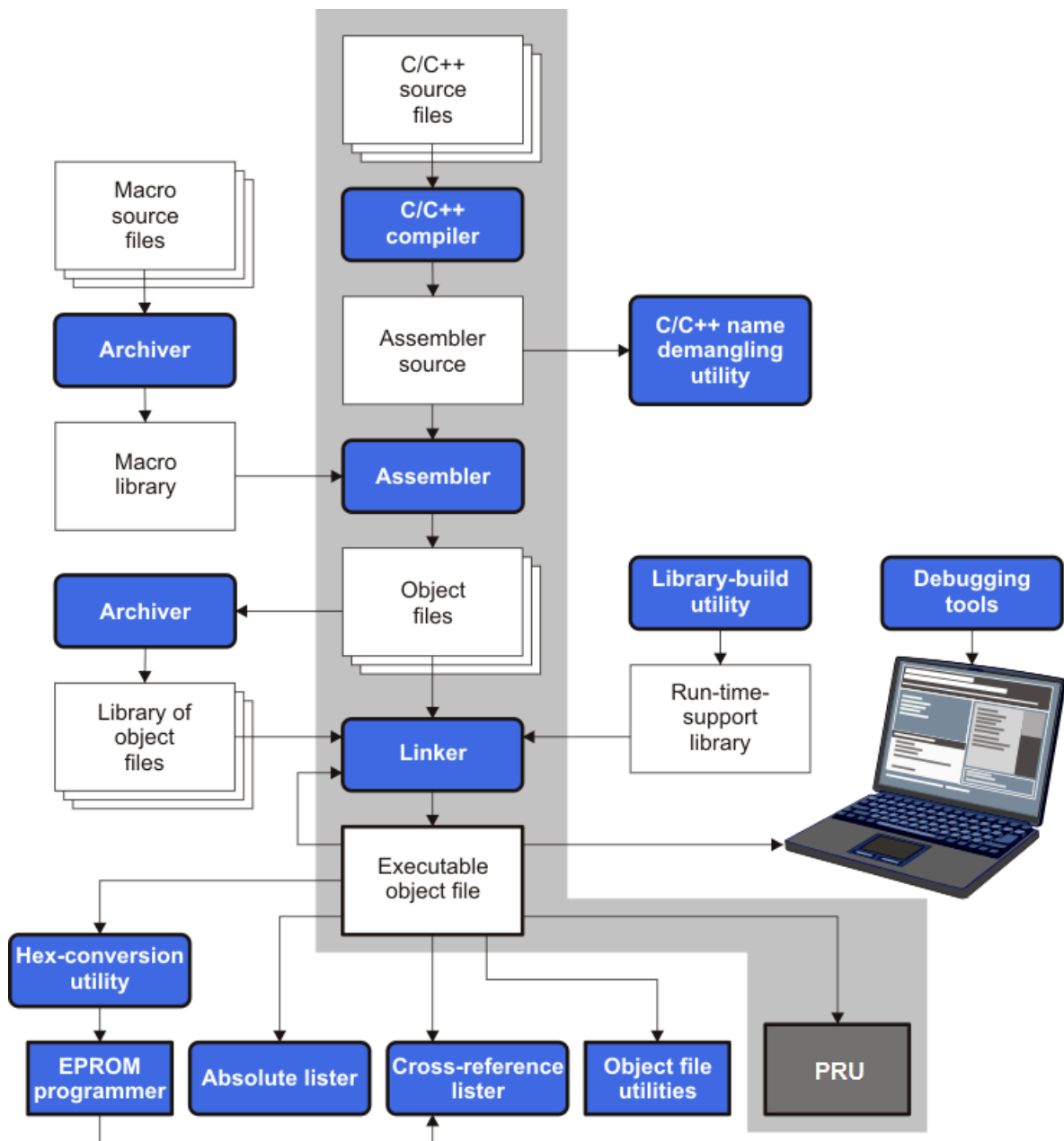


Figura A.3: Fluxo de desenvolvimento de software da PRU.

Fonte: TEXAS INSTRUMENTS

Apêndice B

Sensor de temperatura DS18B20

O sensor de temperatura DS18B20, projetado pela Maxim, é um sensor digital que utiliza somente um pino de dados para comunicação com o dispositivo *master*, cujas regras são estabelecidas pelo protocolo proprietário de comunicação *1-Wire*[®] [39].

1-Wire é um protocolo de comunicação serial *half-duplex* que utiliza uma linha para transmissão de dados, além da referência. Ele é do tipo *master/slave*, ou seja, um dispositivo *master* inicia a comunicação e os dispositivos *slave* somente respondem aos seus comandos. No caso deste protocolo específico, todo dispositivo possui um identificador de 64 bits único, dentro do qual um byte indica o tipo do dispositivo. Devido ao seu modo de funcionamento, quando a linha de dados é desconectada, os dispositivos *slave* entram em estado de *reset*, o que favorece seu uso em aplicações de contato. Enquanto a maioria destes dispositivos tem uma faixa de alimentação de 2,80-5,25V e não possui pino para alimentação [40] – utilizando um sistema de alimentação parasita – o sensor DS18B20 tem a possibilidade de alimentação parasita ou não e sua faixa tolerada para funcionamento é de 3-5,5V [39].

Uma vez que este protocolo de comunicação foi concebido para operações nas quais os dispositivos *master* e *slave* estão fisicamente muito próximos, alguns cuidados devem ser observados ao utilizar linhas de transmissão de longas distâncias para manter a boa performance do sistema [41]. Como o estudo da Maxim acerca das implicações do uso de linhas de longa distância negligencia este aspecto para redes com poucos dispositivos e cabeamento curto (menor que 10m), os cuidados de projeto das redes *1-wire* não serão aqui especificados, deixando um aviso para trabalhos futuros que visem aplicações de longas distâncias ou com número elevado de dispositivos, a exemplo de uma microcervejaria de porte industrial.

Com relação a aspectos específicos do sensor de temperatura, este apresenta precisão de 0,5°C para a faixa de temperaturas de -10°C a +85°C, e de 2,0°C para a faixa completa de

operação, de -55°C a $+125^{\circ}\text{C}$. Quanto à resolução, esta pode ser programada de 9 a 12 bits, sendo 8 bits para a parte inteira e 1 a 4 bits decimais. O custo da melhor resolução é o aumento no tempo de conversão de uma leitura para valor digital, variando de 93,75ms a 750ms para os casos extremos [39].

A arquitetura interna do CI é apresentada na figura B.1. Nota-se que o dispositivo, por utilizar o protocolo *1-Wire*, tem uma ROM de 64 bits para identificação [40], além de uma região de memória de rascunho, que consiste de uma memória SRAM de 64 bits, dividida em 8 regiões de 1 byte, para promover a interface com as funcionalidades do CI: sensor de temperatura, alarmes programáveis para estouro de valores mínimo e máximo das medições, registrador de configuração da resolução e gerador de código CRC de 8 bits [39]. A figura B.2 apresenta o mapa de memória do DS18B20 – nota-se que os registradores de configuração e alarme são copiados para uma memória não volátil (EEPROM) após configurados pelo *master*, retendo seus valores mesmo em caso de corte na alimentação. Outra informação útil obtida a partir da figura B.2 é o valor da temperatura lido após uma queda de alimentação, que é de $+85^{\circ}\text{C}$ até que o dispositivo esteja pronto para uso [39].

As leituras de temperatura são feitas em dois bytes, LSB e MSB, representados na figura B.3, e que juntos são uma representação em complemento de dois, sendo que os 5 bits mais significativos são redundantes e, para resoluções menores do que 12 bits, o valor dos bits menos significativos é indefinido (1 bit indefinido para 11 bits de resolução; 2 bits indefinidos para 10 bits de resolução, etc).

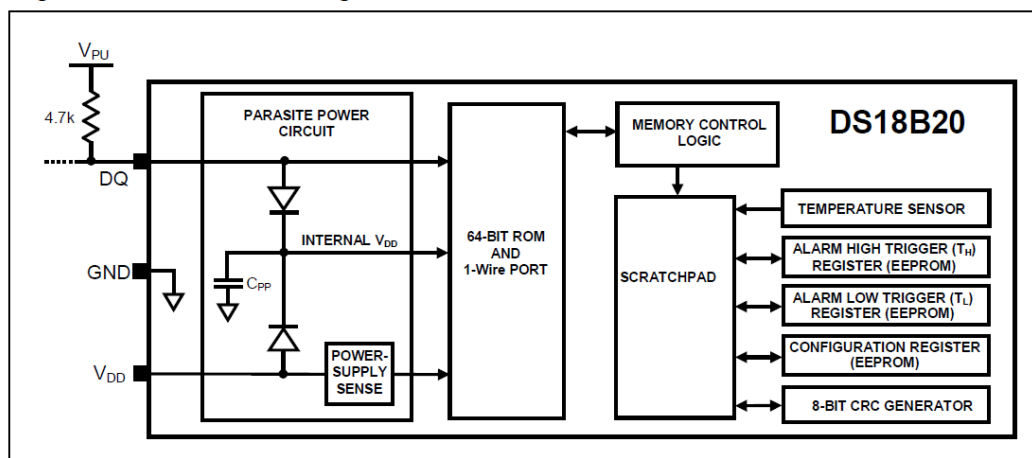


Figura B.1: Arquitetura interna do DS18B20
Fonte: MAXIM INTEGRATED

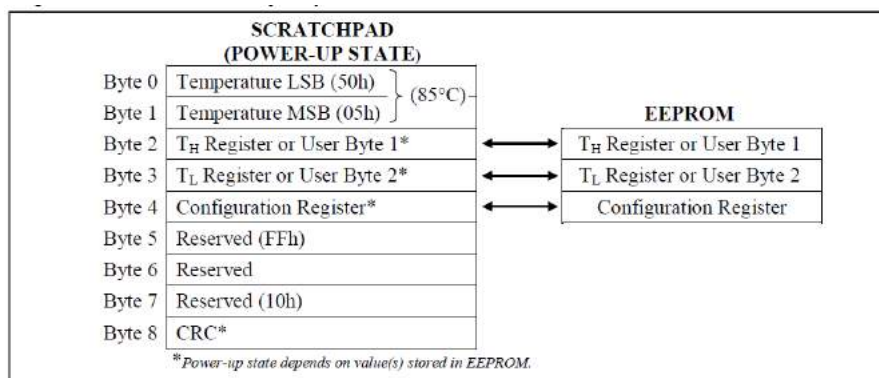


Figura B.2: Mapa de memória do DS18B20.

Fonte: MAXIM INTEGRATED

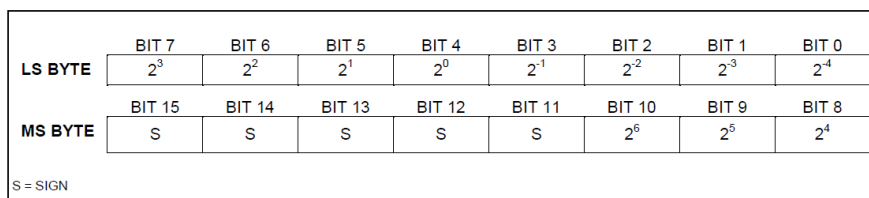


Figura B.3: Registradores de temperatura do DS18B20.

Fonte: MAXIM INTEGRATED

O Kernel do Linux possui *device drivers* que suportam parcialmente o uso do sensor DS18B20, portanto não há a necessidade de descrever em detalhes o protocolo *1-Wire*. O *device driver w1-gpio* controla o barramento (*master*) *1-wire* utilizando a API GPIO para controlar a linha, sendo que a porta utilizada é especificada na descrição de hardware do sistema embarcado (*device tree*) abordada na seção 2.3. O *device driver w1-therm* suporta algumas famílias de sensores de temperatura da Maxim (*slave*), dentre elas a DS18*20, fazendo conversões básicas de temperatura e fornecendo-as ao sistema por meio do FS (*File System* ou Sistema de Arquivos). O resultado da abertura e leitura do arquivo correspondente a um sensor, é a conversão de temperatura pelo *driver* e posterior fornecimento dos dados recebidos do sensor — status da checagem do CRC e temperatura em °C/1000. Este driver não suporta conversão de temperatura simultânea de múltiplos sensores nem redução da precisão de conversão, portanto só são possíveis leituras de 12 bits, o que leva o acesso pelo FS a demorar cerca de 750ms. Em caso de alimentação parasita, somente um sensor pode estar conectado à linha por vez e, caso o sistema embarcado ofereça suporte a *pull-up* interno, o driver é capaz de usá-lo [42].

Apêndice C

BeagleBone Black

BBB é uma placa de sistema embarcado desenvolvida para a comunidade de código aberto (*open-source*), iniciantes e quaisquer pessoas interessadas em um sistema equipado com um processador ARM Cortex-A8 32-bits de baixo custo, seja para teste de conceito ou mesmo uso pessoal e profissional, embora o último não seja o mais adequado. Embora o sistema tenha sido desenvolvido com um número reduzido de funcionalidades, de modo a proporcionar uma experiência de uso simples, este SBC não é uma plataforma completa de desenvolvimento nem é voltada para o desenvolvimento de um produto específico, mas sim para a experimentação e aprendizado nas áreas de *software* e *hardware* [16]. A figura C.1 apresenta a placa e seus principais componentes, em conjunto com a tabela C.1, onde estão contidas as especificações gerais de hardware e a figura C.2, na qual está contido um diagrama de blocos do sistema, de alto nível de abstração. O diagrama esquemático do sistema pode ser obtido no Anexo I.

É imprescindível atentar para o fato de que a BBB possui diversas revisões de *hardware*, sendo que a versão utilizada neste trabalho é a **revisão C**.

A combinação do processador ARM Cortex-A8 com o projeto de *hardware* da placa possibilita o uso de um sistema operacional embarcado – geralmente uma distribuição Linux – que fornece facilidades de programação, a exemplo de uma IDE acessada pelo navegador da internet chamada Cloud9 em combinação com uma biblioteca de acesso ao hardware em *server-side* Javascript (Node.js), possibilitando a prototipagem rápida de soluções embarcadas de produtos voltados ao mundo real. À medida que o desenvolvedor do sistema se torna mais experiente, é possível desenvolver soluções mais complicadas em diversas linguagens de programação, utilizando-se para isto do sistema operacional Linux embarcado na placa. [43]. Além disso, projetos como o do carro de controle remoto via web, desenvolvido por

[43] com o intuito de confirmar as capacidades deste SBC e seu potencial uso no ensino de sistemas embarcados, reforçam a escolha da BBB como plataforma para este projeto. É importante notar que alguns dos pontos fortes deste SBC, ou computador de placa única, estão implícitos em sua classificação: por ser um computador, ele apresenta os benefícios de uma plataforma que suporta um universo de múltiplas aplicações — embora restrito pelo reduzido poder de processamento se comparado a um computador tradicional — e, no caso da escolha do Linux como sistema operacional, traz os benefícios de um *kernel* amplamente estável.

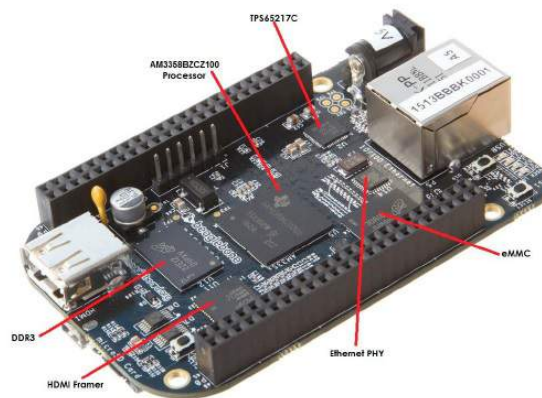


Figura C.1: BeagleBone Black e principais componentes.
Fonte: COLEY (2014)

- **Sitara AM3358BZCZ100** - SoC (*system on chip*)
- **Micron 512MB DDR3L ou Kingston 512MB DDR3** - memória RAM
- **TPS65217C PMIC** - controlador de alimentação dos diferentes componentes do sistema
- **SMSC Ethernet PHY** - interface física à rede Ethernet
- **Micron eMMC** - memória não volátil MMC de 4GB
- **HDMI Framer** - controlador para uso com display HDMI ou DVI-D

Tabela C.1: Especificações Gerais da BeagleBone Black.

Fonte: adaptado de COLEY(2014)

	Especificação
Processador	Sitara AM3358BZCZ100, 1GHz, 2000 MIPS
Motor Gráfico	SGX530 3D, 20M polígonos/s
Memória SDRAM	512MB DDR3L 800MHz
Memória Flash	4GB, 8bit eMMC
CI Gerenciador de Alimentação	TPS65217C + regulador adicional (linear)
Suporte a <i>debug</i>	Interface serial, Conector CTI JTAG opcional de 20 pinos
Fonte de Alimentação	mini USB, conector DC ou 5VDC na barra de pinos
PCB	8,64cm x 5,33cm (3,4"x 2,1") - 6 <i>layers</i>
Leds Indicadores	1 para alimentação; 2 para Ethernet; 4 acessíveis ao usuário
USB 2.0 <i>Client</i>	Acesso à USB0 via miniUSB
USB 2.0 <i>Host</i>	Acesso à USB1, soquete tipo A, 500mA LS/FS/HS
Serial	Acesso à UART0 via <i>header</i> de 6 pinos, TTL 3.3V
Ethernet	10/100 RJ45
SD/MMC	Conector microSD, 3,3V
Chaves	Botões push de reset, boot e alimentação
Saída de vídeo	16b HDMI, 1280x1024 (MAX), 1024x768, 1280x720, 1440x900, 1920x1080@24Hz
Audio	Via interface HDMI, estéreo
Conectores de expansão	Alimentação 5V, 3,3V e VDD_ADC(1,8V) 3,3V para todos os sinais de E/S McASP0, SPI, I2C, até 69 pinos de GPIO, LCD, GPMC, MMC1, MMC2, 7 entradas para conversor A/D (1,8V MAX), 4 <i>timers</i> , 4 UART, CAN, PWM via <i>hardware</i> , interrupção XDMA
Peso	39,68g (1,4oz)
Consumo@5VDC	Ocioso - 280mA Carregando página web - 430mA O consumo varia conforme o desempenho

Cada pino de GPIO digital da BBB possui até 7 modos diferentes de operação, que são configurados utilizando uma ferramenta do Linux chamada *Device Tree*. A figura C.3 apresenta a configuração padrão dos pinos, ou seja, quando é usada a distribuição Debian do Linux, pré-compilada e que é fornecida com a placa. Com relação aos pinos de GPIO, é importante notar que sua lógica opera com níveis de tensão de 0V e 3,3V para os níveis baixo e alto, respectivamente e portanto aplicar uma tensão negativa ou superior a 3,3V pode danificar permanentemente a placa [44].

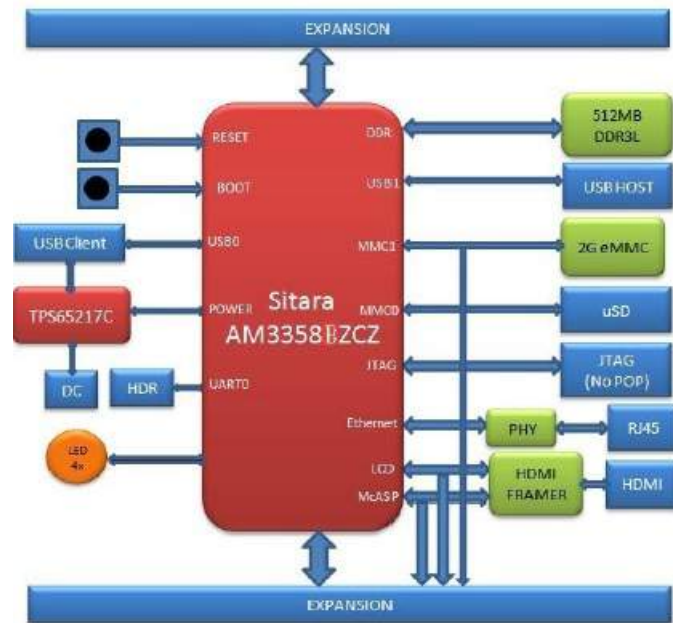


Figura C.2: Diagrama de blocos de alto nível da BeagleBone Black.
Fonte: COLEY (2014)

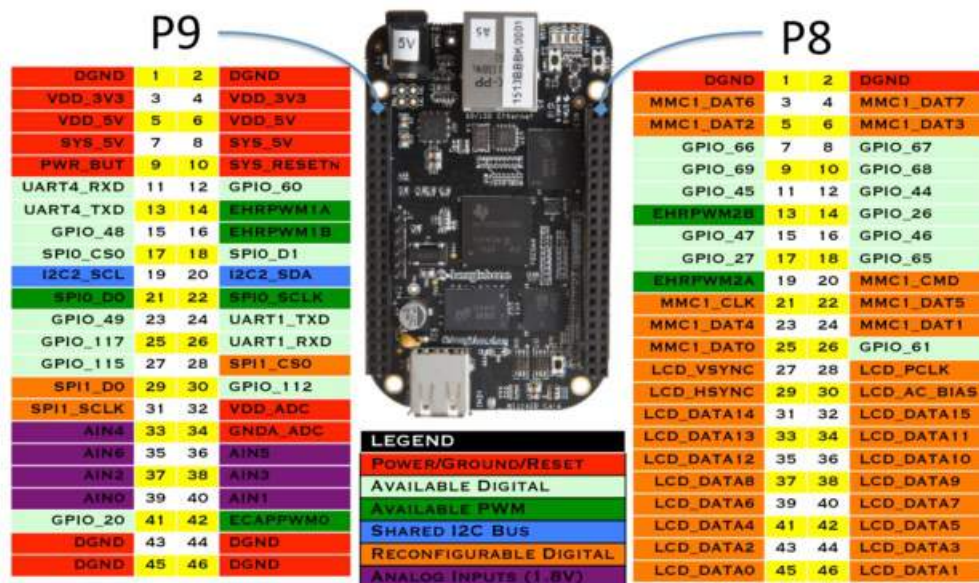


Figura C.3: Configuração padrão dos pinos da BeagleBone Black.
Fonte: Site oficial da Fundação BeagleBoard.org¹

C.0.1 Programmable Real-time Unit – PRU-ICSS

A PRU-ICSS – *Programmable Real-Time Unit Subsystem and Industrial Communications Subsystem* (Unidade Programável de Tempo-Real e Subsistema de Comunicação Industrial,

¹Disponível em <http://beagleboard.org/Support/bone101>

em tradução livre) é um subsistema do processador AM3358 que equipa a BBB, fabricado pela *Texas Instruments*, embora não possua suporte oficial da mesma [16]. Em uma fase inicial de desenvolvimento do projeto, foi vislumbrado o uso desta unidade para implementação de um controlador de temperatura e, embora o desenvolvimento deste tenha sido interrompido, foi realizada uma revisão teórica da PRU, exposta no apêndice A.

C.0.2 Configuração da BeagleBone Black

O primeiro passo para o desenvolvimento do projeto foi a configuração da BeagleBone Black, que é o coração deste sistema. Para tal, foi escolhido o procedimento de gravar a distribuição Debian pré-compilada do sistema operacional Linux na memória de armazenamento eMMC de 4GB da placa. Esta distribuição foi escolhida em função da facilidade de obtenção de suporte em fóruns da internet, artigos e livros, dentre outros, além de ser o SO oficial da BBB [45].

NOTA: Na ocasião da realização deste procedimento, havia uma diferenciação entre a imagem fornecida para uso diretamente a partir do cartão μ SD e a imagem para gravação no eMMC, porém agora o procedimento adotado pela fundação BeagleBoard.org é fornecer somente uma imagem pronta para uso diretamente do cartão μ SD e, no caso de o usuário decidir gravar a distribuição no eMMC, o arquivo `uEnv.txt` na partição de *boot* do cartão μ SD deve ser editado conforme instruções antes de ser inserido na BBB.

Tanto a última versão estável do Debian para BBB (v. 7.9), conhecida como *Wheezy* e a última versão em desenvolvimento (v. 8.2), conhecida como *Jessie*, quanto a versão utilizada neste projeto (v. 7.5) podem ser obtidas a partir do site oficial da BBB [46], conforme ilustrado na figura C.4. A instalação da versão 7.5 se resume a:

1. Baixar a imagem Debian 7.5 para eMMC do site beagleboard.org/latest-images
2. Descomprimir a imagem
3. Gravar a imagem em um cartão μ SD formatado (não basta copiar, é preciso utilizar um software que grave a imagem corretamente. Neste caso foi utilizado o *Win32 Disk Imager*)
4. Inserir o cartão μ SD na BBB desligada
5. Pressionar o botão de boot e energizar a placa com ele ainda pressionado
6. Esperar um dos leds de status acender antes de liberar o botão

7. Esperar os 4 leds de status ficarem continuamente acesos (durante o processo de gravação eles trabalharão de maneira intermitente)

8. Desenergizar a BBB e retirar o cartão µSD

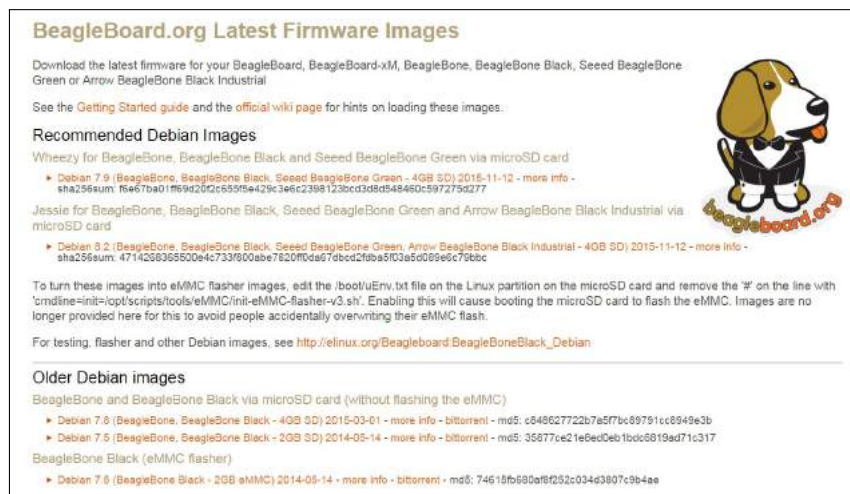


Figura C.4: Últimas imagens pré-compiladas disponíveis para a BBB.
Fonte: Adaptado do site oficial da Fundação BeagleBoard.org²

Após este procedimento, a BBB estará pronta para uso a partir da memória interna eMMC. A verificação do funcionamento do sistema foi feita conectando a placa a um computador instalado com Windows 7 e conectado à internet, via interface USB e posterior acesso através do software de código aberto para acesso via SSH *Putty*, disponível para *download* no site oficial putty.org [47] (qualquer software similar que possibilite o acesso via SSH pode ser utilizado para este fim). Para isso, é preciso também instalar no computador o driver que dá acesso à BBB via USB, disponível em beagleboard.org/getting-started. O acesso via SSH se dá utilizando o endereço de IP **192.168.7.2**, usuário: **debian** e senha: **debian** [44]. A figura C.5 mostra o ambiente de configuração do *Putty* e a figura C.6 apresenta a tela após uma tentativa de login bem sucedida.

²Disponível em <http://beagleboard.org/latest-images>

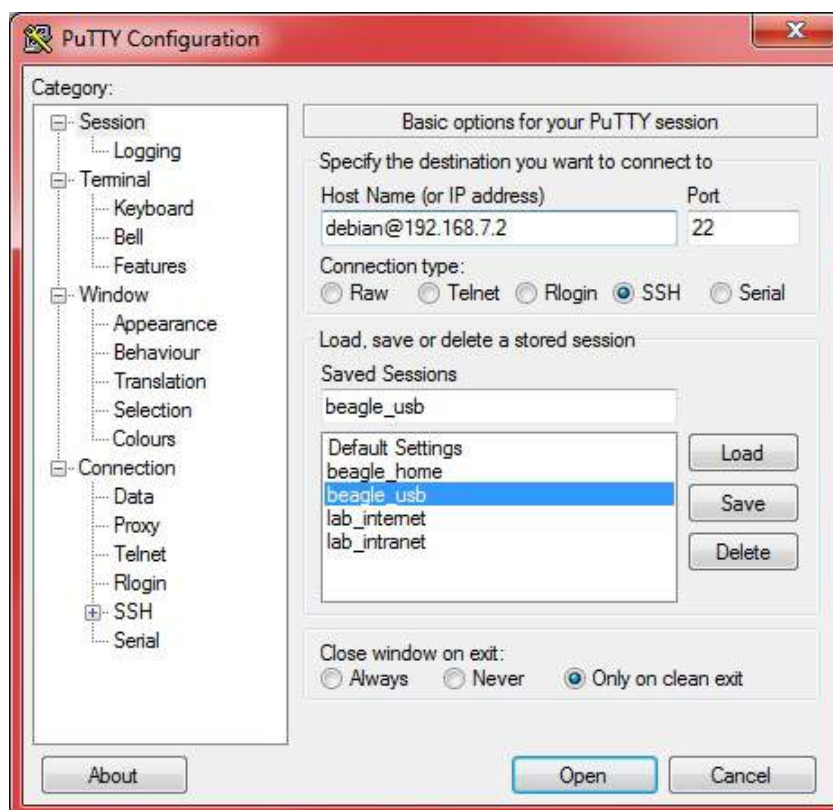


Figura C.5: Configuração do *Putty* para acesso SSH via USB

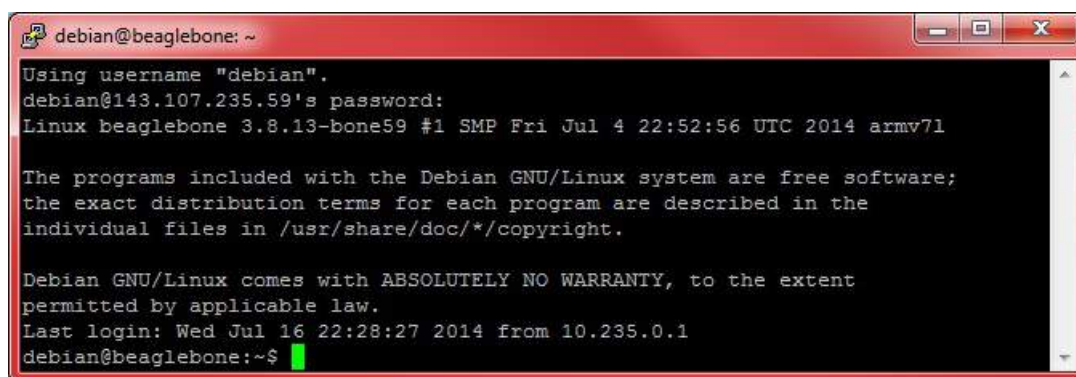


Figura C.6: Tentativa de login bem sucedida pelo *Putty*

C.0.3 Ajustes de rede para uso do adaptador Wi-Fi/USB

Para utilizar o adaptador Wi-Fi USB, em primeiro lugar este foi conectado à BBB. Em seguida, no terminal, o dispositivo foi listado para confirmar que o sistema estava identificando-o corretamente e ativado, permitindo assim obter a lista de redes Wi-Fi próximas. Com isso, foi possível obter os dados da rede Wi-Fi de interesse, para posterior edição do arquivo */etc/network/interfaces*, responsável pelas configurações de rede do sistema – este pode ser obtido no apêndice H. O código C.1 ilustra a sequência de comandos executados:

```

1 lsusb # verifica se o dispositivo USB foi detectado
2 sudo ifconfig wlan up # ativa a interface de rede USB
3 sudo iwlist scan # lista as redes Wi-Fi disponíveis
4
5 # Identificar os valores da rede de interesse, e.g.
6 # ESSID:"Nome_da_rede" -> nome da rede
7 # IE: IEEE 802.11i/WPA2 Version 1 -> encriptação
8 # E então é modificado o arquivo de configuração
9
10 # abre o arquivo para edição
11 sudo nano /etc/network/interfaces
12 # Após modificar o arquivo e salvá-lo:
13
14 sudo ifup wlan0 # ativa a interface de rede Wi-Fi
15 sudo ifdown eth0 # desativa a interface Ethernet

```

Código-fonte C.1: Passos para configuração do Wi-Fi

No arquivo `/etc/network/interfaces`, as configurações utilizadas entre as linhas 19 e 35 do código são referentes à configuração do Wi-Fi para configuração da BBB com endereço de IP estático 192.168.1.155, permitindo o acesso à plataforma remotamente pela internet e não somente dentro da intranet. Observe-se que o acesso a este IP da intranet se dá pelo IP externo 143.107.235.59, que é o endereço do departamento da engenharia elétrica da USP de São Carlos.

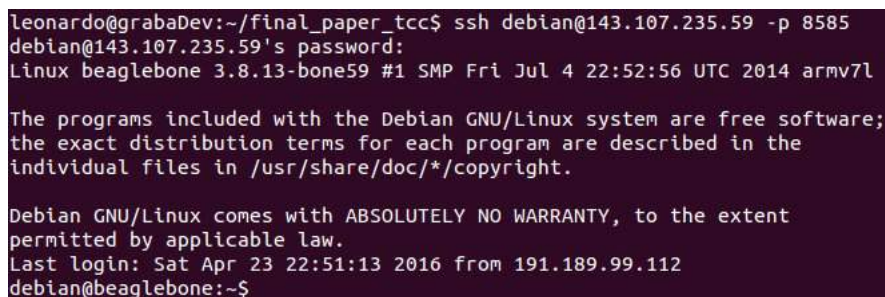
Depois de todo este procedimento, o dispositivo ainda não estava funcionando após a operação de *reboot*, portanto foi utilizado um *script* de reset do Wi-Fi executado durante o *boot*, cujas instruções de uso, obtidas em <https://learn.adafruit.com/setting-up-wifi-with-beaglebone-configuration>, estão descritas no código-fonte C.2. A figura C.7 indica sucesso de conexão via SSH usando uma máquina com sistema operacional Linux Ubuntu 14.04 LTS.

```

1 git clone https://github.com/adafruit/wifi-reset.git #
   download do script
2 cd wifi-reset/ #entra no diretório do script
3 chmod +x install.sh #permissão de execução para o script
4 sudo ./install.sh #executa o script

```

Código-fonte C.2: Configuração para reset do Wi-Fi após o boot



```

leonardo@grabaDev:~/final_paper_tcc$ ssh debian@143.107.235.59 -p 8585
debian@143.107.235.59's password:
Linux beaglebone 3.8.13-bone59 #1 SMP Fri Jul 4 22:52:56 UTC 2014 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Apr 23 22:51:13 2016 from 191.189.99.112
debian@beaglebone:~$

```

Figura C.7: Acesso à BBB via SSH

C.0.4 Data/hora

Para a configuração automática de data e hora da BBB, foi escolhido o uso do protocolo NTP – *Network Time Protocol* ou Protocolo de Tempo para Redes – cujo objetivo é sincronizar os relógios de dispositivos conectados a uma rede a partir de fontes precisas [48]. No Brasil, a confiabilidade deste protocolo é responsabilidade do Observatório Nacional (ON), responsável legal por garantir a Hora Legal Brasileira, em conjunto com o Núcleo de Informação e Coordenação do Ponto BR, administrador e operador do domínio ".br"[49, 50].

Primeiramente foi realizada a instalação do protocolo, seguida da configuração do arquivo */etc/ntp.conf* e da modificação do arquivo que indica a *timezone*, sendo este um link simbólico para o verdadeiro arquivo. Os comandos executados estão descritos no código-fonte C.3:

```
1 sudo apt-get install ntp #instala o NTP
2 sudo nano /etc/ntp.conf #abre o arquivo de configuração para
   edição
3 sudo mv /etc/localtime /etc/localtime-old #backup do arquivo
   antigo da timezone
4 sudo ln -s /usr/share/zoneinfo/America/Sao_Paulo /etc/
   localtime #cria link simbólico para arquivo da nova
   timezone
```

Código-fonte C.3: Instalação e configuração do NTP

Na edição do arquivo */etc/ntp.conf*, a única modificação feita foi a substituição dos servidores de hora padrão para os servidores brasileiros, listados no endereço <http://ntp.br>, responsável pelo oferecimento da **Hora Legal Brasileira** e cuja estrutura de oferecimento do serviço está apresentada na figura C.8. No final de cada endereço foi adicionada a instrução *iburst*, conforme descrito no website oficial do protocolo (<http://support.ntp.org/bin/view/Support/ConfiguringNTP>) e que, embora não esteja claro seu funcionamento, foi essencial para que o horário fosse corretamente configurado a cada *reboot*. O trecho de código-fonte C.4 indica as modificações no arquivo */etc/ntp.conf* adequando aos servidores brasileiros:

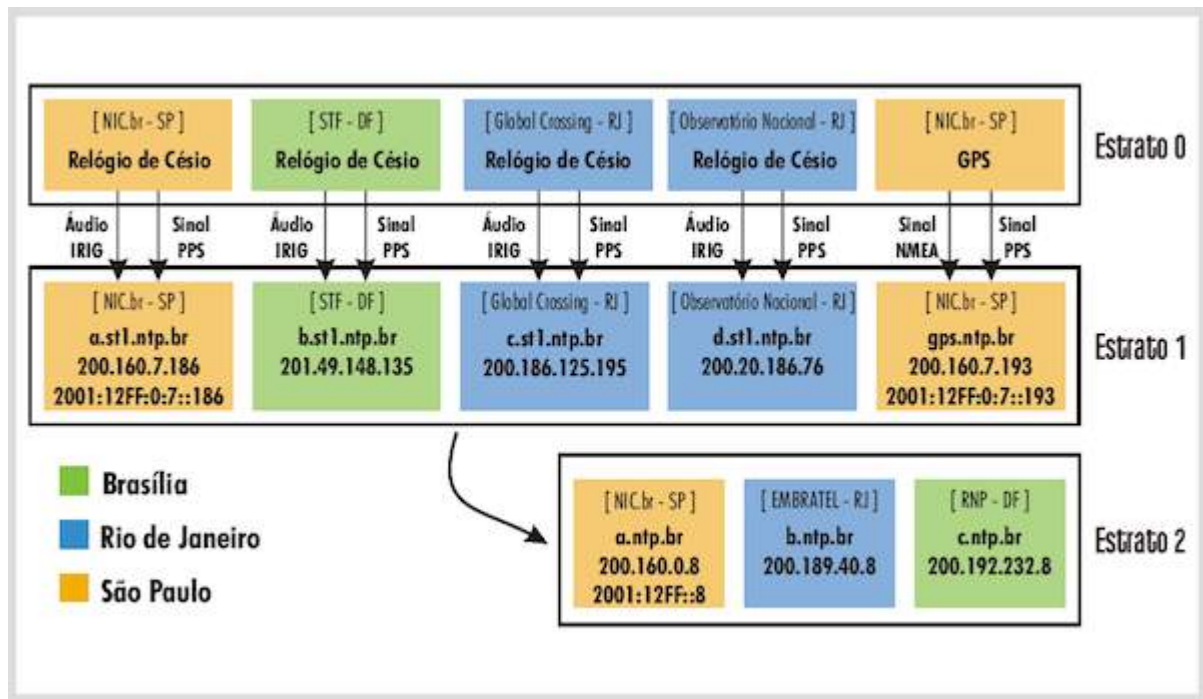


Figura C.8: Estrutura do serviço de data e hora do Ntp.br
Disponível em <http://ntp.br/img-ntp/ntp-estrutura.png>

```

1 # You do need to talk to an NTP server or two (or three).
2 #server ntp.your-provider.example
3
4 # pool.ntp.org maps to about 1000 low-stratum NTP servers.
   Your server will
5 # pick a different set every time it starts up. Please
   consider joining the
6 # pool: <http://www.pool.ntp.org/join.html>
7 server a.st1.ntp.br iburst
8 server b.st1.ntp.br iburst
9 server c.st1.ntp.br iburst
10 server d.st1.ntp.br iburst
11 server a.ntp.br iburst
12 server b.ntp.br iburst
13 server c.ntp.br iburst
14 server d.ntp.br iburst
15 server gps.ntp.br iburst

```

Código-fonte C.4: Servidores brasileiros do NTP

C.0.5 Sensor DS18B20

Uma vez que o sensor de temperatura DS18B20 já é suportado pelo Kernel e incluído como *driver* na distribuição padrão da BBB, somente foi necessário criar uma sobreposição de *device tree* indicando em que pino o sensor está ligado. Tanto o mux quanto o OCP do SoC precisaram ser configurados e, para a compilação do arquivo texto em binário, foi necessária a instalação do DTC, descrita no código-fonte C.5:

```

1 sudo wget -c https://raw.githubusercontent.com/RobertCNelson/tools/
  master/pkggs/dtc.sh
2 sudo chmod +x dtc.sh
3 ./dtc.sh

```

Código-fonte C.5: Instalação do DTC

O arquivo com a sobreposição da *device tree* é descrito na seção H.2, porém o código-fonte C.6 apresenta um trecho relevante deste, no qual observa-se a configuração do multiplexador do SoC. Note-se que esta configuração significa que o *pull-up* interno de 10kda BBB está ativado, eliminando a necessidade de um resistor externo. O valor mínimo de *pull-up* segundo a folha de dados do sensor é de 4,7k, mas não foi observada perda de desempenho do mesmo na presente configuração.

```

1 bb_wl_pins: pinmux_bb_wl_pins {
2     pinctrl-single,pins = <0x70 0x37>;
3 };

```

Código-fonte C.6: Fragmento de device tree para o DS18B20

A compilação da sobreposição é executada com o comando presente no trecho de código C.7. Adicionar -00A0" ao nome do arquivo de saída DTBO é fundamental. Note-se também que o arquivo DTBO é copiado para o diretório no qual ficam todas as sobreposições de *device tree*. Para carregar a sobreposição no boot, é preciso adicionar a linha *echo wl > /sys/devices/bone_capemgr.*slots* ao arquivo */etc/rc.local*.

```

1 sudo dtc -O dtb -o wl-00A0.dtbo -b 0 -@ wl.dts
2 sudo cp wl-00A0.dtbo /lib/firmware

```

Código-fonte C.7: Compilação de *device tree*

Após o reboot e com o sensor conectado à BBB, é possível obter o valor da temperatura realizando a leitura do arquivo */sys/devices/wl_bus_master1/28-*/wl_slave*, conforme ilustrado na figura C.9. Observe-se que na figura ao invés do asterisco foi utilizado o identificador único deste sensor específico — quando é utilizado um asterisco, ele é substituído por todos os possíveis nomes de sensores, portanto é útil para leitura de diversos sensores de uma vez. Para saber quantos dispositivos com o protocolo *1-wire* estão conectados à mesma linha e assim descobrir os números referentes a eles, é possível ler o arquivo */sys/devices/wl_bus_master1/wl_master_slaves* ou então verificar o conteúdo do diretório */sys/devices/wl_bus_master1/*.

```

debian@beaglebone:~$ cat /sys/devices/wl_bus_master1/28-000004ee8ded/wl_slave
c1 01 4b 46 7f ff 0f 10 38 : crc=38 YES
c1 01 4b 46 7f ff 0f 10 38 t=28062

```

Figura C.9: Leitura do sensor DS18B20 pelo terminal

C.0.6 Webserver Apache

Embora o Apache venha instalado na distribuição padrão da imagem do Debian para a BBB, este serviço foi desabilitado, de maneira a poupar recursos de hardware e evitar possíveis conflitos com o servidor implementado em Node.js. O comando utilizado para desabilitar o Apache está descrito na caixa C.8.

```
1 sudo update-rc.d -f apache2 remove
```

Código-fonte C.8: Desabilitando servidor web Apache

Apêndice D

IDE e sistema de controle de versão

Para facilitar o desenvolvimento e a documentação de *softwares*, duas ferramentas foram adotadas além do acesso ao sistema embarcado via SSH: *Cloud9*, cujo uso é descrito na seção D.1 e *Git/GitHub*, na seção D.2.

D.1 *Cloud9*

Cloud9 é um IDE do tipo *software as a service* ou software como um serviço (SaaS), ou seja, um software que está na nuvem e cujo acesso é feito por meio de um navegador de internet. Embora o serviço padrão seja hospedado nos próprios servidores da *Cloud9 IDE Inc.*, um SDK é disponibilizado, no qual está incluso o núcleo do projeto como código-aberto, livre para uso não comercial e desenvolvido em Node.js. Isto possibilitou a hospedagem do serviço do *Cloud9* localmente na BBB[51]. Esta é uma nítida vantagem para uma plataforma *headless* — sem periféricos do tipo *human interface device* (HID) — e com acesso à rede, a exemplo da BBB neste projeto, pois permite a programação direta no dispositivo por meio de uma interface gráfica via web.

Tanto o núcleo quanto as informações para instalação podem ser encontradas no repositório do projeto do *GitHub* (<https://github.com/c9/core>). A primeira tentativa de instalação realizado na BBB é apresentado na caixa de código-fonte D.1:

```
1 cd ~/
2 git clone git://github.com/c9/core.git c9sdk
3 cd c9sdk
4 scripts/install-sdk.sh
```

Código-fonte D.1: Primeira tentativa de instalação do *Cloud9 IDE core* na BBB

Ao tentar executar a aplicação apontando para o diretório */var/www* como diretório de

projeto, ou seja, o diretório raiz ao qual o *Cloud9* tem acesso, não foi possível acessar os arquivos uma vez que o diretório em si pertencia ao usuário *www-data* e os arquivos ao *root*.

Ao rodar a aplicação como administrador, outros erros ocorreram, o que levou à tentativa de instalar outro script presente no endereço <https://github.com/c9/install> do repositório do *Cloud9*, que permite a conexão da IDE a um servidor SSH. Ainda assim, os erros persistiram, o que levou à abertura de uma questão em <https://github.com/c9/core/issues/143>, na qual está descrito o processo adotado para solucionar o problema. Na caixa de código-fonte D.2 é apresentado o processo de instalação executado com sucesso:

```
1 cd ~/
2 wget --no-check-certificate https://raw.githubusercontent.com
  /c9/install/master/install.sh
3 sudo bash install.sh
```

Código-fonte D.2: Instalação bem sucedida do *Cloud9 IDE core* na BBB

Note-se que muitas tentativas foram executadas para obter o resultado final esperado, por isso em caso de tentativa de reprodução do processo de instalação aqui descrito é recomendado tentar usar somente o processo da caixa de código D.2 e, em caso de falha, partir para a etapa descrita anteriormente. Ainda assim, outro processo de instalação mais direto e que deve ser tentado prioritariamente ao aqui descrito, mas que não foi testado, está descrito em <https://cloud9-sdk.readme.io/docs/running-the-sdk>.

Para rodar a aplicação no background e continuar executando mesmo após desconectar da sessão SSH, é preciso executar o servidor do *Cloud9* como root usando o comando descrito na caixa D.3. O parâmetro *-l* indica o endereço de IP do servidor, *-p* a porta, *-a* usuário e senha e *-w* o diretório de trabalho. Também são feitas redireções das saídas *stdout* e *stderr* para os arquivos */var/www/log/cl9.out* e */var/www/log/cl9.err*, respectivamente.

```
1 sudo su #precisa ser root
2 cd /home/debian #pasta na qual esta instalado o Cloud9
3 nohup node c9sdk/server.js -l 143.107.xxx.xxx -p xxxx -a
  usuario:senha -w /var/www/ > /var/www/log/cl9.out 2> /var/
  www/log/cl9.err < /dev/null &
```

Código-fonte D.3: Execução do *Cloud9*

Ao realizar o acesso pelo navegador usando o IP e porta selecionados, ou seja, digitando *IP:porta* na URL do navegador, o resultado é apresentado na figura D.1. Note-se que o ambiente apresentado é a configuração de IDE completa, mas que é completamente personalizável, além de existirem outros *presets*, dentre os quais um modo minimalista no qual somente o editor de texto ocupa a tela, bom para momentos de desenvolvimento intensivo de um arquivo específico.

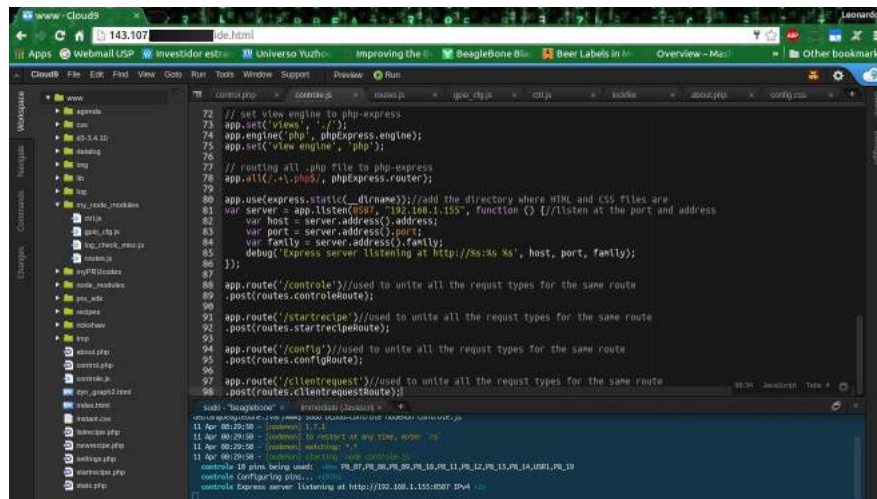


Figura D.1: Ambiente completo do IDE *Cloud9*

D.2 *Git/GitHub*

Git é um sistema de controle de versão distribuído (DVCS) e também é referido como um SCM (gerenciador de códigos-fonte), ou seja, uma ferramenta cuja finalidade é gerenciar o desenvolvimento de *software*. Um dos principais objetivos de um sistema como este é a documentação das mudanças feitas nos arquivos do projeto ao longo do seu desenvolvimento, o que permite facilmente reverter arquivos ou mesmo o projeto inteiro para um estado anterior, comparar mudanças em função do tempo, verificar quem foi a última pessoa a modificar um arquivo e possivelmente introduzir um bug, dentre outros recursos. E o mais importante, executar todas estas funções de maneira leve no que diz respeito a recursos computacionais e fácil do ponto de vista do usuário [52].

Esta ferramenta foi escolhida dentre muitas outras DVCS disponíveis por ser de código-aberto e extremamente popular, não somente pelas suas qualidades mas também pelo fato de ser a DVCS usada para o controle de versão do Kernel do Linux, sendo que o *Git* foi criado pela própria comunidade de manutenção do Kernel, incluindo Linus Torvalds [52]. Embora a referência bibliográfica consultada sobre este assunto possua mais de 500 páginas, demonstrando o nível de complexidade que a ferramenta pode atingir, neste projeto somente o essencial será abordado — o que significa compreender o uso de alguns comandos básicos.

Com relação ao *GitHub*, este nada mais é do que um serviço de hospedagem de repositórios *Git*. Uma de suas grandes vantagens é a popularidade, o que o faz ser uma fonte repleta de recursos de código-aberto. No que diz respeito a este trabalho, dentre suas vantagens estão o fato de o serviço facilitar o processo de *backup* de todo o desenvolvimento realizado

no âmbito de *software*, prover uma estrutura que permite a replicação do projeto com poucos comandos e também o uso deste serviço para a redação da presente monografia sem a preocupação de ter que levar os arquivos fisicamente a diversas localidades ou depender de serviços de armazenamento de propósito geral, que não apresentam as vantagens de um DVCS.

Para configurar uma conta no *GitHub*, o primeiro passo foi a realização do cadastro em <https://github.com>, seguida da criação de um repositório online, conforme ilustrado na figura D.2. Após isto, o repositório foi clonado na BBB e os arquivos relevantes do sistema foram copiados para ele. Observe-se que esta abordagem foi improvisada devido ao fato de arquivos de configuração, dentre outros, encontrarem-se em repositórios específicos do sistema, já que o mais adequado seria que todos os arquivos do projeto fossem criados na pasta do *Git*. Para instalar, configurar o *Git* e clonar o repositório recém criado do *GitHub* para a BBB, foram executados os comandos descritos na caixa de código-fonte D.4.

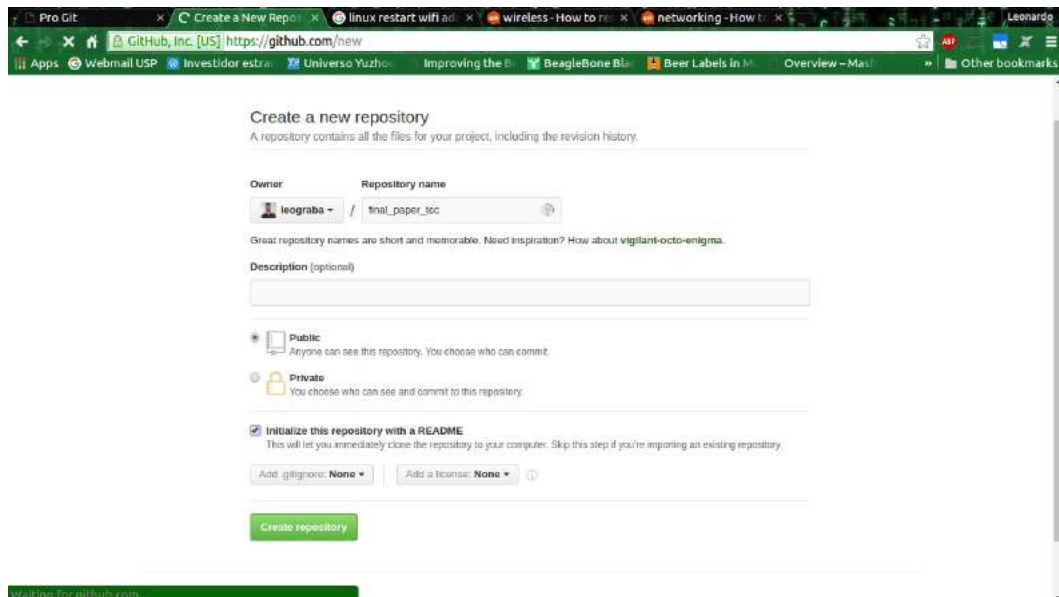


Figura D.2: Criação de repositório no *GitHub*

```

1 sudo apt-get install git
2 git configure --global user.name "Nome"
3 git configure --global user.email email@exemplo.com
4 cd ~
5 git clone https://github.com/leograba/final_paper_tcc.git

```

Código-fonte D.4: Instalação do *Git* e cópia do repositório do *GitHub* para a BBB

Após a clonagem do diretório *final_paper_tcc* do *GitHub* para a BBB, todos os arquivos relevantes ao projeto foram copiados para ele e, subsequentemente, adicionados ao *Git*: sempre que um arquivo é adicionado ou modificado em um repositório *Git*, ele é ignorado a menos que seja executado o comando *git add arquivo* — isto permite flexibilidade, principal-

mente com relação a arquivos que não devem ser adicionados ao repositório, como arquivos auxiliares gerados por compiladores, por exemplo. Após cada mudança, para torná-la efetiva é preciso executar o comando `git commit -m "mensagem para registro"`, que além da data e hora da modificação, permite adicionar uma mensagem para identificar o que foi modificado desde o último `commit`. Por fim, foi preciso enviar as mudanças do repositório local para o *GitHub*. O processo de adicionar arquivos novos/modificados, fazer `commit` nas mudanças e atualizar a origem remota está descrito no código-fonte D.5 e foi usado sempre que mudanças foram feitas ao projeto, desde implementações de funcionalidades a correções de *bugs*. Um exemplo de `commit` após mudanças incrementais é apresentado na figura D.3.

```
1 git add .
2 git commit "Primeiro commit"
3 git push origin master
```

Código-fonte D.5: Primeiro `git commit`

```
debian@beaglebone:~/final_paper_tcc$ git status
# On branch develop
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   www/controle.js
#
no changes added to commit (use "git add" and/or "git commit -a")
debian@beaglebone:~/final_paper_tcc$ git add www/controle.js
debian@beaglebone:~/final_paper_tcc$ git commit -m "Melhorias incrementais ao controle da brassagem"
[develop d149273] Melhorias incrementais ao controle da brassagem
 1 file changed, 2 insertions(+), 2 deletions(-)
debian@beaglebone:~/final_paper_tcc$ git push origin develop
Username for 'https://github.com': leogveiga@gmail.com
Password for 'https://leogveiga@gmail.com@github.com':
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 397 bytes, done.
Total 4 (delta 3), reused 0 (delta 0)
To https://github.com/leogveiga/final_paper_tcc.git
   deae892..d149273  develop -> develop
debian@beaglebone:~/final_paper_tcc$
```

Figura D.3: Fazendo `git commit` após mudanças incrementais

D.3 Considerações finais sobre o uso das ferramentas Git e GitHub

Nesta seção são descritas as considerações finais referentes ao uso da ferramenta de controle de versão distribuído, *Git* em conjunto com a plataforma *GitHub*, que se encontra disponível no endereço https://github.com/leogveiga/final_paper_tcc. A figura D.4 apresenta a página inicial do projeto no *GitHub*, ou seja, o *branch master*.

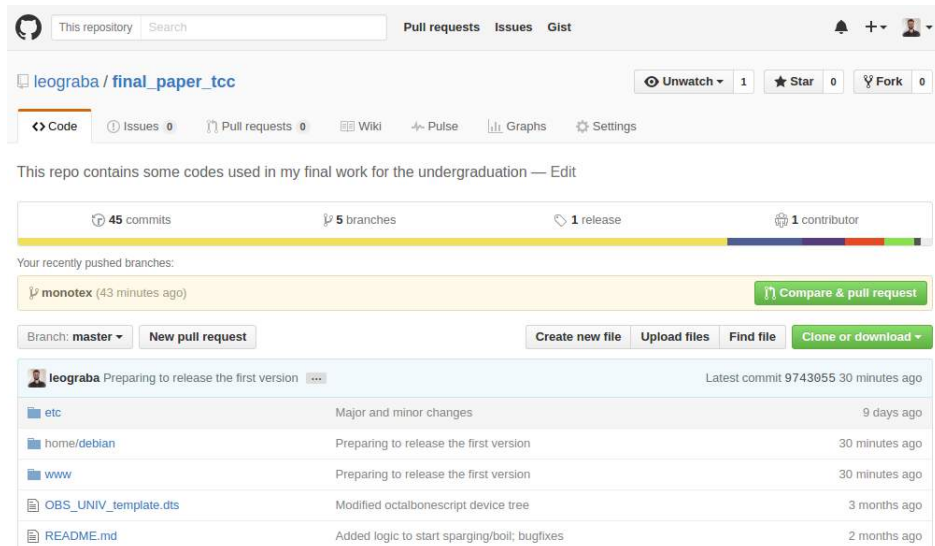


Figura D.4: Ramo *master* do projeto no github

Observa-se que foram feitos 45 *commits*, ou seja, foram submetidas 45 mudanças de algum aspecto do projeto, desde o início do uso da ferramenta *Git*. A figura D.5 apresenta detalhes de 5 *commits* realizados nos dias 26 e 27 de março de 2016. Também pode ser observado na figura D.4 que foram criados 5 *branches* no total, sendo estes:

- **master** — o ramo principal do projeto. Isto significou, para este projeto, que este ramo conteve sempre os últimos códigos testados e funcionando.
- **develop** — o ramo de desenvolvimento. Neste foi mantido o último status do projeto, ou seja, o mais avançado em termos de desenvolvimento, mas que por vezes estava incompleto ou sem ser testado.
- **legacy** — ramo no qual foi adicionado código legado de uma aplicação anterior desenvolvida para a plataforma Raspberry Pi.
- **minimal** — neste ramo foi adicionado somente o código essencial para o funcionamento da aplicação, com o objetivo de portá-la do SBC de desenvolvimento/aprendizado BBB para a solução de SBC industrial customizada Toradex Colibri VF61 + placa base Viola. Esta portabilidade foi abandonada em sua fase inicial e, portanto, o ramo ficou inativo.
- **monotex** — ramo criado para conter somente arquivos utilizados para a construção da monografia, em sua maioria arquivos *.tex* e figuras.

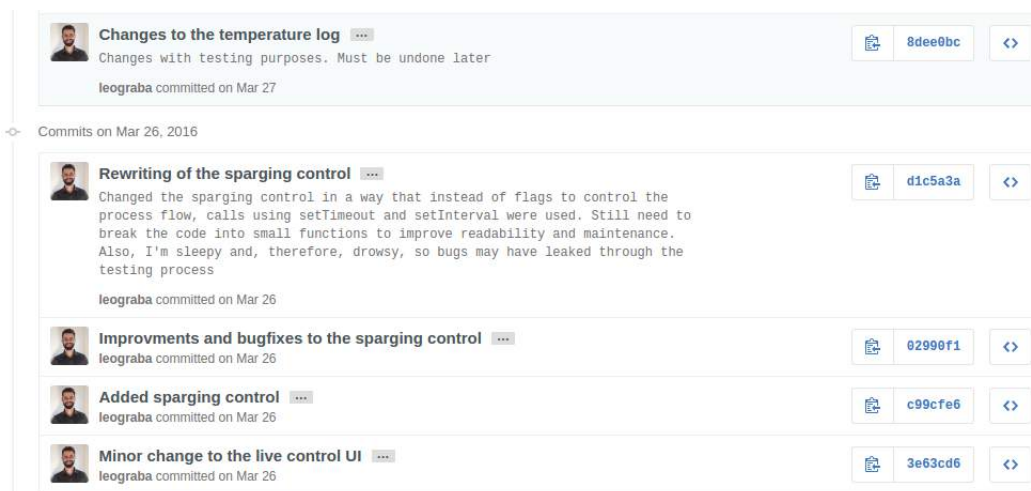


Figura D.5: Registro de contribuições para o projeto na plataforma *GitHub*

Dentre os resultados específicos do uso da ferramenta *GitHub* estão o gráfico do número de contribuições em função do tempo, apresentado na figura D.6. Outra estatística interessante é o gráfico de dispersão da densidade de contribuições em função de hora do dia e dia da semana, apresentado na figura D.7 e referido como *punchcard* (bater o cartão ou bater o ponto, em tradução livre). Este indicou em quais dias e horários da semana o ritmo de trabalho foi mais ou menos intenso: observou-se que desde sexta-feira à noite até segunda-feira à noite foram os dias da semana de desenvolvimento mais ativo e, também, que independente de dia da semana, os horários de maior ritmo de trabalho ficaram concentrados entre 17:00h e 01:00h.

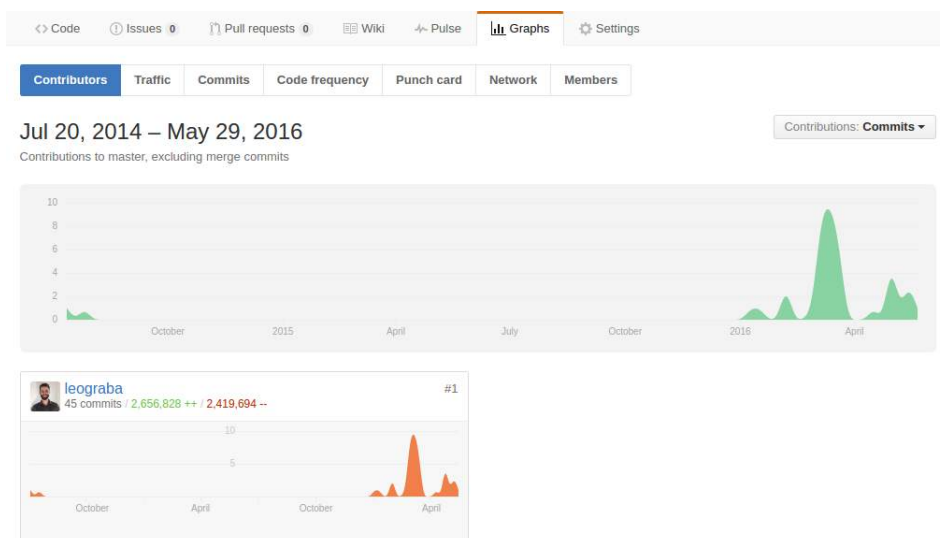


Figura D.6: Gráfico das contribuições para o projeto na plataforma *GitHub* em função do tempo



Figura D.7: Distribuição das contribuições para o projeto no *GitHub* ao longo da semana

Com relação ao sistema de controle de versão *Git* e o serviço online *GitHub*, foi observado o poder que estas ferramentas proporcionam à documentação e manutenção de um projeto que envolve *software*. O simples fato de documentar cada mudança feita ao código-fonte de maneira simples, no nível de detalhamento que o usuário considerar adequado, já justificou o seu uso: este nível de documentação e sua facilidade são um dos pontos essenciais para a facilidade de documentação de código, pois o desenvolvedor sabe o que foi modificado e quando, gastando um mínimo esforço.

A possibilidade de desenvolver esta monografia empregando o *GitHub* como um meio de *backup* e de ter os arquivos necessários disponíveis em qualquer lugar também foi um dos pontos fortes observados que justificou o uso da ferramenta — além de que os registros incrementais ajudaram a recuperar o raciocínio em períodos nos quais a escrita foi deixada em segundo plano em função do desenvolvimento do projeto.

Embora tenha sido lógico o resultado estatístico do *punchcard*, apresentado na seção D.3 — em função de tarefas de maior prioridade executadas durante os dias úteis da semana, foi observado que o desenvolvimento do TCC se deu nos horários livres — é possível extrapolar o uso desta ferramenta estatística no monitoramento de uma equipe de desenvolvimento de software, que não precisa necessariamente trabalhar em um mesmo local físico.

Por fim, com relação ao *Git*, durante alguns momentos do projeto, foi constatado que apesar de a ferramenta ser excelente para o desenvolvimento de *software*, em última instância ela reflete a disciplina e o hábito do programador no uso desta. Isto foi notado em momentos nos quais mudanças simples, mas importantes para o código-fonte, eram ignoradas no sentido de

atualizá-las no *Git*. Outra questão importante com relação ao uso que se faz desta ferramenta é que, ao trabalhar em uma equipe, provavelmente seja melhor combinar com os colaboradores qual o nível de detalhamento desejado para a documentação e quais as técnicas de uso, e.g. criar uma ramificação para consertar um determinado bug ou somente para implementar uma nova funcionalidade, dentre outras possibilidades.

Apêndice E

Scripts Python

E.1 Módulo com as funções para *log* da temperatura

```

1 #!/usr/bin/env python
2  #-*- coding: utf-8 -*-
3
4  #importa os módulos que serão usados no script
5  import time
6  import os.path
7  import graficos_png as graph
8  import datetime
9  import wiringpi2 as wpi
10
11 def tread():
12     """lê temperatura do DS18B20 e retorna em celsius"""
13     tfile = open("/sys/bus/w1/devices/28-000004ee8ded/w1_slave")
14     tfile.readline()
15     temp_raw = tfile.readline()
16     tfile.close()
17     temp_raw = temp_raw.split(" ")[9]
18     temp_raw = float(temp_raw[2:])
19     return temp_raw/1000
20
21 def tprint_all(tcelsius):
22     """imprime a temperatura em diversas escalas no formato HTML"""
23     print "Temperatura:<br />",
24     print "%.2f &degC<br />" % tcelsius, #celsius
25     print "%.2f &degF<br />" % (tcelsius*1.8+32), #fahrenheit
26     print "%.2f K<br />" % (tcelsius+273.15), #kelvin
27     print "%.2f &degR<br />" % (tcelsius*1.8+32+459.67), #rankine
28
29 def tprint_all_terminal(tcelsius):
30     """imprime a temperatura em diversas escalas formatado para o terminal"""
31     print "Temperatura:"
32     print "%.2f" % tcelsius, u"\u00B0C"#celsius
33     print "%.2f" % (tcelsius*1.8+32), u"\u00B0F"#fahrenheit
34     print "%.2f K" % (tcelsius+273.15) #kelvin
35     print "%.2f" % (tcelsius*1.8+32+459.67), u"\u00B0R"#rankine
36
37 def tlog(file = "/var/www/datalog/default.csv"):
38     """salva temperatura e Unix Time em .csv """
39     #arquivo padrão CSV com cabeçalho
40
41
42     #file = raw_input("digite o nome do arquivo e.g. log-data\n")
43     #file += ".log"
44     #tsample = float(raw_input("digite tempo de amostragem em segundos: "))
45
46
47     #os comandos acima foram ignorados para poder usar o nohup (que nao recebe input)
48     tsample = 0.2202 #amostra a cada x segundos
49     #amostras = 5000#numero de amostras a serem coletadas
50     buff_temp = tread()#guarda o último valor lido
51     exist = os.path.isfile(file)
52     #buffer = open(file,"a")#mesma funcao da linha abaixo, porem menos recomendada
53     with open(file, 'a', 1) as log:#desse jeito, o arquivo será fechado
54         #mesmo que haja uma excessao, diferente de usar file.close()
55         #o arg. 1 indica que o buffer antes de escrever no arquivo eh 1 linha

```

```

56 if exist is False:#se vai criar o arquivo agora
57     log.write("temperatura,data\n")
58 while True: #loop infinito
59     #while amostras > 0:#coleta o numero de amostras
60     amostras = amostras - 1#decrementa variavel de controle
61     temp_celsius = tread()
62     epoch = time.time()#lê a data/hora do sistema
63     #nowis = time.ctime(epoch)#converte para string
64     if temp_celsius >= 0:#evita leitura errada
65     #essa leitura errada é intermitente e causa desconhecida
66         log.write("%f,%f\n" % (temp_celsius, epoch))
67         tlog_instant(temp_celsius, epoch)#escreve arquivo com ultima leitura
68         #graph.graph_gen()# atualiza gráfico da temperatura
69         #escreve temperatura e data/hora no .txt
70         time.sleep(tsample)#espera n segundos
71         print "registrando temperatura!",temp_celsius
72
73
74 def tlog_instant(temperature, epoch, file = "/var/www/datalog/instant.csv"):
75     """salva última temperatura e Unix Time em arquivo .csv"""
76     with open(file, 'w', 1) as log:#sobrescreve o arquivo toda vez
77         #temperature = tread()#lê a temperatura
78         #epoch = int(time.time())#lê o Unix Time do sistema
79         log.write("temperatura,data\n")
80         log.write("%f,%f\n" % (temperature, epoch))
81
82 def tlog_test(file = "/var/www/datalog/default.csv"):
83     """salva temperatura e Unix Time em .csv """
84     #arquivo padrão CSV com cabeçalho
85     tsample = 1 #valor de amostragem para teste
86     exist = os.path.isfile(file)
87     with open(file, 'a', 1) as log:#desse jeito, o arquivo será fechado
88         if exist is False:#se vai criar o arquivo agora
89             log.write("temperatura,data\n")
90             temp_celsius = 25.00#somente para teste
91             temp_sparge = 20.00#somente para teste
92             ramp_section = 0#sequencia de funcoes que gera as rampas/degraus
93             while True: #loop infinito
94                 if ramp_section == 0:#aquece medio ate 29 graus
95                     temp_celsius += 0.2
96                     if temp_celsius >= 29:
97                         ramp_section = 1
98                 elif ramp_section == 1:#aquece devagar ate 30 graus
99                     temp_celsius += 0.1
100                     if temp_celsius >= 30:
101                         ramp_section = 2
102                 elif ramp_section == 2:#espera usuario adicionar maltes
103                     time.sleep(15)#30 segundos
104                     ramp_section = 3
105                 elif ramp_section == 3:#aquece medio ate 35 graus
106                     temp_celsius += 0.2
107                     if temp_celsius >= 35:
108                         ramp_section = 4
109                 elif ramp_section == 4:#retrai aquecimento ate 33 graus
110                     temp_celsius -= 0.2
111                     if temp_celsius <= 33:
112                         ramp_section = 5
113                 elif ramp_section == 5:#degrau 1, aquece sparge durante 1 minuto
114                     temp_sparge += 0.5
115                     if temp_sparge >= 50:
116                         ramp_section = 6
117                 elif ramp_section == 6:#aquece rapido ate 42 graus
118                     temp_celsius += 0.3
119                     if temp_celsius >= 42:
120                         ramp_section = 7
121                 elif ramp_section == 7:#aquece medio ate 59 graus
122                     temp_celsius += 0.2
123                     if temp_celsius >= 59:
124                         ramp_section = 8
125                 elif ramp_section == 8:#aquece devagar ate 60 graus
126                     temp_celsius += 0.1
127                     if temp_celsius >= 60:
128                         ramp_section = 9
129                 else:#temperatura de fervura
130                     if temp_sparge <= 98.5:
131                         temp_sparge += 0.3
132             epoch = time.time()#lê a data/hora do sistema
133             log.write("%f,%f\n" % (temp_celsius, epoch))
134             tlog_instant(temp_celsius, epoch)#escreve arquivo com ultima leitura
135             tlog_instant(temp_sparge, epoch, "/var/www/datalog/instant_bk.csv")#somente para teste por enquanto
136             #escreve temperatura e data/hora no .txt
137             time.sleep(tsample)#espera n segundos
138             print "registrando temperatura!",temp_celsius,temp_sparge

```

Código-fonte E.1: Módulo com as funções para *log* da temperatura

E.2 Script para plotagem de gráfico

```

1  #exemplo traduzido e adaptado de
2  #matplotlib.org/examples/pylab_examples/multipage_pdf.html
3  import time
4  import matplotlib
5  matplotlib.use('AGG') #muda a output para um renderer ao inves de backend
6  import numpy as np
7  import matplotlib.pyplot as plt
8  from scipy import interpolate
9  #from matplotlib.backends.backend_pdf import PdfPages
10
11 def graph_gen(file = '/var/www/default.csv', graph = '/var/www/tplot.png'):
12     #le arquivo com as temperaturas armazenadas
13     #file = '/var/www/default.csv'
14     tvalues=[] #declara lista das temperaturas
15     ttimes=[] #declara lista dos tempos
16     x_scale=[] #declara lista do eixo-x
17     #legenda=[] #declara lista de legendas
18     title=""
19     last_line=['', '']
20     control=0 #variavel de controle do loop
21     with open(file, 'r') as tfile:
22         for line in tfile: #le linha por linha o log da temperatura
23             if control != 0: #se esta lendo a primeira linha
24                 tvalues.append(line.split(',')[0]) #separa o valor da temperatura
25                 ttimes.append(line.split(',')[1]) #separa o epoch
26                 last_line[0] = last_line[1] #ultima linha salva
27                 last_line[1] = line.split(',')[1] #ultima linha lida-data/hora
28                 if control == 1: #pega a data/hora da primeira linha
29                     title = line.split(',')[1] #esta em epoch time
30                     control += 1
31                 #ttimes.append(line[23:31]) #separa a hora da leitura
32                 title = time.ctime(float(title)) # de epoch para string formatada
33                 title += ' - ' #separa tinicio e tfim
34                 title += time.ctime(float(last_line[1])) #pega a data/hora da ultima linha
35                 #nao da pra fazer slice do tipo [n:], porque nao exclui o \n do fim da linha
36
37     #plota o grafico da temperatura
38     fig = plt.figure(figsize=(9, 4)) #, dpi=300)
39     ax = fig.add_subplot(111) #subplot p/ mudar cor de fundo
40     ax.set_axis_bgcolor('#BEC5C2') #muda cor do fundo do plot
41     plt.grid() #plota o grafico com grid ligado
42     #se tratando de salvar a figura, o dpi so faz diferenca no savefig
43
44     #ajusta a escala do tempo baseado no tempo total de amostragem dos dados
45     size_x = int(float(ttimes[len(ttimes)-1]) - float(ttimes[0])) #tempo final menos tempo inicial
46     if size_x < 7200: #ate duas horas, escala em minutos
47         for tms in ttimes: #itera pelo array do eixo-x
48             x_scale.append((float(tms) - float(ttimes[0])) / 60) #transforma em minutos
49         plt.xlabel('tempo (minutos)') #imprime label indicando minutos
50     elif size_x < 129600: #ate um dia e meio, escala em horas
51         for tms in ttimes: #itera pelo array do eixo-x
52             x_scale.append((float(tms) - float(ttimes[0])) / (60*60)) #transforma em horas
53         plt.xlabel('tempo (horas)') #imprime label indicando horas
54     elif size_x < 7776000: #ate tres meses, escala em dias
55         for tms in ttimes: #itera pelo array do eixo-x
56             x_scale.append((float(tms) - float(ttimes[0])) / (60*60*24)) #transforma em horas
57         plt.xlabel('tempo (dias)') #imprime label indicando dias
58     else: #caso contrario, escala em meses
59         for tms in ttimes: #itera pelo array do eixo-x
60             x_scale.append((float(tms) - float(ttimes[0])) / (60*60*24*30)) #transforma em horas
61         plt.xlabel(u'tempo (meses -> 30 dias por m\u00EAs)') #imprime label indicando meses
62
63     #gerando interpolacao
64     #tvalues = np.array(tvalues)
65     #x_smooth = np.linspace(x_scale.min(), x_scale.max(), 10)
66     #y_smooth = spline(x_scale, tvalues, x_smooth)
67     #tck = interpolate.splrep(x_scale, tvalues)
68     #print tck
69
70     #plotando os graficos
71     plt.plot(x_scale, tvalues, '--', color='b')
72     #plt.plot(ttimes, tvalues, '--', color='b')
73
74     #trabalha as legendas
75     # tiks = ax.get_xticks().tolist() #le o eixo das legendas
76     # size_tks = len(tiks) #ve quantas posicoes tem pra legenda
77     # for lgd in tiks: #itera essas posicoes
78     #     legenda.append(x_scale[size_tks])
79     # tiks[:] = ['bla', 'ble', 'bli', 'blo', 'blu', 'bal', 'bel', 'bil', 'bol', 'bul']
80     # print tiks[1]
81     # ax.set_xticklabels(tiks)
82     # ax.axis_date()

```

```
84 | #plt.tight_layout(pad=4.0, w_pad=0.5, h_pad=1.0)
85 | #plt.plot(x_smooth, y_smooth, '-', color='r')
86 | plt.title(title)
87 | plt.ylabel(u'temperatura(\u00B0C)')
88 | plt.savefig(graph, dpi=150, facecolor='none')#='#97D9CC')
89 | plt.close()
90 |
91 | while True:
92 |     delta_t = time.time()#guarda tempo inicial
93 |     graph_gen()
94 |     delta_t = time.time() - delta_t#calcula tempo total
95 |     print 'tempo para gerar grafico (s) = %.3f' %delta_t
96 |     time.sleep(300)
```

Código-fonte E.2: Script para plotagem de gráfico

Apêndice F

Códigos-fonte Node.js

F.1 Código-fonte raiz da aplicação em Node.js

```

1  "use strict";
2  //load modules
3  var bodyParser = require('body-parser');
4  var express = require("express");
5  var app = express();
6  var debug = require('debug')('controle');
7  var pruu = require("node-pru-extended");
8  var fs = require("fs");
9  var exec = require("child_process").exec;
10 var phpExpress = require('php-express')({ // assumes php is in your PATH
11 // must specify options hash even if no options provided!
12   binPath: 'php'
13 });
14 //load modules written for this application
15 var gpioCfg = require('./my_node_modules/gpio_cfg.js');
16 var ctrl = require('./my_node_modules/ctrl.js');
17 var log = require('./my_node_modules/log_check_misc.js');
18 var routes = require('./my_node_modules/routes.js');
19
20 //global variables that should also be saved to a backup file periodically
21 global.environmentVariables = {
22   warn: "", //if not empty holds some warning message
23   code: "", //tells the same as msg, but as an index, easier to check programatically
24   tmpMT: "", //mash tun temperature
25   tmpMTsetp: "", //mash tun current setpoint
26   tmpBK: "", //brewing kettle temperature, also the "hot liquor tank" for sparging
27   tmpBKsetp: "", //brewing kettle/hot liquor tank current setpoint
28   timeLeft: "", //helping variable to tell the client the time left for the step rests or the boil, etc
29   readyForNextStep: false, //set whenever the system is ready for the next step
30   auto: true, //whether the process is running automatically or there is human intervention
31   processFail: false, //flag is set if the process fails irreversibly
32   msg: "", //holds some explanatory message
33   timestamps: { //notable timestamps
34     start: "", //epoch time of the first request to start a recipe
35     startHeating: "", //epoch time of the start to heat the mash water
36     finishHeating: "", //epoch for the finish of the heating of mash water
37     //start of the ramps
38     sRamp0: "", sRamp1: "", sRamp2: "", sRamp3: "", sRamp4: "", sRamp5: "", sRamp6: "", sRamp7: "",
39     //finish of the ramps
40     fRamp0: "", fRamp1: "", fRamp2: "", fRamp3: "", fRamp4: "", fRamp5: "", fRamp6: "", fRamp7: "",
41     startDrain: "", //start of the sparging process, when the mash is parcially drained to the BK
42     startSparge: "", //here the recirculation pump starts working
43     heatingBoil: "", //started to heat the wort after sparging
44     boilStart: "", //time when temperature is near enough boiling (>96°C)
45     boilFinishScheduled: "", //time when the boil is scheduled to finish
46     curr: "", //epoch time of the current variables state
47   },
48   ioStatus: gpioCfg.all_io, //also records the IO status
49   okToStart: false, //true if a recipe is ok enough to start a production
50   recipe: "" //recipe name
51 };
52 /*
53 //Trying to access and/or use the PRU
54 console.log(pruu);
55 pruu.init(); //initialize the communications

```

```

56 | //pru.loadDatafile(0,"/var/www/myPRUcodes/data.bin");
57 | pru.execute(0,"/var/www/myPRUcodes/pisca_text.bin",0);
58 | //console.log(pru.getSharedRAM());
59 | pru.exit(0);//force the PRU code to terminate
60 | */
61 |
62 | // Pin configuration
63 | debug(gpioCfg.ioStatus.total + " pins being used: ", gpioCfg.all_io_pins.toString());
64 | debug("Configuring pins...");
65 | gpioCfg.pinsConfiguration();
66 |
67 | //Using Express to create a server
68 | app.use(bodyParser.urlencoded({//to support URL-encoded bodies, MUST come before routing
69 |   extended: true
70 | }));
71 |
72 | // set view engine to php-express
73 | app.set('views', './');
74 | app.engine('php', phpExpress.engine);
75 | app.set('view engine', 'php');
76 |
77 | // routing all .php file to php-express
78 | app.all(/.+\.php$/, phpExpress.router);
79 |
80 | app.use(express.static(__dirname));//add the directory where HTML and CSS files are
81 | var server = app.listen(8587, "192.168.1.155", function () { //listen at the port and address
82 |   var host = server.address().address;
83 |   var port = server.address().port;
84 |   var family = server.address().family;
85 |   debug('Express server listening at http://%s:%s %s', host, port, family);
86 | });
87 |
88 | app.route('/controle')//used to unite all the request types for the same route
89 | .post(routes.controleRoute);
90 |
91 | app.route('/startrecipe')//used to unite all the request types for the same route
92 | .post(routes.startrecipeRoute);
93 |
94 | app.route('/config')//used to unite all the request types for the same route
95 | .post(routes.configRoute);
96 |
97 | app.route('/clientrequest')//used to unite all the request types for the same route
98 | .post(routes.clientrequestRoute);

```

Código-fonte F.1: Código-fonte raiz da aplicação em Node.js

F.2 Módulo de gerenciamento de GPIO e PWM

```

1 | "use strict";
2 |
3 | //var m = require('./controle.js');//not sure it is needed to add the requester to share variables
4 | var b = require('octalbonescript');
5 | var debug = require('debug')('gpio');
6 |
7 | //Variables
8 | var led = module.exports.led = new PinObjectIO("USR1");
9 | var mash_pump = module.exports.mash_pump = new PinObjectIO("P8_07");
10 | var boil_pump = module.exports.boil_pump = new PinObjectIO("P8_08");
11 | var pumps = module.exports.pumps = {mash_pump:mash_pump, boil_pump:boil_pump};
12 |
13 | var mash_valve = module.exports.mash_valve = new PinObjectIO("P8_09");
14 | var boil_valve = module.exports.boil_valve = new PinObjectIO("P8_10");
15 | var chill_valve = module.exports.chill_valve = new PinObjectIO("P8_11");
16 | var water_valve = module.exports.water_valve = new PinObjectIO("P8_12");
17 | var valves = module.exports.valves = { mash_valve:mash_valve, boil_valve:boil_valve,
18 |   chill_valve:chill_valve, water_valve:water_valve};
19 |
20 | var mash_heat = module.exports.mash_heat = new PinObjectIO("P8_13");
21 | var boil_heat = module.exports.boil_heat = new PinObjectIO("P8_14");
22 | var heaters = module.exports.heaters = {mash_heat:mash_heat, boil_heat:boil_heat};
23 |
24 | //for servo, use 0.0325 < duty < 0.11 to protect servo integrity
25 | var servo_pwm = module.exports.servo_pwm = {id:"P8_19", state:{duty:0.11, freq:50}, cfg:b.ANALOG_OUTPUT}; //state of PWM is
   |   duty-cycle and frequency
26 | //var servo_pwm = {id:"P8_19", state:{duty:0.11, freq:60}}; //this is for the SSR test
27 |
28 | var all_io = module.exports.all_io = collect(pumps,valves,heaters,{led:led},{servo_pwm:servo_pwm});
29 | //debug(all_io);
30 | var all_io_objects = module.exports.all_io_objects = Object.keys(all_io);//get all the keys, because cannot access object
   |   by index, e.g all_io[2]
31 | var all_io_pins = module.exports.all_io_pins = [];//all the pins used as an array

```

```

32 for (var i = 0; i < all_io_objects.length; i++){//get the pins one by one
33   all_io_pins[i] = all_io[all_io_objects[i]].id;//and add to the array
34 }
35
36 var ioStatus = module.exports.ioStatus = {cfgok:0, gpio:0, pwm:0, analog:0, interrupt:0, total:all_io_objects.length, //
  helping object
37   //some functions just to exercise the use of methods
38   newGpio: function(){ this.cfgok++; this.gpio++;},
39   newPwm: function(){ this.cfgok++; this.pwm++;},
40   newAnalog: function(){ this.cfgok++; this.analog++;},
41   newInterrupt: function(){ this.cfgok++; this.analog++;},
42   gpio2pwm: function(){ this.gpio--; this.pwm++;},
43   gpio2interrupt: function(){ this.gpio--; this.interrupt++;},
44   pwm2gpio: function(){ this.pwm--; this.gpio++;},
45   pwm2interrupt: function(){ this.pwm--; this.interrupt++;},
46   interrupt2gpio: function(){ this.interrupt--; this.gpio++;},
47   interrupt2pwm: function(){ this.interrupt--; this.pwm++;}
48 };
49
50 //functions
51 module.exports.changeStatusIO = function (pin, val){
52   if(val == "true"){//if button is checked
53     all_io[pin].state = b.HIGH;//turn corresponding pin HIGH
54     b.digitalWriteSync(all_io[pin].id, all_io[pin].state);
55     debug("Pin " + pin + " turned HIGH");
56   }
57   else if(val == "false"){//if button is unchecked
58     all_io[pin].state = b.LOW;//turn corresponding pin LOW
59     b.digitalWriteSync(all_io[pin].id, all_io[pin].state);
60     debug("Pin " + pin + " turned LOW");
61   }
62   else{//if it is not a button
63     if(pin == "servo_pwm"){//check if this is the PWM
64       servo_pwm.state.duty = 0.0325 + (0.0775/180)*val;//turn degree to duty-cycle
65       b.analogWrite(servo_pwm.id, servo_pwm.state.duty, servo_pwm.state.freq, function(err_wr){//set PWM
66         if(err_wr){//if anything goes wrong
67           debug(err_wr);//print the error
68         }
69         else{//otherwise
70           debug("Servo angle set to: " + val);//print the angle the PWM was set to
71         }
72       });
73     }
74   }
75 };
76
77 module.exports.getSystemStatus = function (){
78   var all_io_status = {};//object with all the pin pairs key:value
79   for(var i = 0; i < all_io_objects.length; i++){//get the pair key:value pin by pin
80     all_io_status[all_io_objects[i]] = all_io[all_io_objects[i]].state;
81   }
82   return(all_io_status);
83 };
84
85 // I/O pins
86 function PinObjectIO(pinId){//function to create pin object, should receive pin ID
87   if(pinId){//if the variable is passed to function or not empty
88     this.id = pinId; this.state = b.LOW; this.cfg = b.OUTPUT;
89   }
90   else{//if no variable is passed or passed empty
91     debug("No variable passed to create pin object");
92   }
93 }
94
95 module.exports.pinsConfiguration = function (){
96   for (var i = 0; i < all_io_objects.length; i++){//set all pins as outputs
97     (function (pinIndex){//need to create a scope for the current pin variable, because it is asynchronous
98       debug(" pin passed is " + this + "; pin index passed is " + pinIndex);//"this" is the first parameter passed ->
99       the current pin
100       b.pinMode(this, all_io[all_io_objects[pinIndex]].cfg, function(err, pin){//configure and callback function
101         if(err){//if by the end of executing pinMode function, there is an error
102           console.error(err.message);//then the error is printed
103         }
104         else{//initial state, everyone LOW because HIGH state means ON
105           if(all_io[all_io_objects[pinIndex]].cfg == b.OUTPUT){//if pin is configured as digital output
106             debug(' pin ' + pin + ' ready[OUTPUT], ' + (ioStatus.cfgok+1) + "/" + ioStatus.total + "pins configured");
107             b.digitalWriteSync(pin, all_io[all_io_objects[pinIndex]].state);//state is LOW because of initial values
108             //ioStatus.cfgok++;
109             ioStatus.newGpio();//indicates one more pin is configured as gpio
110             if(ioStatus.cfgok == ioStatus.total){//if all pins are already configured
111               //interval = setInterval(function(){ioTest()}, 5000);//then start the blinking test very slow
112               debug(ioStatus.gpio + " pins as GPIO, " + ioStatus.pwm + " as PWM, " +
113                 ioStatus.analog + " as ANALOG, " + ioStatus.interrupt + " as INTERRUPT");
114             }
115           }
116           else if(all_io[all_io_objects[pinIndex]].cfg == b.ANALOG_OUTPUT){//if pin is configured as PWM
117             debug(' pin ' + pin + ' ready[PWM], ' + (ioStatus.cfgok+1) + "/" + ioStatus.total + "pins configured");
118             b.analogWrite(servo_pwm.id, servo_pwm.state.duty, servo_pwm.state.freq, function(err_wr){//try to configure

```

```

117         if(err_wr){//if error
118             debug(err_wr);
119         }
120         else{//if ok
121             ioStatus.newPwm();//indicates one more pin is configured as PWM
122             if(ioStatus.cfgok == ioStatus.total){//if all pins are already configured
123                 //interval = setInterval(function(){ioTest()}, 5000);//then start the blinking test very slow
124                 debug(ioStatus.gpio + " pins as GPIO, " + ioStatus.pwm + " as PWM, " +
125                     ioStatus.analog + " as ANALOG, " + ioStatus.interrupt + " as INTERRUPT");
126             }
127         }
128     });
129 }
130 }
131
132 //everyone is output
133 //get all the object keys as vector and use it to access all the object keys
134 }).call(all_io[all_io_objects[i]].id,i);//passes pin as "this" and index as pinIndex, only works properly in strict
    mode
135 }
136 };
137
138 module.exports.ioTest = function () {
139     /*This function do some blinking thing, it should be called inside setInterval(),like:
140     setInterval(ioTest, interval);//period = interval*IOpins = 500ms*9 = 4.5s*/
141     var on_count = 0;//number of io turned b.HIGH
142     var last_on; //number of last io b.HIGH
143
144     for(var i = 0; i < all_io_objects.length; i++){//check all
145         if(all_io[all_io_objects[i]].state == b.HIGH){//if HIGH
146             on_count++;//one more io HIGH
147             last_on = i;//this is the last io HIGH
148         }
149     }
150     if(on_count == 1){//if only one IO HIGH
151         if(last_on == 8){//if last pin is the one HIGH
152             //debug('Activating pin 0');
153             all_io[all_io_objects[last_on]].state = b.LOW;//turn it LOW
154             all_io[all_io_objects[0]].state = b.HIGH;//and next one HIGH, which means the first
155             b.digitalWriteSync(all_io[all_io_objects[last_on]].id, all_io[all_io_objects[last_on]].state);
156             b.digitalWriteSync(all_io[all_io_objects[0]].id, all_io[all_io_objects[0]].state);
157         }
158         else{//if any pin but the last is the one HIGH
159             //debug('Activating pin ' + (last_on+1));
160             all_io[all_io_objects[last_on]].state = b.LOW;//turn it LOW
161             all_io[all_io_objects[last_on+1]].state = b.HIGH;//and next one HIGH
162             b.digitalWriteSync(all_io[all_io_objects[last_on]].id, all_io[all_io_objects[last_on]].state);
163             b.digitalWriteSync(all_io[all_io_objects[last_on+1]].id, all_io[all_io_objects[last_on+1]].state);
164         }
165     }
166     else{//if there is more than one IO HIGH
167         for(i = 1; i < all_io_objects.length; i++){
168             all_io[all_io_objects[i]].state = b.LOW;//turn all but first LOW
169             b.digitalWriteSync(all_io[all_io_objects[i]].id, all_io[all_io_objects[i]].state);
170         }
171         all_io[all_io_objects[0]].state = b.HIGH;//turn first HIGH
172         b.digitalWriteSync(all_io[all_io_objects[0]].id, all_io[all_io_objects[0]].state);
173     }
174 };
175
176 function collect() {function concat objects
177     var ret = {};//the new object
178     var len = arguments.length;//the total number of objects passed to collect
179     for (var i=0; i<len; i++) {do it for every object passed
180         for (var p in arguments[i]) {iterate the i-eth object passed
181             if (arguments[i].hasOwnProperty(p)) {whenever there is a property
182                 ret[p] = arguments[i][p];add the property to the new object
183             }
184         }
185     }
186     return ret;
187 }

```

Código-fonte F.2: Módulo de gerenciamento de GPIO e PWM

F.3 Módulo de roteamento do servidor web

```

1  'use strict';
2
3  var debug = require('debug')('routes');
4  var exec = require('child_process').exec;

```



```

5 | var fs = require('fs');
6 | var gpioCfg = require('./gpio_cfg.js');
7 | var log = require('./log_check_misc.js');
8 | var ctrl = require('./ctrl.js');
9 |
10 | module.exports.controleRoute = function (req, res) {
11 |     var serverResponse = {command:req.body.command, btn:req.body.btn, val:req.body.val};
12 |     var command = req.body.command, pin = req.body.btn, val = req.body.val;
13 |
14 |     if(command == "pinSwitch"){//if command passed by client is to switch a pin configuration
15 |         gpioCfg.changeStatusIO(pin, val);//change the IO status according to the data recieved
16 |         res.send(serverResponse);//echo the recieved data to the server
17 |     }
18 |     else if(command == "getStatus"){
19 |         global.environmentVariables.ioStatus = gpioCfg.all_io;//all of the pins status
20 |         res.send(global.environmentVariables);
21 |     }
22 | };
23 |
24 | module.exports.startrecipeRoute = function (req, res) {
25 |     var serverResponse = {resp:"success"};
26 |     var command = req.body.command;
27 |     debug("Command: " + command);
28 |     var recipesPath = "./recipes";//path to the recipes directory
29 |     var bkpFile = "./datalog/backup.log";//path to the backup logs file
30 |     var lockFile = "./datalog/lockfile";//file that tells if recipe is running
31 |     var recipeName ;
32 |     if(command == "getRecipes"){//if command passed by client is to get the recipe names
33 |         log.sendRecipeNames(recipesPath, res);//get it and return to the client
34 |     }
35 |     else if(command == "startRequest"){//if the client wants to start a recipe
36 |         recipeName = req.body.recipe + ".recipe";//get the recipe name
37 |         log.checkRecipeIntegrity(recipeName, recipesPath, res);
38 |     }
39 |     else if(command == "startRecipe"){//start the recipe; should only be requested after the "startRequest" command
40 |         recipeName = req.body.recipe + ".recipe";//recipe name sent from the client
41 |         ctrl.startMashingProcess(recipeName, res, lockFile, recipesPath, function(err, nextStep){
42 |             if(err){
43 |                 debug("Ops, something wrong. It's so annoying that the error isn't explained here, isn't it?");
44 |                 return;
45 |             }
46 |             debug(nextStep);//here the next step, i.e. controlling the mash steps, should be called
47 |         });//start the production
48 |     }
49 |     else if(command == "InProgress"){//checks if there is a recipe running
50 |         fs.readFile(lockFile, function(err, data){
51 |             if(err){
52 |                 serverResponse.resp = "false";//assumes the error means no recipe is in progress
53 |                 res.send(serverResponse);
54 |             }
55 |             else{
56 |                 if(+data == 1){//if there is a recipe in progress
57 |                     serverResponse.resp = "true";
58 |                     fs.readFile(bkpFile, "utf-8", function(err, data){
59 |                         if(err){//if contents of log could not be retrieved
60 |                             serverResponse.recipe = "unknown";//not the best thing to do
61 |                             res.send(serverResponse);
62 |                         }
63 |                         else{
64 |                             var lines = data.trim().split('\n');//separate line by line
65 |                             debug(lines.slice(-1)[0]);
66 |                             var lastLine = lines.slice(-1)[0];//get the last log line
67 |                             var properties = JSON.parse(lastLine);//JSON to object
68 |                             properties.recipe = properties.recipe.replace(".recipe", "");//remove the file extension
69 |                             properties.recipe = properties.recipe.replace(/_/g, " ");//replace underlines with spaces
70 |                             serverResponse.recipe = properties.recipe;//send recipe name to the client
71 |                             debug(serverResponse);
72 |                             res.send(serverResponse);
73 |                         }
74 |                     });
75 |                 }
76 |                 else{
77 |                     serverResponse.resp = "false";
78 |                     res.send(serverResponse);
79 |                 }
80 |             }
81 |         });
82 |     }
83 |     else{
84 |         debug("command not found, doing nothing");
85 |         serverResponse.resp = "fail";
86 |         res.send(serverResponse);
87 |     }
88 | };
89 |
90 | module.exports.configRoute = function (req, res) {
91 |     var request = req.body.request;

```

```

92 | var nd = req.body.newdate;
93 | var serverResponse = {datetime:""};
94 | if(request == "datetime"){//send the system time/date to the client
95 |   serverResponse.datetime = Date();
96 |   res.send(serverResponse);
97 | }
98 | else if(request == "setdatetime"){//change the system time/date
99 |   exec('sudo date -s ' + nd + ' | hwclock -w',//execute shell command
100 |   function(error, stdout, stderr) {
101 |     debug('stdout: ' + stdout);
102 |     debug('stderr: ' + stderr);
103 |     if (error !== null) {//if new date wasn't correctly set
104 |       debug('exec error: ' + error);//print error
105 |       serverResponse.datetime = "error";//send fail to client
106 |       res.send(serverResponse);
107 |     }
108 |     else{//if date was correctly set, send ok to client
109 |       debug("date changed to: " + Date(nd));//print
110 |       serverResponse.datetime = "ok";
111 |       res.send(serverResponse);
112 |     }
113 |   });
114 | }
115 | };
116 |
117 | module.exports.clientrequestRoute = function(req, res){
118 |   var command = req.body.command;
119 |   var serverResponse = {resp:"success"};
120 |   debug("Client request: " + command);
121 |   if(command == "startMashRamp"){//set flag to tell ctrl.startMashingProcess() to go to the next step
122 |     global.environmentVariables.readyForNextStep = true;
123 |     res.send(serverResponse);
124 |   }
125 |   else{
126 |     serverResponse.resp = "fail";
127 |     res.send(serverResponse);
128 |   }
129 | };

```

Código-fonte F.3: Módulo de roteamento do servidor web

F.4 Módulo de registros e verificações em geral

```

1 | 'use strict';
2 |
3 | var debug = require('debug')('log');
4 | var fs = require('fs');
5 | var spawn = require("child_process").spawn;
6 |
7 | //variables
8 | var temperatureLogHandler;//variable to handle the python "log.py" script
9 |
10 | module.exports.startTemperatureLogging = function () {
11 |   //starts the python script that logs temperature to file
12 |   //sudo kill $(ps aux | grep log.py | grep -v grep | awk '{print $2}') //to stop the process from terminal
13 |   fs.unlink("./datalog/instant.csv", function(err){//try to delete the old last saved value
14 |     if(err){//this may happen if the old log was already deleted
15 |       debug("file could not be deleted!");//probably nothing to worry about
16 |     }
17 |     //start the logging process anyway
18 |     temperatureLogHandler = spawn("python", ["/home/debian/brewing/log_test.py"]);//starts to log
19 |     debug("Temperature logging started!");
20 |     temperatureLogHandler.on('close', function(code, signal){
21 |       debug("Temperature logging stopped!");
22 |     });
23 |   });
24 | };
25 |
26 | module.exports.stopTemperatureLogging = function(){
27 |   debug("Stopping the temperature logging!");
28 |   temperatureLogHandler.kill('SIGHUP');//kill the process and stop logging
29 | };
30 |
31 | module.exports.getTemperatureReading = function (logFilePath, logReadHandler, callback){
32 |   // var logReadHandler = {
33 |   //   lastValidTemperature: 0,//the last valid temperature reading (defaults to zero, not the better solution)
34 |   //   lastReadingsTimestamp: [0, 0, 0, 0, 0],//last 5 timestamps, to check if readings are going ok
35 |   //   errorCount: 0//counts the file reading errors
36 |   // };
37 |   fs.readFile(logFilePath, "utf-8", function(err, data){//gets the most recent temperature reading

```

```

38 | var parsedData ;//variable to hold the relevant data (temperature and its timestamp)
39 | var callbackError;
40 | if(err){//could not read temperature
41 |   logReadHandler.errorCount++;//increment the errorCount variable
42 |   if(logReadHandler.errorCount >= 180){//180 arbitrarily chosen - it is 5% of 1 hour logging readings
43 |     //oh my god thigs are bad! User, you must take over from here
44 |     callbackError = "Too many temperature reading errors, something may be going awry!";
45 |     debug("Too many temperature reading errors, something may be going awry!");
46 |     callback(err, null);
47 |   }
48 |   else{
49 |     callbackError = "Single wrong reading, nothing to worry yet.";
50 |     debug("Single wrong reading, nothing to worry yet.");
51 |     callback(err, null);
52 |   }
53 | }
54 | else{//if temperature reading from file was successful
55 |   parsedData = data.trim().split('\n').slice(-1)[0].split(',');
56 |   if((parsedData[0] == "temperature") || (typeof parsedData[0] != "string") || isNaN(+parsedData[0])){//wrong reading
57 |     //because of wrong logging
58 |     debug("Wrong temperature reading: " + parsedData[0]);
59 |     parsedData[0] = logReadHandler.lastValidTemperature;//then use the last valid temperature reading
60 |     logReadHandler.errorCount++;//increment the errorCount variable
61 |     if(logReadHandler.errorCount >= 180){//180 arbitrarily chosen - it is 5% of 1 hour logging readings
62 |       //oh my god thigs are bad! User, you must take over from here
63 |       callbackError = "Too many temperature reading errors, something may be going awry!";
64 |       debug("Too many temperature reading errors, something may be going awry!");
65 |       callback(err, null);
66 |     }
67 |     else{
68 |       callbackError = "Single wrong reading, nothing to worry yet.";
69 |       debug("Single wrong reading, nothing to worry yet.");
70 |       callback(err, null);
71 |     }
72 |   }
73 |   else{
74 |     parsedData[0] = +parsedData[0];//string to number
75 |     logReadHandler.lastValidTemperature = +parsedData[0];//save the last valid temperature reading
76 |   }
77 |   logReadHandler.lastReadingsTimestamp[4] = parsedData[1];//updates last temperature reading timestamp
78 |   for(var i = 0; i < 4; i++){//updates the older timestamps also
79 |     logReadHandler.lastReadingsTimestamp[i] = logReadHandler.lastReadingsTimestamp[i+1];
80 |   }
81 |   if(logReadHandler.lastReadingsTimestamp[0] >= logReadHandler.lastReadingsTimestamp[4]){//if the reading is the same
82 |     //within 4s
83 |     //hey user check this, because something may be going awry!
84 |     callbackError = "Log not updating, check temperature probe connection!";
85 |     debug("Log not updating, check temperature probe connection!");
86 |     logReadHandler.errorCount++;//increment the errorCount variable
87 |     if(logReadHandler.errorCount >= 180){//180 arbitrarily chosen - it is 5% of 1 hour logging readings
88 |       //oh my god thigs are bad! User, you must take over from here
89 |       callbackError = "Too many temperature reading errors, something may be going awry!";
90 |       debug("Too many temperature reading errors, something may be going awry!");
91 |       callback(err, null);
92 |     }
93 |   }
94 |   else{//then we can act in order to get to the temperature setpoint
95 |     //callback second argument is the temperature
96 |     callback(null, parsedData[0]);//successfully got the temperature reading
97 |   }
98 | }
99 | };
100 | module.exports.checkRecipeIntegrity = function (recipe, path, res){
101 |   var serverResponse = {resp: "success", warn: "", err: ""};//tells the client if everything is ok
102 |   var recipeContents ;
103 |   global.environmentVariables.recipe = recipe;//save the recipe name
104 |   fs.readFile(path + "/" + recipe, function(err, recipeFileContents){
105 |     var okToStartFlag = 1;
106 |     if(err){//if file contents could not be retrieved
107 |       serverResponse.resp = "couldntReadFile";//tells the client
108 |       res.send(serverResponse);
109 |     }
110 |     else{//if file was successfully read
111 |       fs.readFile("./dataLog/lockfile", function(err, lockFileContents){
112 |         if(err){//if lockfile could not be read for some reason
113 |           serverResponse.resp = err;
114 |           //res.send(serverResponse);
115 |           okToStartFlag = 0;//unable to start the recipe
116 |           debug("couldn't read lockfile, unable to start recipe.");
117 |           return;
118 |         }
119 |         if(+lockFileContents == 0){//if there is no recipe in progress
120 |           debug("ready to rock!");
121 |           recipeContents = recipeFileContents.toString("UTF8").split("\n");//split contents to array
122 |           for(var i = 0; i < recipeContents.length; i++){//get only the relevant data

```

```

123 recipeContents[i] = recipeContents[i].split('')[1];
124 if(!recipeContents[i]){//if some recipe line is blank
125     switch(i){//and this line is important, then:
126         case 0://recipe name not set (warning)
127             serverResponse.warn += "nome da receita; ";
128             break;
129         case 1://beer style not set (warning)
130             serverResponse.warn += "estilo; ";
131             break;
132         case 2://beer yeast not set (warning)
133             serverResponse.warn += "levedura; ";
134             break;
135         case 3://mash water not set (error)
136             serverResponse.err += "água de brassagem; ";
137             global.environmentVariables.okToStart = false;
138             okToStartFlag = 0;
139             break;
140         case 4://sparging water not set (warning)
141             serverResponse.warn += "água de sparging; ";
142             break;
143         case 5://sparging water temperature not set (warning)
144             serverResponse.warn += "temperatura de sparging; ";
145             break;
146         case 6://boiling time not set (error)
147             serverResponse.err += "tempo da fervura; ";
148             global.environmentVariables.okToStart = false;
149             okToStartFlag = 0;
150             break;
151         case 7://mash initial temperature not set (error)
152             serverResponse.err += "temperatura inicial de brassagem; ";
153             global.environmentVariables.okToStart = false;
154             okToStartFlag = 0;
155             break;
156         case 8://mash first step not set (error)
157             serverResponse.err += "primeiro degrau de temperatura; ";
158             global.environmentVariables.okToStart = false;
159             okToStartFlag = 0;
160             break;
161         case 9://mash first step time not set (error)
162             serverResponse.err += "tempo do primeiro degrau; ";
163             global.environmentVariables.okToStart = false;
164             okToStartFlag = 0;
165             break;
166         case 24://malt 1 not set (error)
167             serverResponse.err += "malte 1; ";
168             global.environmentVariables.okToStart = false;
169             okToStartFlag = 0;
170             break;
171         case 25://malt 1 quantity not set (error)
172             serverResponse.err += "quantidade do malte 1; ";
173             global.environmentVariables.okToStart = false;
174             okToStartFlag = 0;
175             break;
176         case 40://hop 1 not set (error)
177             serverResponse.err += "lúpulo 1; ";
178             global.environmentVariables.okToStart = false;
179             okToStartFlag = 0;
180             break;
181         case 41://hop 1 quantity not set (error)
182             serverResponse.err += "quantidade do lúpulo 1; ";
183             global.environmentVariables.okToStart = false;
184             okToStartFlag = 0;
185             break;
186         case 42://hop 1 adding time not set (error)
187             serverResponse.err += "tempo de adição do lúpulo 1; ";
188             global.environmentVariables.okToStart = false;
189             okToStartFlag = 0;
190             break;
191     }
192 }
193 }
194 if(okToStartFlag){//if recipe can be started
195     global.environmentVariables.okToStart = true;//add this info to the global variables
196 }
197 logToFile("request to start production", 0, "start");
198 res.send(serverResponse);//answer to the client (may have error msg or not)
199 debug(serverResponse);
200 }
201 else{//there is a recipe in progress, cannot start
202     //okToStartFlag = 0;
203     serverResponse.resp = "recipe in progress. Unable to start mashing";
204     res.send(serverResponse);//answer error to the client
205 }
206 });
207 }
208 });
209 };

```

```

210
211 var logToFile = module.exports.logToFile = function (message, code, notableTimestamp){
212   var d = new Date();
213   var logTime = d.getTime();
214   var dataToSave ;
215   var logFile = "./datalog/backup.log"; //path to the backup logs file
216   global.environmentVariables.msg = message; //explanatory message to be logged
217   global.environmentVariables.code = code; //code referring to the message
218   global.environmentVariables.timestamps.curr = logTime; //logs the request to start recipe timestamp
219   if(notableTimestamp in global.environmentVariables.timestamps){ //checks if variable is declared
220     debug(notableTimestamp + " is declared. Logging notable timestamp");
221     global.environmentVariables.timestamps[notableTimestamp] = logTime;
222   }
223   //global.environmentVariables.ioStatus = gpioCfg.all_io; //all of the pins status
224   //the above commented line may not be needed since the main script attributes one variable to the other
225   //and so one variable points to the other, not needing to refresh it. *They are pointers, not values!
226   dataToSave = JSON.stringify(global.environmentVariables) + "\n";
227   debug(global.environmentVariables.msg + " - code: " + global.environmentVariables.code + " - auto: " + global.
     environmentVariables.auto);
228   if(code == 0){ //if it is the first line to be logged, overwrite log file
229     fs.writeFile(logFile, dataToSave, function(err){ //overwrite previous backup
230       if(err){ //if was unable to log to the file
231         debug("could not write to the backup file!"); //well my friend, you're on your own!
232       }
233       else{
234         debug("backup log started!");
235       }
236     });
237   }
238   else{ //otherwise, just append the data
239     fs.appendFile(logFile, dataToSave, function(err){ //overwrite previous backup
240       if(err){ //if was unable to log to the file
241         debug("could not write to the backup file!"); //well my friend, you're on your own!
242       }
243       else{
244         debug("backup log updated!");
245       }
246     });
247   }
248 };
249
250 module.exports.sendRecipeNames = function (path, res){ //try to read files in directory
251   fs.readdir(path, function(err, files){
252     var deletedIndexes = new Array(); // variable that holds the indexes of the deleted recipes
253     var serverResponse = {resp: "success"};
254     if(err){ //if something is wrong
255       debug(err); //print the error
256       serverResponse.resp = "error";
257       serverResponse.recipes = err;
258     }
259     else{
260       for(var i = 0; i < files.length; i++){ //iterate the array of names
261         if(files[i].indexOf(".del") > 0){ //if the server reads a deleted file
262           deletedIndexes.push(i);
263         }
264         files[i] = files[i].replace(".recipe", ""); //remove the file extension
265         files[i] = files[i].replace(/_/g, " "); //replace underlines with spaces
266       }
267       for(i = (deletedIndexes.length)-1; i >= 0; i--){ //iterate the array of deleted recipes
268         //debug("index: " + deletedIndexes[i]);
269         files.splice(deletedIndexes[i], 1); //deletes the file name from the array
270       }
271       debug("deleted recipes indexes: " + deletedIndexes);
272       serverResponse.recipes = files;
273     }
274     res.send(serverResponse); //send the recipes if successful, otherwise sends the error
275   }); //get it and return to the client
276 };

```

Código-fonte F.4: Módulo de registros e verificações em geral

F.5 Módulo de controle do processo de brassagem

```

1 'use strict';
2
3 var debug = require('debug')('ctrl');
4 var fs = require('fs');
5 var b = require('octalbonescript');
6 var gpioCfg = require('./gpio_cfg.js');
7 var log = require('./log_check_misc.js');
8

```

```

9 module.exports.startMashingProcess = function (recipe, res, lockFile, recipesPath, callback){
10 /*Starts the mashing process, checking if everything is in order, starting
11 the temperature logging and the heating of the mash water.
12 The callback function is passed as callback to the heatMashWater function*/
13 var serverResponse = {resp:"success"};
14 var recipeContents ;
15 if(global.environmentVariables.okToStart){//first check is if the recipe is ok to start
16 if(recipe == global.environmentVariables.recipe){//recipe name should match also
17 fs.writeFile(lockFile, 1, function(err){//write to lockfile telling there is a recipe in progress
18 if(err){
19 serverResponse.resp = "could not write to lockfile";
20 res.send(serverResponse);
21 callback(serverResponse.resp,null);
22 return;
23 }
24 log.logToFile("production starting", 1);//log to file
25 fs.readFile(recipesPath + "/" + global.environmentVariables.recipe, function(err, data){
26 if(err){//if file contents could not be retrieved
27 serverResponse.resp = "couldn't read recipe file";//tells the client
28 res.send(serverResponse);
29 callback(serverResponse.resp,null);
30 return;
31 }//if file was successfully read
32 log.startTemperatureLogging();
33 recipeContents = data.toString("UTF8").split("\n");//split contents to array
34 for(var i = 0; i < recipeContents.length; i++){//get only the relevant data
35 recipeContents[i] = recipeContents[i].split('"')[1];
36 }
37 serverResponse.resp = "success";//tells the client everything went alright
38 res.send(serverResponse);
39 heatMashWater(recipeContents);//start to heat the mash water
40 //The callback is called when the heat starts, not after it finishes
41 callback(null,"Started to heat mash water");
42 });
43 });
44 }
45 else{//if recipe name doesn't match the one from "startRequest"
46 serverResponse.resp = "failed";
47 res.send(serverResponse);
48 callback(serverResponse.resp,null);
49 }
50 }
51 else{//if recipe isn't ok to start
52 serverResponse.resp = "failed";
53 res.send(serverResponse);
54 callback(serverResponse.resp,null);
55 }
56 };
57
58 function heatMashWater(recipeContents){
59 //heats the mashing water to the starting setpoint
60 var mashSetpoint = recipeContents[7];
61 var instantPath = "./datalog/instant.csv";
62 var lockFile = "./datalog/lockfile";//file that tells if recipe is running
63 var reachedSetpoint = false;//var set to true the first time the checkpoint is reached
64 var logReadHandler = {
65 lastValidTemperature: 0,//the last valid temperature reading (defaults to zero, not the better solution)
66 lastReadingsTimestamp: [0, 0, 0, 0, 0],//last 5 timestamps, to check if readings are going ok
67 errorCount: 0//counts the file reading errors
68 };
69 global.heatingPower = 2;
70 global.environmentVariables.tmpMTsetp = mashSetpoint;//stores the setpoints
71 gpioCfg.changeStatusIO("mash_pump", "true");//turn the recirculation pump on
72 log.logToFile("heating mash water", 2, "startHeating");//log to file, first time get notable timestamp
73 var logTimer = setInterval(function(){//logs the heating process every ~5s
74 log.logToFile("heating mash water", 2);//log to file
75 }, 5000);
76 var readTmpTimer = setInterval(function(){//read temperature every second
77 debug("Executing temperature control loop[1]");
78 log.getTemperatureReading(instantPath, logReadHandler, function(err, temperature){
79 if(err){
80 debug("something went wrong");
81 return;
82 }
83 updateHeatingConditions("MT", mashSetpoint, temperature, function(){
84 if(!reachedSetpoint){//if it is the first time the setpoint is reached
85 debug('Heating of the mash water finished. Now the user must add the grains.');//
86 debug(" Current temperature = " + temperature + "\n"
87 + " Reading errors = " + logReadHandler.errorCount);
88 reachedSetpoint = true;//holds this information
89 clearInterval(logTimer);//stop the logging
90 log.logToFile("waiting for grains", 3, "finishHeating");//log to file at least once
91 logTimer = setInterval(function(){//logs that system is waiting for user input
92 log.logToFile("waiting for grains", 3);//log to file
93 }, 5000);
94 }
95 //while the temperature is above the setpoint, do nothing - just wait for the user

```

```

96     });
97   });
98   //this if must be outside the updateHeatingConditions() call, otherwise recursive behavior mess it up
99   if(global.environmentVariables.readyForNextStep){//wait for the temperature reach setpoint and user add grains
100     global.environmentVariables.readyForNextStep = false;//prepare the flag for the next step request
101     clearInterval(logTimer);//stop the "waiting for grains" logging
102     clearInterval(readTmpTimer);//stop the temperature adjusting loop
103     rampControl(recipeContents);
104   }
105 },1000);
106 //just do the next things after the user added the grains
107 }
108
109 function rampControl(recipeContents){
110   var lockFile = "./datalog/lockfile";//file that tells if recipe is running
111   var instantPath = "./datalog/instant.csv";
112   var instantBK = "./datalog/instant_bk.csv";
113   var currentSetpointPointer = 1;
114   var finishedRamp = false;
115   var mashSetpoint = recipeContents[8];//get the first setpoint
116   var spargeSetpoint = recipeContents[5];
117   var spargeSet = false;// only set this if the sparge temperature is set
118   var stepTime = 60*1000*recipeContents[9];//get the first step rest time in ms
119   global.environmentVariables.tmpMTsetp = mashSetpoint;//stores the setpoints
120   global.environmentVariables.tmpBKsetp = spargeSetpoint;
121   var logReadHandler = {
122     lastValidTemperature: 0,//the last valid temperature reading (defaults to zero, not the better solution)
123     lastReadingsTimestamp: [0, 0, 0, 0, 0],//last 5 timestamps, to check if readings are going ok
124     errorCount: 0//counts the file reading errors
125   };
126   var logReadHandlerBK = {
127     lastValidTemperature: 0,//the last valid temperature reading (defaults to zero, not the better solution)
128     lastReadingsTimestamp: [0, 0, 0, 0, 0],//last 5 timestamps, to check if readings are going ok
129     errorCount: 0//counts the file reading errors
130   };
131   if(spargeSetpoint){//if there is a sparge temperature
132     spargeSet = true;//then assumes there will be sparging water to heat
133   }
134   fs.writeFile(lockFile, 0, function(err){//release system to start new production (for testing purpose only)
135     if(err) return;
136   });
137   debug("Ramp control started!");
138   log.logToFile("mash ramp in progress", 4, "sRamp0");//log immediately and hold notable timestamp
139   var logTimer = setInterval(function(){//logs that the mash ramp is in progress
140     log.logToFile("mash ramp[" + currentSetpointPointer + "] in progress", 4);//log to file
141   }, 5000);
142   var readTmpTimer = setInterval(function (){//read temperature every second
143     debug("Executing temperature control loop[2]");
144     if(global.environmentVariables.readyForNextStep && currentSetpointPointer < 8){//if there are still mashing steps
145       debug("setpoint pointer: " + currentSetpointPointer);//cannot enter here when currentSetpointPointer == 1
146       global.environmentVariables.readyForNextStep = false;//wait for the step to finish before checking next step
147       if(recipeContents[8 + 2*currentSetpointPointer]){//and the value is not empty
148         mashSetpoint = recipeContents[8 + 2*currentSetpointPointer];//get the new setpoint
149         stepTime = 60*1000*recipeContents[9 + 2*currentSetpointPointer];//get the next step rest time in ms
150         global.environmentVariables.tmpMTsetp = mashSetpoint;//stores the setpoints
151         debug("Starting next ramp");
152         clearInterval(logTimer);//stop logging step to log ramp
153         log.logToFile("mash ramp in progress", 4, "sRamp" + currentSetpointPointer);//log immediately and hold notable timestamp
154         logTimer = setInterval(function(){//logs that the mash ramp is in progress
155           log.logToFile("mash ramp[" + currentSetpointPointer + "] in progress", 4);//log to file
156         }, 5000);
157       }
158       else{
159         //gets here for all the lasting setpoints (currentSetpointPointer <= 7)
160         global.environmentVariables.readyForNextStep = true;//ready for the sparging/boil
161         debug("End of the mash ramps/steps process " + currentSetpointPointer);
162       }
163       currentSetpointPointer++;//increment to point to the next setpoint
164     }
165     else if(global.environmentVariables.readyForNextStep){//start the next step, which is sparging or boil
166       global.environmentVariables.readyForNextStep = false;//prepare for the next step
167       gpioCfg.changeStatusIO("mash_heat", "false");//just to make sure, turn off heating elements
168       gpioCfg.changeStatusIO("boil_heat", "false");//just to make sure, turn off heating elements
169       gpioCfg.changeStatusIO("mash_pump", "false");//also shutdown the recirculation pump
170       clearInterval(readTmpTimer);//stop everything, for now at least
171       clearInterval(logTimer);
172       if(spargeSet){//if there will be sparging, then start it
173         spargingControl(recipeContents);//start the sparging process
174       }
175       else{//goes straight to the boil, without sparging
176         debug("No sparging: starting the boil process.");
177         theBoil(recipeContents);
178       }
179     }
180     else{//temperature control code, no need to execute when it is getting the next setpoint
181       log.getTemperatureReading(instantPath, logReadHandler, function(err, temperature){//read current temperature

```

```

182     if(err){
183         debug("something went wrong");
184         return;
185     }
186     updateHeatingConditions("MT", mashSetpoint, temperature, function(){//control temperature for current setpoint
187         if(!finishedRamp){//if the ramp just reached its setpoint
188             //log and save the notable timestamp fRamp
189             finishedRamp = true;
190             clearInterval(logTimer);//stop logging ramp to log step
191             log.logToFile("mash step[" + currentSetpointPointer + "] in progress", 5, "fRamp" + currentSetpointPointer);
192             logTimer = setInterval(function(){//logs that the mash ramp is in progress
193                 log.logToFile("mash step[" + currentSetpointPointer + "] in progress", 5);//log to file
194             }, 5000);
195             debug('Ramp finished, keeping temperature for the duration of the step.');//
196             debug( "          Current temperature = " + temperature + "\n"
197                 + "          Reading errors = " + logReadHandler.errorCount);
198         }
199         if((global.environmentVariables.timestamps.curr - global.environmentVariables.timestamps["fRamp" +
200             currentSetpointPointer]) > stepTime){
201             //if the step rest is finished
202             finishedRamp = false;
203             global.environmentVariables.readyForNextStep = true;//if setpoint is reached
204             debug('Step finished, starting next ramp or going to sparging.');//
205             debug( "          Current temperature = " + temperature + "\n"
206                 + "          Reading errors = " + logReadHandler.errorCount);
207         }
208         else{//if it is in the middle of a step, heat the sparging water and tell the user the step time left
209             global.environmentVariables.timeLeft = //in minutes (not round) = total time - elapsed time
210             (stepTime - (global.environmentVariables.timestamps.curr - global.environmentVariables.timestamps["fRamp" +
211                 currentSetpointPointer]))/60000;
212             if(spargeSet){//only heat sparging water if there is sparging water to heat
213                 log.getTemperatureReading(instantBK, logReadHandlerBK, function(err, temperature){//read current temperature
214                     if(err){
215                         debug("something went wrong");
216                         return;
217                     }
218                     updateHeatingConditions("BK", spargeSetpoint, temperature, function(){//heat the sparging water
219                         //});
220                     });
221                 }
222             });
223         }
224     },1000);
225 }
226
227 function spargingControl(recipeContents){
228     //function heavily based in timing. The time it takes for the water to flow is pretty predictable.
229     //The time for the mash to drain depends on the recipe, hence the mash tun overflow watching device.
230     //Because the pump flow is higher then the drain flow, the pump must be controlled to adjust.
231     //The pump must start some time after the drain, so the grain bed is almost uncovered by the mash.
232     //Also, in case of overflow, there must be a predefined time or time function to wait until the recirculation restarts.
233     var logMsg = { notableTimestamp:"startDrain",//if there is the need to log a notable timestamp, put its name here
234         message:"sparging in progress",//log message must be put here
235         code:6//code identifier
236     };
237     var mashTunOverflow = false;
238     var finishedFirstDrain = false;//should modify to use setTimeout instead of setInterval
239     var finishedSpargeTime = false;//should modify to use setTimeout instead of setInterval
240     var justAnotherFlag = false;//terrible thing to name a flag like this! Hue Hue BR trolling myself from the future
241     //the time setpoints are based on the volume of water used in mashing and sparging
242     var drainTimeSetp = 500*recipeContents[3];//time in ms that should wait before showering with sparge water
243     var spargeTimeSetp = 3000*recipeContents[4];//time in ms that should wait before finishing the process
244     var overflowTimeSetp = 30000;//time to wait before pumping the sparge water again
245     var realSpargeTime = 0;//approximately the time that the recirculation pump is working in ms
246
247     debug("Starting the sparging process");
248     log.logToFile(logMsg.message, logMsg.code, logMsg.notableTimestamp);//log the startDrain notable timestamp
249     var logTimer = setInterval(function(){//log periodically
250         if(logMsg.notableTimestamp){//if a notable timestamp is set
251             log.logToFile(logMsg.message, logMsg.code, logMsg.notableTimestamp);//log with notable timestamp
252             logMsg.notableTimestamp = "";//clear the notable timestamp so it doesn't overwrite
253         }
254         else{
255             log.logToFile(logMsg.message, logMsg.code);//log without notable timestamp
256         }
257     },5000);//every half second? really? it can be done every 5s without major concerns!
258
259     var overflowTimer = setInterval(function(){//watch periodically for MT overflow
260         if(!mashTunOverflow && global.environmentVariables.ioStatus["led"].state == b.HIGH){//if MT is about to overflow
261             debug("Overflow detected, pumping on hold");
262             mashTunOverflow = true;//set overflow flag
263             gpioCfg.changeStatusIO("boil_pump", "false");//stop pumping
264             logMsg.message = "mash tun overflow"; logMsg.code = 7;//log to file that the mash tun is almost overflowing
265             setTimeout(function(){//wait some time before turning the recirculation pump on again
266                 mashTunOverflow = false;

```



```

267 //gpioCfg.changeStatusIO("boil_pump", "true");//this is risky since the pump could be already off before overflow
268 debug("Fake overflow OFF for testing purposes");
269 gpioCfg.changeStatusIO("led", "false");//fake overflow for testing purposes
270 },overflowTimeSetp);
271 }
272 },500);
273
274 gpioCfg.changeStatusIO("mash_valve", "true");//open the drain valve
275 var spargeTimer = setTimeout(function(){//wait until the first mash drain is finished before spraying sparge water
276 debug("Finished the first drain");
277 logMsg.message = "sparging in progress"; logMsg.code = 6;//log the notable startSparge timestamp
278 logMsg.notableTimestamp = "startSparge";
279 gpioCfg.changeStatusIO("boil_valve", "true");//open the recirculation valve
280 gpioCfg.changeStatusIO("boil_pump", "true");//get the pump to run, consider some method to control the flow rate
281 spargeTimer = setInterval(function(){//must wait for the sparge time in a loop because of the overflow check
282 if(!mashTunOverflow){//if there is no overflow
283 logMsg.message = "sparging in progress"; logMsg.code = 6;//this is just needed after an overflow
284 gpioCfg.changeStatusIO("boil_pump", "true");//this is just needed after an overflow
285 realSpargeTime += 500;//add approximately the loop time
286 if(realSpargeTime == 500){//fake an overflow to check code
287 debug("Fake overflow ON for testing purposes : " + realSpargeTime);
288 gpioCfg.changeStatusIO("led", "true");//fake overflow for testing purposes
289 }
290 if(realSpargeTime > spargeTimeSetp){//if the running time is enough
291 debug("Stopping the sparging process. Waiting for the mash to completely drain");
292 gpioCfg.changeStatusIO("boil_valve", "false");//close the recirculation valve
293 gpioCfg.changeStatusIO("boil_pump", "false");//stop pumping
294 clearInterval(overflowTimer);//from now on there is no more overflow risk
295 clearInterval(spargeTimer);//also stop the time increment loop
296 setTimeout(function(){//wait for all the mash to drain to the BK
297 debug("mash drained, sparging process finished");
298 gpioCfg.changeStatusIO("mash_valve", "false");//close the drain valve
299 gpioCfg.changeStatusIO("boil_valve", "false");//for security check
300 gpioCfg.changeStatusIO("boil_pump", "false");//for security check
301 clearInterval(logTimer);//stop periodic logging
302 //for now, I think there is no problem to call the boil here cause the only nested
303 //functions are setInterval and setTimeout
304 theBoil(recipeContents);//maybe it is better to call this outside the loops?
305 },2.5*spargeTimeSetp);
306 }
307 }
308 },500);
309 },drainTimeSetp);
310 }
311
312 function theBoil(recipeContents){
313 //Straightforward part of the process, no need to do a lot of checks, etc
314 //something important is remember the user to remove the grains from the MT
315 //because the chill water will be stored there
316 var boilFlag = false;//set when the boil starts
317 var endBoilTime;//hold the timestamp of when the boil must end
318 var instantBK = ".datalog/instant_bk.csv";
319 var logReadHandlerBK = {
320 lastValidTemperature: 0,//the last valid temperature reading (defaults to zero, not the better solution)
321 lastReadingsTimestamp: [0, 0, 0, 0, 0],//last 5 timestamps, to check if readings are going ok
322 errorCount: 0//counts the file reading errors
323 };
324 debug("Heating wort until it boils");
325 log.logToFile("heating wort to boil", 8, "heatingBoil");//log the process start
326 var logTimer = setInterval(function(){//log while waiting for the boil
327 log.logToFile("heating wort to boil", 8);//log periodically
328 },5000);
329 gpioCfg.changeStatusIO("boil_heat", "true");//full power
330 var readTmpTimer = setInterval(function(){//read temperature every second
331 var hopAddSetp;//timestamp of when to add the next hop
332 log.getTemperatureReading(instantBK, logReadHandlerBK, function(err, temperature){
333 if(err){
334 debug("something went wrong");
335 return;
336 }
337 if(temperature > 96 && !boilFlag){//if it is near enough boiling
338 debug("boiling started! Burn, baby, burn!");
339 clearInterval(logTimer);//stop logging code 7
340 log.logToFile("boiling wort", 9, "boilStart");//log the process start
341 logTimer = setInterval(function(){//log the boil
342 log.logToFile("boiling wort", 9);//log periodically
343 },5000);
344 endBoilTime = 60*1000*recipeContents[6] + global.environmentVariables.timestamps.curr;//end of boiling
345 global.environmentVariables.timestamps.boilFinishScheduled = endBoilTime;
346 for(var hopPointer = 0; hopPointer < 8; hopPointer++){//check all possible hop addition times
347 hopAddSetp = 60*1000*recipeContents[42 + 3*hopPointer];//time from the beginning in ms
348 if(hopAddSetp){//if there is a next hop to add
349 //this way, if the hops are out of order of addition, it doesn't matter
350 debug("Hop " + recipeContents[40 + 3*hopPointer] + "scheduled to be added in " + hopAddSetp + "ms");
351 setTimeout(function(){//then add it when it is time
352 debug("Adding hop " + recipeContents[40 + 3*hopPointer]);
353 controlHopCompartment(hopPointer + 1);

```

```

354 |         log.logToFile("added hop [" + hopPointer + "]", 10); //log periodically
355 |     }, hopAddSetp);
356 |     }
357 | }
358 | }
359 | });
360 | if(global.environmentVariables.timestamps.curr > endBoilTime){ //end of the boil
361 |     debug("end of the boil");
362 |     gpioCfg.changeStatusIO("boil_heat", "false"); //shutdown the BK heater
363 |     controlHopCompartment(0); //close the hops compartment
364 |     log.stopTemperatureLogging(); //for testing purposes
365 | }
366 | }, 1000);
367 | }
368 | }
369 | function controlHopCompartment(pos) { //control the servo to open the hops compartment until requested position
370 |     //maybe it's better to close everytime a hop is added, think about it!
371 |     var angle = pos * 11.25; //11.25 degree per compartment, if pos=0, close everything
372 |     gpioCfg.changeStatusIO("servo_pwm", angle); //change the compartment opening
373 | }
374 | }
375 | function updateHeatingConditions(vessel, setp, temperature, callback) {
376 |     //This function must be called in a loop, every loop iteration
377 |     if(vessel == "mash_heat" || vessel == "MT" || vessel == "mash tun") {
378 |         vessel = "mash_heat"; //if wants to heat the mash tun
379 |         global.environmentVariables.tmpMT = temperature; //update the mash tun temperature
380 |         if(global.environmentVariables.ioStatus["boil_heat"].state == b.HIGH) { //security check
381 |             gpioCfg.changeStatusIO("boil_heat", "false"); //turn off the other heating element
382 |             debug("Security shutdown of the BK heater");
383 |         }
384 |     }
385 |     else if(vessel == "boil_heat" || vessel == "BK" || vessel == "brewing kettle" || vessel == "copper") {
386 |         vessel = "boil_heat"; //if wants to heat the brewing kettle
387 |         global.environmentVariables.tmpBK = temperature; //update the brewing kettle temperature
388 |         if(global.environmentVariables.ioStatus["mash_heat"].state == b.HIGH) { //security check
389 |             gpioCfg.changeStatusIO("mash_heat", "false"); //turn off the other heating element
390 |             debug("Security shutdown of the MT heater");
391 |         }
392 |     }
393 |     else { //if vessel is unknown
394 |         callback("unknown vessel");
395 |         return; //don't even try to heat
396 |     }
397 |     debug("Heating power variable: " + global.heatingPower);
398 |     if(temperature < 0.7 * setp) { //if temperature is less then 0.7 of the setpoint
399 |         debug(vessel + ": 100% heating power");
400 |         gpioCfg.changeStatusIO(vessel, "true"); //give 100% power to the heating resistor
401 |     }
402 |     else if(temperature < 0.9 * setp) { //if temperature between 0.7 and 0.9 of the setpoint
403 |         debug(vessel + ": 66% heating power");
404 |         if(global.heatingPower == 0) { //give 66% power to the heating resistor
405 |             gpioCfg.changeStatusIO(vessel, "false"); // 1/3 of the time off
406 |             global.heatingPower = 2; // heatingPower goes from 0 to 2
407 |         }
408 |         else {
409 |             gpioCfg.changeStatusIO(vessel, "true"); // 2/3 of the time on
410 |             global.heatingPower--;
411 |         }
412 |     }
413 |     else if(temperature < setp) { //if temperature between 0.9 and 1.0 of the setpoint
414 |         debug(vessel + ": 33% heating power");
415 |         if(global.heatingPower == 0) { //give 33% power to the heating resistor
416 |             gpioCfg.changeStatusIO(vessel, "true"); // 1/3 of the time on
417 |             global.heatingPower = 2; // heatingPower goes from 0 to 2
418 |         }
419 |         else {
420 |             gpioCfg.changeStatusIO(vessel, "false"); // 2/3 of the time off
421 |             global.heatingPower--;
422 |         }
423 |     }
424 |     else { //in this case, the temperature got to the setpoint
425 |         debug(vessel + ": 0% heating power");
426 |         gpioCfg.changeStatusIO(vessel, "false"); // turn the heating element off
427 |         callback(null); //choose what to do
428 |     }
429 | }

```

Código-fonte F.5: Código-fonte raiz da aplicação em Node.js

Apêndice G

Códigos-fonte da aplicação web

G.1 Módulo PHP que cria a barra de navegação da interface web

```

1 <?php
2 error_reporting(-1); ini_set("display_errors", "On");//debug on
3
4 //echo '{"0":"true"}';
5 if(isset($_POST["url"])){//workaround because Express doesn't set http_host and request_uri
6     echo generateHeader();
7 }
8
9 function generateHeader(){
10     if(isset($_POST["url"])){//workaround because Express doesn't set http_host and request_uri
11         $url = parse_url($_POST["url"]);
12         $short_url = $url["path"];
13     }
14     else{
15         $url = parse_url($_SERVER["HTTP_HOST"]);//needed because of the different port used for startrecipe
16         $short_url = $_SERVER["REQUEST_URI"];
17     }
18     $links = array(
19         ' href='http://$url[host]:8587/listrecipe.php',
20         ' href='http://$url[host]:8587/startrecipe.php',
21         ' href='http://$url[host]:8587/stats.php',
22         ' href='http://$url[host]:8587/settings.php',
23         ' href='http://$url[host]:8587/about.php');
24
25     $selected = "style='background:#595450;' href='#';";
26
27     switch($short_url){//highlight div and disable click to current page
28         case "/listrecipe.php":
29             $links[0] = $selected;
30             break;
31         case "/startrecipe.php":
32             $links[1] = $selected;
33             break;
34         case "/stats.php":
35             $links[2] = $selected;
36             break;
37         case "/settings.php":
38             $links[3] = $selected;
39             break;
40         case "/about.php":
41             $links[4] = $selected;
42             break;
43     }
44     // Navbar structure
45     return "
46     <div class='myheader'>
47         <div class='headercell'>
48             <a class='headerlink' $links[0]>receitas</a>
49         </div>
50         <div class='headercell'>
51             <a class='headerlink' $links[1]>brassagem</a>
52         </div>
53         <div class='headercell'>
54             <a class='headerlink' $links[2]>estatísticas</a>
55         </div>
56         <div class='headercell'>
57             <a class='headerlink' $links[3]>configurações</a>
58         </div>
59     </div>";

```

```

56 |         </div>
57 |         <div class='headercell'>
58 |             <a class='headerlink' $links[4]>sobre</a>
59 |         </div>
60 |     </div>;
61 | }
62 | ?>

```

Código-fonte G.1: Módulo PHP que cria a barra de navegação da interface web

G.2 Função que passa a URL da página atual para um módulo em PHP

```

1 | function headerPHP (pathToHeader) { //create the header of the page
2 |     $.post (pathToHeader, {url:document.URL} , function (data, status) { //send the page URL to PHP
3 |         if (status == "success") { //if POST is successfull
4 |             $('body').prepend($(data)); //add the received header to the top of the page
5 |             $('body').show(); //and displays the hidden page afterwards
6 |         }
7 |     });
8 | }

```

Código-fonte G.2: Função que passa a URL da página atual para um módulo em PHP

G.3 Página inicial

```

1 | <!DOCTYPE html>
2 |
3 | <head>
4 |     <title>Hello!</title>
5 |     <link rel="icon" type="image/png" href="img/beer2.png">
6 |     <style>
7 |     html {
8 |         background-image: url("../img/beer_compiler.png");
9 |         background-repeat: no-repeat;
10 |         background-size: cover;
11 |     }
12 |     a {
13 |         position: absolute;
14 |         top: 0;
15 |         left: 0;
16 |         width: 100%;
17 |         height: 100%;
18 |     }
19 |     </style>
20 | </head>
21 |
22 | <body>
23 |     <a href="./about.php"></a>
24 | </body></html>

```

Código-fonte G.3: Página inicial da interface web

G.4 Página sobre

```

1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |     <meta charset="UTF-8">
5 |     <title>About</title>
6 |     <link href='https://fonts.googleapis.com/css?family=Roboto&subset=latin,latin-ext' rel='stylesheet' type='text/css'>
7 |     <link rel="stylesheet" type="text/css" href="./css/config.css">
8 |     <link rel="stylesheet" type="text/css" href="./css/buttons.css">
9 |     <link rel="icon" type="image/png" href="img/beer2.png">
10 |     <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
11 |     <script type="text/javascript" src="./lib/header.js"></script>
12 | <script>
13 |     $(function () { headerPHP ("./lib/header.php"); }); //add header

```

```

14 </script>
15 </head>
16
17 <body style="display:none;">
18 <?php // include './lib/header.php'; echo generateHeader();?>
19 <h1>Cervejaria do Futuro</h1>
20 <p>Trabalho de Conclusão de Curso de Engenharia Elétrica - Ênfase em Eletrônica</p>
21 <p>Escola de Engenharia de São Carlos - Universidade de São Paulo (EESC-USP)</p>
22 <p>Professor orientador: Evandro Luís Linhari Rodrigues</p>
23 <p>Aluno: Leonardo Graboski Veiga</p>
24 <ul>
25 <li>Email: leogveiga@gmail.com</li>
26 <li>Github: <a style="font-size:1em;" href="https://github.com/leograba/final_paper_tcc">https://github.com/
leograba/final_paper_tcc</a></li>
27 </ul>
28
29 <br>
30
31 <p>Objetivo do Projeto: Desenvolver um sistema inovador de produção de cerveja
32 , fácil de usar, divertido, confiável e disponível em qualquer lugar</p>
33 <br>
34
35 <p style="font-size:0.75em;">Esta interface foi desenvolvida para ser visualizada em desktop, com o navegador Google
Chrome.
36 Outros navegadores e/ou dispositivos móveis podem apresentar problemas de design ou funcionalidade</p>
37 </body>
38 </html>

```

Código-fonte G.4: Página *sobre* da interface web

G.5 Página de estatísticas

```

1 <html>
2 <head>
3 <meta charset="utf-8"/>
4 <!--Many tries to avoid caching this page-->
5 <meta http-equiv="cache-control" content="max-age=0" />
6 <meta http-equiv="cache-control" content="no-cache" />
7 <meta http-equiv="expires" content="0" />
8 <meta http-equiv="expires" content="Tue, 01 Jan 1980 1:00:00 GMT" />
9 <meta http-equiv="pragma" content="no-cache" />
10
11 <title>Temperatura</title>
12 <link rel="stylesheet" type="text/css" href="./css/config.css">
13 <link rel="stylesheet" type="text/css" href="./css/buttons.css">
14 <link rel="icon" type="image/png" href="./img/beer2.png">
15 <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
16 <script type="text/javascript" src="./lib/header.js"></script>
17 <script>
18 $(function(){gambiarraHeaderPHP("./lib/header.php");});//add header
19 </script>
20 </head>
21
22 <body style="display:none;">
23 <div id="mytemp">
24 <h2 style="display:inline-block;">Gráfico Dinâmico / Temperatura Agora: </h2>
25 <h2 id="tnow" style="display:inline-block;"></h3>
26 </div>
27 <iframe id="realtime_graph" src="./dyn_graph2.html" width="100%" height="100%" frameborder="0"></iframe>
28 <h2>Gráfico do histórico de temperatura</h2>
29 
30 <map name="plot">
31 <area shape="rect" coords="0,0,1300,577" href="./img/tplot.png" alt="?tplot.png">
32 </map>
33 <br>
34 </body>
35
36 <script>
37 $(function(){
38 //acesso ao iframe
39 //var myFrame = document.getElementById("realtime_graph");
40 //var insideMyFrame = myFrame.contentDocument || myFrame.contentWindow.document;
41 var temperatura = document.getElementById("tnow");
42 var grafico = document.getElementById("static_graph");
43
44 setInterval(function(d){//atualiza temperatura a cada 1s
45 myFrame = document.getElementById("realtime_graph");
46 insideMyFrame = myFrame.contentDocument || myFrame.contentWindow.document;
47 d = insideMyFrame.getElementById("temp_now").innerHTML;
48 console.log("teste.php - lendo iframe:"+d);

```

```

49     console.log(insideMyFrame.getElementById("temp_now").innerHTML);
50     temperatura.innerHTML = d;
51     },1000);//repete a cada 1000ms
52
53     setInterval(function(){//atualiza grafico estatico a cada 2,5min
54         grafico.src = "/tplot.png?" + new Date().getTime();
55         },150000);//repete a cada 2,5min
56     });
57 </script>
58 </html>

```

Código-fonte G.5: Página de estatísticas

G.6 Gráfico dinâmico de temperatura

```

1  <!doctype html>
2  <head>
3      <meta charset="utf-8"/>
4      <meta http-equiv="cache-control" content="max-age=0" />
5      <meta http-equiv="cache-control" content="no-cache, no-store, must-revalidate" />
6      <meta http-equiv="expires" content="0" />
7      <meta http-equiv="expires" content="Tue, 01 Jan 1980 1:00:00 GMT" />
8      <meta http-equiv="pragma" content="no-cache" />
9
10     <link type="text/css" rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8/themes/base/jquery-ui.css"
11         >
12     <link type="text/css" rel="stylesheet" href="./rickshaw/src/css/graph.css">
13     <link type="text/css" rel="stylesheet" href="./rickshaw/src/css/detail.css">
14     <link type="text/css" rel="stylesheet" href="./rickshaw/src/css/legend.css">
15     <link type="text/css" rel="stylesheet" href="./rickshaw/examples/css/extensions.css?v=2">
16
17     <script src="./rickshaw/vendor/d3.v3.js"></script>
18
19     <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.min.js"></script>
20     <script>
21         jQuery.noConflict();
22     </script>
23
24     <script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.8.15/jquery-ui.min.js"></script>
25
26     <script src="./rickshaw/src/js/Rickshaw.js"></script>
27     <script src="./rickshaw/src/js/Rickshaw.Class.js"></script>
28     <script src="./rickshaw/src/js/Rickshaw.Compat.ClassList.js"></script>
29     <script src="./rickshaw/src/js/Rickshaw.Graph.js"></script>
30     <script src="./rickshaw/src/js/Rickshaw.Graph.Renderer.js"></script>
31     <script src="./rickshaw/src/js/Rickshaw.Graph.Renderer.Area.js"></script>
32     <script src="./rickshaw/src/js/Rickshaw.Graph.Renderer.Line.js"></script>
33     <script src="./rickshaw/src/js/Rickshaw.Graph.Renderer.Bar.js"></script>
34     <script src="./rickshaw/src/js/Rickshaw.Graph.Renderer.ScatterPlot.js"></script>
35     <script src="./rickshaw/src/js/Rickshaw.Graph.Renderer.Stack.js"></script>
36     <script src="./rickshaw/src/js/Rickshaw.Graph.RangeSlider.js"></script>
37     <script src="./rickshaw/src/js/Rickshaw.Graph.RangeSlider.Preview.js"></script>
38     <script src="./rickshaw/src/js/Rickshaw.Graph.HoverDetail.js"></script>
39     <script src="./rickshaw/src/js/Rickshaw.Graph.Annotate.js"></script>
40     <script src="./rickshaw/src/js/Rickshaw.Graph.Legend.js"></script>
41     <script src="./rickshaw/src/js/Rickshaw.Graph.Axis.Time.js"></script>
42     <script src="./rickshaw/src/js/Rickshaw.Graph.Behavior.Series.Toggle.js"></script>
43     <script src="./rickshaw/src/js/Rickshaw.Graph.Behavior.Series.Order.js"></script>
44     <script src="./rickshaw/src/js/Rickshaw.Graph.Behavior.Series.Highlight.js"></script>
45     <script src="./rickshaw/src/js/Rickshaw.Graph.Smooth.js"></script>
46     <script src="./rickshaw/src/js/Rickshaw.Fixtures.Time.js"></script>
47     <script src="./rickshaw/src/js/Rickshaw.Fixtures.Time.Local.js"></script>
48     <script src="./rickshaw/src/js/Rickshaw.Fixtures.Number.js"></script>
49     <script src="./rickshaw/src/js/Rickshaw.Fixtures.RandomData.js"></script>
50     <script src="./rickshaw/src/js/Rickshaw.Fixtures.Color.js"></script>
51     <script src="./rickshaw/src/js/Rickshaw.Color.Palette.js"></script>
52     <script src="./rickshaw/src/js/Rickshaw.Graph.Axis.Y.js"></script>
53
54     <script src="./rickshaw/examples/js/extensions.js"></script>
55 </head>
56 <body>
57 <div id="temp_now" style="display:none;"></div>
58 <div id="content">
59
60     <form id="side_panel">
61         <!--<h1>Temperatura da Producao</h1-->
62         <section><div id="legend"><div></div></section>
63         <section>
64             <div id="renderer_form" class="toggler">
65                 <input type="radio" name="renderer" id="area" value="area" checked>

```

```

66     <label for="area">area</label>
67     <input type="radio" name="renderer" id="line" value="line">
68     <label for="line">linha</label>
69 </div>
70 </section>
71 <section>
72     <div id="offset_form">
73         <label for="stack">
74             <input type="radio" name="offset" id="stack" value="zero" checked>
75             <span>pilha</span>
76         </label>
77         <label for="value">
78             <input type="radio" name="offset" id="value" value="value">
79             <span>valor</span>
80         </label>
81     </div>
82     <div id="interpolation_form">
83         <label for="cardinal">
84             <input type="radio" name="interpolation" id="cardinal" value="cardinal" checked>
85             <span>cardinal</span>
86         </label>
87         <label for="linear">
88             <input type="radio" name="interpolation" id="linear" value="linear">
89             <span>linear</span>
90         </label>
91         <label for="step">
92             <input type="radio" name="interpolation" id="step" value="step-after">
93             <span>degrau</span>
94         </label>
95     </div>
96 </section>
97 <section>
98     <h6>Smoothing</h6>
99     <div id="smoother"></div>
100 </section>
101 <section>
102     <h6>Mostrar ultimos:</h6>
103     <label for="fiveminutes">
104         <input type="radio" name="maxpoints" id="fiveminutes" value="300">
105         <span>5 min</span>
106     </label>
107     <label for="thirtyminutes">
108         <input type="radio" name="maxpoints" id="thirtyminutes" value="1800">
109         <span>30 min</span>
110     </label>
111     <label for="onehour">
112         <input type="radio" name="maxpoints" id="onehour" value="3600" checked>
113         <span>hora</span>
114     </label>
115 </section>
116 <section></section>
117 </form>
118
119 <div id="chart_container">
120     <div id="chart"></div>
121     <div id="timeline"></div>
122     <div id="preview"></div>
123 </div>
124
125 </div>
126
127 <script>
128
129
130 //preisa ter inicializado os dados na hora de plotar, senao da erro
131 var seriesData = [[], []];
132 var seriesDataBkp = [[], []];
133 var graph;//o esquema eh declarar essa global, pq vai modificar dentro de funcao
134 //variaveis para formatar a string da data para o formato data
135 var fdata = d3.time.format("%a %b %e %X %Y");
136 var dt = new Date();
137 var max_data = 3600;//numero maximo de pontos do grafico
138 //var diff = [0, 0];//diferenca entre os valores atual e anterior de numero de pontos
139 //var last_size = [0, 0];//tamanho do array anterior
140 var last_data = [];//ultima data lida
141
142 //le o arquivo csv do log e plota o grafico
143 //aqui dentro inicializa tudo
144 d3.csv("./datalog/default.csv", function(data) { //le o arquivo .csv
145     console.log("First read from .csv");
146     data.forEach(function(d) { //para cada valor lido, edita os dados
147         d.temperatura = +d.temperatura; //essa string vira numero
148         d.data = +d.data; //essa vira Epoch time
149         for (var i = 0; i < 2; i++) { //le para array temporario
150             seriesData[i].push({x:d.data, y:d.temperatura});
151             seriesDataBkp[i].push({x:d.data, y:d.temperatura});
152             last_data[i] = d.data; //ultima data escrita no array

```

```

153 |     } // fim do for
154 | }); // fim do forEach
155 | // limita o numero de dados plotados excluindo das primeiras posicoes
156 |     seriesData[0].splice(0, seriesData[0].length - max_data);
157 |     seriesData[1].splice(0, seriesData[1].length - max_data);
158 |     seriesDataBkp[0].splice(0, seriesData[0].length - max_data);
159 |     seriesDataBkp[1].splice(0, seriesData[1].length - max_data);
160 |     lets_plot(); // plota o grafico
161 |     updateit();
162 |     graph.render(); // renderiza pela primeira vez
163 |     console.log(graph);
164 |     console.log(seriesData);
165 | }); // fim do d3.csv
166 |
167 |
168 | // loop que le o arquivo csv com a ultima temperatura valida
169 | setInterval(function() { // realiza o loop a cada x milisegundos
170 |     d3.csv("./data/log/instant.csv", function(data) { // le o arquivo .csv
171 |         var current = [];
172 |         var first_time = 0;
173 |         var j = [];
174 |         var con = [];
175 |         var new_data = [];
176 |         var last_temp;
177 |         var last_temp = document.getElementById("temp_now");
178 |         data.forEach(function(d) { // para cada valor lido, edita os dados
179 |             d.temperatura = +d.temperatura; // essa string vira numero
180 |             d.data = +d.data; // essa vira Epoch time
181 |             for (var i = 0; i < 2; i++) { // le para array temporario
182 |                 current[i] = {x: d.data, y: d.temperatura};
183 |                 new_data[i] = d.data; // ultima data lida
184 |             } // fim do for
185 |             last_temp = d.temperatura; // quando acaba o forEach, tem a ultima temp
186 |             console.log(new_data);
187 |         }); // fim do forEach, funcao d
188 |         document.getElementById("temp_now").innerHTML = last_temp + "&deg;C"; // atualiza valor da temp
189 |         console.log(temp_now);
190 |         // console.log("ultima: " + last_temp);
191 |         for (var i = 0; i < 2; i++) { // escreve do temporario pro fixo
192 |             if (new_data[i] > last_data[i]) { // se a data lida eh posterior a ultima data do grafico
193 |                 last_data[i] = new_data[i]; // atualiza a ultima data do grafico
194 |             }
195 |             if (seriesDataBkp[i].length >= 3600) { // backup guarda os ultimos 3600 pontos
196 |                 seriesDataBkp[i].splice(0, seriesDataBkp[i].length - 3600);
197 |                 seriesDataBkp[i].push(current[i]);
198 |             }
199 |             else {
200 |                 seriesDataBkp[i].push(current[i]);
201 |             }
202 |             if (seriesData[i].length >= max_data) { // nunca plota mais que o maximo de pontos
203 |                 seriesData[i].splice(0, seriesData[i].length - max_data);
204 |                 seriesData[i].push(current[i]);
205 |             }
206 |             else {
207 |                 if (seriesDataBkp[i].length > (seriesData[i].length + 1)) { // se mudar escala
208 |                     if (seriesDataBkp[i].length < max_data) { // se tem menos pontos que o maximo
209 |                         j = 0;
210 |                         con = seriesDataBkp[i].length - seriesData[i].length;
211 |                     }
212 |                     else {
213 |                         j = seriesDataBkp[i].length - max_data;
214 |                         con = seriesDataBkp[i].length - seriesData[i].length;
215 |                     }
216 |                     for (; j < con; con--) {
217 |                         seriesData[i].unshift(seriesDataBkp[i][con]); // completa os elementos anteriores
218 |                     } // por exemplo, se a escala estava 300 e vai pra 1800, mas ja tem 500 pontos, adiciona os 200 pontos iniciais que haviam
219 |                     // sido excluidos
220 |                 }
221 |                 else {
222 |                     seriesData[i].push(current[i]);
223 |                 }
224 |             }
225 |         }
226 |     }
227 |     // manda pro console pra debugar
228 |     console.log(seriesData);
229 |     graph.update();
230 | }); // fim do d3.csv
231 | // }, 30000); // repete a cada 15 segundos, metade do tempo do log (30s)
232 | // }, 600); // repete mais de uma vez por segundo
233 |
234 |
235 | // lista dos botoes para ajustar numero de pontos do grafico
236 | var ultimos = document.forms['side_panel'].elements['maxpoints'];
237 | // loop pela lista de botoes
238 | for (var i = 0, len = ultimos.length; i < len; i++) {

```



```

239 |     ultimos[i].onclick = function(){
240 |         max_data = this.value; //muda o valor do numero maximo de pontos
241 |         console.log("max points changed to max_data"+max_data);
242 |     }
243 | }
244 |
245 | // instantiate our graph!
246 |
247 | function lets_plot(){
248 |     var palette = new Rickshaw.Color.Palette( { scheme: 'cool' } );
249 |
250 |     graph = new Rickshaw.Graph( {
251 |         element: document.getElementById("chart"),
252 |         width: 650,
253 |         height: 300,
254 |         min: 'auto',
255 |         renderer: 'area',
256 |         stroke: true,
257 |         preserve: true,
258 |         series: [
259 |             {
260 |                 color: palette.color(),
261 |                 data: seriesData[0],
262 |                 name: 'Temperatura'
263 |             }, {
264 |                 color: palette.color(),
265 |                 data: seriesData[1],
266 |                 name: 'Valvula'
267 |             }
268 |         ]
269 |     } );
270 |
271 |     //graph.render();
272 | }
273 |
274 | function updateit(){
275 |     var preview = new Rickshaw.Graph.RangeSlider( {
276 |         graph: graph,
277 |         element: document.getElementById('preview'),
278 |     } );
279 |
280 |     var hoverDetail = new Rickshaw.Graph.HoverDetail( {
281 |         graph: graph,
282 |         xFormatter: function(x) {
283 |             return new Date(x * 1000).toString();
284 |         }
285 |     } );
286 |
287 |     var annotator = new Rickshaw.Graph.Annotate( {
288 |         graph: graph,
289 |         element: document.getElementById('timeline')
290 |     } );
291 |
292 |     var legend = new Rickshaw.Graph.Legend( {
293 |         graph: graph,
294 |         element: document.getElementById('legend')
295 |     } );
296 |
297 |
298 |     var shelving = new Rickshaw.Graph.Behavior.Series.Toggle( {
299 |         graph: graph,
300 |         legend: legend
301 |     } );
302 |
303 |     var order = new Rickshaw.Graph.Behavior.Series.Order( {
304 |         graph: graph,
305 |         legend: legend
306 |     } );
307 |
308 |     var highlighter = new Rickshaw.Graph.Behavior.Series.Highlight( {
309 |         graph: graph,
310 |         legend: legend
311 |     } );
312 |
313 |     var smoother = new Rickshaw.Graph.Smoother( {
314 |         graph: graph,
315 |         element: document.querySelector('#smoother')
316 |     } );
317 |
318 |     var ticksTreatment = 'glow';
319 |
320 |     var xAxis = new Rickshaw.Graph.Axis.Time( {
321 |         graph: graph,
322 |         ticksTreatment: ticksTreatment,
323 |         timeFixture: new Rickshaw.Fixtures.Time.Local()
324 |         //timeFixture: fdata
325 |     } );

```

```

326 |
327 | xAxis.render();
328 |
329 | var yAxis = new Rickshaw.Graph.Axis.Y( {
330 |   graph: graph,
331 |   //pixelsPerTick: graph.height/10,
332 |   tickSize: 10,
333 |   tickFormat: Rickshaw.Fixtures.Number.formatKMBT,
334 |   ticksTreatment: ticksTreatment,
335 |
336 | } );
337 |
338 |
339 | yAxis.render();
340 |
341 |
342 | var controls = new RenderControls( {
343 |   element: document.querySelector('form'),
344 |   graph: graph
345 | } );
346 |
347 | // add some data every so often
348 |
349 | var messages = [
350 |   "Changed home page welcome message",
351 |   "Minified JS and CSS",
352 |   "Changed button color from blue to green",
353 |   "Refactored SQL query to use indexed columns",
354 |   "Added additional logging for debugging",
355 |   "Fixed typo",
356 |   "Rewrite conditional logic for clarity",
357 |   "Added documentation for new methods"
358 | ];
359 |
360 | /*setInterval( function() {
361 |   random.removeData(seriesData);
362 |   random.addData(seriesData);
363 |   graph.update();
364 |
365 | }, 3000 );
366 |
367 | function addAnnotation(force) {
368 |   if (messages.length > 0 && (force || Math.random() >= 0.95)) {
369 |     annotator.add(seriesData[2][seriesData[2].length-1].x, messages.shift());
370 |     annotator.update();
371 |   }
372 | }
373 |
374 | addAnnotation(true);
375 | setTimeout( function() { setInterval( addAnnotation, 6000 ) }, 6000 );
376 |
377 | var previewXAxis = new Rickshaw.Graph.Axis.Time({
378 |   graph: preview.previews[0],
379 |   timeFixture: new Rickshaw.Fixtures.Time.Local(),
380 |   ticksTreatment: ticksTreatment
381 | });
382 |
383 | previewXAxis.render();*/
384 |
385 | }
386 |
387 |
388 |
389 | </script>
390 |
391 | </body>

```

Código-fonte G.6: Gráfico dinâmico de temperatura

G.7 Página gerenciadora de receitas da interface web

```

1 | <!DOCTYPE HTML>
2 | <html>
3 | <head>
4 |   <meta charset="UTF-8">
5 |   <title>Receitas</title>
6 |   <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
7 |   <link rel="stylesheet" type="text/css" href="./css/config.css">
8 |   <link rel="icon" type="image/png" href="./img/beer2.png">
9 |   <?php

```

```

10 error_reporting(-1); ini_set("display_errors", "On");//debug on
11
12 function listFiles(){
13     $receitas = scandir("./recipes");//read all recipes
14     $fileId=0;//help variables
15     foreach ($receitas as $valor){//iterates over array, with auto-index
16         if(strpos($valor,".del") == false){//if file is not a backup
17             if(strpos($valor,".recipe") != false){//if current file extension is compatible
18                 //$filePath="./recipes/".$valor;//caminho ate a receita
19                 //edit the name to human readable:
20                 $valor = str_replace(".recipe","", $valor);//exclude extension
21                 $nome = str_replace("_", " ", $valor);//all underlines -> spaces
22                 echo "<div id='recipe".$fileId."><a class='fileList big' href=newrecipe.php?name=$valor>$nome<br></a> ";//echo
23                     the links to the recipes inside identifiable DIV
24                 //echo "<input id='_".$valor.'" class='del' type='button' value='X' onClick='deleteRecipe(this);'></div>";//
25                     create identifiable delete button
26                 echo "<input id='_".$valor.'" class='del' type='button' value='X'></div>";//create identifiable delete button with
27                     recipe name as ID
28                 $fileId++;//increment to set id number of next recipe
29             }
30         }
31     }
32 }
33
34 <script type="text/javascript" src="./lib/listrecipe.js"></script>
35 <script type="text/javascript" src="./lib/header.js"></script>
36 <script type="text/javascript">
37 // POST request using AJAX
38 $(function(){ //wait for the page to load
39     //WANT TO DELETE OR UNDELETE RECIPE
40     $(".del").click(function(){//if any del/undo button is clicked
41         deleteRecipe($(this));//delete or recover recipe
42     });
43     //WANT TO CREATE NEW RECIPE
44     $("#new_add").click(function addRecipeName(){//if want to create recipe
45         $(this).hide();//hide the button to add new recipe
46         $(this).siblings().show(200).css("display","inline-block");//show the text input and add button
47     });
48     $("#new_done").click(function addRecipe(){//if name was put and want to create recipe
49         if($("#new_name").val().trim()){//if name is not empty, replace spaces with underscore signs
50             window.location.href = "newrecipe.php?name="+$("#new_name").val().replace(/ /g, "_");
51         }
52         else{//if name is empty or whitespaces
53             //
54         }
55     });
56     //RECIPE PREVIEW
57     $(".fileList").mouseover(function(){//if needs to load the recipe
58         recipePreview($(this));//pass its name (through the current element)
59     });
60     $(".fileList").mouseout(function(){//when mouse leaves recipe name
61         $("#preview").hide();
62     });
63     headerPHP("./lib/header.php");//add header
64 });
65 </script>
66 </head>
67 <body style="display:none;">
68 <h1>Gerenciador de Receitas</h1>
69 <div>
70     <input id="new_add" class="add" type="button" value="Criar nova receita"/>
71     <input id="new_name" class="long" type="text" maxlength="50" placeholder="Escreva o nome da sua receita" style="display:none;"/>
72     <input id="new_done" class="enter" type="button" value="+" style="display:none;"/>
73 </div>
74 <div id="status_message" style="display:none;">
75     <p style="width:30%;display:inline-block;">Teste</p>
76 </div>
77 <section>
78     <div id="recipeList" style="display:inline-block; width:50%; float:left;">
79         <p>Receitas Cadastradas:<br></p>
80         <?php listFiles();?>
81         <br><br><br><br>
82     </div>
83     <div id="preview" style="display:none; width:50%; float:left; background:#595450; border-radius:10px;">
84         <p class="prevhead">Nome da Receita:</p><p class="prev" id="nome_da_receita"></p><br>
85         <p class="prevhead">Estilo:</p><p class="prev" id="estilo"></p><br>
86         <p class="prevhead">Levedura:</p><p class="prev" id="levedura"></p><br>
87         <p class="prevhead">Água de mosturação (l):</p><p class="prev" id="mosto"></p><br>
88         <p class="prevhead">Água de lavagem (l):</p><p class="prev" id="lavagem"></p><br>
89         <p class="prevhead">Tempo de fervura (min):</p><p class="prev" id="fervura"></p><br>
90         <p class="prevhead">Maltes:</p><p class="prev" id="maltes"></p><br>
91         <p class="prevhead">Lúpulos:</p><p class="prev" id="lupulos"></p>
92     </div>
93 </section>

```

```

94 </body>
95 </html>

```

Código-fonte G.7: Página gerenciadora de receitas da interface web

G.8 Funções auxiliares do gerenciador de receitas

```

1 function deleteRecipe(element){//if any del/undo button is clicked
2 //var recipeId = $(this).parent().attr("id");//save the parent ID so it can be used in callback if necessary
3 var btnId = $(element).attr("id");//save the button ID too
4 var whatToDo = "undo";//variable points which action should take, by default undo
5 if($(element).attr("value") == "X"){//if intends to delete the recipe
6     whatToDo = "delete";//points it should be deleted
7 }
8 //http POST request, to file deleterecipe.php, sends the recipe to be deleted info, and callback on success
9 $.post("../lib/deleterecipe.php", {recipe:btnId, stat:whatToDo}, function(data, status){
10     if(data == 0 && whatToDo == "delete"){//check if recipe was successfully deleted
11         $("#"+ btnId).attr("value","desfazer");//the user have the option to undelete the recipe
12     }
13     else if(data == 0 && whatToDo == "undo"){//check if recipe was successfully deleted
14         $("#"+ btnId).attr("value","X");//the user have the option to undelete the recipe
15     }
16     else{//tell the user there was some error
17         $("#status_message").children("p").text("Receita não foi deletada. Erro " + data);//place status message in HTML
18             hidden paragraph
19         $("#status_message").show(500);//shows the paragraph with the status message with fade-in effect
20     }
21 });
22 }
23 function recipePreview(element){//when user put mouse into recipe name
24     var fileName = $(element).attr("href").split("?")[1].split("=")[1];//get recipe name
25     $("#preview").show().css("display","inline-block");
26     $.post("../lib/previewrecipe.php", {name:fileName}, function(data, status){//request recipe contents to server
27         if(status == "success"){
28             $("#preview").children().each(function setNameInHTML(){
29                 var idOfElement = $(this).attr("id");
30                 $(this).text(data[idOfElement]);//put content inside corresponding HTML element
31                 if(idOfElement == "maltes"){//put all malts in the same div
32                     $(this).text("");//empty the text inside MALTES
33                     if(data["malte" + 1]){//if property is not empty
34                         $(this).append(data["malte" + 1].replace(" ", "&nbsp;"));//append content to MALTES
35                     }
36                     for (var i = 2; i < 8; i++){//iterate though all malts
37                         if(data["malte" + i]){//if property is not empty
38                             $(this).append("&nbsp;" + data["malte" + i].replace(" ", "&nbsp;"));//append content to MALTES
39                         }
40                     }
41                 }
42                 if(idOfElement == "lupulos"){//put all malts in the same div
43                     $(this).text("");//empty the text inside LUPULOS
44                     if(data["lupulo" + 1]){//if property is not empty
45                         $(this).append(data["lupulo" + 1].replace(" ", "&nbsp;"));//append content to LUPULOS
46                     }
47                     for (i = 2; i < 8; i++){//iterate though all malts
48                         if(data["lupulo" + i]){//if property is not empty
49                             $(this).append("&nbsp;" + data["lupulo" + i].replace(" ", "&nbsp;"));//append content to LUPULOS
50                         }
51                     }
52                 }
53             });
54         }
55         else{
56             console.log("oops failed!");
57         }
58     }, "json");//server return data as JSON encoded
59 }

```

Código-fonte G.8: Funções auxiliares do gerenciador de receitas

G.9 Código HTML do editor de receitas

```

1 <!DOCTYPE HTML>
2 <html>

```

```

3 <head>
4 <meta charset="UTF-8">
5 <title>Nova Receita</title>
6 <link rel="stylesheet" type="text/css" href="./css/config.css">
7 <link rel="icon" type="image/png" href="./img/beer2.png">
8 <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
9 <script type="text/javascript" src="./lib/newrecipe.js"></script>
10 <script type="text/javascript" src="./lib/header.js"></script>
11 </script>
12 $(function(){//wait for the page to load
13     headerPHP("./lib/header.php");//add the header
14
15     rearrangeAll();//show all filled fields
16     $("#name").val(getNameFromURL().replace(/_/g, " "));//set the recipe name got from GET
17     $(".myform").focusout(function(){//everytime the form focus change
18         rearrange(fieldsToRearrange(event.target.id));//rearrange the part of the form that was changed
19         saveOnDemand($(this));//save the form
20     });
21
22 });
23 </script>
24 </head>
25
26 <body style="display:none">
27 <h1>Crie ou edite sua receita!</h1>
28 <p id="statusMsg" style="text-align:right;">receita salva</p>
29
30
31 <?php
32     error_reporting(-1);//set debug options on
33     ini_set("display_errors", "On");//set debug options on
34     require './lib/newrecipe.php';//file that does the magic, i.e. form handling, file saving, etc
35     $all_variables = loadFormData();//load recipe if there is a GET name
36     ?>
37
38 <form class="myform" id="form" name="form" method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
39     <div>
40         <label class="myform long" for="name">Nome da Receita:</label>
41         <input class="myform long" type="text" id="name" maxlength="50" name="nome_da_receita" readonly>
42     </div>
43     <div>
44         <label class="myform long" for="beerStyle">Estilo da Cerveja:</label>
45         <input class="myform long" type="text" id="beerStyle" maxlength="50" name="estilo" value="<?php echo $all_variables["estilo"];?>"><span class="error">* <?php echo $estiloErr;?></span><br>
46     </div>
47     <div>
48         <label class="myform long" for="yeast">Levedura:</label>
49         <input class="myform long" type="text" id="yeast" maxlength="50" name="levedura" value="<?php echo $all_variables["levedura"];?>"><span class="error">* <?php echo $leveduraErr;?></span><br>
50     </div>
51     <div>
52         <label class="myform long" for="mash_water">Água - mosturação(l):</label>
53         <input class="myform long" type="number" id="mash_water" min="0" max="50" step="0.5" name="mosto" value="<?php echo $all_variables["mosto"];?>"><span class="error">* <?php echo $mostoErr;?></span><br>
54     </div>
55     <div>
56         <label class="myform long" for="sparge_water">Água - <i>sparging</i> (l):</label>
57         <input class="myform long" type="number" id="sparge_water" min="0" max="50" step="0.5" name="lavagem" value="<?php echo $all_variables["lavagem"];?>">
58         <label class="myform short" for="sparge_temp">Temperatura:</label>
59         <input class="myform short" type="number" id="sparge_temp" min="0" max="99" step="0.5" name="tlavagem" value="<?php echo $all_variables["tlavagem"];?>"><span class="error">* <?php echo $lavagemErr;?></span><br>
60     </div>
61     <div>
62         <label class="myform long" for="boil_time">Fervura do mosto(min):</label>
63         <input class="myform long" type="number" id="boil_time" min="0" max="300" step="1" name="fervura" value="<?php echo $all_variables["fervura"];?>"><span class="error">* <?php echo $fervuraErr;?></span><br>
64     </div>
65
66 <h4>Maltes</h4>
67 <div id="m1t1">
68     <label class="myform long" for="nm1t1">Malte 1:</label>
69     <input class="myform long" type="text" id="nm1t1" maxlength="25" name="malte1" value="<?php echo $all_variables["malte1"];?>">
70     <label class="myform short" for="qm1t1">Quantidade (kg):</label>
71     <input class="myform short" type="number" id="qm1t1" min="0" max="100" step="0.01" name="qtd1" value="<?php echo $all_variables["qtd1"];?>"><span class="error">* <?php echo $malte1Err;?></span>
72 </div>
73 <div id="m1t2" style="display:none">
74     <label class="myform long" for="nm1t2">Malte 2:</label>
75     <input class="myform long" type="text" id="nm1t2" maxlength="25" name="malte2" value="<?php echo $all_variables["malte2"];?>">
76     <label class="myform short" for="qm1t2">Quantidade (kg):</label>
77     <input class="myform short" type="number" id="qm1t2" min="0" max="100" step="0.01" name="qtd2" value="<?php echo $all_variables["qtd2"];?>">
78 </div>
79 <div id="m1t3" style="display:none">
80     <label class="myform long" for="nm1t3">Malte 3:</label>

```

```

82 <input class="myform long" type="text" id="nmlt3" maxlength="25" name="malte3" value="<?php echo $all_variab
83 <label class="myform short" for="qmlt3">Quantidade (kg):</label>
84 <input class="myform short" type="number" id="qmlt3" min="0" max="100" step="0.01" name="qtd3" value="<?php echo
85 $all_variab[ "qtd3" ];?>">
86 </div>
87 <div id="mlt4" style="display:none;">
88 <label class="myform long" for="nmlt4">Malte 4:</label>
89 <input class="myform long" type="text" id="nmlt4" maxlength="25" name="malte4" value="<?php echo $all_variab[ "malte4
90 <label class="myform short" for="qmlt4">Quantidade (kg):</label>
91 <input class="myform short" type="number" id="qmlt4" min="0" max="100" step="0.01" name="qtd4" value="<?php echo
92 $all_variab[ "qtd4" ];?>">
93 </div>
94 <div id="mlt5" style="display:none;">
95 <label class="myform long" for="nmlt5">Malte 5:</label>
96 <input class="myform long" type="text" id="nmlt5" maxlength="25" name="malte5" value="<?php echo $all_variab[ "malte5
97 <label class="myform short" for="qmlt5">Quantidade (kg):</label>
98 <input class="myform short" type="number" id="qmlt5" min="0" max="100" step="0.01" name="qtd5" value="<?php echo
99 $all_variab[ "qtd5" ];?>">
100 </div>
101 <div id="mlt6" style="display:none;">
102 <label class="myform long" for="nmlt6">Malte 6:</label>
103 <input class="myform long" type="text" id="nmlt6" maxlength="25" name="malte6" value="<?php echo $all_variab[ "malte6
104 <label class="myform short" for="qmlt6">Quantidade (kg):</label>
105 <input class="myform short" type="number" id="qmlt6" min="0" max="100" step="0.01" name="qtd6" value="<?php echo
106 $all_variab[ "qtd6" ];?>">
107 </div>
108 <div id="mlt7" style="display:none;">
109 <label class="myform long" for="nmlt7">Malte 7:</label>
110 <input class="myform long" type="text" id="nmlt7" maxlength="25" name="malte7" value="<?php echo $all_variab[ "malte7
111 <label class="myform short" for="qmlt7">Quantidade (kg):</label>
112 <input class="myform short" type="number" id="qmlt7" min="0" max="100" step="0.01" name="qtd7" value="<?php echo
113 $all_variab[ "qtd7" ];?>">
114 </div>
115 <div id="mlt8" style="display:none;">
116 <label class="myform long" for="nmlt8">Malte 8:</label>
117 <input class="myform long" type="text" id="nmlt8" maxlength="25" name="malte8" value="<?php echo $all_variab[ "malte8
118 <label class="myform short" for="qmlt8">Quantidade (kg):</label>
119 <input class="myform short" type="number" id="qmlt8" min="0" max="100" step="0.01" name="qtd8" value="<?php echo
120 $all_variab[ "qtd8" ];?>">
121 </div>
122 <h4>Lúpulos</h4>
123 <div id="lup1">
124 <label class="myform long" for="nlup1">Lupulo 1:</label>
125 <input class="myform long" type="text" id="nlup1" maxlength="25" name="lupulo1" value="<?php echo $all_variab[ "
126 lupulo1" ];?>">
127 <label class="myform short" for="qlup1">Quantidade (g):</label>
128 <input class="myform short" type="number" id="qlup1" min="0" max="10000" step="1" name="lqtd1" value="<?php echo
129 $all_variab[ "lqtd1" ];?>">
130 <label class="myform long" for="tlup1">Tempo de adicao (min):</label>
131 <input class="myform short" type="number" id="tlup1" min="0" max="300" step="1" name="tlupulo1" value="<?php echo
132 $all_variab[ "tlupulo1" ];?>"><span class="error">* <?php echo $lupulo1Err;?></span>
133 </div>
134 <div id="lup2" style="display:none;">
135 <label class="myform long" for="nlup2">Lupulo 2:</label>
136 <input class="myform long" type="text" id="nlup2" maxlength="25" name="lupulo2" value="<?php echo $all_variab[ "
137 lupulo2" ];?>">
138 <label class="myform short" for="qlup2">Quantidade (g):</label>
139 <input class="myform short" type="number" id="qlup2" min="0" max="10000" step="1" name="lqtd2" value="<?php echo
140 $all_variab[ "lqtd2" ];?>">
141 <label class="myform long" for="tlup2">Tempo de adicao (min):</label>
142 <input class="myform short" type="number" id="tlup2" min="0" max="300" step="1" name="tlupulo2" value="<?php echo
143 $all_variab[ "tlupulo2" ];?>">
144 </div>
145 <div id="lup3" style="display:none;">
146 <label class="myform long" for="nlup3">Lupulo 3:</label>
147 <input class="myform long" type="text" id="nlup3" maxlength="25" name="lupulo3" value="<?php echo $all_variab[ "
148 lupulo3" ];?>">
149 <label class="myform short" for="qlup3">Quantidade (g):</label>
150 <input class="myform short" type="number" id="qlup3" min="0" max="10000" step="1" name="lqtd3" value="<?php echo

```

```

151 <label class="myform long" for="nlup5">Lupulo 5:</label>
152 <input class="myform long" type="text" id="nlup5" maxlength="25" name="lupulo5" value="<?php echo $all_variables["
153 lupulo5"];?>">
154 <label class="myform short" for="qlup5">Quantidade (g):</label>
155 <input class="myform short" type="number" id="qlup5" min="0" max="10000" step="1" name="lqtd5" value="<?php echo
156 $all_variables["lqtd5"];?>">
157 <label class="myform long" for="tlup5">Tempo de adicao (min):</label>
158 <input class="myform short" type="number" id="tlup5" min="0" max="300" step="1" name="tlupulo5" value="<?php echo
159 $all_variables["tlupulo5"];?>">
160 </div>
161 <div id="lup6" style="display:none;">
162 <label class="myform long" for="nlup6">Lupulo 6:</label>
163 <input class="myform long" type="text" id="nlup6" maxlength="25" name="lupulo6" value="<?php echo $all_variables["
164 lupulo6"];?>">
165 <label class="myform short" for="qlup6">Quantidade (g):</label>
166 <input class="myform short" type="number" id="qlup6" min="0" max="10000" step="1" name="lqtd6" value="<?php echo
167 $all_variables["lqtd6"];?>">
168 <label class="myform long" for="tlup6">Tempo de adicao (min):</label>
169 <input class="myform short" type="number" id="tlup6" min="0" max="300" step="1" name="tlupulo6" value="<?php echo
170 $all_variables["tlupulo6"];?>">
171 </div>
172 <div id="lup7" style="display:none;">
173 <label class="myform long" for="nlup7">Lupulo 7:</label>
174 <input class="myform long" type="text" id="nlup7" maxlength="25" name="lupulo7" value="<?php echo $all_variables["
175 lupulo7"];?>">
176 <label class="myform short" for="qlup7">Quantidade (g):</label>
177 <input class="myform short" type="number" id="qlup7" min="0" max="10000" step="1" name="lqtd7" value="<?php echo
178 $all_variables["lqtd7"];?>">
179 <label class="myform long" for="tlup7">Tempo de adicao (min):</label>
180 <input class="myform short" type="number" id="tlup7" min="0" max="300" step="1" name="tlupulo7" value="<?php echo
181 $all_variables["tlupulo7"];?>">
182 </div>
183 <div id="lup8" style="display:none;">
184 <label class="myform long" for="nlup8">Lupulo 8:</label>
185 <input class="myform long" type="text" id="nlup8" maxlength="25" name="lupulo8" value="<?php echo $all_variables["
186 lupulo8"];?>">
187 <label class="myform short" for="qlup8">Quantidade (g):</label>
188 <input class="myform short" type="number" id="qlup8" min="0" max="10000" step="1" name="lqtd8" value="<?php echo
189 $all_variables["lqtd8"];?>">
190 <label class="myform long" for="tlup8">Tempo de adicao (min):</label>
191 <input class="myform short" type="number" id="tlup8" min="0" max="300" step="1" name="tlupulo8" value="<?php echo
192 $all_variables["tlupulo8"];?>">
193 </div>
194 <h4>Cozimento do mosto</h4>
195 <div id="tpo0">
196 <label class="myform long" for="tperaturaini">Temperatura inicial:</label>
197 <input class="myform short" type="number" id="tperaturaini" name="temperaturaini" min="0" max="99" step="0.5" value="<?
198 php echo $all_variables["temperaturaini"];?>"><span class="error">* <?php echo $tiniErr;?></span>
199 </div>
200 <div id="tpo1">
201 <label class="myform long" for="tperatural">Temperatura 1:</label>
202 <input class="myform short" type="number" id="tperatural" name="temperatural" min="0" max="99" step="0.5" value="<?php
203 echo $all_variables["temperatural"];?>">
204 <label class="myform short" for="tmpo1">Tempo (min):</label>
205 <input class="myform short" type="number" id="tmpo1" min="0" step="1" name="tempo1" value="<?php echo $all_variables["
206 tmpo1"];?>"><span class="error">* <?php echo $tempErr;?></span>
207 </div>
208 <div id="tpo2" style="display:none;">
209 <label class="myform long" for="tperatura2">Temperatura 2:</label>
210 <input class="myform short" type="number" id="tperatura2" name="temperatura2" min="0" max="99" step="0.5" value="<?php
211 echo $all_variables["temperatura2"];?>">
212 <label class="myform short" for="tmpo2">Tempo (min):</label>
213 <input class="myform short" type="number" id="tmpo2" min="0" step="1" name="tempo2" value="<?php echo $all_variables["
214 tmpo2"];?>">
215 </div>
216 <div id="tpo3" style="display:none;">
217 <label class="myform long" for="tperatura3">Temperatura 3:</label>
218 <input class="myform short" type="number" id="tperatura3" name="temperatura3" min="0" max="99" step="0.5" value="<?php
219 echo $all_variables["temperatura3"];?>">
220 <label class="myform short" for="tmpo3">Tempo (min):</label>
221 <input class="myform short" type="number" id="tmpo3" min="0" step="1" name="tempo3" value="<?php echo $all_variables["
222 tmpo3"];?>">
223 </div>
224 <div id="tpo4" style="display:none;">
225 <label class="myform long" for="tperatura4">Temperatura 4:</label>
226 <input class="myform short" type="number" id="tperatura4" name="temperatura4" min="0" max="99" step="0.5" value="<?php
227 echo $all_variables["temperatura4"];?>">
228 <label class="myform short" for="tmpo4">Tempo (min):</label>
229 <input class="myform short" type="number" id="tmpo4" min="0" step="1" name="tempo4" value="<?php echo $all_variables["
230 tmpo4"];?>">
231 </div>
232 <div id="tpo5" style="display:none;">
233 <label class="myform long" for="tperatura5">Temperatura 5:</label>
234 <input class="myform short" type="number" id="tperatura5" name="temperatura5" min="0" max="99" step="0.5" value="<?php
235 echo $all_variables["temperatura5"];?>">
236 <label class="myform short" for="tmpo5">Tempo (min):</label>
237 <input class="myform short" type="number" id="tmpo5" min="0" step="1" name="tempo5" value="<?php echo $all_variables["
238 tmpo5"];?>">
239 </div>
240 <div id="tpo6" style="display:none;">
241 <label class="myform long" for="tperatura6">Temperatura 6:</label>

```

```

220 <input class="myform short" type="number" id="tperatura6" name="temperatura6" min="0" max="99" step="0.5" value="<?php
      echo $all_variables["temperatura6"];?>">
221 <label class="myform short" for="tmpo6">Tempo (min) :</label>
222 <input class="myform short" type="number" id="tmpo6" min="0" step="1" name="tempo6" value="<?php echo $all_variables["
      tempo6"];?>">
223 </div>
224 <div id="tpo7" style="display:none;">
225 <label class="myform long" for="tperatura7">Temperatura 7:</label>
226 <input class="myform short" type="number" id="tperatura7" name="temperatura7" min="0" max="99" step="0.5" value="<?php
      echo $all_variables["temperatura7"];?>">
227 <label class="myform short" for="tmpo7">Tempo (min) :</label>
228 <input class="myform short" type="number" id="tmpo7" min="0" step="1" name="tempo7" value="<?php echo $all_variables["
      tempo7"];?>">
229 </div>
230 <div id="tpo8" style="display:none;">
231 <label class="myform long" for="tperatura8">Temperatura 8:</label>
232 <input class="myform short" type="number" id="tperatura8" name="temperatura8" min="0" max="99" step="0.5" value="<?php
      echo $all_variables["temperatura8"];?>">
233 <label class="myform short" for="tmpo8">Tempo (min) :</label>
234 <input class="myform short" type="number" id="tmpo8" min="0" step="1" name="tempo8" value="<?php echo $all_variables["
      tempo8"];?>">
235 </div>
236 <br><br>
237
238 <input class="myform" type="button" value="Voltar" onClick="window.location='./listreceipe.php'">
239 </form>
240 </body>
241 </html>

```

Código-fonte G.9: Código HTML do editor de receitas

G.10 Código PHP do editor de receitas

```

1 <?php
2 error_reporting(-1);
3 ini_set("display_errors", "On");
4
5 function test_input($data){
6 //verifica integridade dos dados{
7 $data = trim($data); //exclui caracteres desnecessarios
8 $data = stripslashes($data); //exclui backslashes "\"
9 $data = htmlspecialchars($data); //evita receber codigo malicioso
10 return $data;
11 }
12
13 function requireSalva($variables){
14 //funcao criada soh para poder passar os valores para salvareceita.php
15 //require 'salvareceita.php';
16 if(!empty($variables["nome_da_receita"])){//check if recipe has a name
17 $filename = "/var/www/recipes/"; //absolute PATH to the recipe
18 $filename .= str_replace(" ", "_", $variables["nome_da_receita"]); //mount the name
19 $filename .= ".recipe"; //add file extension
20 }
21 else{//if there is no name
22 return '{"status":"1","msg":"Erro ao salvar receita! Sem nome!"}'; //echo error
23 }
24 $s="";
25 foreach($variables as $v){ //put one content in each line
26 $s .= serialize($v).PHP_EOL;
27 }
28
29 $result = file_put_contents($filename, $s);
30 if ($result)
31 {return '{"status":"0","msg":"Receita salva!"}';}
32 else {return '{"status":"1","msg":"Erro ao salvar receita! Não deu pra gravar!"}';}
33 }
34
35 function requireCarrega(){
36 //funcao que carrega os valores salvos na receita
37 $path="./recipes/".$_GET["name"].".recipe";
38 $variables = array( "nome_da_receita" => "", "estilo" => "", "levedura" => "",
39 "mosto" => "", "lavagem" => "", "tlavagem" => "", "fervura" => "", "temperaturaini" => "",
40 "temperatura1" => "", "tempo1" => "", "temperatura2" => "", "tempo2" => "",
41 "temperatura3" => "", "tempo3" => "", "temperatura4" => "", "tempo4" => "",
42 "temperatura5" => "", "tempo5" => "", "temperatura6" => "", "tempo6" => "",
43 "temperatura7" => "", "tempo7" => "", "temperatura8" => "", "tempo8" => "",
44 "malte1" => "", "qtd1" => "", "malte2" => "", "qtd2" => "",
45 "malte3" => "", "qtd3" => "", "malte4" => "", "qtd4" => "",
46 "malte5" => "", "qtd5" => "", "malte6" => "", "qtd6" => "",
47 "malte7" => "", "qtd7" => "", "malte8" => "", "qtd8" => "",
48 "lupulo1" => "", "lqtd1" => "", "tlupulo1" => "",
49 "lupulo2" => "", "lqtd2" => "", "tlupulo2" => "",

```



```

50     "lupulo3" => "", "lqtd3" => "", "tlupulo3" => "",
51     "lupulo4" => "", "lqtd4" => "", "tlupulo4" => "",
52     "lupulo5" => "", "lqtd5" => "", "tlupulo5" => "",
53     "lupulo6" => "", "lqtd6" => "", "tlupulo6" => "",
54     "lupulo7" => "", "lqtd7" => "", "tlupulo7" => "",
55     "lupulo8" => "", "lqtd8" => "", "tlupulo8" => "",
56 );
57 //require 'carregareceita.php';
58 if(!empty($path)){
59     $file=fopen($path,"r");//abre arquivo da receita
60 }
61 else{
62     echo "Erro ao carregar receita!";
63     return ;
64 }
65 $s = file($path, FILE_IGNORE_NEW_LINES);//le arquivo da receita salva
66 $count = 0;//variavel de iteracao
67 foreach($variables as &$v){//apesar de variables ter key => value, usar soh value
68     $v = unserialize($s[$count]);//copia um array no outro
69     $count++;
70 }
71
72 $result = fclose($file);
73 if (!$result)
74 {echo "Erro ao carregar receita!";}
75 return $variables;
76 }
77
78 function loadFormData(){
79     if(isset($_GET["name"])){//check if recipe name was passed through URL
80         $filename = "recipes/".$_GET["name"].".recipe";//mount the filename with relative PATH
81         if (file_exists($filename)){//if it is the first time loading and file exists
82             return requireCarrega();//then load the file values
83         }
84         else{//not sure if anything should be done here
85
86         }
87     }
88 }
89
90 // define variaveis para guardar os dados recebidos e verificados
91 $all_variables = array( "nome_da_receita" => "", "estilo" => "", "levedura" => "",
92     "mosto" => "", "lavagem" => "", "tlavagem" => "", "fervura" => "", "temperaturaini" => "",
93     "temperatura1" => "", "tempo1" => "", "temperatura2" => "", "tempo2" => "",
94     "temperatura3" => "", "tempo3" => "", "temperatura4" => "", "tempo4" => "",
95     "temperatura5" => "", "tempo5" => "", "temperatura6" => "", "tempo6" => "",
96     "temperatura7" => "", "tempo7" => "", "temperatura8" => "", "tempo8" => "",
97     "malte1" => "", "qtd1" => "", "malte2" => "", "qtd2" => "",
98     "malte3" => "", "qtd3" => "", "malte4" => "", "qtd4" => "",
99     "malte5" => "", "qtd5" => "", "malte6" => "", "qtd6" => "",
100    "malte7" => "", "qtd7" => "", "malte8" => "", "qtd8" => "",
101    "lupulo1" => "", "lqtd1" => "", "tlupulo1" => "",
102    "lupulo2" => "", "lqtd2" => "", "tlupulo2" => "",
103    "lupulo3" => "", "lqtd3" => "", "tlupulo3" => "",
104    "lupulo4" => "", "lqtd4" => "", "tlupulo4" => "",
105    "lupulo5" => "", "lqtd5" => "", "tlupulo5" => "",
106    "lupulo6" => "", "lqtd6" => "", "tlupulo6" => "",
107    "lupulo7" => "", "lqtd7" => "", "tlupulo7" => "",
108    "lupulo8" => "", "lqtd8" => "", "tlupulo8" => "",
109 );
110 $nameErr = $estiloErr = $malte1Err = $lupulo1Err = $tempErr = $iniErr = "";
111 $leveduraErr = $mostoErr = $lavagemErr = $fervuraErr = "";
112
113 if ($_SERVER["REQUEST_METHOD"] == "POST"){//se recebeu formulario
114 {
115     //echo json_encode($_POST);//echo content recieved for debug/verification
116     if (empty($_POST["nome_da_receita"])){//verifica campo vazio
117         $nameErr = "Nome eh necessario";//string indicando o erro
118     }
119     else{//se campo nao esta vazio
120         $all_variables["nome_da_receita"] = test_input($_POST["nome_da_receita"]);//testa a integridade do dado
121         if (!preg_match("/^[a-zA-Z 0-9]*$/",$all_variables["nome_da_receita"])){//varifica se eh valido
122             $nameErr = "Somente letras e numeros";//string indicando erro
123         }
124         //else {$nameErr = "";}//livre de erro
125     }
126 }
127
128 if (empty($_POST["estilo"])){
129     $estiloErr = "Estilo eh necessario";
130 }
131 else{
132     $all_variables["estilo"] = test_input($_POST["estilo"]);
133     if (!preg_match("/^[a-zA-Z ]*$/",$all_variables["estilo"])){
134         $estiloErr = "Somente letras";
135     }
136     //else {$estiloErr = "";}
137 }

```

```

137
138 if (empty($_POST["levedura"])){
139     $leveduraErr = "Levedura eh necessario";
140 }
141 else{
142     $all_variables["levedura"] = test_input($_POST["levedura"]);
143     if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variables["levedura"])){
144         $leveduraErr = "Somente letras e numeros";
145     }
146     //else {$leveduraErr = "";}
147 }
148
149 if(empty($_POST["mosto"])){
150     $mostoErr = "Adicione a quantidade de agua do mosto";
151 }
152 else{
153     $all_variables["mosto"] = test_input($_POST["mosto"]);
154     if (!preg_match("/^[0-9.]*$/", $all_variables["mosto"])){
155         $mostoErr = "Somente numeros e virgula";
156     }
157     //else {$mostoErr = "";}
158 }
159
160 if(empty($_POST["lavagem"])){
161     $lavagemErr = "Adicione a quantidade de agua";
162 }
163 else{
164     $all_variables["lavagem"] = test_input($_POST["lavagem"]);
165     if (!preg_match("/^[0-9.]*$/", $all_variables["lavagem"])){
166         $lavagemErr = "Somente numeros e virgula";
167     }
168     //else {$lavagemErr = "";}
169 }
170
171 if(empty($_POST["tlavagem"])){
172     $lavagemErr = "Temperatura eh necessario";
173 }
174 else{
175     $all_variables["tlavagem"] = test_input($_POST["tlavagem"]);
176     if (!preg_match("/^[0-9.]*$/", $all_variables["tlavagem"])){
177         $lavagemErr = "Somente numeros e virgula";
178     }
179     //else {$lavagemErr = "";}
180 }
181
182 if(empty($_POST["fervura"])){
183     $fervuraErr = "Informe o tempo de fervura do mosto";
184 }
185 else{
186     $all_variables["fervura"] = test_input($_POST["fervura"]);
187     if (!preg_match("/^[0-9.]*$/", $all_variables["fervura"])){
188         $fervuraErr = "Somente numeros e virgula";
189     }
190     //else {$fervuraErr = "";}
191 }
192
193 if(empty($_POST["malte1"])){
194     $malte1Err = "Adicione, no minimo, Malte 1";
195 }
196 else{
197     $all_variables["malte1"] = test_input($_POST["malte1"]);
198     if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variables["malte1"])){
199         $malte1Err = "Somente letras e numeros";
200     }
201     //else {$malte1Err = "";}
202 }
203
204 if(empty($_POST["qtd1"])){
205     $malte1Err = "Adicione, no minimo, Malte 1";
206 }
207 else{
208     $all_variables["qtd1"] = test_input($_POST["qtd1"]);
209     if (!preg_match("/^[0-9.]*$/", $all_variables["qtd1"])){
210         $malte1Err = "Somente numeros e virgula";
211     }
212     //else {$malte1Err = "";}
213 }
214
215 $all_variables["malte2"] = test_input($_POST["malte2"]);
216 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variables["malte2"])){
217     $malte1Err="Somente letras e numeros";
218 }
219 //else {$malte1Err = "";}
220 $all_variables["qtd2"] = test_input($_POST["qtd2"]);
221 if (!preg_match("/^[0-9.]*$/", $all_variables["qtd2"])){
222     $malte1Err="Somente numeros e virgula";
223 }
224 //else {$malte1Err = "";}
225
226 $all_variables["malte3"] = test_input($_POST["malte3"]);
227 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variables["malte3"])){
228     $malte1Err="Somente letras e numeros";
229 }
230 //else {$malte1Err = "";}
231 $all_variables["qtd3"] = test_input($_POST["qtd3"]);

```

```

224 if (!preg_match("/^[0-9.]*$/", $all_variáveis["qtd3"]))
225 { $malte1Err="Somente numeros e virgula";}
226 //else { $malte1Err = "";}
227
228 $all_variáveis["malte4"] = test_input($_POST["malte4"]);
229 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variáveis["malte4"]))
230 { $malte1Err="Somente letras e numeros";}
231 //else { $malte1Err = "";}
232 $all_variáveis["qtd4"] = test_input($_POST["qtd4"]);
233 if (!preg_match("/^[0-9.]*$/", $all_variáveis["qtd4"]))
234 { $malte1Err="Somente numeros e virgula";}
235 //else { $malte1Err = "";}
236
237 $all_variáveis["malte5"] = test_input($_POST["malte5"]);
238 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variáveis["malte5"]))
239 { $malte1Err="Somente letras e numeros";}
240 //else { $malte1Err = "";}
241 $all_variáveis["qtd5"] = test_input($_POST["qtd5"]);
242 if (!preg_match("/^[0-9.]*$/", $all_variáveis["qtd5"]))
243 { $malte1Err="Somente numeros e virgula";}
244 //else { $malte1Err = "";}
245
246 $all_variáveis["malte6"] = test_input($_POST["malte6"]);
247 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variáveis["malte6"]))
248 { $malte1Err="Somente letras e numeros";}
249 //else { $malte1Err = "";}
250 $all_variáveis["qtd6"] = test_input($_POST["qtd6"]);
251 if (!preg_match("/^[0-9.]*$/", $all_variáveis["qtd6"]))
252 { $malte1Err="Somente numeros e virgula";}
253 //else { $malte1Err = "";}
254
255 $all_variáveis["malte7"] = test_input($_POST["malte7"]);
256 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variáveis["malte7"]))
257 { $malte1Err="Somente letras e numeros";}
258 //else { $malte1Err = "";}
259 $all_variáveis["qtd7"] = test_input($_POST["qtd7"]);
260 if (!preg_match("/^[0-9.]*$/", $all_variáveis["qtd7"]))
261 { $malte1Err="Somente numeros e virgula";}
262 //else { $malte1Err = "";}
263
264 $all_variáveis["malte8"] = test_input($_POST["malte8"]);
265 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variáveis["malte8"]))
266 { $malte1Err="Somente letras e numeros";}
267 //else { $malte1Err = "";}
268 $all_variáveis["qtd8"] = test_input($_POST["qtd8"]);
269 if (!preg_match("/^[0-9.]*$/", $all_variáveis["qtd8"]))
270 { $malte1Err="Somente numeros e virgula";}
271 //else { $malte1Err = "";}
272
273 if(empty($_POST["lupulo1"]))
274 { $lupulo1Err = "Adicione, no minimo, Lupulo 1";}
275 else
276 { $all_variáveis["lupulo1"] = test_input($_POST["lupulo1"]);
277   if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variáveis["lupulo1"]))
278   { $lupulo1Err = "Somente letras e numeros";}
279   //else { $lupulo1Err = "";}
280 }
281
282 if(empty($_POST["lqtd1"]))
283 { $lupulo1Err = "Adicione, no minimo, Lupulo 1";}
284 else
285 { $all_variáveis["lqtd1"] = test_input($_POST["lqtd1"]);
286   if (!preg_match("/^[0-9.]*$/", $all_variáveis["lqtd1"]))
287   { $lupulo1Err="Somente numeros e virgula";}
288   //else { $lupulo1Err = "";}
289 }
290
291 if(empty($_POST["tlupulo1"]))
292 { $lupulo1Err = "Adicione, no minimo, Lupulo 1";}
293 else
294 { $all_variáveis["tlupulo1"] = test_input($_POST["tlupulo1"]);
295   if (!preg_match("/^[0-9]*$/", $all_variáveis["tlupulo1"]))
296   { $lupulo1Err="Somente numeros";}
297   //else { $lupulo1Err = "";}
298 }
299
300 $all_variáveis["lupulo2"] = test_input($_POST["lupulo2"]);
301 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variáveis["lupulo2"]))
302 { $lupulo1Err="Somente letras e numeros";}
303 //else { $lupulo1Err = "";}
304 $all_variáveis["lqtd2"] = test_input($_POST["lqtd2"]);
305 if (!preg_match("/^[0-9.]*$/", $all_variáveis["lqtd2"]))
306 { $lupulo1Err="Somente numeros e virgula";}
307 //else { $lupulo1Err = "";}
308 $all_variáveis["tlupulo2"] = test_input($_POST["tlupulo2"]);
309 if (!preg_match("/^[0-9]*$/", $all_variáveis["tlupulo2"]))
310 { $lupulo1Err="Somente numeros";}

```

```

311 //else {$lupulo1Err = "";}
312
313 $all_variables["lupulo3"] = test_input($_POST["lupulo3"]);
314 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variables["lupulo3"]))
315 { $lupulo1Err="Somente letras e numeros";}
316 //else {$lupulo1Err = "";}
317 $all_variables["lqtd3"] = test_input($_POST["lqtd3"]);
318 if (!preg_match("/^[0-9]*$/", $all_variables["lqtd3"]))
319 { $lupulo1Err="Somente numeros e virgula";}
320 //else {$lupulo1Err = "";}
321 $all_variables["tlupulo3"] = test_input($_POST["tlupulo3"]);
322 if (!preg_match("/^[0-9]*$/", $all_variables["tlupulo3"]))
323 { $lupulo1Err="Somente numeros";}
324 //else {$lupulo1Err = "";}
325
326 $all_variables["lupulo4"] = test_input($_POST["lupulo4"]);
327 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variables["lupulo4"]))
328 { $lupulo1Err="Somente letras e numeros";}
329 //else {$lupulo1Err = "";}
330 $all_variables["lqtd4"] = test_input($_POST["lqtd4"]);
331 if (!preg_match("/^[0-9]*$/", $all_variables["lqtd4"]))
332 { $lupulo1Err="Somente numeros e virgula";}
333 //else {$lupulo1Err = "";}
334 $all_variables["tlupulo4"] = test_input($_POST["tlupulo4"]);
335 if (!preg_match("/^[0-9]*$/", $all_variables["tlupulo4"]))
336 { $lupulo1Err="Somente numeros";}
337 //else {$lupulo1Err = "";}
338
339 $all_variables["lupulo5"] = test_input($_POST["lupulo5"]);
340 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variables["lupulo5"]))
341 { $lupulo1Err="Somente letras e numeros";}
342 //else {$lupulo1Err = "";}
343 $all_variables["lqtd5"] = test_input($_POST["lqtd5"]);
344 if (!preg_match("/^[0-9]*$/", $all_variables["lqtd5"]))
345 { $lupulo1Err="Somente numeros e virgula";}
346 //else {$lupulo1Err = "";}
347 $all_variables["tlupulo5"] = test_input($_POST["tlupulo5"]);
348 if (!preg_match("/^[0-9]*$/", $all_variables["tlupulo5"]))
349 { $lupulo1Err="Somente numeros";}
350 //else {$lupulo1Err = "";}
351
352 $all_variables["lupulo6"] = test_input($_POST["lupulo6"]);
353 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variables["lupulo6"]))
354 { $lupulo1Err="Somente letras e numeros";}
355 //else {$lupulo1Err = "";}
356 $all_variables["lqtd6"] = test_input($_POST["lqtd6"]);
357 if (!preg_match("/^[0-9]*$/", $all_variables["lqtd6"]))
358 { $lupulo1Err="Somente numeros e virgula";}
359 //else {$lupulo1Err = "";}
360 $all_variables["tlupulo6"] = test_input($_POST["tlupulo6"]);
361 if (!preg_match("/^[0-9]*$/", $all_variables["tlupulo6"]))
362 { $lupulo1Err="Somente numeros";}
363 //else {$lupulo1Err = "";}
364
365 $all_variables["lupulo7"] = test_input($_POST["lupulo7"]);
366 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variables["lupulo7"]))
367 { $lupulo1Err="Somente letras e numeros";}
368 //else {$lupulo1Err = "";}
369 $all_variables["lqtd7"] = test_input($_POST["lqtd7"]);
370 if (!preg_match("/^[0-9]*$/", $all_variables["lqtd7"]))
371 { $lupulo1Err="Somente numeros e virgula";}
372 //else {$lupulo1Err = "";}
373 $all_variables["tlupulo7"] = test_input($_POST["tlupulo7"]);
374 if (!preg_match("/^[0-9]*$/", $all_variables["tlupulo7"]))
375 { $lupulo1Err="Somente numeros";}
376 //else {$lupulo1Err = "";}
377
378 $all_variables["lupulo8"] = test_input($_POST["lupulo8"]);
379 if (!preg_match("/^[a-zA-Z 0-9]*$/", $all_variables["lupulo8"]))
380 { $lupulo1Err="Somente letras e numeros";}
381 //else {$lupulo1Err = "";}
382 $all_variables["lqtd8"] = test_input($_POST["lqtd8"]);
383 if (!preg_match("/^[0-9]*$/", $all_variables["lqtd8"]))
384 { $lupulo1Err="Somente numeros e virgula";}
385 //else {$lupulo1Err = "";}
386 $all_variables["tlupulo8"] = test_input($_POST["tlupulo8"]);
387 if (!preg_match("/^[0-9]*$/", $all_variables["tlupulo8"]))
388 { $lupulo1Err="Somente numeros";}
389 //else {$lupulo1Err = "";}
390
391 if(empty($_POST["temperaturaini"]))
392 { $tiniErr = "Informe a temperatura inicial de cozimento do mosto";}
393 else
394 { $all_variables["temperaturaini"] = test_input($_POST["temperaturaini"]);
395   if (!preg_match("/^[0-9]*$/", $all_variables["temperaturaini"]))
396   { $tiniErr = "Somente numeros e virgula";}
397   //else {$tiniErr = "";}

```

```

398 }
399
400 if(empty($_POST["temperatura1"]))
401 { $tempErr = "Adicione, no minimo, Temperatura 1"; }
402 else
403 { $all_variables["temperatura1"] = test_input($_POST["temperatura1"]);
404   if (!preg_match("/^[0-9.]*$/", $all_variables["temperatura1"]))
405     { $tempErr = "Somente numeros e virgula"; }
406   //else { $tempErr = ""; }
407 }
408
409 if(empty($_POST["tempo1"]))
410 { $tempErr = "Adicione, no minimo, Temperatura 1"; }
411 else
412 { $all_variables["tempo1"] = test_input($_POST["tempo1"]);
413   if (!preg_match("/^[0-9.]*$/", $all_variables["tempo1"]))
414     { $tempErr = "Somente numeros"; }
415   //else { $tempErr = ""; }
416 }
417
418 $all_variables["temperatura2"] = test_input($_POST["temperatura2"]);
419 if (!preg_match("/^[0-9.]*$/", $all_variables["temperatura2"]))
420   { $tempErr="Somente numeros e virgula"; }
421 //else { $tempErr = ""; }
422 $all_variables["tempo2"] = test_input($_POST["tempo2"]);
423 if (!preg_match("/^[0-9.]*$/", $all_variables["tempo2"]))
424   { $tempErr="Somente numeros"; }
425 //else { $tempErr = ""; }
426
427 $all_variables["temperatura3"] = test_input($_POST["temperatura3"]);
428 if (!preg_match("/^[0-9.]*$/", $all_variables["temperatura3"]))
429   { $tempErr="Somente numeros e virgula"; }
430 //else { $tempErr = ""; }
431 $all_variables["tempo3"] = test_input($_POST["tempo3"]);
432 if (!preg_match("/^[0-9.]*$/", $all_variables["tempo3"]))
433   { $tempErr="Somente numeros"; }
434 //else { $tempErr = ""; }
435
436 $all_variables["temperatura4"] = test_input($_POST["temperatura4"]);
437 if (!preg_match("/^[0-9.]*$/", $all_variables["temperatura4"]))
438   { $tempErr="Somente numeros e virgula"; }
439 //else { $tempErr = ""; }
440 $all_variables["tempo4"] = test_input($_POST["tempo4"]);
441 if (!preg_match("/^[0-9.]*$/", $all_variables["tempo4"]))
442   { $tempErr="Somente numeros"; }
443 //else { $tempErr = ""; }
444
445 $all_variables["temperatura5"] = test_input($_POST["temperatura5"]);
446 if (!preg_match("/^[0-9.]*$/", $all_variables["temperatura5"]))
447   { $tempErr="Somente numeros e virgula"; }
448 //else { $tempErr = ""; }
449 $all_variables["tempo5"] = test_input($_POST["tempo5"]);
450 if (!preg_match("/^[0-9.]*$/", $all_variables["tempo5"]))
451   { $tempErr="Somente numeros"; }
452 //else { $tempErr = ""; }
453
454 $all_variables["temperatura6"] = test_input($_POST["temperatura6"]);
455 if (!preg_match("/^[0-9.]*$/", $all_variables["temperatura6"]))
456   { $tempErr="Somente numeros e virgula"; }
457 //else { $tempErr = ""; }
458 $all_variables["tempo6"] = test_input($_POST["tempo6"]);
459 if (!preg_match("/^[0-9.]*$/", $all_variables["tempo6"]))
460   { $tempErr="Somente numeros"; }
461 //else { $tempErr = ""; }
462
463 $all_variables["temperatura7"] = test_input($_POST["temperatura7"]);
464 if (!preg_match("/^[0-9.]*$/", $all_variables["temperatura7"]))
465   { $tempErr="Somente numeros e virgula"; }
466 //else { $tempErr = ""; }
467 $all_variables["tempo7"] = test_input($_POST["tempo7"]);
468 if (!preg_match("/^[0-9.]*$/", $all_variables["tempo7"]))
469   { $tempErr="Somente numeros"; }
470 //else { $tempErr = ""; }
471
472 $all_variables["temperatura8"] = test_input($_POST["temperatura8"]);
473 if (!preg_match("/^[0-9.]*$/", $all_variables["temperatura8"]))
474   { $tempErr="Somente numeros e virgula"; }
475 //else { $tempErr = ""; }
476 $all_variables["tempo8"] = test_input($_POST["tempo8"]);
477 if (!preg_match("/^[0-9.]*$/", $all_variables["tempo8"]))
478   { $tempErr="Somente numeros"; }
479 //else { $tempErr = ""; }
480
481
482 //if(($nameErr == "") and ($estiloErr == "") and ($malte1Err == "") and ($lupulo1Err == "") and ($tempErr == "")
    and ($tiniErr == "") and ($leveduraErr == "") and ($mostoErr == "") and ($lavagemErr == "") and ($fervuraErr
    == "")){

```

```

483 | if($nameErr === ""){//só não salva se o nome estiver errado, se for outra coisa, salva mesmo assim
484 | $sts = json_decode(requireSalva($all_variables),true);//se não tem nenhum erro, salva
485 | if($estiloErr || $maltelErr || $lupulolErr || $tempErr || $tiniErr || $leveduraErr || $mostoErr || $lavagemErr ||
    | $fervuraErr){
486 |     $sts["msg"] = "Receita salva, mas há campos necessários não preenchidos.";
487 | }
488 | echo json_encode($sts);
489 | }
490 | else{
491 |     echo "Falha ao salvar, verifique possíveis erros!\n";
492 |     echo "$nameErr, $estiloErr, $maltelErr, $lupulolErr, $tempErr, $tiniErr, $leveduraErr, $mostoErr, $lavagemErr,
    | $fervuraErr";
493 | }
494 | }
495 |
496 | ?>

```

Código-fonte G.10: Código PHP do editor de receitas

G.11 Código javascript do editor de receitas

```

1 | function fieldsToRearrange(lastActive){
2 |     if(lastActive.indexOf("mlt") > 0){//if it was a malt field that changed
3 |         return "#mlt";//return malt id as string
4 |     }
5 |     else if(lastActive.indexOf("lup") > 0){//if it was a hop field that changed
6 |         return "#lup";//return hop id as string
7 |     }
8 |     else{//if it was any other field that changed
9 |         return "#tpo";//return temperature id string
10 |     }
11 | }
12 |
13 | function rearrange(catg, line){//rearrange from LINE passed to the last line
14 |     //the first line should be passed as value zero!
15 |     var nextLineEmpty = 0;//zero if all next fields are empty
16 |     var firstLineEmpty = 0;//zero if first line is empty, 2 if fully filled
17 |     if(typeof line === "undefined") {line = 0;}//optional parameter, defaults to zero
18 |     if(typeof line === "undefined") {return;}//mandatory parameter, should be passed
19 |     console.log("catg: " + catg + "; line: " + line);
20 |     line++;//the first line isn't zero indexed
21 |     console.log("starting line: " + line);
22 |     /*if(line == -1){//if last call was the last needed call
23 |         console.log("nothing more to do - the end " + line);
24 |         return -1;//return success, nothing more to do
25 |     }*/
26 |     for(var i = line; i < 8; i++){//do it for all the lines, except the last
27 |         $(catg + (i)).children("input").each(function checkIfEmpty(){
28 |             if($(this).val()){//if input isn't empty
29 |                 if(i == line){//if it is the first line
30 |                     firstLineEmpty++;//tells it isn't empty
31 |                 }
32 |                 else{//if it is one of the next lines
33 |                     if(!nextLineEmpty){//if it is still empty
34 |                         nextLineEmpty = i;//variable non-zero tells next non-empty line
35 |                     }
36 |                 }
37 |             }
38 |         });
39 |     }
40 |     console.log("First line - " + firstLineEmpty);
41 |     console.log("Next line - " + nextLineEmpty);
42 |     if(!nextLineEmpty){//if all next lines are empty
43 |         if(!firstLineEmpty){//if a line that wasn't the last filled line is deleted
44 |             line--;//correct the index, so it points to the last filled line
45 |         }
46 |         else if(firstLineEmpty != $(catg + (line+1)).children().length/2){//if last filled line is incomplete
47 |             $(catg + (line+1)).hide();//do not show the next empty line
48 |         }
49 |         else{//but if last filled line is fully filled
50 |             $(catg + (line+1)).show();//show the next line to the user
51 |         }
52 |         for(i = 1; i <= line; i++){//show the non-empty lines
53 |             $(catg + i).show();
54 |         }
55 |         for(i = (line+2); i <= 8; i++){//hide the other empty lines
56 |             $(catg + i).hide();
57 |         }
58 |         console.log("nothing more to do - " + line);
59 |         return -1;//success, do nothing more
60 |     }

```

```

61 | else{//if there are next line(s) not empty
62 |     if(!firstLineEmpty){//and first line is empty
63 |         //copy next line value and erase from next line
64 |         $("#nmlt" + line).val($("#nmlt" + (nextLineEmpty)).val());
65 |         $("#nmlt" + (nextLineEmpty)).val("");
66 |         $("#qmlt" + line).val($("#qmlt" + (nextLineEmpty)).val());
67 |         $("#qmlt" + (nextLineEmpty)).val("");
68 |         console.log("Current line empty rearranged, try next line");
69 |         rearrange(catg, line);//do it again for the next line
70 |     }
71 |     else{//if first line have contents already
72 |         console.log("Current line not empty, try next line");
73 |         rearrange(catg, line);//try rearranging next line
74 |     }
75 | }
76 | }
77 |
78 | function rearrangeAll(){
79 |     rearrange("#mlt");
80 |     rearrange("#lup");
81 |     rearrange("#tpo");
82 | }
83 |
84 | function getNameFromURL(){
85 |     var sPageURL = window.location.search.substring(1);//get URL
86 |     return sPageURL.split('=')[1];
87 | }
88 |
89 | function saveOnDemand(element){
90 |     console.log($(element));
91 |     //var dataInput = $(this).serialize();
92 |     var dataRaw = $(element).serializeArray();
93 |     var dataInput = {};
94 |     $.each(dataRaw, function(index, value){//transform to object with pair key:value
95 |         //console.log("nome: " + value.name + " valor: " + value.value);
96 |         dataInput[value.name] = value.value;
97 |     });
98 |     $("#statusMsg").text("salvando receita...");//tell the user that the recipe is being saved
99 |     $.post("./lib/newrecipe.php", dataInput, function(data, status){//send form data to php
100 |         console.log(data);
101 |         if(status == "success"){//if could contact server
102 |             if(data.status == 0){//if successfully saved
103 |                 $("#statusMsg").text(data.msg);
104 |             }
105 |             else{//otherwise
106 |                 $("#statusMsg").text(data.msg);
107 |             }
108 |         }
109 |         else{
110 |             $("#statusMsg").text("erro ao contatar servidor");
111 |         }
112 |     }, "json");
113 | }

```

Código-fonte G.11: Código javascript do editor de receitas

G.12 Gerenciador de brassagem

```

1 | <!DOCTYPE HTML>
2 | <html>
3 | <head>
4 |     <meta charset="UTF-8">
5 |     <title>Controle</title>
6 |     <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
7 |     <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">
8 |     <link rel="stylesheet" type="text/css" href="./css/config.css">
9 |     <link rel="stylesheet" type="text/css" href="./css/buttons.css">
10 |    <link rel="icon" type="image/png" href="./img/beer2.png">
11 |    <script type="text/javascript" src="./lib/header.js"></script>
12 |    <script type="text/javascript" src="./lib/listrecipe.js"></script>
13 | <script>
14 |     checkRecipeInProgress(recipeInProgress);//it may be done before page is loaded, no worries
15 |     $(function main(){//when document is fully loaded
16 |         headerPHP("./lib/header.php");
17 |         getAvailableRecipes("#recipeSel");
18 |
19 |         $("#input[type='button'][value='iniciar']").click(startRecipeRequest);
20 |     });
21 |
22 |     function checkRecipeInProgress(callback){

```

```

23     $.post("/startrecipe", {command:"inProgress", recipeInProgress,"json"};
24 }
25
26 function recipeInProgress(data, status){
27     if(status == "success"){//if server responds ok
28     if(data.resp == "true"){//if the recipes are successfully recieved
29         window.location.replace("http://beaglebrewing.servebeer.com:8587/control.php");
30     }
31 }
32
33 }
34
35 function startRecipeRequest(){
36     var recipeName = $("#recipeSel").val().replace(/ /g, "_");
37     $("#previewData").attr("href","?" + recipeName);//replace spaces with underlines
38     recipePreview($("#previewData"));//pass its name (through the current element)
39     setTimeout(function(){
40         $.post("/startrecipe", {command:"startRequest", recipe:recipeName}, function(data, status){//ask the server for
the recipe names
41         if(status == "success"){//if server responds ok
42         if(data.resp == "success"){//if the recipes are successfully recieved
43             console.log(data);
44             if(errorWarningHandler(data, "#errors", "#warnings", "#messages")){//if recipe can be started
45                 startRecipe(recipeName);
46                 //window.location.replace("http://beaglebrewing.servebeer.com:8587/control.php?start=true");
47             }
48         }
49     }
50 }, "json");
51 }, 1000);
52 }
53
54 function startRecipe(recipeName){
55     console.log("Recipe name from startRecipe: " + recipeName);
56     $.post("/startrecipe", {command:"startRecipe", recipe:recipeName}, function(data, status){
57         if(status == "success"){//if server responds ok
58             console.log(data);
59             if(data.resp == "success"){//if the recipes are successfully recieved
60                 console.log("starting the requested recipe...");
61                 window.location.replace("http://beaglebrewing.servebeer.com:8587/control.php");
62             }
63         }
64     }, "json");
65 }
66
67 function getAvailableRecipes(recipeSelDivId){
68     $.post("/startrecipe", {command:"getRecipes"}, function(data, status){//ask the server for the recipe names
69     if(status == "success"){//if server responds ok
70         if(data.resp == "success"){//if the recipes are successfully recieved
71             for (var i = 0; i < data.recipes.length; i++){//create one option for each recipe in the dropdown list
72                 //console.log(data.recipes[i]);
73                 $(recipeSelDivId).append("<option value='"+ + data.recipes[i] + "'> " + data.recipes[i] + "</option>");
74             }
75         }
76         else if(data.resp == "error"){
77
78         }
79         //console.log(data);
80     }
81 }, "json");
82 }
83
84 function errorWarningHandler(data, errDivId, warnDivId, msgDivId){
85     if(data.err && data.warn){//if there are errors and warnings to give
86         $(errDivId).text("Essenciais: " + data.err).show();
87         $(warnDivId).text("Facultativos: " + data.warn).show();
88         $(msgDivId).show();
89         return 0;//don't start the recipe
90     }
91     else if(data.err){//if there are only erros
92         $(errDivId).text("Essenciais: " + data.err).show();
93         $(msgDivId).show();
94         return 0;//don't start the recipe
95     }
96     else if(data.warn){//if there are only warnings
97         $(warnDivId).text("Facultativos: " + data.warn).show();
98         $(msgDivId).text("Alguns itens da receita não foram preenchidos."
99         + "Deseja continuar mesmo assim?").show();
100         if (confirm("Alguns itens da receita não foram preenchidos."
101         + "Deseja realmente iniciar?") == true) {
102             return 1;//start the recipe
103         }
104         else{
105             return 0;//don't start the recipe
106         }
107     }
108 }
else{//if everything is right, start the recipe

```



```

109         return 1;//recipe ready to start
110     }
111 }
112 </script>
113 </head>
114
115 <body style="display:none;">
116     <h1>Iniciar Brassagem</h1>
117     <div class="warning">
118         <div ><i class="material-icons custom">warning</i></div>
119         <p class="warning">Antes de iniciar uma receita, certifique-se de que o equipamento está
120             devidamente limpo e sanitizado, e de que a água e ingredientes estão a postos para a produção.
121         </p>
122     </div>
123     <select id="recipeSel"></select><br><br>
124     <input class="leftselect" type="button" value="iniciar"/>
125
126     <div id="preStart">
127         <p id="messages" style="display:none;">Alguns itens da receita não foram preenchidos. Preencha os
128             obrigatórios antes de continuar.</p>
129         <p id="warnings" style="display:none;"></p>
130         <p id="errors" style="display:none;"></p>
131
132         <div id="preview" style="display:none; width:50%; float:left; background:#595450; border-radius:10px;">
133             <p class="prevhead">Nome da Receita:</p><p class="prev" id="nome_da_receita"></p><br>
134             <p class="prevhead">Estilo:</p><p class="prev" id="estilo"></p><br>
135             <p class="prevhead">Levedura:</p><p class="prev" id="levedura"></p><br>
136             <p class="prevhead">Água de mosturação (l):</p><p class="prev" id="mosto"></p><br>
137             <p class="prevhead">Água de lavagem (l):</p><p class="prev" id="lavagem"></p><br>
138             <p class="prevhead">Tempo de fervura (min):</p><p class="prev" id="fervura"></p><br>
139             <p class="prevhead">Maltes:</p><p class="prev" id="maltes"></p><br>
140             <p class="prevhead">Lúpulos</p><p class="prev" id="lupulos"></p>
141         </div>
142     </div>
143
144     <a id="previewData" style="display:none;"></a>
145
146     <h2>Controle do sistema</h2>
147     <p>Ao iniciar uma receita você será automaticamente redirecionado para o controle.</p>
148     <p>É possível acessar os controles a qualquer momento:</p>
149     <input class="leftselect" type="button" value="controle" onClick="window.location='./control.php' />
150
151 </body>
152 </html>

```

Código-fonte G.12: Gerenciador de brassagem

G.13 Painel de controle de brassagem

```

1  <!DOCTYPE HTML>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <title>Controle</title>
6      <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
7      <link rel="stylesheet" type="text/css" href="./css/config.css">
8      <link rel="icon" type="image/png" href="./img/beer2.png">
9      <script type="text/javascript" src="./lib/header.js"></script>
10     <script>
11         $(function() { //when document is fully loaded
12             headerPHP("./lib/header.php"); //add the header
13             refreshSystemStatus(); //get the status fo the GPIO, PWM, etc
14             var refreshHandler = setInterval(refreshSystemStatus, 500); //and then do it periodically
15
16             $(".btn").click(function() { console.log($(this).attr("id")); });
17             //valveToggle;
18
19             $("#confirmStartNextStep").click(function startMashRamp() { //whenever button is clicked
20                 $.post("/clientrequest", {command:"startMashRamp"}, function(data, status) {
21                     if(status == "success") { //if server responds ok
22                         console.log("Starting the ramp control process");
23                     }
24                 }, "json");
25             });
26
27             //Display the slider value dinamicallly
28             $(".slider").on("input", function() { //whenever user is changing value
29                 $(".slider-value").html($(this).val() + "°"); //change the display value
30             });
31             $(".slider").on("change", function() {

```

```

32     $.post("/controle", {command:"pinSwitch", btn:$(this).attr("id"), val:$(this).val()}, function(data, status){
33         if(status == "success"){//if server responds ok
34             console.log("Server - ID: " + data.btn + "; VALUE: " + data.val);
35         }
36     }, "json");
37 });
38
39
40 function valveToggle(){//whenever button is clicked
41     //Post the button ID and VALUE
42     $.post("/controle", {command:"pinSwitch", btn:$(this).attr("id"), val:$(this).is(":checked")}, function(data,
43         status){
44         if(status == "success"){//if server responds ok
45             console.log("Server - ID: " + data.btn + "; VALUE: " + data.val);
46         }
47     }, "json");
48 }
49
50 function refreshSystemStatus(){
51     $.post("/controle", {command:"getStatus"}, function(data, status){//ask for the server status
52         var degreeAngle;//var to temporarily store the servo_motor angle in degree
53         if(status == "success"){//if server responds ok
54             for(var obj in data.ioStatus){//iterate through all object keys
55                 if(data.ioStatus[obj].state == 1 && obj != "servo_pwm"){//if it is set and isn't the pwm
56                     $("##" + obj).prop("checked", true);//check the corresponding checkbox
57                 }
58                 else if(data.ioStatus[obj].state == 0 && obj != "servo_pwm"){//if it is clear and isn't the pwm
59                     $("##" + obj).prop("checked", false);//check the corresponding checkbox
60                 }
61                 else if(obj == "servo_pwm"){//if it is the pwm
62                     degreeAngle = Math.round((data.ioStatus[obj].state.duty - 0.0325)*180/0.0775);//calculate the pwm angle
63                     from duty to degree
64                     $("##" + obj).val(degreeAngle);//set the slider position by setting its value
65                     $(".slider-value").html(degreeAngle + "°");//refresh the indicator value
66                     $(".pwm_slider").show();//then show the div correctly set
67                 }
68             }
69             if(data.auto){//tells the process is in automatic mode
70                 $("#auto").prop("checked", true);
71                 $(".btn").off("click");
72             }
73             else{//automatic mode turned off
74                 $("#auto").prop("checked", false);
75                 $(".btn").off("click");//does it otherwise many click events are set per button
76                 $(".btn").on("click", valveToggle);
77             }
78             updateStatusMessage(data);
79         }, "json");
80     }
81 }
82
83 function updateStatusMessage(data){
84     if(!data.processFail){//if the process is running smoothly
85         if(data.code){//if something is going on
86             switch(data.code){
87                 case 2://mash water being heated
88                     $("#current_status").text("Esquentando água da brassagem: ");
89                     $("#current_status_helper").html(data.tmpMT + "°C &rarr; " + data.tmpMTsetp + "°C");
90                     break;
91                 case 3://waiting for the user to add the grains
92                     $("#current_status").text("Adicione os maltes e clique em prosseguir!");
93                     $("#current_status_helper").text("");
94                     $(".confirmationButton").show();//wait for the user to click this button to continue
95                     break;
96                 case 4://if the ramp control is going on
97                     $(".confirmationButton").hide();
98                     $("#current_status").text("Esquentando mosto: ");
99                     $("#current_status_helper").html(data.tmpMT + "°C &rarr; " + data.tmpMTsetp + "°C");
100                     break;
101                 case 5://if the step rest is going on
102                     $("#current_status").text("Degrau de repouso: ");
103                     if(data.tmpBKsetp){//if sparging is set
104                         if(data.timeLeft >= 1){
105                             console.log("minutos");
106                             $("#current_status_helper").html(data.tmpMT + "°C &rarr; " +
107                                 Math.floor(data.timeLeft) + ":" + (data.timeLeft % 1)*60 +
108                                 " minutos restantes. Temperatura de sparging: " + data.tmpBK + "°C");
109                         }
110                         else{
111                             console.log("segundos");
112                             $("#current_status_helper").html(data.tmpMT + "°C &rarr; " +
113                                 Math.round((data.timeLeft % 1)*60) + " segundos restantes. " +
114                                 "Temperatura de sparging: " + data.tmpBK + "°C");
115                         }
116                     }
117                     else{//no sparging, no sparging water being heated
118                         if(data.timeLeft >= 1){

```

```

117         $("#current_status_helper").html(data.tmpMT + "°C &arr; " +
118         Math.floor(data.timeLeft) + ":" + (data.timeLeft % 1)*60 +
119         " minutos restantes");
120     }
121     else{
122         $("#current_status_helper").html(data.tmpMT + "°C &arr; " +
123         Math.round((data.timeLeft % 1)*60) + " segundos restantes");
124     }
125 }
126 break;
127 case 6://if the sparging process is running (without overflow)
128     $("#current_status").html("<i>Sparging</i> em andamento. ");
129     $("#current_status_helper").text("");
130     break;
131 case 7://if there is an overflow
132     $("#current_status").html("<i>Sparging</i> em andamento. ");
133     $("#current_status_helper").text("Tina do mosto cheia, drenando...");
134     break;
135 case 8://heating for the boil
136     $("#current_status").text("Aquecendo mosto para a fervura. ");
137     $("#current_status_helper").html(data.tmpBK + "°C");
138     break;
139 case 9://boiling the wort
140     $("#current_status").text("Fervura em andamento. ");
141     $("#current_status_helper").html((data.timestamps.boilFinishScheduled - data.timestamps.curr)/60000 + "
142     minutos restantes");
143     break;
144 case 10://hop added
145     $("#current_status").text("Fervura em andamento. ");
146     $("#current_status_helper").text("Lúpulo adicionado!");
147     break;
148 }
149 $("#h2").show();//show some information/status message
150 }
151 else{//if nothing is going on
152     $("#current_status").text("Sistema parado");//display idle message
153     $("#current_status_helper").text("");//display idle message
154 }
155 }
156 else{//if there is some error that caused the production to stop irreversibly
157     $("#current_status").text("Erro no sistema.");
158     $("#current_status_helper").text("Algo impediu que esta receita continue.");
159 }
160 }
161 </script>
162 </head>
163 <body style="display:none;">
164 <!--<h1>Controle do Sistema</h1-->
165 <h2 style="display:inline-block;" id="current_status"></h2>
166 <h2 style="display:inline-block;" id="current_status_helper"></h2>
167 <form>
168     <div class="slideThree">
169         <input type="checkbox" value="None" id="auto" name="check" />
170         <label for="auto"></label>
171         <span>AUTO</span>
172     </div>
173     <div class="slideThree id="confirmationButton" style="display:none;">
174         <input type="checkbox" value="None" id="confirmStartNextStep" name="check" />
175         <label for="confirmStartNextStep"></label>
176         <span>PROSSEGUIR</span>
177     </div>
178     <hr>
179     <div class="slideThree">
180         <input type="checkbox" class="btn" value="None" id="led" name="check" />
181         <label for="led"></label>
182         <span>LED PLACA</span>
183     </div>
184     <div class="slideThree">
185         <input type="checkbox" class="btn" value="None" id="mash_pump" name="check" />
186         <label for="mash_pump"></label>
187         <span>BOMBA DO MOSTO</span>
188     </div>
189     <div class="slideThree">
190         <input type="checkbox" class="btn" value="None" id="boil_pump" name="check" />
191         <label for="boil_pump"></label>
192         <span>BOMBA DA FERVURA</span>
193     </div>
194     <div class="slideThree">
195         <input type="checkbox" class="btn" value="None" id="mash_valve" name="check" />
196         <label for="mash_valve"></label>
197         <span>VÁLVULA DO MOSTO</span>
198     </div>
199     <div class="slideThree">
200         <input type="checkbox" class="btn" value="None" id="boil_valve" name="check" />
201         <label for="boil_valve"></label>
202         <span>VÁLVULA DA FERVURA</span>

```

```

203 </div><div class="slideThree">
204 <input type="checkbox" class="btn" value="None" id="chill_valve" name="check" />
205 <label for="chill_valve"></label>
206 <span>VÁLVULA DO CHILLER</span>
207 </div>
208 <div class="slideThree">
209 <input type="checkbox" class="btn" value="None" id="water_valve" name="check" />
210 <label for="water_valve"></label>
211 <span>VÁLVULA DA ÁGUA</span>
212 </div>
213 <div class="slideThree">
214 <input type="checkbox" class="btn" value="None" id="mash_heat" name="check" />
215 <label for="mash_heat"></label>
216 <span>AQUECEDOR DO MOSTO</span>
217 </div>
218 <div class="slideThree">
219 <input type="checkbox" class="btn" value="None" id="boil_heat" name="check" />
220 <label for="boil_heat"></label>
221 <span>AQUECEDOR DA FERVURA</span>
222 </div><br>
223 <div id="pwm_slider" style="display:none;">
224 <input type="range" class="slider" id="servo_pwm" name="servo_range" min="0" max="180" value="0" step="1"/>
225 <span class="slider-value"></span>
226 <label class="pwm" for="servo_pwm">SERVO-MOTOR</label>
227 </div>
228 </form><br>
229 </body>
230 </html>

```

Código-fonte G.13: Painel de controle de brassagem

G.14 CSS do formulário

```

1 form.myform
2 {
3     background: #595450; /* Old browsers */
4     background: -moz-linear-gradient(top, #595450 0%, #47413d 100%); /* FF3.6+ */
5     background: -webkit-gradient(linear, left top, left bottom, color-stop(0%,#595450), color-stop(100%,#47413d)); /* Chrome
6     ,Safari4+ */
7     background: -webkit-linear-gradient(top, #595450 0%,#47413d 100%); /* Chrome10+,Safari5.1+ */
8     background: -o-linear-gradient(top, #595450 0%,#47413d 100%); /* Opera 11.10+ */
9     background: -ms-linear-gradient(top, #595450 0%,#47413d 100%); /* IE10+ */
10    background: linear-gradient(to bottom, #595450 0%,#47413d 100%); /* W3C */
11    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#595450', endColorstr='#47413d',GradientType=0 ); /*
12    IE6-9 */
13
14    margin:auto;
15    position:relative;
16    font-family: Tahoma, Geneva, sans-serif;
17    font-size: 14px;
18    line-height: 24px;
19    font-weight: bold;
20    color: #F5CC7A;
21    text-decoration: none;
22    -webkit-border-radius: 10px;
23    -moz-border-radius: 10px;
24    border-radius: 10px;
25    padding:10px;
26    -webkit-box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3);
27    -moz-box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3);
28    box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3);
29 }
30
31 label.myform
32 {
33     display:inline-block;
34     width:20%;
35     /*padding-right:10px;
36     text-align:right;
37     float:left*/
38 }
39
40 input.myform, select.myform, textarea.myform
41 {
42     vertical-align: middle;
43     font-family: Arial, Helvetica, Sans-Serif;
44     font-style: normal;
45     font-weight: bold;
46     font-size: 13px;

```

```

47 | input[type=number].myform
48 | {
49 |     border-color:#5C5653;
50 |     background-color:#C4C2C1;
51 |     /*width:20%;*/
52 |     box-sizing: border-box;
53 |     display:inline-block;
54 |     vertical-align: middle;
55 |     font-family: Arial, Helvetica, Sans-Serif;
56 |     font-style: normal;
57 |     font-weight: bold;
58 |     font-size: 13px;
59 |     -webkit-border-radius: 5px;
60 |     -moz-border-radius: 5px;
61 |     border-radius: 5px;
62 | }
63 |
64 | input[type=text].myform
65 | {
66 |     border-color:#5C5653;
67 |     background-color:#C4C2C1;
68 |     /*width:20%;*/
69 |     box-sizing: border-box;
70 |     display:inline-block;
71 |     vertical-align: middle;
72 |     font-family: Arial, Helvetica, Sans-Serif;
73 |     font-style: normal;
74 |     font-weight: bold;
75 |     font-size: 13px;
76 |     -webkit-border-radius: 5px;
77 |     -moz-border-radius: 5px;
78 |     border-radius: 5px;
79 | }
80 |
81 | input[type=submit].myform
82 | {
83 |     width:100px;
84 |     position:relative;
85 |     margin:5px;
86 |     bottom:35px;
87 |     float:right;
88 |     background:#33282C;
89 |     color:#f5cc7a;
90 |     font-family: Tahoma, Geneva, sans-serif;
91 |     height:30px;
92 |     -webkit-border-radius: 15px;
93 |     -moz-border-radius: 15px;
94 |     border-radius: 15px;
95 | }
96 | input[type=submit].myform:hover {
97 |     background:#fff;
98 |     color:#33282C;
99 | }
100 |
101 | input[type=reset].myform
102 | {
103 |     width:100px;
104 |     position:relative;
105 |     margin:5px;
106 |     bottom:35px;
107 |     float:right;
108 |     background:#33282C;
109 |     color:#f5cc7a;
110 |     font-family: Tahoma, Geneva, sans-serif;
111 |     height:30px;
112 |     -webkit-border-radius: 15px;
113 |     -moz-border-radius: 15px;
114 |     border-radius: 15px;
115 | }
116 | input[type=reset].myform:hover {
117 |     background:#fff;
118 |     color:#33282C;
119 | }
120 |
121 | input[type=button].myform
122 | {
123 |     width:100px;
124 |     position:relative;
125 |     margin:5px;
126 |     bottom:35px;
127 |     float:right;
128 |     background:#33282C;
129 |     color:#f5cc7a;
130 |     font-family: Tahoma, Geneva, sans-serif;
131 |     height:30px;
132 |     -webkit-border-radius: 15px;
133 |     -moz-border-radius: 15px;

```

```

134 |     border-radius: 15px;
135 | }
136 | input[type=button].myform:hover {
137 |     background:#fff;
138 |     color:#33282C;
139 | }
140 |
141 | .long{width:20%;}/*For widths that don't need to be too long*/
142 | .short{width:10%;}/*For widths that don't need to be too long*/

```

Código-fonte G.14: CSS do formulário

G.15 CSS específico para alguns botões

```

1 | input[type=button].leftselect
2 | {
3 |     vertical-align:middle;
4 |     font-family: Arial, Helvetica, Sans-Serif;
5 |     font-style: normal;
6 |     font-weight: bold;
7 |     font-size: 13px;
8 |     text-transform: uppercase;
9 |     width:150px;
10 |    position:relative;
11 |    margin:5px;
12 |    bottom:46px;
13 |    float:right;
14 |    background:#33282C;
15 |    color:#f5cc7a;
16 |    font-family: Tahoma, Geneva, sans-serif;
17 |    height:30px;
18 |    -webkit-border-radius: 15px;
19 |    -moz-border-radius: 15px;
20 |    border-radius: 15px;
21 |    border: 1p solid #999;
22 | }
23 | input[type=button].leftselect:hover {
24 |     background:#fff;
25 |     color:#33282C;
26 | }
27 |
28 | input[type=button].okbtn
29 | {
30 |     vertical-align:middle;
31 |     font-family: Arial, Helvetica, Sans-Serif;
32 |     font-style: normal;
33 |     font-weight: bold;
34 |     font-size: 13px;
35 |     text-transform: uppercase;
36 |     width:50px;
37 |     margin:5px;
38 |     background:#33282C;
39 |     color:#f5cc7a;
40 |     font-family: Tahoma, Geneva, sans-serif;
41 |     height:25px;
42 |     -webkit-border-radius: 15px;
43 |     -moz-border-radius: 15px;
44 |     border-radius: 15px;
45 |     border: 1p solid #999;
46 | }
47 | input[type=button].okbtn:hover {
48 |     background:#fff;
49 |     color:#33282C;
50 | }

```

Código-fonte G.15: CSS específico para alguns botões

G.16 CSS do template da aplicação web

```

1 | .long{width:18%;}/*For widths that don't need to be too long*/
2 | .short{width:10%;}/*For widths that don't need to be too long*/
3 | .big{width:60%;}
4 | .full{width:100%;}

```

```

5 |
6 | /*Warning google symbol and paragraph side by side*/
7 | div.warning{
8 |     border: 3px solid orange;
9 |     -webkit-border-radius: 6px;
10 |     -moz-border-radius: 6px;
11 |     border-radius: 6px;
12 |     margin-bottom:10px;
13 | }
14 | div.warning > div{
15 |     float:left;
16 | }
17 | div.warning > p{
18 |     position:relative;
19 |     margin: 0px 0px 0px 60px;
20 | }
21 | .material-icons.custom{
22 |     font-size: 40px;
23 |     color: #E65C00;
24 | }
25 |
26 | body{
27 |     background-color:#A2A2A2;
28 |     margin: 8px;
29 | }
30 |
31 | h1{
32 |     font-family:"Arial", Helvetica, sans-serif;
33 |     font-family: 'Roboto', sans-serif;
34 |     font-size:2.5em;
35 |     color:#FFD633;
36 | }
37 |
38 | h2{
39 |     font-family:"Arial", Helvetica, sans-serif;
40 |     font-family: 'Roboto', sans-serif;
41 |     font-size:2.0em;
42 |     color:#FFD633;
43 | }
44 |
45 | p{
46 |     font-family:"Arial", Helvetica, sans-serif;
47 |     font-family: 'Roboto', sans-serif;
48 |     font-size:1.15em;
49 |     color:#494949;
50 | }
51 |
52 | li{
53 |     font-family: 'Roboto', sans-serif;
54 |     font-family:"Arial", Helvetica, sans-serif;
55 |     font-size:1.15em;
56 |     color:#494949;
57 | }
58 |
59 | .prevhead{
60 |     font-family: 'Roboto', sans-serif;
61 |     font-family:"Arial", Helvetica, sans-serif;
62 |     font-size:13px;
63 |     color:#FFD633;
64 |     display:inline-block;
65 |     margin:0 0 0 10px;
66 |     width:25%;
67 |     vertical-align:top;
68 | }
69 |
70 | .prev{
71 |     font-family: 'Roboto', sans-serif;
72 |     font-family:"Arial", Helvetica, sans-serif;
73 |     font-size:13px;
74 |     color:#FFFFFF;
75 |     display:inline-block;
76 |     margin:0 0 0 10px;
77 |     width:70%;
78 | }
79 |
80 | a:link, a:visited{
81 |     font-family: 'Roboto', sans-serif;
82 |     font-family:"Arial", Helvetica, sans-serif;
83 |     font-size:1.15em;
84 |     text-decoration:none;
85 |     display:inline-block;
86 |     color:#FFD633;
87 | }
88 |
89 | a:hover, a:active{
90 |     background-color:#616161;
91 | }

```

```

92 |
93 | ol{
94 |     font-family: 'Roboto', sans-serif;
95 |     font-family:"Arial", Helvetica, sans-serif;
96 |     font-size:1.15em;
97 |     color:#FFD633;
98 | }
99 |
100 | .enter, .del{
101 |     font-family: 'Roboto', sans-serif;
102 |     font-family:"Arial", Helvetica, sans-serif;
103 |     font-weight:bold;
104 |     font-size:10px;
105 |     color:red;
106 |     border-radius:12px;
107 |     border-style:outset;
108 | }
109 |
110 | .enter:hover, .del:hover{
111 |     border-style:inset;
112 | }
113 |
114 | .add{
115 |     vertical-align:middle;
116 |     font-style: normal;
117 |     font-weight: bold;
118 |     font-size: 13px;
119 |     text-transform: uppercase;
120 |     width:30%;
121 |     background:#33282C;
122 |     color:#f5cc7a;
123 |     font-family: 'Roboto', sans-serif;
124 |     font-family: Tahoma, Geneva, sans-serif;
125 |     height:30px;
126 |     -webkit-border-radius: 15px;
127 |     -moz-border-radius: 15px;
128 |     border-radius: 15px;
129 | }
130 | .add:hover {
131 |     background:#fff;
132 |     color:#33282C;
133 | }
134 |
135 | input[type=text]
136 | {
137 |     border-color:#5C5653;
138 |     background-color:#C4C2C1;
139 |     -webkit-box-sizing:border-box;
140 |     -moz-box-sizing: border-box;
141 |     box-sizing: border-box;
142 |     display:inline-block;
143 |     vertical-align: middle;
144 |     font-family: 'Roboto', sans-serif;
145 |     font-family: Tahoma, Geneva, sans-serif;
146 |     font-style: normal;
147 |     font-weight: bold;
148 |     font-size: 13px;
149 |     -webkit-border-radius: 5px;
150 |     -moz-border-radius: 5px;
151 |     border-radius: 5px;
152 | }
153 |
154 | select
155 | {
156 |     border-color:#5C5653;
157 |     background-color:#C4C2C1;
158 |     /*width:20%;*/
159 |     box-sizing: border-box;
160 |     display:inline-block;
161 |     vertical-align: middle;
162 |     font-family: 'Roboto', sans-serif;
163 |     font-family: Arial, Helvetica, Sans-Serif;
164 |     font-style: normal;
165 |     font-weight: bold;
166 |     font-size: 13px;
167 |     -webkit-border-radius: 5px;
168 |     -moz-border-radius: 5px;
169 |     border-radius: 5px;
170 | }
171 |
172 | :-webkit-input-placeholder {color: #33282C;}
173 | :-moz-placeholder { /* Firefox 18- */color: #33282C;}
174 | :-moz-placeholder { /* Firefox 19+ */color: #33282C;}
175 | :-ms-input-placeholder {color: #33282C;}
176 |
177 | form.myform
178 | {

```



```

179 | background: #595450; /* Old browsers */
180 | background: -moz-linear-gradient(top, #595450 0%, #47413d 100%); /* FF3.6+ */
181 | background: -webkit-gradient(linear, left top, left bottom, color-stop(0%,#595450), color-stop(100%,#47413d)); /* Chrome
    | ,Safari4+ */
182 | background: -webkit-linear-gradient(top, #595450 0%,#47413d 100%); /* Chrome10+,Safari5.1+ */
183 | background: -o-linear-gradient(top, #595450 0%,#47413d 100%); /* Opera 11.10+ */
184 | background: -ms-linear-gradient(top, #595450 0%,#47413d 100%); /* IE10+ */
185 | background: linear-gradient(to bottom, #595450 0%,#47413d 100%); /* W3C */
186 | filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#595450', endColorstr='#47413d',GradientType=0 ); /*
    | IE6-9 */
187 |
188 | margin:auto;
189 | position:relative;
190 | font-family: 'Roboto', sans-serif;
191 | font-family: Tahoma, Geneva, sans-serif;
192 | font-size: 14px;
193 | line-height: 24px;
194 | font-weight: bold;
195 | color: #F5CC7A;
196 | text-decoration: none;
197 | -webkit-border-radius: 10px;
198 | -moz-border-radius: 10px;
199 | border-radius: 10px;
200 | padding:10px;
201 | -webkit-box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3);
202 | -moz-box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3);
203 | box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3);
204 | }
205 |
206 | label.myform
207 | {
208 |     display:inline-block;
209 |     /*padding-right:10px;
210 |     text-align:right;
211 |     float:left*/
212 | }
213 |
214 | input.myform, select.myform, textarea.myform
215 | {
216 |     vertical-align: middle;
217 |     font-family: 'Roboto', sans-serif;
218 |     font-family: Arial, Helvetica, Sans-Serif;
219 |     font-style: normal;
220 |     font-weight: bold;
221 |     font-size: 13px;
222 | }
223 |
224 | input[type=number].myform
225 | {
226 |     border-color:#5C5653;
227 |     background-color:#C4C2C1;
228 |     /*width:20%;*/
229 |     box-sizing: border-box;
230 |     display:inline-block;
231 |     vertical-align: middle;
232 |     font-family: 'Roboto', sans-serif;
233 |     font-family: Arial, Helvetica, Sans-Serif;
234 |     font-style: normal;
235 |     font-weight: bold;
236 |     font-size: 13px;
237 |     -webkit-border-radius: 5px;
238 |     -moz-border-radius: 5px;
239 |     border-radius: 5px;
240 | }
241 |
242 | input[type=text].myform
243 | {
244 |     border-color:#5C5653;
245 |     background-color:#C4C2C1;
246 |     /*width:20%;*/
247 |     box-sizing: border-box;
248 |     display:inline-block;
249 |     vertical-align: middle;
250 |     font-family: 'Roboto', sans-serif;
251 |     font-family: Arial, Helvetica, Sans-Serif;
252 |     font-style: normal;
253 |     font-weight: bold;
254 |     font-size: 13px;
255 |     -webkit-border-radius: 5px;
256 |     -moz-border-radius: 5px;
257 |     border-radius: 5px;
258 | }
259 |
260 | input[type=datetime-local].myform
261 | {
262 |     border-color:#5C5653;
263 |     background-color:#C4C2C1;

```

```

264 | /*width:20%;*/
265 | box-sizing: border-box;
266 | display:inline-block;
267 | vertical-align: middle;
268 | font-family: 'Roboto', sans-serif;
269 | font-family: Arial, Helvetica, Sans-Serif;
270 | font-style: normal;
271 | font-weight: bold;
272 | font-size: 13px;
273 | -webkit-border-radius: 5px;
274 | -moz-border-radius: 5px;
275 | border-radius: 5px;
276 | }
277 |
278 | input[type=submit].myform
279 | {
280 |     width:100px;
281 |     position:relative;
282 |     margin:5px;
283 |     bottom:35px;
284 |     float:right;
285 |     background:#33282C;
286 |     color:#f5cc7a;
287 |     font-family: 'Roboto', sans-serif;
288 |     font-family: Tahoma, Geneva, sans-serif;
289 |     height:30px;
290 |     -webkit-border-radius: 15px;
291 |     -moz-border-radius: 15px;
292 |     border-radius: 15px;
293 | }
294 | input[type=submit].myform:hover {
295 |     background:#fff;
296 |     color:#33282C;
297 | }
298 |
299 | input[type=reset].myform
300 | {
301 |     width:100px;
302 |     position:relative;
303 |     margin:5px;
304 |     bottom:35px;
305 |     float:right;
306 |     background:#33282C;
307 |     color:#f5cc7a;
308 |     font-family: 'Roboto', sans-serif;
309 |     font-family: Tahoma, Geneva, sans-serif;
310 |     height:30px;
311 |     -webkit-border-radius: 15px;
312 |     -moz-border-radius: 15px;
313 |     border-radius: 15px;
314 | }
315 | input[type=reset].myform:hover {
316 |     background:#fff;
317 |     color:#33282C;
318 | }
319 |
320 | input[type=button].myform
321 | {
322 |     width:100px;
323 |     position:relative;
324 |     margin:5px;
325 |     bottom:35px;
326 |     float:right;
327 |     background:#33282C;
328 |     color:#f5cc7a;
329 |     font-family: 'Roboto', sans-serif;
330 |     font-family: Tahoma, Geneva, sans-serif;
331 |     height:30px;
332 |     -webkit-border-radius: 15px;
333 |     -moz-border-radius: 15px;
334 |     border-radius: 15px;
335 | }
336 | input[type=button].myform:hover {
337 |     background:#fff;
338 |     color:#33282C;
339 | }
340 |
341 | label.pwm {
342 |     position:relative;
343 |     white-space:nowrap;
344 |
345 |     font: 12px/26px Arial, sans-serif;
346 |     color: #000;
347 |     font-weight: bold;
348 |     text-shadow: 1px 1px 0px rgba(255,255,255,.15);
349 | }
350 |

```

```

351 | /*HERE STARTS THE CHECKBOX ON-OFF BUTTON*/
352 | input[type=checkbox] {
353 |     visibility: hidden;
354 | }
355 |
356 | /* SLIDE THREE */
357 | .slideThree {
358 |     width: 80px;
359 |     height: 26px;
360 |     background: #595450;
361 |     margin: 20px;
362 |     margin-right: 200px;
363 |
364 |     -webkit-border-radius: 50px;
365 |     -moz-border-radius: 50px;
366 |     border-radius: 50px;
367 |     position: relative;
368 |     display:inline-block;
369 |
370 |     -webkit-box-shadow: inset 0px 1px 1px rgba(0,0,0,0.5), 0px 1px 0px rgba(255,255,255,0.2);
371 |     -moz-box-shadow: inset 0px 1px 1px rgba(0,0,0,0.5), 0px 1px 0px rgba(255,255,255,0.2);
372 |     box-shadow: inset 0px 1px 1px rgba(0,0,0,0.5), 0px 1px 0px rgba(255,255,255,0.2);
373 | }
374 |
375 | .slideThree:after {
376 |     content: 'OFF';
377 |     font: 12px/26px Arial, sans-serif;
378 |     color: #000;
379 |     position: absolute;
380 |     right: 10px;
381 |     z-index: 0;
382 |     font-weight: bold;
383 |     text-shadow: 1px 1px 0px rgba(255,255,255,.15);
384 | }
385 |
386 | .slideThree:before {
387 |     content: 'ON';
388 |     font: 12px/26px Arial, sans-serif;
389 |     color: #00bf00;
390 |     position: absolute;
391 |     left: 10px;
392 |     z-index: 0;
393 |     font-weight: bold;
394 | }
395 |
396 | .slideThree label {
397 |     display: block;
398 |     width: 34px;
399 |     height: 20px;
400 |
401 |     -webkit-border-radius: 50px;
402 |     -moz-border-radius: 50px;
403 |     border-radius: 50px;
404 |
405 |     -webkit-transition: all .4s ease;
406 |     -moz-transition: all .4s ease;
407 |     -o-transition: all .4s ease;
408 |     -ms-transition: all .4s ease;
409 |     transition: all .4s ease;
410 |     cursor: pointer;
411 |     position: absolute;
412 |     top: 3px;
413 |     left: 3px;
414 |     z-index: 1;
415 |
416 |     -webkit-box-shadow: 0px 2px 5px 0px rgba(0,0,0,0.3);
417 |     -moz-box-shadow: 0px 2px 5px 0px rgba(0,0,0,0.3);
418 |     box-shadow: 0px 2px 5px 0px rgba(0,0,0,0.3);
419 |     background: #fcfff4;
420 |
421 |     background: -webkit-linear-gradient(top, #fcfff4 0%, #dfe5d7 40%, #b3bead 100%);
422 |     background: -moz-linear-gradient(top, #fcfff4 0%, #dfe5d7 40%, #b3bead 100%);
423 |     background: -o-linear-gradient(top, #fcfff4 0%, #dfe5d7 40%, #b3bead 100%);
424 |     background: -ms-linear-gradient(top, #fcfff4 0%, #dfe5d7 40%, #b3bead 100%);
425 |     background: linear-gradient(top, #fcfff4 0%, #dfe5d7 40%, #b3bead 100%);
426 |     filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#fcfff4', endColorstr='#b3bead',GradientType=0 );
427 | }
428 |
429 | .slideThree input[type=checkbox]:checked + label {
430 |     left: 43px;
431 | }
432 |
433 | .slideThree span {
434 |     position: absolute;
435 |     /*top:3px;*/
436 |     left:83px;
437 |     white-space: nowrap;

```

```

438 |
439 |     font: 12px/26px Arial, sans-serif;
440 |     color: #000;
441 |     font-weight: bold;
442 |     text-shadow: 1px 1px 0px rgba(255,255,255,.15);
443 | }
444 | /*HERE ENDS THE CHECKBOX ON-OFF BUTTON*/
445 |
446 |
447 | /*HERE STARTS THE SLIDER CSS*/
448 | input[type=range].slider {
449 |     -webkit-appearance: none;
450 |     width: 50%;
451 |     margin: 8.8px 0;
452 | }
453 | input[type=range].slider:focus {
454 |     outline: none;
455 | }
456 | input[type=range].slider::-webkit-slider-runnable-track {
457 |     width: 100%;
458 |     height: 8.4px;
459 |     cursor: pointer;
460 |     box-shadow: 1px 1px 1px #000000, 0px 0px 1px #0d0d0d;
461 |     background: #595450;
462 |     border-radius: 7.8px;
463 |     border: 1px solid #010101;
464 | }
465 | input[type=range].slider::-webkit-slider-thumb {
466 |     box-shadow: 0.9px 0.9px 5px #000031, 0px 0px 0.9px #00004b;
467 |     border: 3px solid #ffd633;
468 |     height: 26px;
469 |     width: 26px;
470 |     border-radius: 13px;
471 |     background: #ffd633;
472 |     cursor: pointer;
473 |     -webkit-appearance: none;
474 |     margin-top: -9.8px;
475 | }
476 | input[type=range].slider:focus::-webkit-slider-runnable-track {
477 |     background: #3E3B38;
478 | }
479 | input[type=range].slider::-moz-range-track {
480 |     width: 100%;
481 |     height: 8.4px;
482 |     cursor: pointer;
483 |     box-shadow: 1px 1px 1px #000000, 0px 0px 1px #0d0d0d;
484 |     background: #595450;
485 |     border-radius: 7.8px;
486 |     border: 1px solid #010101;
487 | }
488 | input[type=range].slider::-moz-range-thumb {
489 |     box-shadow: 0.9px 0.9px 5px #000031, 0px 0px 0.9px #00004b;
490 |     border: 1px solid #ffd633;
491 |     height: 22px;
492 |     width: 22px;
493 |     border-radius: 11px;
494 |     background: #ffd633;
495 |     cursor: pointer;
496 | }
497 | input[type=range].slider::-ms-track {
498 |     width: 100%;
499 |     height: 8.4px;
500 |     cursor: pointer;
501 |     background: transparent;
502 |     border-color: transparent;
503 |     color: transparent;
504 | }
505 | input[type=range].slider::-ms-fill-lower {
506 |     background: #1f181b;
507 |     border: 1px solid #010101;
508 |     border-radius: 15.6px;
509 |     box-shadow: 1px 1px 1px #000000, 0px 0px 1px #0d0d0d;
510 | }
511 | input[type=range].slider::-ms-fill-upper {
512 |     background: #595450;
513 |     border: 1px solid #010101;
514 |     border-radius: 15.6px;
515 |     box-shadow: 1px 1px 1px #000000, 0px 0px 1px #0d0d0d;
516 | }
517 | input[type=range].slider::-ms-thumb {
518 |     box-shadow: 0.9px 0.9px 5px #000031, 0px 0px 0.9px #00004b;
519 |     border: 3px solid #ffd633;
520 |     height: 26px;
521 |     width: 26px;
522 |     border-radius: 13px;
523 |     background: #ffd633;
524 |     cursor: pointer;

```

```

525 | height: 8.4px;
526 | }
527 | input[type=range].slider:focus::-ms-fill-lower {
528 |     background: #595450;
529 | }
530 | input[type=range].slider:focus::-ms-fill-upper {
531 |     background: #3E3B38;
532 | }
533 |
534 | .slider-value {
535 |     display: inline-block;
536 |     position: relative;
537 |     width: 60px;
538 |     color: #fff;
539 |     font-size: 16px;
540 |     line-height: 20px;
541 |     text-align: center;
542 |     border-radius: 3px;
543 |     background: #595450;
544 |     padding: 5px 10px;
545 |     margin-left: 7px;
546 | }
547 | /*HERE ENDS THE SLIDER CSS*/
548 |
549 | /*HERE STARTS THE HEADER MENU CSS*/
550 | div.myheader{
551 |     /* Retro compatibility*/
552 |     width:80%;
553 |     /* Firefox */
554 |     width: -moz-calc(80% + 16px);
555 |     /* WebKit */
556 |     width: -webkit-calc(80% + 16px);
557 |     /* Opera */
558 |     width: -o-calc(80% + 16px);
559 |     /* Standard */
560 |     width: calc(80% + 16px);
561 |
562 |     margin:-8px -8px;
563 |     padding-left:20px;
564 |     height:40px;
565 |     background:#33282C;
566 | }
567 |
568 | div.headercell{
569 |     display:inline-block;
570 |     height:inherit;
571 |     width:15%;
572 | }
573 |
574 | a.headerlink{
575 |     display:inline-block;
576 |     text-align:center;
577 |     line-height: 40px;
578 |     color:#f5cc7a;
579 |     font-family: 'Roboto', sans-serif;
580 |     font-family: Tahoma, Geneva, sans-serif;
581 |     font-weight:bold;
582 |     width:100%;
583 | }
584 | /*HERE ENDS THE HEADER MENU CSS*/

```

Código-fonte G.16: CSS do template da aplicação web

Apêndice H

Arquivos de configuração do sistema

H.1 Arquivo de configuração de rede

```

1  # This file describes the network interfaces available
2  # on your system and how to activate them.
3  # For more information, see interfaces(5).
4
5  # The loopback network interface
6  auto lo
7  iface lo inet loopback
8
9  # The primary network interface
10 #auto eth0
11 #iface eth0 inet dhcp
12 # Example to keep MAC address between reboots
13 #hwaddress ether DE:AD:BE:EF:CA:FE
14
15 # The secondary network interface
16 #auto eth1
17 #iface eth1 inet dhcp
18
19 # WiFi Example
20 auto wlan0
21 allow-hotplug wlan0
22 iface wlan0 inet static
23 #allow-hotplug wlan0
24 #auto lo
25 #iface lo inet loopback
26 #auto eth0
27 #iface eth0 inet static
28 address 192.168.1.155
29 netmask 255.255.252.0
30 network 192.168.1.0
31 gateway 192.168.1.1
32 #pre-up ifconfig eth0 hw ether 00:01:02:03:05:14
33 dns-nameservers 143.107.225.6 143.107.182.2 8.8.8.8
34 wpa-ssid "nome_da_rede"
35 wpa-psk "senha1"
36 #iface wlan0 inet dhcp
37 # wpa-ssid "outra_rede"
38 # wpa-psk "senha2"
39
40 # Ethernet/RNDIS gadget (g_ether)
41 # ... or on host side, usbnet and random hwaddr
42 # Note on some boards, usb0 is automatically
43 #setup with an init script
44 iface usb0 inet static
45 address 192.168.7.2
46 netmask 255.255.255.0
47 network 192.168.7.0
48 gateway 192.168.7.1

```

Código-fonte H.1: /etc/network/interfaces

H.2 Device Tree para o DS18B20

```

1  /*
2  * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
3  *
4  * This program is free software; you can redistribute it and/or modify
5  * it under the terms of the GNU General Public License version 2 as
6  * published by the Free Software Foundation.
7  *
8  * Modified by Russell Senior from the weather cape's DTS file.
9  * Minor formatting by C W Rose.
10 * Modified by Leonardo Graboski Veiga to change the DS18B20 pin and pin configuration
11 */
12 /dts-v1/;
13 /plugin/;
14
15 / {
16     compatible = "ti,beaglebone", "ti,beaglebone-black";
17     part-number = "BB-w1";
18     version = "00A0";
19
20     exclusive-use = "P9.11";
21
22     fragment@0 {
23         target = <am33xx_pinmux>;
24         __overlay__ {
25             bb_w1_pins: pinmux_bb_w1_pins {
26                 pinctrl-single,pins = <0x70 0x37>;
27             };
28         };
29     };
30
31     fragment@1 {
32         target = <ocp>;
33         __overlay__ {
34             onewire@0 {
35                 status = "okay";
36                 compatible = "w1-gpio";
37                 pinctrl-names = "default";
38                 pinctrl-0 = <bb_w1_pins>;
39
40                 gpios = <gpio1 30 0>;
41             };
42         };
43     };
44 };

```

Código-fonte H.2: w1.dts

Apêndice I

Caracterização do funcionamento do servo-motor com AT89S52

Servomotor é um dispositivo de conversão eletromecânica que surgiu da necessidade de controle de posição angular de um motor de corrente contínua — tanto é que um servomotor é um motor DC com controle e realimentação, ou seja, um dispositivo de malha fechada. Além da tensão de alimentação, o servomotor recebe um sinal de controle proporcional à posição desejada do eixo e atua no motor DC até que, através do ramo de realimentação, atinja-se a posição angular de entrada. A Figura I.1 apresenta um servomotor desmontado e a identificação de suas partes principais.

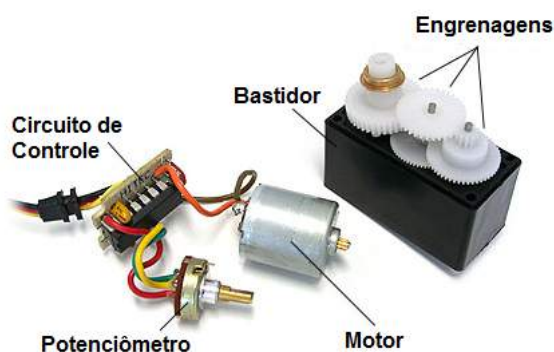


Figura I.1: Servo-motor desmontado

Observa-se que o circuito de controle é um circuito integrado, porém não há ampla documentação acerca de sua arquitetura interna; já o potenciômetro é o elemento sensor do conjunto, cujo objetivo é fornecer a posição angular do eixo através da variação de sua resistência. O motor é a planta do sistema de controle, convertendo energia elétrica em rotação do eixo. O conjunto de engrenagens é responsável por aumentar o torque do mesmo, com a

I.1 Simulação e ensaio em bancada

O "ciclo" referido a seguir é o mínimo incremento do tempo em alto do sinal de controle — baseado em simulação, o código utilizado para gerar o pulso em nível alto apresentou uma limitação de período/incremento de $20\mu\text{s}$ para $f=12\text{MHz}$, portanto o código trabalha com resolução de $20\mu\text{s}$ /incremento em 12MHz . Na figura I.3 é apresentada a forma de onda necessária para controlar o servo motor.

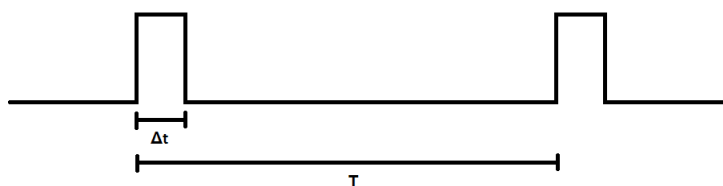


Figura I.3: Forma de onda de acionamento do servo-motor

O período T é fixo, com valor de 20ms , e o tempo em nível alto Δt é de, aprioximadamente $0,5\text{ms} < \Delta t < 2,5\text{ms}$. O objetivo da caracterização do servomotor em questão foi definir com precisão os limites de Δt . Após a simulação bem-sucedida do controle do servo motor incluso na biblioteca padrão do Proteus, o código foi efetivamente gravado no μC AT89S52, com $f=24\text{MHz}$ e o motor foi excursionado até seus limites — e além deles — e seu comportamento foi observado e documentado, conforme é apresentado na figura I.4. O led vermelho ligado mecanicamente ao braço do servo é somente uma referência visual para a posição angular deste.

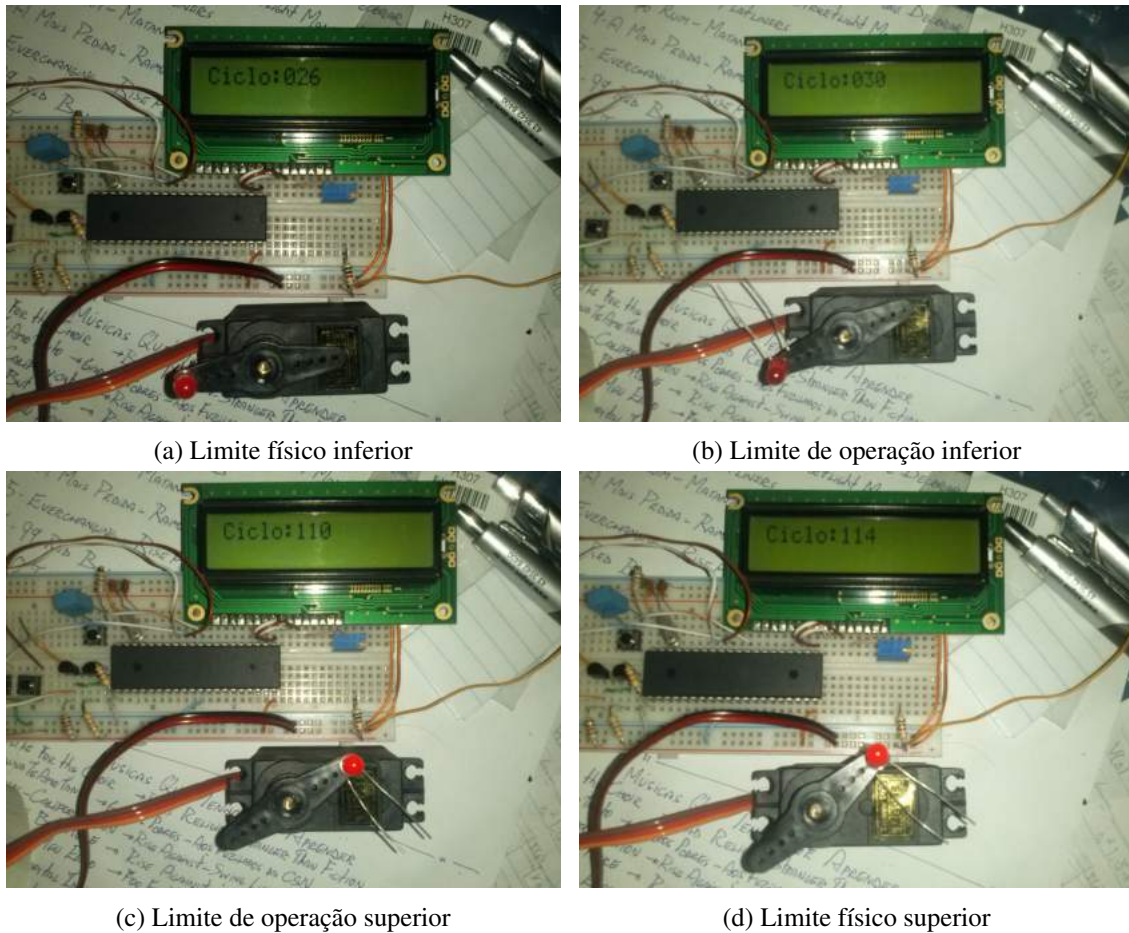


Figura I.4: Limites de excursão do servo-motor

A partir do ensaio realizado, pôde ser observado que quando funcionando muito próximo dos limites de excursão, o motor gera vibração mecânica e calor excessivos. Como a excursão máxima, em graus, é maior do que 180° , a faixa segura de operação foi definida para $30 < ciclo < 110$, tendo em vista que essa faixa tem uma excursão aproximada de 180° e está distante dos valores mínimo e máximo de excursão.

Voltando ao Proteus, ao simular o circuito, foi adicionado um osciloscópio ao pino de controle do servo motor, possibilitando a predição do tempo em alta do sinal. Três simulações para valores notáveis de "ciclo", i.e. incremento mínimo de Δt , foram feitas: para ciclo=1, determinando o offset da temporização, pois há atrasos de chamada de rotina de configuração dos temporizadores; ciclo=25, determinando o valor mínimo de operação e; ciclo=114, determinando o valor máximo de operação. Os resultados são mostrados na Figura I.5, onde é observado no osciloscópio o sinal para ciclo=1 e diversas marcações de temporização para outros valores de ciclo.

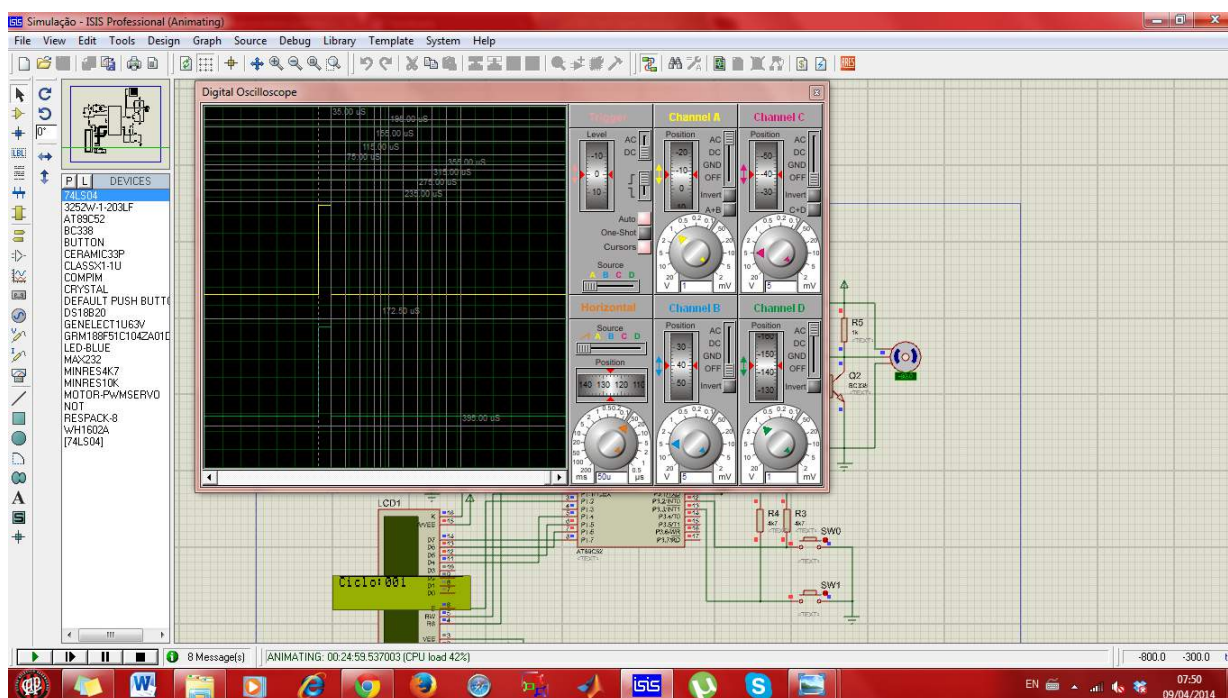


Figura I.5: Simulação para obtenção do valor mínimo de incremento Δt

Diversas outras amostras de Δt foram registradas e seus valores interpolados de maneira a observar a linearidade t_{min} com a qual Δt é incrementado. O gráfico da Figura I.6 apresenta os resultados.

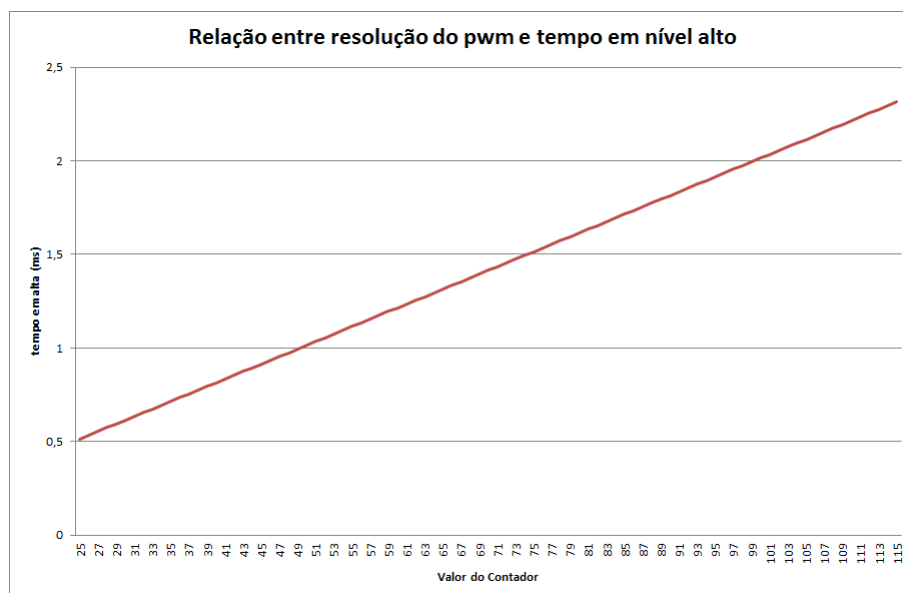


Figura I.6: Gráfico de Δt em função de *ciclo*

A partir do ensaio e da simulação aqui descritos, foram obtidos os limites de operação do servomotor TowerPro MG995 e a fórmula de Δt , demonstrada na equação I.1 em função do valor do seu incremento mínimo (*ciclo*). A tabela I.1 resume os limites de temporização do

servo-motor MG995.

$$\Delta t = (\text{ciclo} - 1) \cdot t_{min} + \text{offset} \quad (\text{I.1})$$

$$t_{min} = 20\mu s$$

$$\text{offset} = 35\mu s$$

Tabela I.1: Limites de temporização do servo-motor TowerPro MG995

Limite	Valor (ms)
período mínimo	10
excursão mínima	0,515
excursão mínima ideal	0,615
excursão máxima	2,295
excursão máxima ideal	2,215

É importante que os limites de temporização sejam respeitados, uma vez que o motor, cuja excursão máxima é limitada mecânicamente, tenta compensá-la requisitando cada vez mais corrente da fonte, até queimar. Geralmente quando os limites são excedidos ou chegam próximos do máximo, o motor começa a vibrar e aquecer, indicando ao projetista atento que há algo errado.

I.2 Código-fonte

```

1  #include <at89s8252.h>
2  /*****DEFINIÇÃO DOS PORTS E CONSTANTES*****/
3  #define RS      P1_0
4  #define RW      P1_1
5  #define E       P1_2
6  #define LCD     P1
7  #define LINE0   0x80
8  #define LINE1   0xC0
9  #define SERVO   P2_0
10 /*****/
11
12 /***** VARIÁVEIS EXTERNAS *****/
13 volatile unsigned int estouro=0;
14 volatile unsigned char ciclo=25;
15 /*****/
16
17 /*****funções de main.c*****/
18 void config();
19 void tmr0() __interrupt 1;
20 void tmr1() __interrupt 3;
21 void ext_int0() __interrupt 0;
22 void ext_int1() __interrupt 2;
23 void inicia_lcd();
24 void envia_lcd(unsigned char);
25 void envia_str_lcd(unsigned char [], unsigned char);
26 void reset_lcd();
27 void atraso(unsigned char);

```



```

28  /*****fim das funções*****/
29  void config(){
30      TMOD=0x12; //timer 1 como timer(16b) e timer 0 como timer(8b auto reload)
31      IE=0x8A; //interrupção geral e por timer 0 e 1 ativadas
32      TL0=TH0=0xCE; //preset para 50us
33      TH1=0xB1; //prepara para 20ms
34      TL1=0xDF;
35      EX0=EX1=1; //habilita interrupção externa
36      TR1=1; //liga ambos os timers
37      TR0=1;
38      PT0=PT1=1; //prioridade de interrupção maior para os timers
39  }
40
41  void tmr0() __interrupt 1{
42      estouro++;
43      if(estouro==ciclo){ //se deu a largura de pulso
44          estouro=0; //zera o controle
45          SERVO=0;
46          TR0=0; //desliga o timer
47      }
48  }
49
50  void tmr1() __interrupt 3{
51      TH1=0xB1; //prepara para 20ms
52      TL1=0xDF;
53      TL0=TH0=0xD8; //preset para 40us(12MHz) ou 20us(24MHz)
54      TR0=1; //liga o timer 0
55      SERVO=1;
56  }
57
58  void ext_int0() __interrupt 0{
59      unsigned int atraso;
60      //if(ciclo<126){ //valor prático para 24MHz -> 2,5ms
61          ciclo++;
62      }
63      for(atraso=0; atraso!=0xffff; atraso++);
64  }
65
66  void ext_int1() __interrupt 2{
67      unsigned int atraso;
68      //if(ciclo>24){ //valor prático para 24MHz -> 0,5ms
69          ciclo--;
70      }
71      for(atraso=0; atraso!=0xffff; atraso++);
72  }
73
74  void inicia_lcd(){
75      const unsigned char instr[]={0x06,0x0C,0x28,0x01};
76      unsigned char colorado, invicta;
77      RS=RW=E=0;
78      for(colorado=0; colorado!=60; colorado++){ //gera um atraso de 15ms de inicialização
79          atraso(250);
80      }
81      E=1;
82      LCD=(0x0f&LCD)|(letra&0xf0); //máscara para os bits de controle
83      E=0;
84      for(invicta=0; invicta!=20; invicta++){ //gera um atraso de 5ms de inicialização
85          atraso(250);
86      }
87      for(colorado=0; colorado!=4; colorado++){ //envia as instruções de inicialização
88          envia_lcd(instr[colorado]);
89          for(invicta=0; invicta!=20; invicta++){ //gera um atraso de 5ms de inicialização
90              atraso(250);
91          }
92      }
93  }
94
95  void envia_lcd(unsigned char letra){
96      E=1;
97      LCD=(0x0f&LCD)|(letra&0xf0); //máscara para os bits de controle
98      E=0;
99      letra<<=4; //faz o swap dos nibbles
100     E=1;
101     LCD=(0x0f&LCD)|(letra&0xf0); //máscara para os bits de controle
102     E=0;
103     atraso(250); //espera enviar o dado
104 }
105
106 void envia_str_lcd(unsigned char string[], unsigned char pos){
107     unsigned char way;
108     RS=0;
109     envia_lcd(pos);
110     RS=1;
111     for(way=0; string[way]!='\0'; way++){
112         envia_lcd(string[way]);
113     }
114 }

```

```

115 |
116 | void reset_lcd(){
117 |     unsigned char coruja;
118 |     RS=RW=0;
119 |     E=1;
120 |     LCD=0x04;
121 |     E=0;
122 |     E=1;
123 |     LCD=0x14;
124 |     E=0;
125 |     for(coruja=0;coruja!=8;coruja++){ //gera um atraso de 2ms de inicialização
126 |         atraso(250);
127 |     }
128 | }
129 |
130 | void atraso(unsigned char tempo){
131 |     //essa rotina não é recomendada para tempos de menos que 30us(22us na verdade, masssss....)
132 |     //as instruções a seguir corrigem o offset da função
133 |     tempo-=15; //1us
134 |     tempo/=6; //8us
135 |     //cada laço while demora 6us
136 |     while(tempo){ //enquanto o tempo em us não for zero
137 |         --tempo; //decrementa a cada ciclo do laço while
138 |     }
139 | }
140 |
141 | void main(){
142 |     unsigned char temp;
143 |     inicia_lcd();
144 |     envia_str_lcd("Ciclo:",0x80);
145 |     config();
146 |     while(1){
147 |         RS=0;
148 |         envia_lcd(0x86);
149 |         RS=1;
150 |         temp=ciclo/100;
151 |         envia_lcd(temp+'0');
152 |         temp=ciclo-temp*100;
153 |         envia_lcd((temp/10)+'0');
154 |         temp-=(temp/10)*10;
155 |         envia_lcd(temp+'0');
156 |     }
157 | }

```

Código-fonte I.1: Código-fonte de caracterização do funcionamento do servo-motor com AT89S52

Apêndice J

Cálculo de OG e IBU de uma receita de cerveja

Neste apêndice é apresentado o cálculo da gravidade original (OG) e do amargor (IBU) de uma receita de cerveja, baseado em [1]. Em primeiro lugar, é importante salientar que o cálculo da OG e do IBU dependem fortemente de dados do equipamento de produção de cerveja e/ou de constantes levantadas empiricamente, o que significa que este roteiro deve ser visto como um guia e, portanto, na necessidade de obter dados exatos, medições feitas por laboratórios especializados são necessárias.

Primeiramente é preciso calcular as unidades de ácidos alfa, que é um meio para quantificar a contribuição de amargor independentemente do tipo de lúpulo. Para isto, o peso em onças é multiplicado pela porcentagem de ácidos alfa, medida em laboratório e fornecida pelo fabricante. A fórmula para cálculo de AAU é apresentada na equação J.1. Outra aplicação interessante é quando uma variedade específica de lúpulo tem sua porcentagem de ácidos alfa diferente em anos diferentes: neste caso, sabendo a AAU da receita original, é fácil recalcular a massa necessária para adequar a receita às novas especificações.

$$AAU = w \cdot aa \quad (J.1)$$

Outro parâmetro de padronização de uma cerveja é a gravidade original, que nada mais é do que a densidade do mosto após o cozimento dos grãos. Ela é baseada nos pontos de contribuição (*pts*) de cada malte, ou seja, uma transformação numérica da gravidade original do malte, que quantifica a contribuição de açúcares do malte em questão para o mosto e indicada em g/cm^3 . O valor da gravidade original do malte é fornecido pelo fabricante geralmente

em pontos, mas a equação J.2 expressa o cálculo a partir do valor da densidade. Com isso é possível calcular a gravidade original do mosto, conforme descrito na equação J.3, na qual n é o número de maltes empregados na receita, η é a eficiência do equipamento, ou capacidade de extração de açúcares fermentáveis dos grãos para o mosto e V é o volume final de cerveja. Observe-se que o volume é expresso em galões e a massa em onças.

$$pts_{malte} = (1 - OG_{malte}) \cdot 1000 \quad (J.2)$$

$$OG = \left\{ \left[\sum_n (pts_n \cdot w_n) \right] \cdot \frac{\eta}{1000 \cdot V} \right\} + 1 \quad (J.3)$$

A utilização é um dos fatores mais críticos para o cálculo adequado da quantidade de IBU da cerveja. Ela é uma função da OG e do tempo de fervura. Uma vez que seu valor é fortemente dependente do vigor da fervura, da química do mosto e de outros parâmetros difíceis de quantificar, valores empíricos de constantes são empregados como um ponto de estimação de IBU — embora com o tempo o operador do equipamento deva fazer um ajuste fino destas constantes para aumentar a precisão do cálculo. O cálculo da utilização u é apresentado no conjunto de equações J.6.

$$u = f(OG) \cdot f(t), \quad \text{onde :} \quad (J.4a)$$

$$f(OG) = 1,65 \cdot 0,000125^{(OG-1)} \quad (J.4b)$$

$$f(t) = \frac{1 - e^{-0,04t}}{4,15} \quad (J.4c)$$

Finalmente, é possível calcular a quantidade de amargor da cerveja usando a fórmula J.5, na qual V é o volume final da produção expresso em galões e 74,89 é o fator de conversão de onças por galão para miligramas por litro, unidade do IBU.

$$IBU = \frac{AAU \cdot u \cdot 74,89}{V} \quad (J.5)$$

J.1 Estimação do IBU para uma receita de 20l

Para saber se 400g de lúpulos são suficientes para uma receita de 20 litros, foi feito um cálculo hipotético considerando situações extremas: densidade original alta ($1,120g/cm^3$), tempo de

fervura alto (120min), lúpulo nobre (baixo teor de ácidos alfa = 5%):

$$AAU = (400 \cdot 0,035274) \cdot 5,0 = 70,55$$

$$f(OG) = 1,65 \cdot 0,000125^{(1,12-1,00)} = 0,56$$

$$f(t) = \frac{1 - e^{-0,04 \cdot 120}}{4,15} = 0,24$$

$$u = 0,56 \cdot 0,24 = 0,135$$

$$IBU = \frac{70,55 \cdot 0,135 \cdot 74,89}{20 \cdot 0,264} = 135mg/l$$

Considerando o guia do BJCP (Beer Judge Certification Program), que é adotado por profissionais ao redor de todo o mundo para participarem de competições de cervejas como juízes, os dois estilos de cerveja com maior amargor são *Imperial India Pale Ale* e *American Barley Wine*, ambos com limite superior de 120 IBU [53] e o livro *How to Brew* que descreve somente um estilo com mais de 120 IBU — justamente o *American Barley Wine* — é razoável assumir que raramente será necessário operar a estrutura de adição de lúpulos em sua capacidade máxima.

Apêndice K

Análise do tempo de leitura do sensor DS18B20 em Python

Uma vez que o sistema operacional Linux não é de tempo real, é sensato concluir que um laço de repetição que gera um registro não terá um tempo de amostragem fixo, mas sim um valor médio, e este foi o caso do presente projeto. A confirmação se deu a partir de um estudo do tempo de amostragem de temperatura do DS18B20, sob as seguintes condições: o *log* de temperaturas foi executado independente do script que gera os gráficos em Python. O servidor em Node.js estava sendo executado porém inativo, o que significa que nenhum tipo de requisição foi feita por meio da interface web.

Foram realizadas amostragens com a prioridade de processo do Linux definida como padrão (*nice*=0) e máxima (*nice*=-20). O script Python foi modificado para que o número de pontos de amostragem fosse definido como 500 pontos, sendo os resultados estatísticos da amostragem descritos na tabela K.1 e no histograma de distribuição dos tempos de amostragem, exposto na figura K.1.

Tabela K.1: Estatísticas referentes ao tempo de amostragem de temperatura na BBB, para 500 pontos em Python

	Sem prioridade (s)	Com prioridade (s)
Média	1,7798	1,7785
Desvio padrão	2,5481e-3	2,2893e-3
Máximo	1,7891	1,7907
Mínimo	1,7773	1,7754
Mediana	1,7793	1,7774
Moda/ocorrências	1,7773 / 6	1,7773 / 8

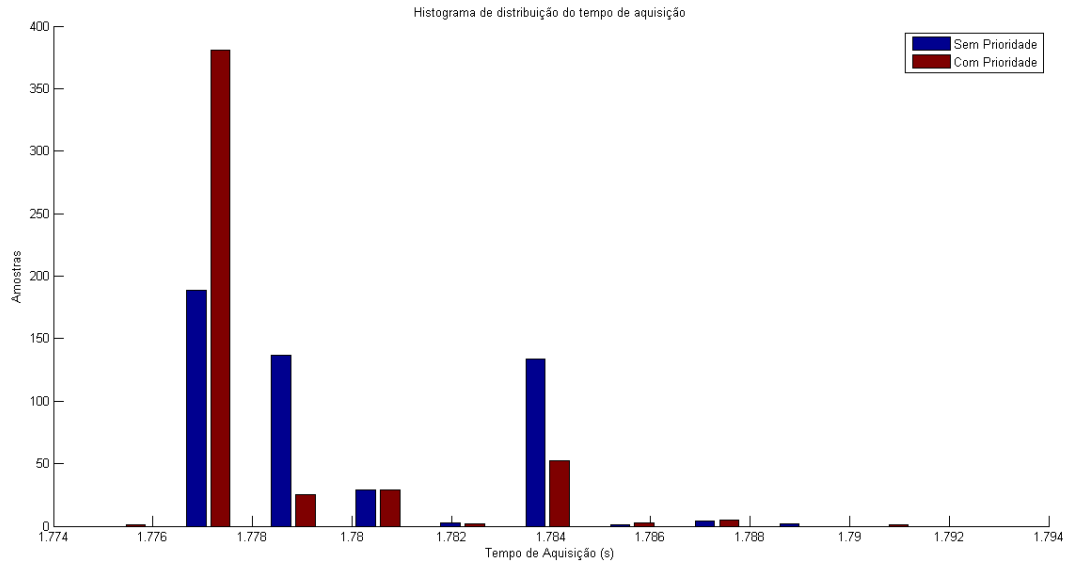


Figura K.1: Histograma de distribuição do tempo de amostragem de temperatura

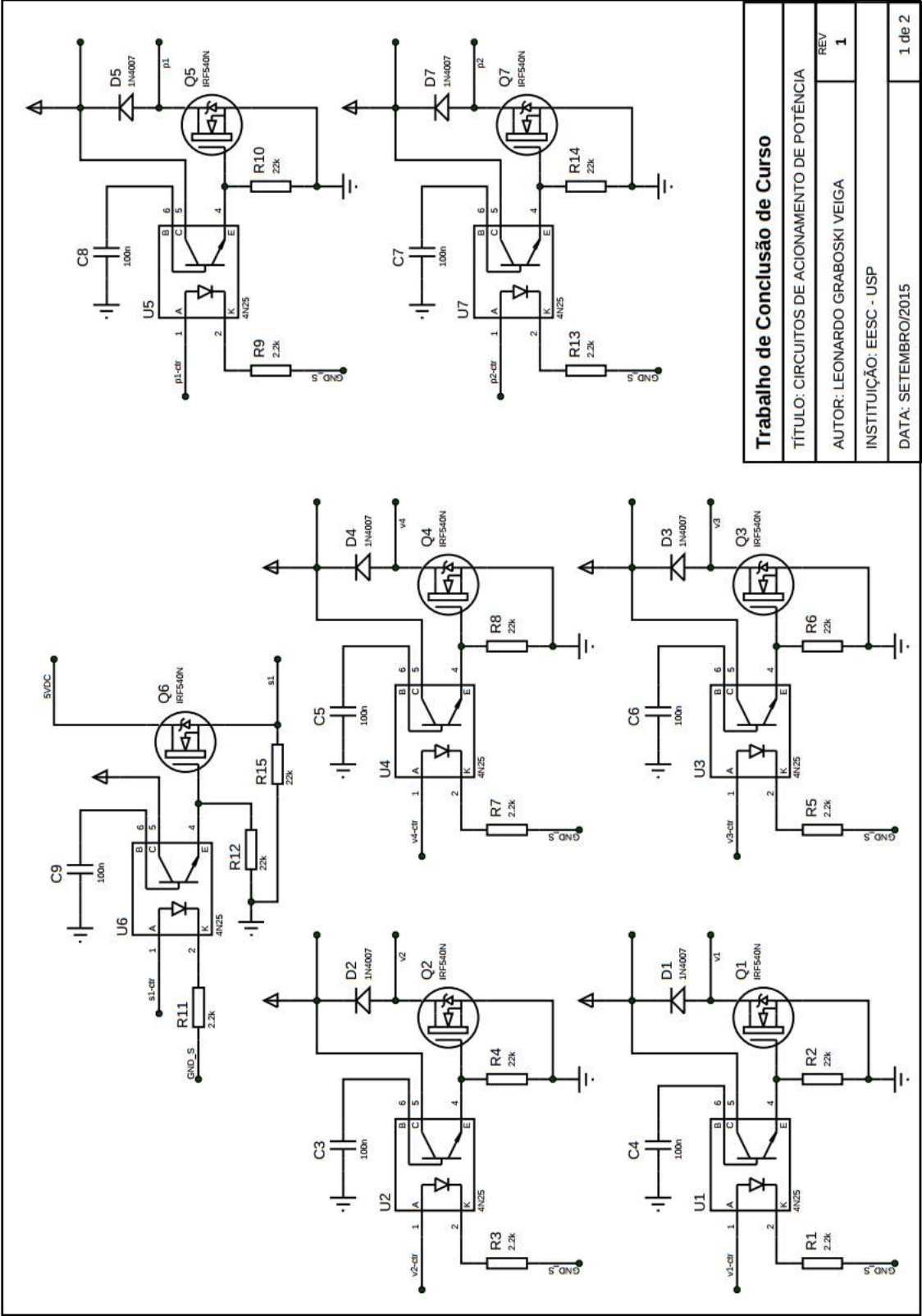
Observando e comparando as estatísticas, é possível observar uma redução de 10% na variância, indicando que a maioria das medidas está mais próxima do valor médio — que também ficou um pouco mais próximo do valor esperado — no caso com prioridade. Este fato já era esperado, pois com a prioridade, maior tempo do processador é alocado ao processo e, por conseguinte, menor é a probabilidade de que ele seja interrompido. A amplitude dos valores foi de 11,8ms sem prioridade contra 15,3ms com prioridade. No código, embora tenha sido introduzido um atraso de somente 1s, o tempo médio entre amostras de cerca de 1,77s ocorreu pois cada leitura do sensor de temperatura leva 750ms, de acordo com a documentação do *driver* do Kernel.

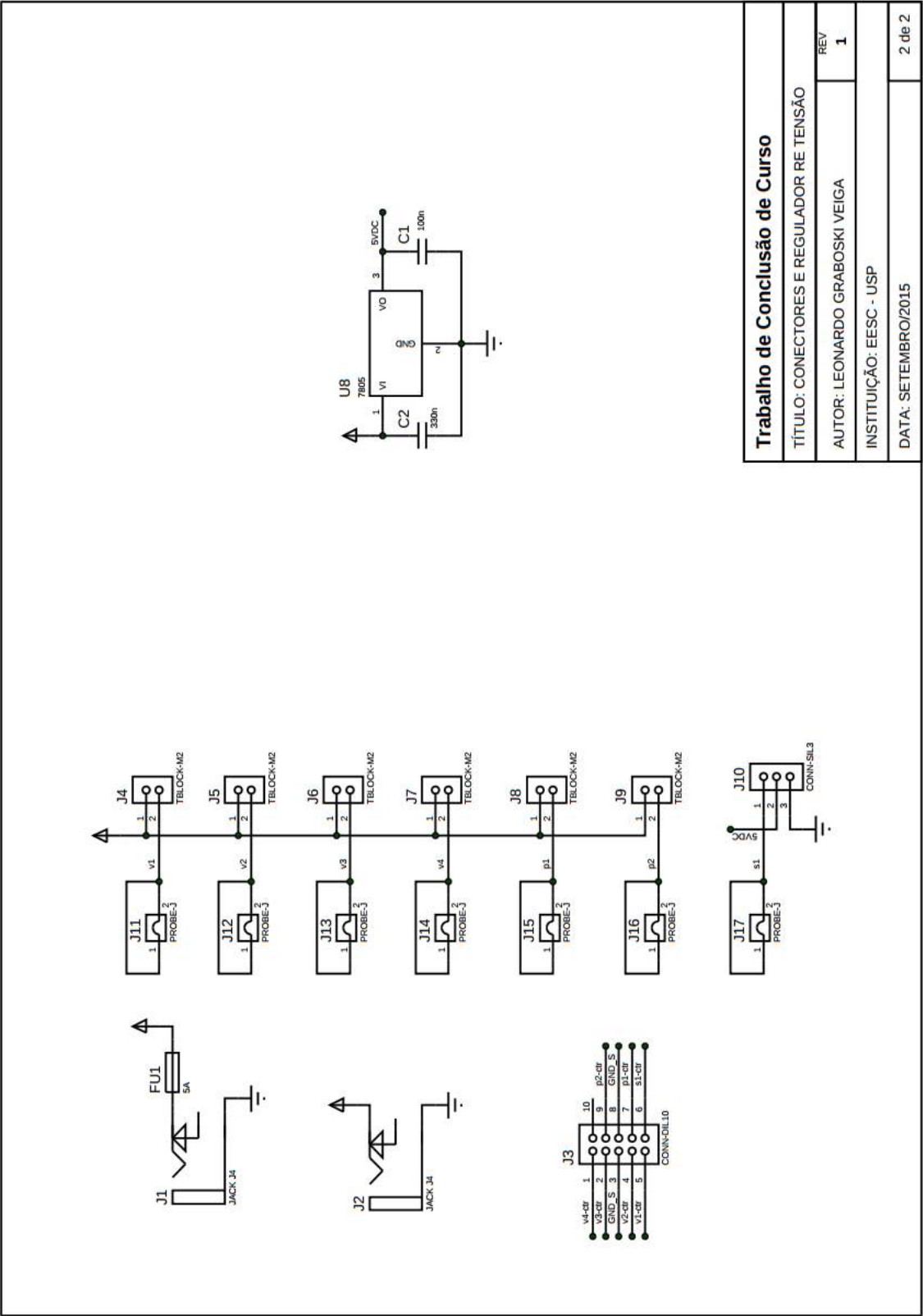
Observa-se que, pela quantidade de ocorrências da moda, ela representou 1,6% do total de medidas no melhor caso. Assim sendo, foi decidido recorrer ao histograma, que apresenta o local de uma faixa de valores. A maior concentração de dados no experimento com prioridade de processo, reafirma seu comportamento mais desejável. Nos dois casos, a maior concentração de valores está próxima da moda, rebatendo a hipótese de que ele é um valor de pouca significância se observado que representa somente 1,6% do total. Por fim, observando o histograma, não foi possível obter uma distribuição de probabilidade conhecida.

Tendo em vista o tempo de amostragem, levado em consideração o tempo de leitura do sensor pelo driver do Kernel, decidiu-se por usar um tempo de amostragem de 0,75s (módulo) + 0,25s (script Python) - 0,0298s (tempo médio de amostragem - tempo esperado de amostragem) = 0,2202s.

Apêndice L

Diagrama esquemático do circuito de acionamentos de potência





Trabalho de Conclusão de Curso	
TÍTULO: CONECTORES E REGULADOR DE TENSÃO	
AUTOR: LEONARDO GRABOSKI VEIGA	REV 1
INSTITUIÇÃO: EESC - USP	
DATA: SETEMBRO/2015	
2 de 2	

Anexo I

Diagrama de conexões elétricas da BeagleBone Black

REV	Description	DATE	BY
A4A	Initial production Release.	11/19/2012	GC
A5	On the initial production release the processors were to be found incorrect as supplied by TI. Parts while marked AM3359 were actually AM3352. This revision uses the correct parts.	1/2/2013	GC
A5A	1. Deleted R29-R44 from the LCD lines. 2. Added 47pf capacitors C156-C173 to LCD data lines to ground. 3. Changed schematic revision to A5A. 4. Changed a few footprints after PCB update for above changes. 5. Added access point for the battery function of the TPS65217C. 6. Added Ferrite beads in series with LED power and 5V power rail of the USB host connector. Required to pass FCC/CE testing due to noise emissions on that pin. 7. Added power button to enable sleep, wakeup, power down and power up features on the system. 8. Added Modification to add 100K ohm resistor to ground to prevent crosstalk when serial cable is not plugged in.	2/8/2013	GC
A5B	1. Added 100K pulldown on J1 pin 4 to prevent crosstalk when serial cable is not connected into PCB layout. 2. Changed the LED resistors to 4.75K to lower the brightness.	5/21/2013	GC
A5C	1. Changed R46, R47, R48 to 0 ohms. 2. Changed R45 to 22 Ohms. Change was made due to production failures on some boards due to differences in impedances.	6/12/2013	GC
A6	1. Moved the enable for the VDD_3V3B regulator to VDD_3V3A rail. Change was made to reduce the delay between the ramp up of the 3.3V rails. 2. Added a AND gate to the SYS_RESETrn circuitry. There is a small chance that on power up the nRESETrOUT signal on the processor may go high, causing the SYS_RESETrn signal to go HI before it should. This change reenforces the reset with the PORZn reset signal. 3. Added optional zero ohm resistor to tie GND_OSC0 to system ground.	7/25/2013	GC
A6A	1. Added optional zero ohm resistor to tie GND_OSC1 to system ground. 2. Changed C106 to a 1uF capacitor. 3. Changed C24 to a 2.2uF capacitor. 4. Made R8 installed and R9 not installed.	12/13/2013	GC
B	1.Changed the processor to the AM3358BZCZ100.	1/20/2014	GC
C	1.Increased the eMMC from 2GB to 4GB.	3/21/2014	GC

THIS SCHEMATIC IS "NOT SUPPORTED" and DOES NOT constitute a reference design. Only "community" support is allowed via resources at BeagleBoard.org/discuss.

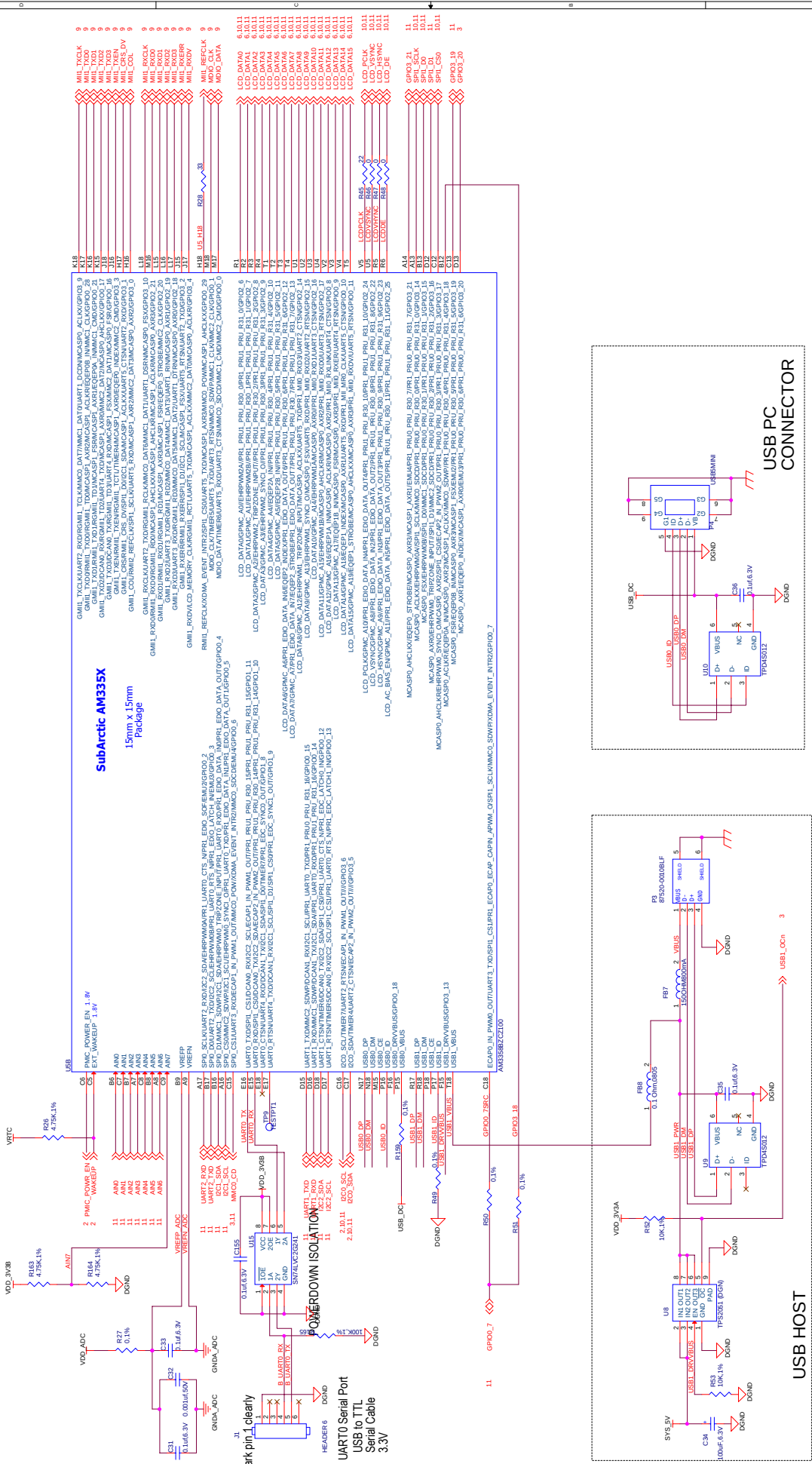
THERE IS NO WARRANTY FOR THIS DESIGN , TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE DESIGN "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE DESIGN IS WITH YOU. SHOULD THE DESIGN PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

PAGE NO.	SCHEMATIC PAGE
1	COVER PAGE
2	POWER MANAGEMENT
3	PROCESSOR 1 OF 3, JTAG HEADER
4	PROCESSOR 2 OF 3, UAB PORTS
5	PROCESSOR 3 OF 3
6	LED, CONFIGURATION AND BUTTON
7	DDR3 MEMORY
8	eMMC FLASH
9	10/100 ETHERNET
10	HDMI FRAMER
11	EXP CONN, uSD

NOTE: PCB Revision for this board is Rev B6



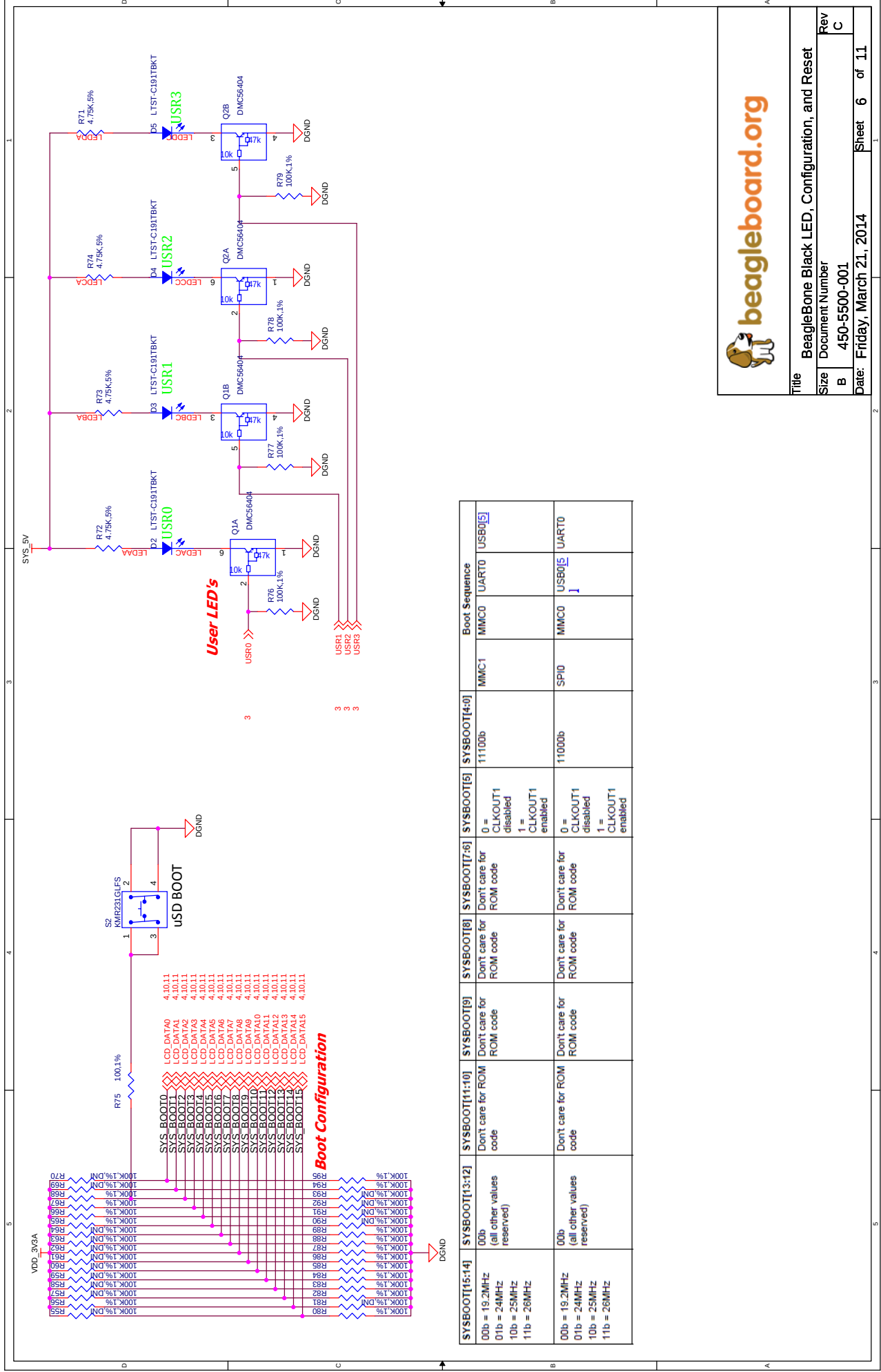
Title		BeagleBone Black Cover Page	
Size	Document Number		Rev
B	450-5500-001		C
Date: Friday, March 21, 2014		Sheet	1 of 11



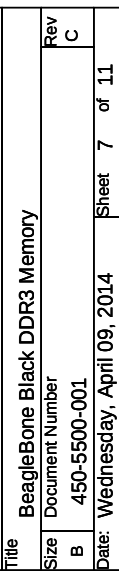
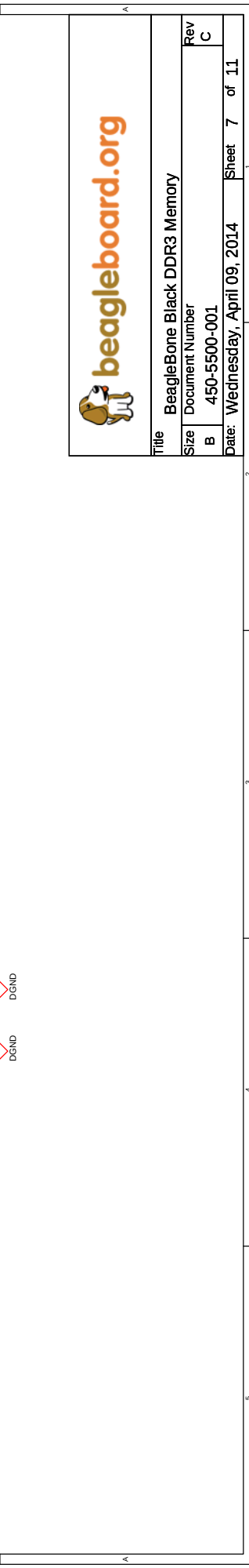
SubArctic AM335X
15mm x 15mm
Package

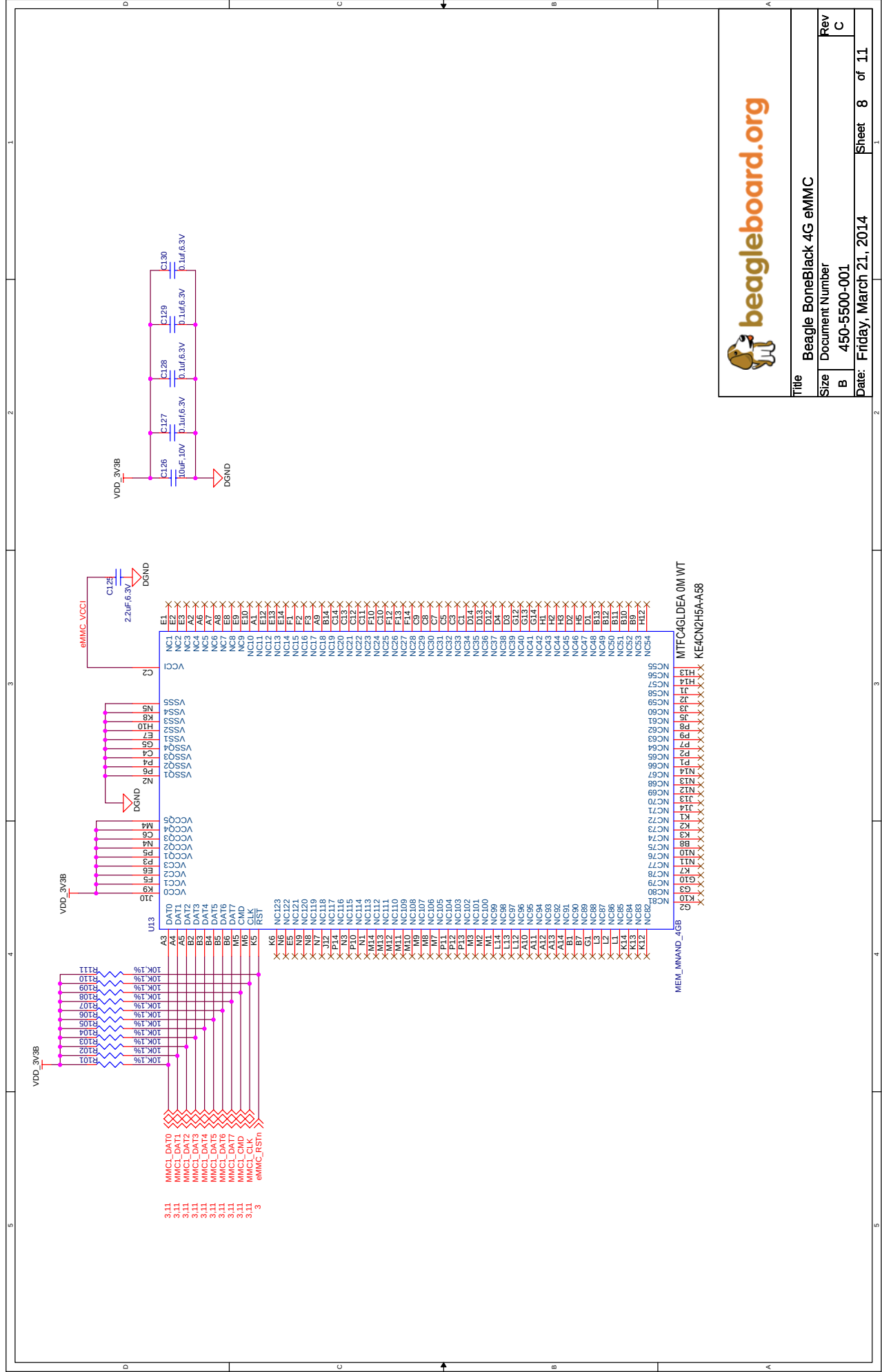


Title		BeagleBone Black Processor 2 of 3, USB, and Serial	
Size	Document Number	C	450-5500-001
Date:	Friday, March 21, 2014	Sheet	4 of 11



Title		BeagleBone Black LED, Configuration, and Reset	
Size	Document Number	Rev	
B	450-5500-001	C	
Date: Friday, March 21, 2014		Sheet	6 of 11





Title		Beagle BoneBlack 4G eMMC
Size		Document Number
B	450-5500-001	Rev C
Date: Friday, March 21, 2014		Sheet 8 of 11

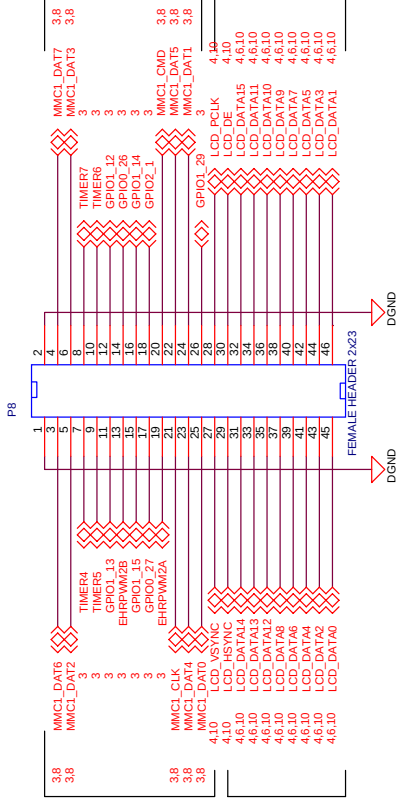


Title		BeagleBone Black Ethernet	Rev	C
Size	Document Number			
B	450-5500-001			
Date: Friday, March 21, 2014			Sheet	9 of 11



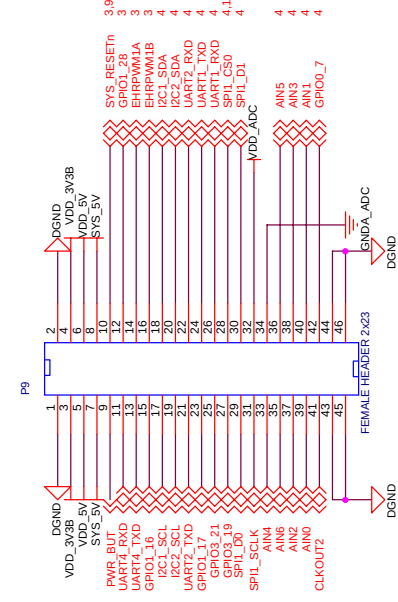
Title		BeagleBone Black HDMI Interface
Size	Document Number	Rev C
B	450-5500-001	
Date: Friday, March 21, 2014		Sheet 10 of 11

CAUTION: USED ON BOARD

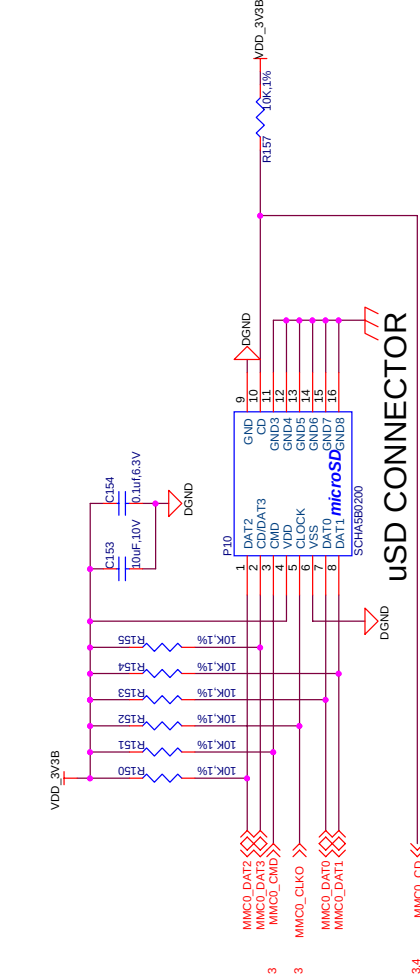


EXPANSION HEADER

CAUTION: USED ON BOARD



EXPANSION HEADER



EXPANSION HEADER

3,4



Title		BeagleBone Black Expansion Headers, uSD and EEPROM
Size	Document Number	Rev C
B	450-5500-001	
Date:	Friday, March 21, 2014	Sheet 11 of 11

Anexo II

Registadores do Módulo de Controle

Tabela II.1: Registradores do módulo de controle.
Fonte: adaptado de TEXAS INSTRUMENTS(2014)

Offset	Acrônimo
800h	conf_gpmc_ad0
804h	conf_gpmc_ad1
808h	conf_gpmc_ad2
80Ch	conf_gpmc_ad3
810h	conf_gpmc_ad4
814h	conf_gpmc_ad5
818h	conf_gpmc_ad6
81Ch	conf_gpmc_ad7
820h	conf_gpmc_ad8
824h	conf_gpmc_ad9
828h	conf_gpmc_ad10
82Ch	conf_gpmc_ad11
830h	conf_gpmc_ad12
834h	conf_gpmc_ad13
838h	conf_gpmc_ad14
83Ch	conf_gpmc_ad15
840h	conf_gpmc_a0
844h	conf_gpmc_a1

Tabela II.1: Registradores do módulo de controle.
 Fonte: adaptado de TEXAS INSTRUMENTS(2014) (continuação)

848h	conf_gpmmc_a2
84Ch	conf_gpmmc_a3
850h	conf_gpmmc_a4
854h	conf_gpmmc_a5
858h	conf_gpmmc_a6
85Ch	conf_gpmmc_a7
860h	conf_gpmmc_a8
864h	conf_gpmmc_a9
868h	conf_gpmmc_a10
86Ch	conf_gpmmc_a11
870h	conf_gpmmc_wait0
874h	conf_gpmmc_wpn
878h	conf_gpmmc_ben1
87Ch	conf_gpmmc_csn0
880h	conf_gpmmc_csn1
884h	conf_gpmmc_csn2
888h	conf_gpmmc_csn3
88Ch	conf_gpmmc_clk
890h	conf_gpmmc_adv_n_ale
894h	conf_gpmmc_oen_ren
898h	conf_gpmmc_wen
89Ch	conf_gpmmc_ben0_cle
8A0h	conf_lcd_data0
8A4h	conf_lcd_data1
8A8h	conf_lcd_data2
8ACh	conf_lcd_data3
8B0h	conf_lcd_data4
8B4h	conf_lcd_data5

Tabela II.1: Registradores do módulo de controle.
 Fonte: adaptado de TEXAS INSTRUMENTS(2014) (continuação)

8B8h	conf_lcd_data6
8BCh	conf_lcd_data7
8C0h	conf_lcd_data8
8C4h	conf_lcd_data9
8C8h	conf_lcd_data10
8CCh	conf_lcd_data11
8D0h	conf_lcd_data12
8D4h	conf_lcd_data13
8D8h	conf_lcd_data14
8DCh	conf_lcd_data15
8E0h	conf_lcd_vsync
8E4h	conf_lcd_hsync
8E8h	conf_lcd_pclk
8ECh	conf_lcd_ac_bias_en
8F0h	conf_mmc0_dat3
8F4h	conf_mmc0_dat2
8F8h	conf_mmc0_dat1
8FCh	conf_mmc0_dat0
900h	conf_mmc0_clk
904h	conf_mmc0_cmd
908h	conf_mmi1_col
90Ch	conf_mmi1_crs
910h	conf_mmi1_rx_er
914h	conf_mmi1_tx_en
918h	conf_mmi1_rx_dv
91Ch	conf_mmi1_txd3
920h	conf_mmi1_txd2
924h	conf_mmi1_txd1

Tabela II.1: Registradores do módulo de controle.
 Fonte: adaptado de TEXAS INSTRUMENTS(2014) (continuação)

928h	conf_mmi1_txd0
92Ch	conf_mmi1_tx_clk
930h	conf_mmi1_rx_clk
934h	conf_mmi1_rxd3
938h	conf_mmi1_rxd2
93Ch	conf_mmi1_rxd1
940h	conf_mmi1_rxd0
944h	conf_rmml1_ref_clk
948h	conf_mdio
94Ch	conf_mdc
950h	conf_spi0_sclk
954h	conf_spi0_d0
958h	conf_spi0_d1
95Ch	conf_spi0_cs0
960h	conf_spi0_cs1
964h	conf_ecap0_in_pwm0_out
968h	conf_uart0_ctsn
96Ch	conf_uart0_rtsn
970h	conf_uart0_rdx
974h	conf_uart0_tdx
978h	conf_uart1_ctsn
97Ch	conf_uart1_rtsn
980h	conf_uart1_rdx
984h	conf_uart1_tdx
988h	conf_i2c0_sda
98Ch	conf_i2c0_scl
990h	conf_mcasp0_aclrx
994h	conf_mcasp0_fsx

Tabela II.1: Registradores do módulo de controle.
 Fonte: adaptado de TEXAS INSTRUMENTS(2014) (continuação)

998h	conf_mcaspo_axr0
99Ch	conf_mcaspo_ahclkr
9A0h	conf_mcaspo_aclkr
9A4h	conf_mcaspo_fsr
9A8h	conf_mcaspo_axr1
9ACh	conf_mcaspo_ahclkx
9B0h	conf_xdma_event_intr0
9B4h	conf_xdma_event_intr1
9B8h	conf_warmrstn
9C0h	conf_nnmi
9D0h	conf_tms
9D4h	conf_tdi
9D8h	conf_tdo
9DCh	conf_tck
9E0h	conf_trstn
9E4h	conf_emu0
9E8h	conf_emu1
9F8h	conf_rtc_pwronrstn
9FCh	conf_pmic_power_en
A00h	conf_ext_wakeup
A04h	conf_rtc_kaldo_enn
A1Ch	conf_usb0_drvvbus
A34h	conf_usb1_drvvbus

Anexo III

Válvula Danfoss EV210BD 032U3620

Ficha técnica

Válvulas solenoides de 2/2 vias de operação direta Tipo EV210B



A EV210B cobre uma ampla linha de válvulas solenoides de 2/2 vias de operação direta para uso universal.

A EV210B é uma válvula muito robusta com alto desempenho e pode ser utilizada em todos os tipos e condições de trabalho, até mesmo nas mais adversas aplicações industriais, tais como controle e fechamento.

Características e versões:

- Para água, óleo, ar comprimido e meios neutros similares.
- Faixa de fluxo: 0 – 8 m³/h
- Pressão diferencial: 0 – 30 bar
- Temperatura do meio: -30 a 140 °C
- Temperatura ambiente: Até 80 °C
- Grau de proteção: Até IP67
- Conexões de rosca: G 1/8 – G 1
- DN 1.5 – 25
- Viscosidade: Até 50 cSt
- A válvula pode ser usada para vácuo
- Versão EV210B em latão para água, óleo, ar comprimido e meios neutros similares
- Versão EV210 em aço inoxidável para líquidos e gases neutros e agressivos.
- Também disponível com conexão NPT.

Corpo da válvula em latão
EV210B, NF



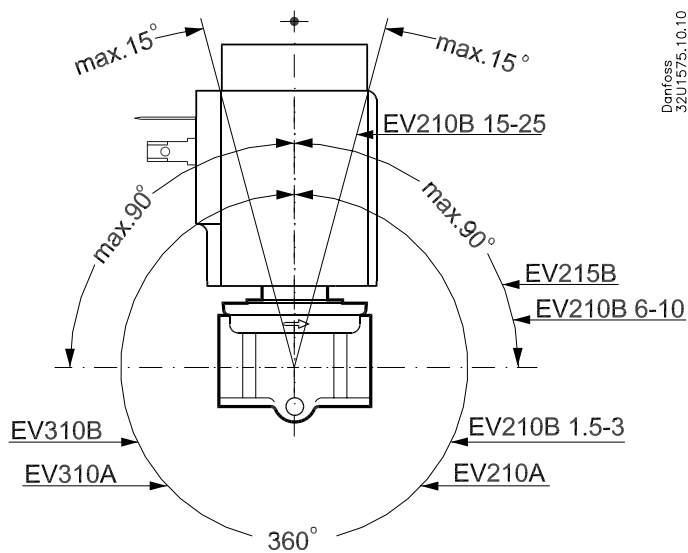
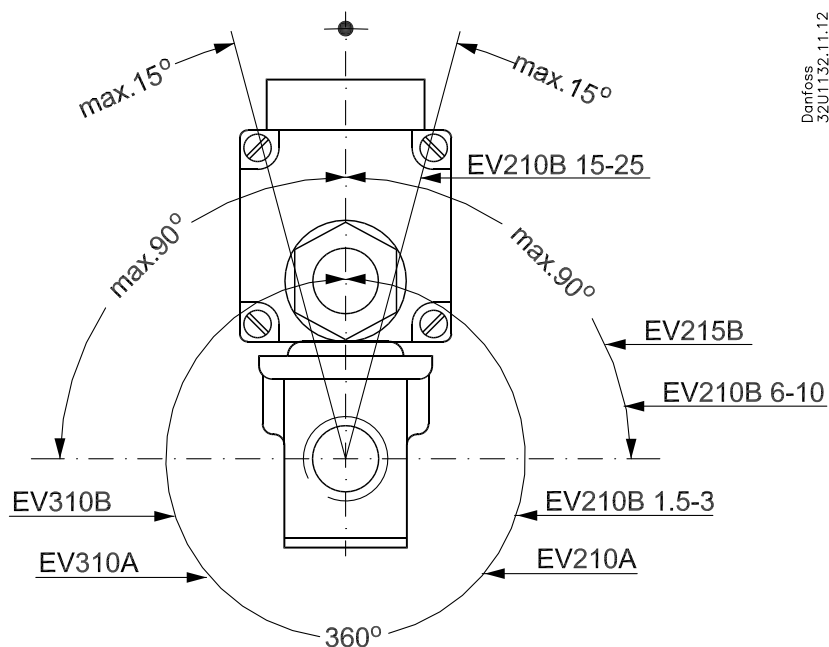
Con- ex. ISO 228/1	Material de vedação	Diâ- metro do orifí- cio	Valor kv [m³/h]	Pressão diferencial mín. a máx. [bar]/ tipo de bobina ²⁾							Temperatura do meio mín. a máx. [°C]	Código
				BA 9 [W c.a.]	BA 15 [W c.c.]	BD 15 [W c.a.]	BB 10 [W c.a.]	BB 18 [W c.c.]	BG 12 [W c.a.]	BG 20 [W c.c.]		
G 1/8	EPDM ¹⁾	1.5	0.08	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	-30 – 120	032U5701
	FKM		0.08	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	-10 – 100	032U5702
	FKM	2.0	0.15	0 – 30	0 – 20	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	-10 – 100	032U5704
	EPDM ¹⁾	3.0	0.30	0 – 15	0 – 9	0 – 24	0 – 20	0 – 13	0 – 30	0 – 25	-30 – 120	032U5705
	FKM		0.30	0 – 15	0 – 9	0 – 24	0 – 20	0 – 13	0 – 30	0 – 25	-10 – 100	032U5706
G 1/4	FKM	1.5	0.08	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	-10 – 100	032U3629
	EPDM ¹⁾	2.0	0.15	0 – 30	0 – 20	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	-30 – 120	032U5707
	FKM		0.15	0 – 30	0 – 20	0 – 30	0 – 30	0 – 30	0 – 30	0 – 30	-10 – 100	032U5708
	EPDM ¹⁾	3.0	0.30	0 – 15	0 – 9	0 – 24	0 – 20	0 – 13	0 – 30	0 – 25	-30 – 120	032U5709
	FKM		0.30	0 – 15	0 – 9	0 – 24	0 – 20	0 – 13	0 – 30	0 – 25	-10 – 100	032U5710
	EPDM ¹⁾	4.5	0.55	0 – 8	0 – 3.5	0 – 12	0 – 10	0 – 4.5	0 – 13	0 – 9	-30 – 120	032U3600
	FKM		0.55	0 – 8	0 – 3.5	0 – 12	0 – 10	0 – 4.5	0 – 13	0 – 9	-10 – 100	032U3601
	EPDM ¹⁾	6.0	0.70	0 – 2.5	0 – 1.0	0 – 3.3	0 – 4.0	0 – 2.0	0 – 6	0 – 4.5	-30 – 120	032U3602
	FKM		0.70	0 – 2.5	0 – 1.0	0 – 3.3	0 – 4.0	0 – 2.0	0 – 6	0 – 4.5	-10 – 100	032U3603
	EPDM ¹⁾	8.0	1.00	0 – 1.5	0 – 0.5	0 – 2.0	0 – 2.0	0 – 1.2	0 – 3	0 – 2.5	-30 – 120	032U3609
G 3/8	FKM		1.00	0 – 1.5	0 – 0.5	0 – 2.0	0 – 2.0	0 – 1.2	0 – 3	0 – 2.5	-10 – 100	032U3610
	EPDM ¹⁾	10.0	1.50	0 – 0.8	0 – 0.3	0 – 1.1	0 – 1.2	0 – 0.6	0 – 1.6	0 – 1.3	-30 – 120	032U3611
	FKM		1.50	0 – 0.8	0 – 0.3	0 – 1.1	0 – 1.2	0 – 0.6	0 – 1.6	0 – 1.3	-10 – 100	032U3612
	EPDM ¹⁾	15.0	2.50	0 – 0.25	-	0 – 0.4	0 – 0.3	0 – 0.15	0 – 0.45	0 – 0.4	-30 – 120	032U3613
	FKM		2.50	0 – 0.25	-	0 – 0.4	0 – 0.3	0 – 0.15	0 – 0.45	0 – 0.4	-10 – 100	032U3614
	EPDM ¹⁾	8.0	1.00	0 – 1.5	0 – 0.5	0 – 2.0	0 – 2.0	0 – 1.2	0 – 3	0 – 2.5	-30 – 120	032U3615
	FKM		1.00	0 – 1.5	0 – 0.5	0 – 2.0	0 – 2.0	0 – 1.2	0 – 3	0 – 2.5	-10 – 100	032U3616
	EPDM ¹⁾	10.0	1.50	0 – 0.8	0 – 0.3	0 – 1.1	0 – 1.2	0 – 0.6	0 – 1.6	0 – 1.3	-30 – 120	032U3617
	FKM		1.50	0 – 0.8	0 – 0.3	0 – 1.1	0 – 1.2	0 – 0.6	0 – 1.6	0 – 1.3	-10 – 100	032U3618
	EPDM ¹⁾	15.0	2.85	0 – 0.25	-	0 – 0.4	0 – 0.3	0 – 0.15	0 – 0.45	0 – 0.4	-30 – 120	032U3619
G 1/2	FKM		2.85	0 – 0.25	-	0 – 0.4	0 – 0.3	0 – 0.15	0 – 0.45	0 – 0.4	-10 – 100	032U3620
G 3/4	EPDM ¹⁾	20.0	4.50	-	-	-	0 – 0.28	0 – 0.12	0 – 0.4	0 – 0.35	-30 – 120	032U3621
	FKM		4.50	-	-	-	0 – 0.28	0 – 0.12	0 – 0.4	0 – 0.35	-10 – 100	032U3622
G 1	EPDM ¹⁾	25.0	8.00	-	-	-	0 – 0.25	0 – 0.09	0 – 0.35	0 – 0.2	-30 – 120	032U3623
	FKM		8.00	-	-	-	0 – 0.25	0 – 0.09	0 – 0.35	0 – 0.2	-10 – 100	032U3624

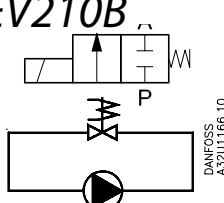
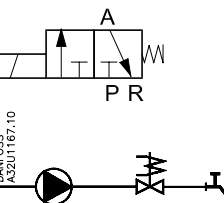
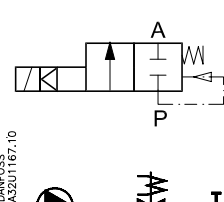
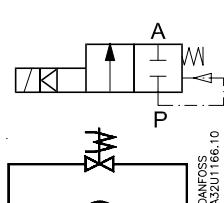
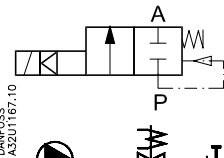
¹⁾ Vapor de baixa pressão de 140 °C / 3.6 bar, orifício DN 1.5 – 4.5.

- Vapor de baixa pressão: DN 1.5 – 3 Usar bobina tipo BB ou BG.
DN 4.5 Usar bobina tipo BG.

²⁾ A faixa de pressão pode ser estendida para usar em um vácuo grosso, normalmente até 99% de vácuo (10 mbar), dependendo da aplicação.

Direct-operated valves



	Medium			
	Air and neutral gases	Water	Oil	Steam
EV210B  	X	X	X	
EV310B  	X	X	X	
EV220B   	X	X	X	
	X	X	X	X
EV250B  	X	X	X	X
EV225B  				X

For other characteristics please see the valve selection guide

Characteristics		Description
Connection [ISO 228/1]	Function	
G 3/8" - G 1"	NC/NO	EV210B covers a wide range of direct-operated 2/2-way solenoid valves for universal use. EV210B is a real robust valve program with high performance and can be used in all kind of tough working conditions.
G 1/8" - G 3/8"	NC/NO	EV310B is a direct-operated 3/2- way solenoid valve. The valve is especially used in connection with air-operated valves to allow air supply and air relief for the air actuator.
G 1/4" - G 1"	NC/NO	EV220B 6-22 is a direct servo-operated 2/2-way solenoid valve program. This program is especially for OEM applications demanding a robust solution and moderate flow rates.
G 1/2" - G 2"	NC/NO	EV220B 15-50 is a universal indirect servo-operated 2/2-way solenoid valve program. Valve body in brass, dezincification resistant brass and stainless steel ensures that a broad variety of applications can be covered.
G 3/8" - G 1"	NC	EV250B with assisted lift is especially to use in closed circuits with low differential pressure, but demanding moderate flow rates. Valve body in DZR brass ensures a long life, even in connection with aggressive steam media.
G 1/4" - G 1"	NC	The EV225B design is based on a PTFE diaphragm and valve body in dezincification resistant brass, ensuring high reliable function and long life even in connection with contaminated steam.

Spare parts set for EV210B NC

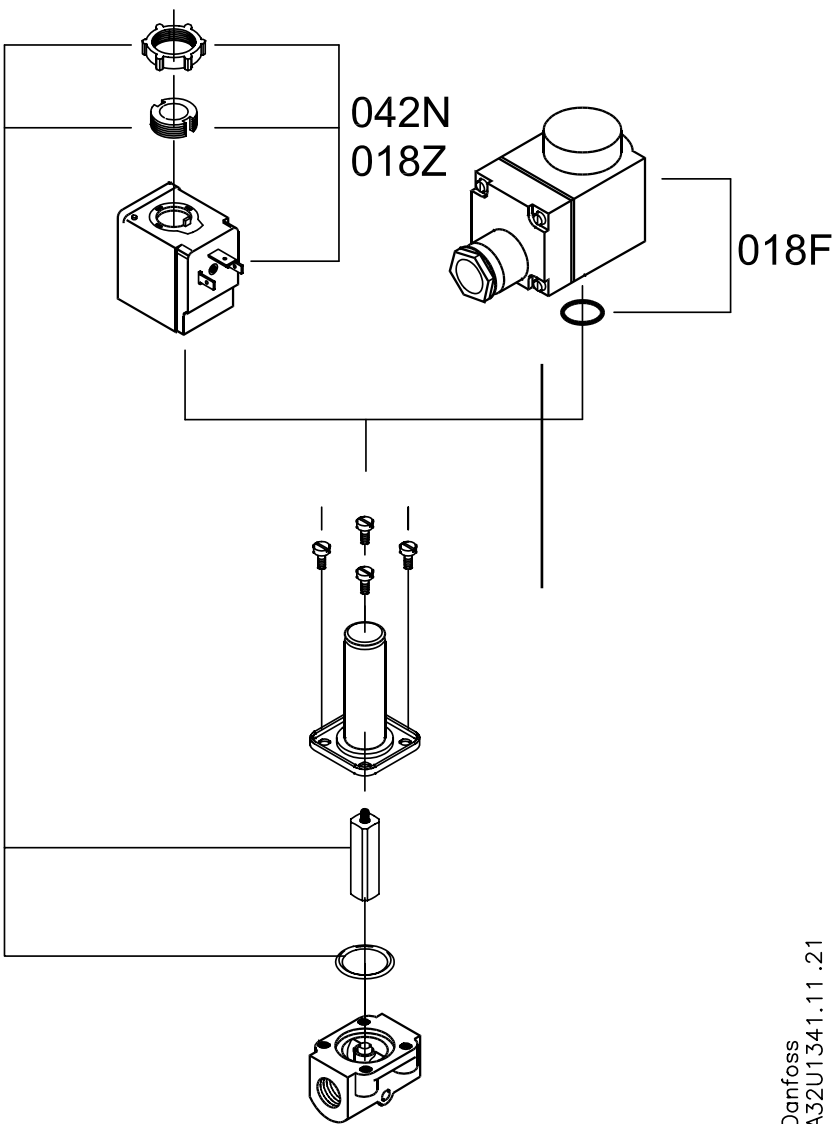
The spare parts set contains locking button and nut for coil, armature with valve plate and spring, and O-rings.

EPDM¹⁾ versions

Type	Code no.
EV210B 1.5 - 4.5	032U6000
EV210B 6,8,10	032U2006

FKM¹⁾ versions

Type	Code no.
EV210B 1.5 - 4.5	032U2003
EV210B 6,8,10	032U2011

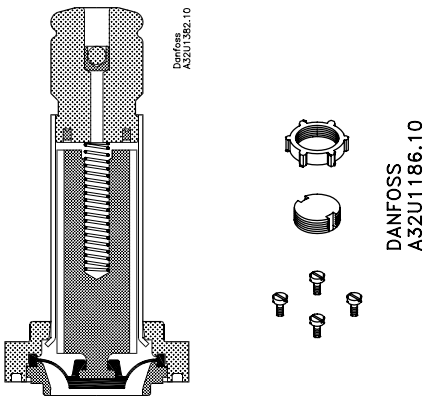


Danfoss
A32U1341.11 .21

¹⁾ See page 20 for description of seal materials

Isolating diaphragm kit for EV210B 1.5-4.5 NC and EV220B 15-50 NC

Avoids build up of contaminates that can block movement of the armature. Permits use of more aggressive media that would normally attack the armature. Gel filled; guarantees operation after long periods on inactivity.



Seal material	Code no.
EPDM ¹⁾	042U1009
FKM ¹⁾	042U1010

¹⁾ See page 20 for description of seal materials

Anexo IV

Bobina Danfoss 042N7550

Data sheet

Solenoid coils



Danfoss solenoid valves and coils are usually ordered separately to allow maximum flexibility, enabling you to select a valve and coil combination to best suit your needs. The Danfoss coil program consists of both the easy-to-handle Clip-On system and traditional coils with threaded fastener. Also, with approvals such as EEx/ATEX and UL, we offer a wide range of application specific coils for e.g. steam or hazardous areas.

Features

- Encapsulated coils with long operating life, even under extreme conditions
- Standard coils for AC or DC
- Standard coils from 12 V to 400 V, 50, 60 or 50/60 Hz
- Coils can be fitted without use of tools
- Coils can only be removed with use of tools
- Standard coils available with:
 - Cable plugs
 - Industrial plugs
 - Terminal box
 - 3 core cable
 - Junction box
 - Conduit hub

BA, High performance coils



- Ambient temperature: Up to 40 °C
- IP00 version with DIN 43650 A spade connectors
- IP20 version with protective cap
- IP65 version with mounted cable plug

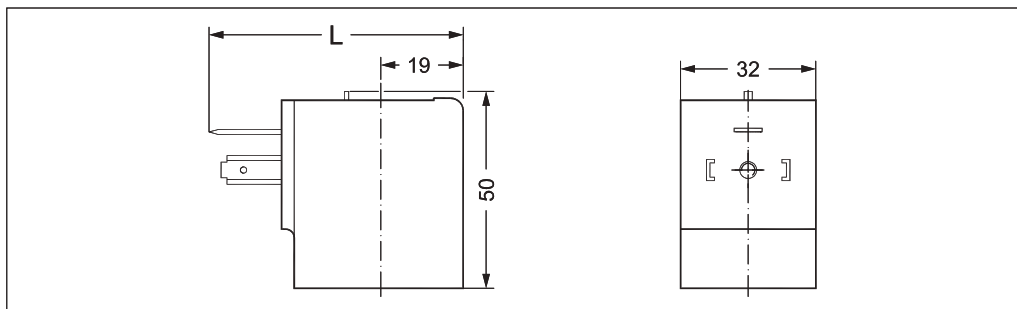
Coil type	Supply voltage		Frequency [Hz]	Power consumption holding [W]	Code no.
	[V AC]	[V DC]			
BA024A	24	–	50	9	042N7508
BA048A	48	–	50	9	042N7510
BA115A	115	–	50	9	042N7512
BA230A	220 – 230	–	50	9	042N7501
BA240A	240	–	50	9	042N7502
BA400A	380 – 400	–	50	9	042N7504
BA024B	24	–	60	9	042N7520
BA115B	115	–	60	9	042N7522
BA220B	220	–	60	9	042N7523
BA012D	–	12	–	15	042N7550
BA024D	–	24	–	15	042N7551

Technical data

Design	In accordance with VDE 0580	
Voltage variation	220/380 V AC	-15%, +10%
	230/400 V AC	-10%, +6%
	Other AC coils with NC valve	-15%, +10%
	Other AC coils with NO valve and all DC	±10%
Power consumption, cut in	39 VA AC coils only	
Insulation of coil windings	Class H according to IEC 85	
Connection	Spade connector in accordance with DIN 43650 form A	
Enclosure, IEC 529	IP00 with spade connec. IP20 with protective cap, IP65 with cable plug	
Ambient temperature	Max. 40 °C	
Duty rating	Continuous	
Plug type	Cable plug	

Dimensions and weight

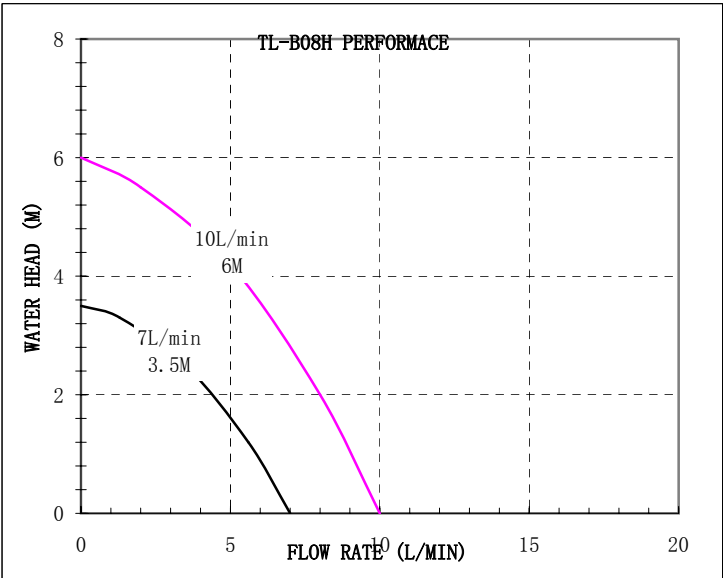
Type	L without cable plug [mm]	L with protective cap [mm]	L with cable plug [mm]	Weight [kg]
BA	54	71	79	0.16



Anexo V

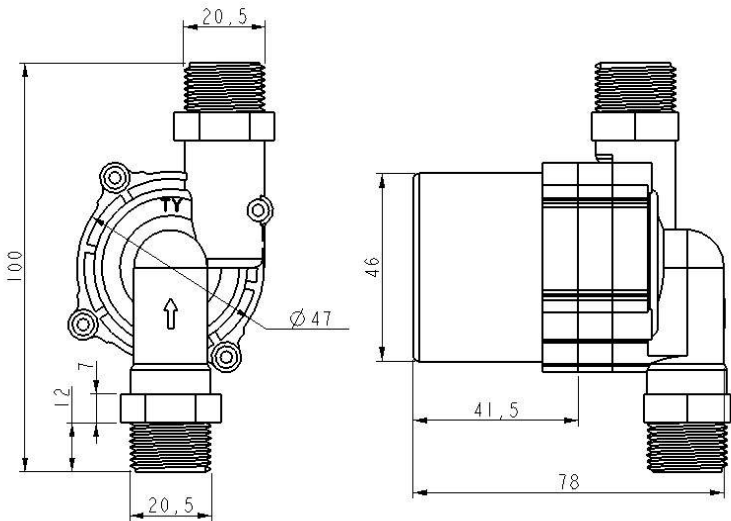
Bomba centrífuga Topsflo B08H121006

TOPSFLO / TL- B08H



Model	Product Code	Max Flow Rate (L/M)	Rated Voltage (DC)	Current (A)	Max Water Head (M)	Power (W)
TL-B08H (100℃)	TL-B08H-12-0703	7	12VDC	0.9	3.5	11
	TL-B08H-12-1006	10	12VDC	1.6	6	20
	TL-B08H-24-0703	7	24VDC	0.5	3.5	12
	TL-B08H-24-1006	10	24VDC	0.8	6	20
	can customize specification of TL-B08H-24-1208, but which only for ≤70℃ liquid only					
TL-B08H/PV (100℃)	TL-B08H/PV-12/24-1006	10	12V (8V~26V)	0.8	6	20
	special designe low starting, can be powered directly by solar panels, apply for both 12V and 24V voltage					
	pump itself meet: IH /F (high temperature 100℃ / food grade) can be customized: IS (submersible) ONLY for liquid ≤60℃, ambient temperature≤40℃					

Important Note:
TOPSFLO not confirmation also not recommend the TL-B08H or H/PV pump used for Solar Hot Water System Circulation, for the pump is belong to normal plastic circulation pump (not with specializing enhanced design) which can not meet 10bar system pressure.
TOPSFLO also not responsible for any quality problems happened in short time or years due to usage of this pump for Solar Hot Water System Circulation which usually with system pressure requirements to 10bar, also with all materials requirements should above 200°C,etc



Anexo VI

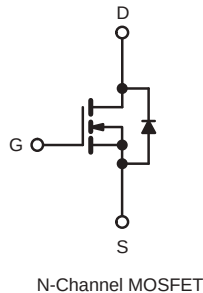
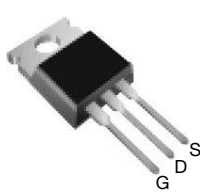
Transistor IRF540N

Power MOSFET

PRODUCT SUMMARY

V_{DS} (V)	100	
$R_{DS(on)}$ (Ω)	$V_{GS} = 10\text{ V}$	0.077
Q_g (Max.) (nC)	72	
Q_{gs} (nC)	11	
Q_{gd} (nC)	32	
Configuration	Single	

TO-220AB



FEATURES

- Dynamic dV/dt Rating
- Repetitive Avalanche Rated
- 175 °C Operating Temperature
- Fast Switching
- Ease of Paralleling
- Simple Drive Requirements
- Compliant to RoHS Directive 2002/95/EC



Available
RoHS*
COMPLIANT

DESCRIPTION

Third generation Power MOSFETs from Vishay provide the designer with the best combination of fast switching, ruggedized device design, low on-resistance and cost-effectiveness.

The TO-220AB package is universally preferred for all commercial-industrial applications at power dissipation levels to approximately 50 W. The low thermal resistance and low package cost of the TO-220AB contribute to its wide acceptance throughout the industry.

ORDERING INFORMATION

Package	TO-220AB
Lead (Pb)-free	IRF540PbF SiHF540-E3
SnPb	IRF540 SiHF540

ABSOLUTE MAXIMUM RATINGS ($T_C = 25\text{ }^{\circ}\text{C}$, unless otherwise noted)

PARAMETER	SYMBOL	LIMIT	UNIT
Drain-Source Voltage	V_{DS}	100	V
Gate-Source Voltage	V_{GS}	± 20	
Continuous Drain Current	V_{GS} at 10 V	$T_C = 25\text{ }^{\circ}\text{C}$	A
		$T_C = 100\text{ }^{\circ}\text{C}$	
Pulsed Drain Current ^a	I_{DM}	110	
Linear Derating Factor		1.0	W/ $^{\circ}\text{C}$
Single Pulse Avalanche Energy ^b	E_{AS}	230	mJ
Repetitive Avalanche Current ^a	I_{AR}	28	A
Repetitive Avalanche Energy ^a	E_{AR}	15	mJ
Maximum Power Dissipation	P_D	150	W
Peak Diode Recovery dV/dt ^c	dV/dt	5.5	V/ns
Operating Junction and Storage Temperature Range	T_J, T_{stg}	- 55 to + 175	$^{\circ}\text{C}$
Soldering Recommendations (Peak Temperature)	for 10 s	300 ^d	
Mounting Torque	6-32 or M3 screw	10	lbf · in
		1.1	N · m

Notes

- Repetitive rating; pulse width limited by maximum junction temperature (see fig. 11).
- $V_{DD} = 25\text{ V}$, starting $T_J = 25\text{ }^{\circ}\text{C}$, $L = 440\text{ }\mu\text{H}$, $R_g = 25\text{ }\Omega$, $I_{AS} = 28\text{ A}$ (see fig. 12).
- $I_{SD} \leq 28\text{ A}$, $dI/dt \leq 170\text{ A}/\mu\text{s}$, $V_{DD} \leq V_{DS}$, $T_J \leq 175\text{ }^{\circ}\text{C}$.
- 1.6 mm from case.

* Pb containing terminations are not RoHS compliant, exemptions may apply

IRF540, SiHF540

Vishay Siliconix



THERMAL RESISTANCE RATINGS

PARAMETER	SYMBOL	TYP.	MAX.	UNIT
Maximum Junction-to-Ambient	R_{thJA}	-	62	°C/W
Case-to-Sink, Flat, Greased Surface	R_{thCS}	0.50	-	
Maximum Junction-to-Case (Drain)	R_{thJC}	-	1.0	

SPECIFICATIONS ($T_J = 25\text{ }^{\circ}\text{C}$, unless otherwise noted)

PARAMETER	SYMBOL	TEST CONDITIONS		MIN.	TYP.	MAX.	UNIT
Static							
Drain-Source Breakdown Voltage	V_{DS}	$V_{GS} = 0\text{ V}$, $I_D = 250\text{ }\mu\text{A}$		100	-	-	V
V_{DS} Temperature Coefficient	$\Delta V_{DS}/T_J$	Reference to $25\text{ }^{\circ}\text{C}$, $I_D = 1\text{ mA}$		-	0.13	-	V/ $^{\circ}\text{C}$
Gate-Source Threshold Voltage	$V_{GS(th)}$	$V_{DS} = V_{GS}$, $I_D = 250\text{ }\mu\text{A}$		2.0	-	4.0	V
Gate-Source Leakage	I_{GSS}	$V_{GS} = \pm 20\text{ V}$		-	-	± 100	nA
Zero Gate Voltage Drain Current	I_{DSS}	$V_{DS} = 100\text{ V}$, $V_{GS} = 0\text{ V}$		-	-	25	μA
		$V_{DS} = 80\text{ V}$, $V_{GS} = 0\text{ V}$, $T_J = 150\text{ }^{\circ}\text{C}$		-	-	250	
Drain-Source On-State Resistance	$R_{DS(on)}$	$V_{GS} = 10\text{ V}$	$I_D = 17\text{ A}^b$	-	-	0.077	Ω
Forward Transconductance	g_{fs}	$V_{DS} = 50\text{ V}$, $I_D = 17\text{ A}^b$		8.7	-	-	S
Dynamic							
Input Capacitance	C_{iss}	$V_{GS} = 0\text{ V}$, $V_{DS} = 25\text{ V}$, $f = 1.0\text{ MHz}$, see fig. 5		-	1700	-	pF
Output Capacitance	C_{oss}			-	560	-	
Reverse Transfer Capacitance	C_{rss}			-	120	-	
Total Gate Charge	Q_g	$V_{GS} = 10\text{ V}$	$I_D = 17\text{ A}$, $V_{DS} = 80\text{ V}$, see fig. 6 and 13 ^b	-	-	72	nC
Gate-Source Charge	Q_{gs}			-	-	11	
Gate-Drain Charge	Q_{gd}			-	-	32	
Turn-On Delay Time	$t_{d(on)}$	$V_{DD} = 50\text{ V}$, $I_D = 17\text{ A}$ $R_g = 9.1\text{ }\Omega$, $R_D = 2.9\text{ }\Omega$, see fig. 10 ^b		-	11	-	ns
Rise Time	t_r			-	44	-	
Turn-Off Delay Time	$t_{d(off)}$			-	53	-	
Fall Time	t_f			-	43	-	
Internal Drain Inductance	L_D	Between lead, 6 mm (0.25") from package and center of die contact 		-	4.5	-	nH
Internal Source Inductance	L_S			-	7.5	-	
Drain-Source Body Diode Characteristics							
Continuous Source-Drain Diode Current	I_S	MOSFET symbol showing the integral reverse p - n junction diode 		-	-	28	A
Pulsed Diode Forward Current ^a	I_{SM}			-	-	110	
Body Diode Voltage	V_{SD}	$T_J = 25\text{ }^{\circ}\text{C}$, $I_S = 28\text{ A}$, $V_{GS} = 0\text{ V}^b$		-	-	2.5	V
Body Diode Reverse Recovery Time	t_{rr}	$T_J = 25\text{ }^{\circ}\text{C}$, $I_F = 17\text{ A}$, $dI/dt = 100\text{ A}/\mu\text{s}^b$		-	180	360	ns
Body Diode Reverse Recovery Charge	Q_{rr}			-	1.3	2.8	μC
Forward Turn-On Time	t_{on}	Intrinsic turn-on time is negligible (turn-on is dominated by L_S and L_D)					

Notes

- a. Repetitive rating; pulse width limited by maximum junction temperature (see fig. 11).
b. Pulse width $\leq 300\text{ }\mu\text{s}$; duty cycle $\leq 2\%$.

Anexo VII

Servo-motor TowerPro MG995

TowerPro MG995 Servo

Basic Information	
Modulation:	Digital
Torque:	4.8V: 130.5 oz-in (9.40 kg-cm) 6.0V: 152.8 oz-in (11.00 kg-cm)
Speed:	4.8V: 0.20 sec/60° 6.0V: 0.16 sec/60°
Weight:	1.94 oz (55.0 g)
Dimensions:	Length: 1.60 in (40.7 mm) Width: 0.78 in (19.7 mm) Height: 1.69 in (42.9 mm)
Motor Type:	3-pole
Gear Type:	Metal
Rotation/Support:	? (add)

Special Notes

- This servo can be purchased with 180 or 360 degree rotation for robotics applications.

Additional Specifications

Rotational Range:	? (add)
Pulse Cycle:	1 ms
Pulse Width:	? (add)
Connector Type:	JR



Brand:	Tower pro
Product Number:	? (add)
Suggested Retail:	? (add)
Street Price:	8.50 USD
Compare:	add

User Reviews

Number of Reviews:	18
Average Rating:	2.7 / 5.0