Marcelo Tomasini

Projeto de Placa Eletrônica Desenvolvida para Gerenciamento de Plataforma Robótica

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

Curso de Engenharia Elétrica com ênfase em
Eletrônica

ORIENTADOR: Evandro Luís Linhari Rodrigues

São Carlos
2011

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Tomasini, Marcelo
T655p Projeto de placa eletrônica desenvolvida para gerenciamento de plataforma robótica / Marcelo Tomasini ; orientador Evandro Luís Linhari Rodrigues -- São Carlos, 2011.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2011.

1. Robótica. 2. Futebol de robôs. 3. Placa principal. 4. Robô. 5. Projeto de placa eletrônica. 6. Eagle 5.4.0. I. Titulo.

Projeto de Placa Eletrônica Desenvolvida para Gerenciamento de Plataforma Robótica

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do título de Engenheiro Eletricista, do curso de Engenharia Elétrica com Ênfase em Eletrônica da Escola de Engenharia de São Carlos da Universidade de São Paulo.

Professor orientador: Evandro Luís Linhari Rodrigues

São Carlos, 2011

“Dedico este trabalho a todos os alunos envolvidos com o Grupo de Estudos Avançados em Robótica – GEAR, com os quais aprendi muito, e àqueles que possuem grande paixão pela eletrônica, em especial pelo ramo da robótica”.

AGRADECIMENTOS

Agradeço primeiramente a Deus, ao meu orientador Evandro L. L. Rodrigues pelo apoio e conselhos, a todos os demais professores, técnicos e funcionários do Departamento de Engenharia Elétrica que, direta ou indiretamente, me ajudaram no desenvolvimento desse projeto, ainda aos amigos, sempre presentes durante minha graduação.

Não menos importante, também agradeço a toda minha família que sempre incentivou meus estudos, além de minha namorada pela compreensão ao empenho que dei a tal projeto e monografia.

"A mente que se abre a uma nova ideia jamais volta ao seu tamanho original."

Albert Einstein

RESUMO

O futebol jogado por robôs passa a ser interessante devido ao fato dessas máquinas não serem controladas por humanos, mas sim por uma inteligência artificial presente em um computador. O fato desse desafio permitir que seus integrantes apliquem os conhecimentos obtidos em sala de aula também colabora no âmbito de incentivar os estudantes a desenvolverem esse complexo sistema.

A monografia trata exclusivamente do projeto da placa principal do robô, responsável por executar a comunicação, o processamento de dados e a movimentação da plataforma robótica. Dessa maneira, temas como mecânica do robô, inteligência artificial e visão computacional estão fora do escopo desse estudo, apesar de serem vitais para o futebol de robôs.

Os principais softwares utilizados para projetar e construir a placa eletrônica foram o MPLAB IDE, para realizar a programação e emulação do código, Eagle 5.4.0, com a finalidade de implementar as interligações entre os dispositivos e seus posicionamentos na placa, e o CircuitCAM 5.2, apenas para converter os arquivos gerados pelo software anterior para um formato compreendido pela fresa.

Ao final da monografia espera-se ter em mãos o projeto de uma placa robusta, capaz de atender a todos os requisitos exigidos pela sua aplicação, para tanto, todos os componentes foram estudados e os módulos testados em *protoboard*, para então consolidar o projeto.

Mesmo com todo o cuidado e estudo para concretização do projeto, melhorias são propostas nos tópicos finais, possíveis por meio de novas abordagens de certos temas, utilização de tecnologias diferentes, devido à maior disponibilidade de recursos monetários ou até mesmo ao maior tempo de dedicação à execução.

Palavras-chave: futebol de robôs, placa principal, robô, projeto, Eagle 5.4.0.

ABSTRACT

The football played by robots becomes interesting due to the fact that these machines are not controlled by humans, but by an artificial intelligence present in a computer. The fact that this challenge enables their members to apply the knowledge obtained from in-classroom also helps in encouraging students to develop this complex system.

The monograph deals exclusively with the design of the main board of the robot, responsible for executing the communication, data processing and movement of the robotic platform. This way, subjects such as mechanical robot, artificial intelligence and computer vision are outside of the scope of this study, despite being vital for robot soccer.

The main softwares used to design and build the electronic board were the MPLAB IDE to perform programming and emulation code, Eagle 5.4.0, in order to implement the interconnections between devices and and their positions on the board, and CircuitCAM 5.2, just to convert the files generated by the previous software into a format understood by the cutter.

At the end of the monograph is expected to have in hand the design of a robust plate, which can meet all requirements for their implementation, thus, all components have been studied and tested in protoboard modules, and then build the project.

Even with all the care and study to completion of the project, improvements are proposed in the final points, possible through new approaches to certain issues, by using different technologies, by the increase of the availability of monetary resources or even the greatest time of dedication to the implementation.

Keywords: robot soccer, main board, robot, project, Eagle 5.4.0.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GEAR	Grupo de Estudos Avançados em Robótica
FIFO	*First In First Out*
RF	Rádio Frequência
PC	*Personal Computer*
MCU	*Micro Control Unit*
PWM	*Pulse Width Modulation*
MLP	Modulação por Largura de Pulso
LED	*Light Emissor Diode*
CRC	*Cyclic Redundancy Check*
CI	Circuito Integrado
ICSP	*In Circuit Serial Programming*
DC	*Direct Current*
BJT	*Bipolar Junction Transistor*
P	Proporcional
PI	Proporcional Integrativo
PID	Proporcional Integrativo Derivativo
DIP	*Dual In-line Package*

SUMÁRIO

INTRODUÇÃO

Esta monografia pretende apresentar o projeto completo de uma placa eletrônica cuja finalidade é realizar a movimentação de uma plataforma robótica, desde a análise das características dos componentes presentes na placa, passando pelo projeto da mesma até chegar em sua programação.

A motivação de tal trabalho surgiu no grupo de robótica do Departamento de Engenharia Elétrica, GEAR, tendo como objetivo desenvolver, aperfeiçoar e aplicar o conhecimento aprendido em sala de aula por meio da construção de um sistema autônomo de futebol de robôs.

O sistema é composto basicamente de quatro áreas, *hardware*, mecânica, visão computacional e inteligência artificial. Esse trabalho se restringe a mostrar o desenvolvimento da placa eletrônica, elemento principal e vital da área de *hardware*.

Dessa maneira, o objetivo da monografia é concretizar o projeto de tal placa eletrônica, de forma que a mesma possua os requisitos exigidos pelo robô para a função ao qual será utilizado, além de possibilitar que os leitores sejam capazes de entender seu funcionamento.

Todos os componentes da placa eletrônica foram analisados minuciosamente desde suas características elétricas, até a disponibilidade do mesmo no mercado. Os módulos da placa foram todos projetados individualmente e testados em *protoboard* antes do projeto final ser concretizado, colaborando dessa maneira para a correção de erros e integração desses módulos.

A programação foi modularizada o máximo possível, gerando dessa maneira códigos mais extensos, porém mais inteligíveis. Bibliotecas e estruturas foram desenvolvidas, facilitando dessa maneira a correção de erros e até a simulação do sistema.

As folhas de dados dos componentes além de manuais de referência do microcontrolador foram essenciais para o desenvolvimento deste trabalho, já que neles é possível encontrar desde informações a respeito do dispositivo, até circuitos e códigos de exemplo, tornando as explicações muito mais elucidativas.

1 OBJETIVOS

O presente trabalho foca a construção de uma placa eletrônica capaz de suprir as necessidades de um robô, com excessão de seu chute, para competições de futebol de robôs na RoboCup categoria Small Size, uma das mais tradicionais no âmbito internacional, porém ainda pouco desenvolvida no Brasil. [1]

O robô deve caber em um cilindro de 180 mm de diâmetro, não podendo ultrapassar 150 mm de altura, conforme exibido na Figura 1. Dessa maneira, as dimensões da placa eletrônica ficam limitadas, tanto em raio quanto em altura. [2]

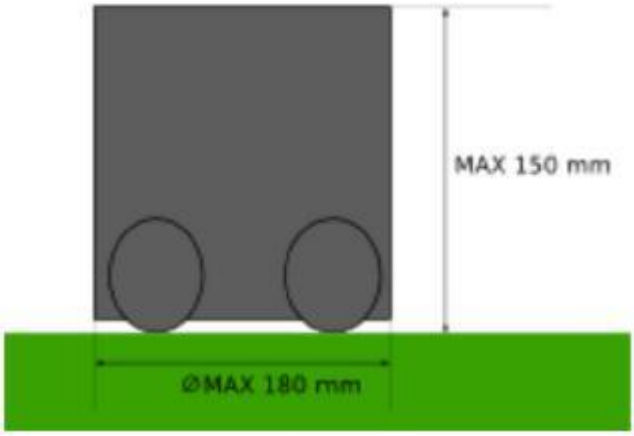


Figura 1: Dimensões Máximas do Jogador [2]

O jogador também deve ser capaz de conduzir uma bola de golfe laranja e executar jogadas com ela, como chutes, passes e dribles, conforme exibido na Figura 2. Dessa maneira, uma segunda placa deve ser projetada, denominada "Placa de Chute", que está fora do escopo deste trabalho.

O mecanismo responsável pela condução da bola é chamado "Dribbler" e tem como função aplicar um *spin* negativo à esfera, fazendo com que ela fique de posse do jogador enquanto esse se locomove pelo campo. [2]

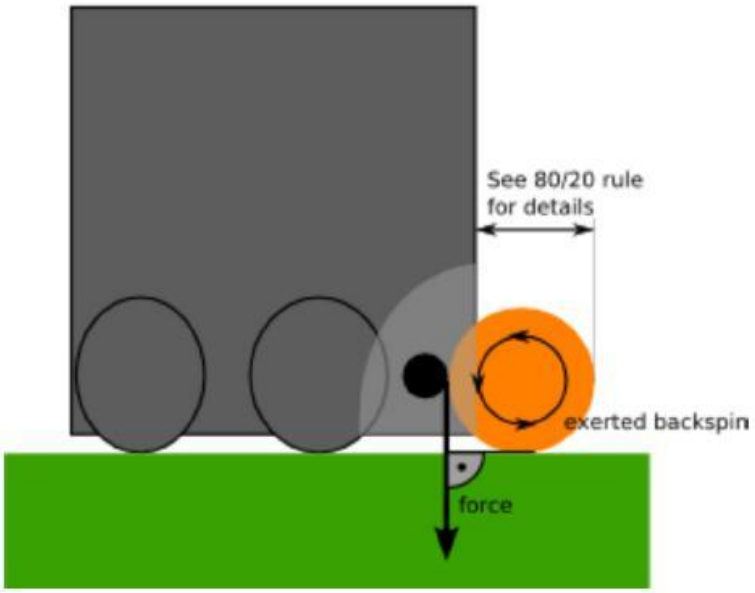


Figura 2: Condução de Bola [2]

Dessa forma o desafio está no projeto e construção de uma placa capaz de receber informações de um computador, processá-las, executar os movimentos solicitados, de maneira controlada, permitir a condução da bola e realizar a telemetria do sistema.

Todo o projeto das interligações entre componentes, denominado esquemático, foi feito com o software Eagle 5.4.0, o mesmo utilizado para posicionar os componentes na placa e traçar as trilhas que os conectam, chamado de *board*.

O programa responsável por converter o arquivo *board*, do Eagle, em uma extensão compreensível pela fresa foi o CircuitCAM 5.2, onde as camadas superior, inferior, de furos e de bordas são acopladas e exportadas em um arquivo com extensão ".LMD", como descrito no Anexo A.

O microcontrolador foi programado e a placa emulada por inteira, utilizando o MPLAB IDE da Microchip e o gravador/emulador PICKIT 3.

2 ESTADO DA ARTE

2.1 Hardware

Para que o robô possa executar as funções solicitadas, é vital que possua como um de seus principais módulos uma placa eletrônica capaz de adquirir as informações enviadas, via rádio frequência, por uma placa de transmissão conectada a um computador, processá-las e realizar o acionamento dos motores.

A telemetria é de grande importância à plataforma robótica, uma vez que medições do nível da bateria, orientação do robô, posse de bola e tensão presente nos capacitores de chute são importantes para que a inteligência artificial tome melhores decisões durante as partidas, justificando dessa maneira a transmissão de dados em ambos os sentidos.

Com base nestes argumentos, antes da execução do projeto, é importante realizar a pesquisa e o estudo de componentes capazes de suprir as necessidades do robô, possibilitando ao mesmo o desempenho exigido por tal aplicação, futebol de robôs.

O presente trabalho divide a placa eletrônica em três grandes módulos: Comunicação, Processamento de Dados e Acionamento dos Motores, que serão abordados nos capítulos 2.2 , 2.3 e 2.4 respectivamente, baseando o projeto do sistema eletrônico.

2.2 Comunicação

Como os robôs necessitam de grande liberdade com relação a fios, para poderem jogar futebol o tipo de comunicação utilizada entre eles e o computador fica limitada às formas de transmissão de dados que utilizam da tecnologia *wireless*, eliminando dessa maneira as cabeadas.

A comunicação *wireless* pode ser feita via rádio frequência, micro-ondas ou infravermelho. [3] No desafio em questão, é interessante a utilização da primeira tecnologia, visto que no mercado de

produtos eletrônicos brasileiro são encontrados dispositivos de pequeno porte capazes de realizar a transmissão de dados, a recepção deles ou até mesmo ambos em um único componente.

2.2.1 Transceptor

Transceiver, ou transceptor, é um dispositivo que possui tanto o sistema transmissor quanto o receptor, combinados no mesmo circuito ou no mesmo encapsulamento. [4]

O TRF-2.4G é composto de uma antena, um sintetizador de frequência integrado, um amplificador de potência, um cristal oscilador e um modulador. Consome baixa corrente e é programado utilizando uma interface de três linhas. [5] A Figura 3 apresenta o transceptor, juntamente com a disposição de seus 10 pinos e suas dimensões.

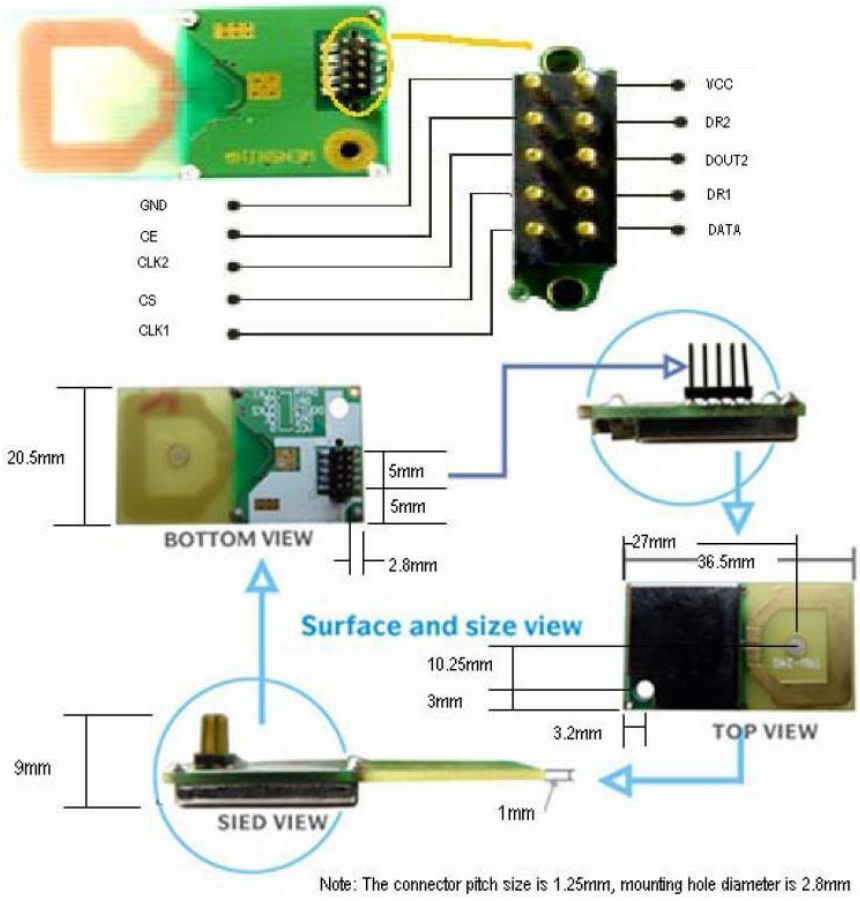


Figura 3: Transceptor TRF 2.4G [5]

A descrição resumida de cada um dos pinos do dispositivo está apresentada na Tabela 1, juntamente com sua nomenclatura.

Tabela 1: Funções dos Pinos do Transceptor

Pinos e suas Funções	
Pino	**Função**
GND	Terra
CE	Habilitação do Chip
CLK2	Clock para RX do canal 2
CS	Seleção do Chip – Ativa o Modo de Configuração
CLK1	Clock para TX e RX do canal 1
DATA	Canal de dados para RX e TX
DR1	Dados de RX prontos no canal 1
DOUT2	Dados de RX do canal 2
DR2	Dados de RX prontos no canal 2
VCC	Alimentação

Os valores mínimos, típicos e máximos dos parâmetros do dispositivo em questão estão apresentados na Tabela 2, possibilitando assim realizar análises a respeito da conexão deste com outras interfaces.

Tabela 2: Características Elétricas do Transceptor

Características Elétricas			
Parâmetro	**Mínimo**	**Típico**	**Máximo**
V_{CC}	1,9V	3,0V	3,6V
V_{IH}	V_{CC}-0,3	-	V_{CC}
V_{IL}	V_{SS}	-	0,3V
V_{OH}	V_{CC}-0,3	-	V_{CC}

V_{OL}	V_{SS}	-	0,3V
f_{OP}	2.400MHz	-	2.524MHz

2.2.2 Modo de Operação

O dispositivo TRF-2.4G pode assumir os modos de operação presentes na Tabela 3, dependendo do valor aplicado aos pinos CE e CS. [5]

Tabela 3: Modos de Operação [5]

Modos de Operação do Dispositivo TRF-2.4G		
Modo	**CE**	**CS**
Ativo – RX/TX	1	0
Configuração	0	1
Espera	0	0

Quando ativo o transceptor pode operar em uma das duas configurações existentes, Modo Direto ou *ShockBurst*™. [5]

A tecnologia *ShockBurst* utiliza o sistema FIFO on-chip, onde é gerada uma pilha sendo que as instruções são processadas na ordem de chegada, para possibilitar a entrada de dados no *transceiver*, em baixa taxa de transferência e transmití-los, no ar, a uma altíssima taxa, podendo atingir 1Mbps, oferecido pela banda de 2.4GHz, sem a necessidade de um microcontrolador de alta velocidade para tanto, além de reduzir consideravelmente o consumo de potência. [5]

A economia, tanto de processamento como de energia, ocorre porque o microcontrolador após configurar o CI para operar no modo *ShockBurst*™ necessita enviar ao transceptor apenas o endereço e os dados que deseja transmitir, de forma que o TRF-2.4G se encarrega de gerar o CRC, realizar a amostragem dos bits e codificar ou decodificar os pacotes, dependendo se será realizada uma transmissão ou recepção. Em seguida o transceptor pode transmitir ou receber as informações a uma velocidade de até 1Mbps e dormir. [6]

A comunicação entre o microcontrolador e o TRF-2.4G é feita por meio de uma interface de 3 linhas, CS, CLK1 e DATA, de forma que a palavra de configuração, quando em modo *ShockBurst*, habilita o transceptor a armazenar o protocolo RF. [5]

Para a utilização do modo *ShockBurst*^{TM} é necessário configurar:

- Tamanho do bloco de dados
- Tamanho do endereço
- Endereço de destino para dados recebidos
- CRC

Com tais dados armazenados nos registradores de configuração, quando for realizada uma transmissão, o pacote gerado será do formato apresentado na Figura 4.

Figura 4: Estrutura do Pacote de Configuração [5]

A transmissão de dados é feita da seguinte maneira: [5]

- Quando o MCU necessita enviar dados, o pino CE deve ser posto em nível lógico alto;
- O endereço do nó de recebimento e os dados devem ser pulsados no TRF-2.4G;
- MCU coloca CE em nível lógico baixo, ativando a transmissão *ShockBurst*;
- *Preamble* é adicionado e CRC é calculado, para completar o pacote;
- Dados são transmitidos em alta velocidade, 250kbps ou 1Mbps;
- O dispositivo TRF-2.4G retorna ao estado de espera quando a transmissão é concluída.

Como pode ser analisado na Figura 5 que contempla o fluxograma de tal operação.

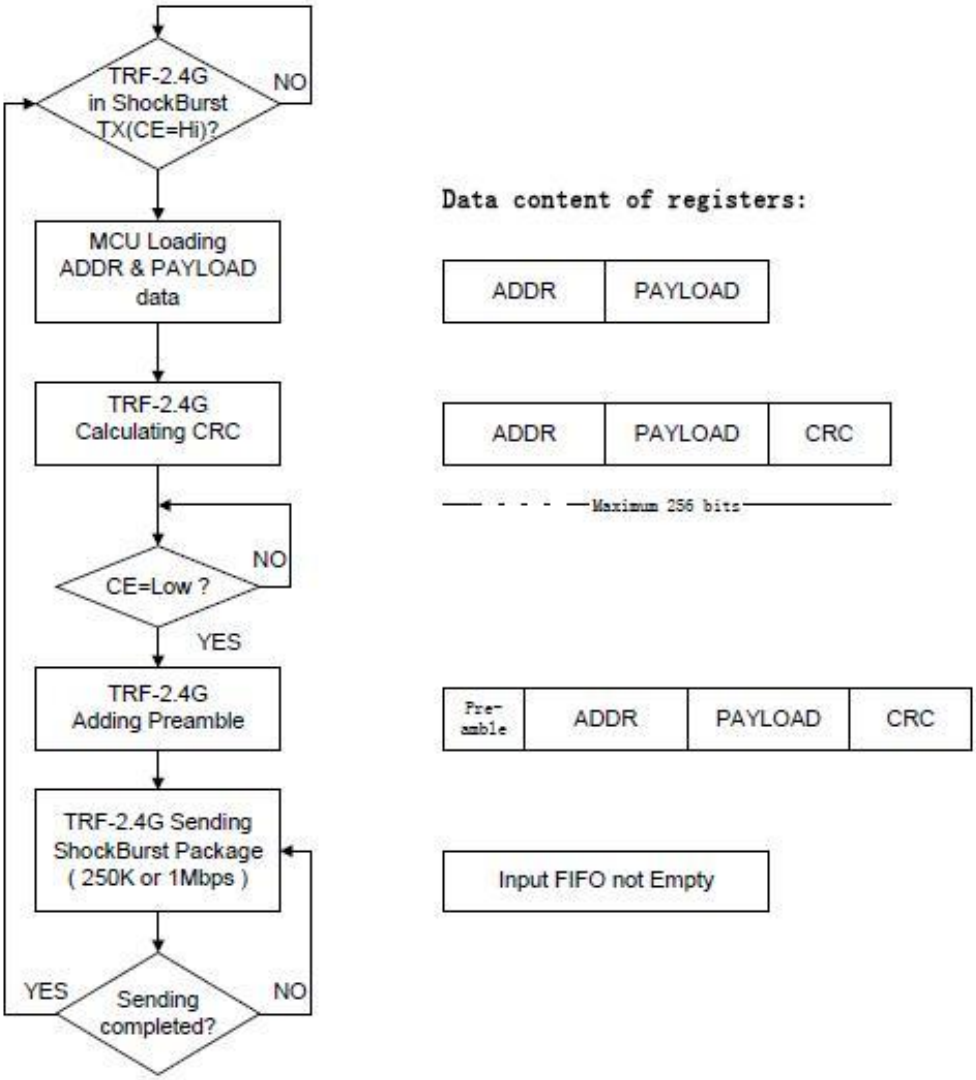


Figura 5: Transmissão de Dados Utilizando *ShockBurst* [5]

A recepção de dados é realizada da seguinte maneira: [5]

O endereço e o tamanho do *payload* dos pacotes RF de entrada são definidos assim que o TRF-2.4G é configurado no modo *ShockBurst*;

- Para habilitar a recepção, o pino CE deve ser colocado em estado lógico 1;
- O dispositivo irá começar a monitorar o ar buscando informações de entrada;
- Quando um pacote válido é encontrado, ou seja, endereço correto e encontra o CRC, o dispositivo remove os bits de *preamble*, endereço e CRC, restando apenas a carga útil;
- O pino DR1 então é colocado em nível lógico alto, notificando o MCU por meio de uma interrupção externa;
- O MCU deve então colocar CE em nível lógico baixo;

- O MCU então retira apenas o *payload* a uma taxa aceitável para o mesmo;
- Quando todos os dados foram coletados DR1 é colocado em nível lógico baixo novamente e o dispositivo está pronto para uma nova recepção caso o CE mantenha-se alto durante o *download* dos dados.

Como pode ser analisado na Figura 6 que contempla o fluxograma de tal operação.

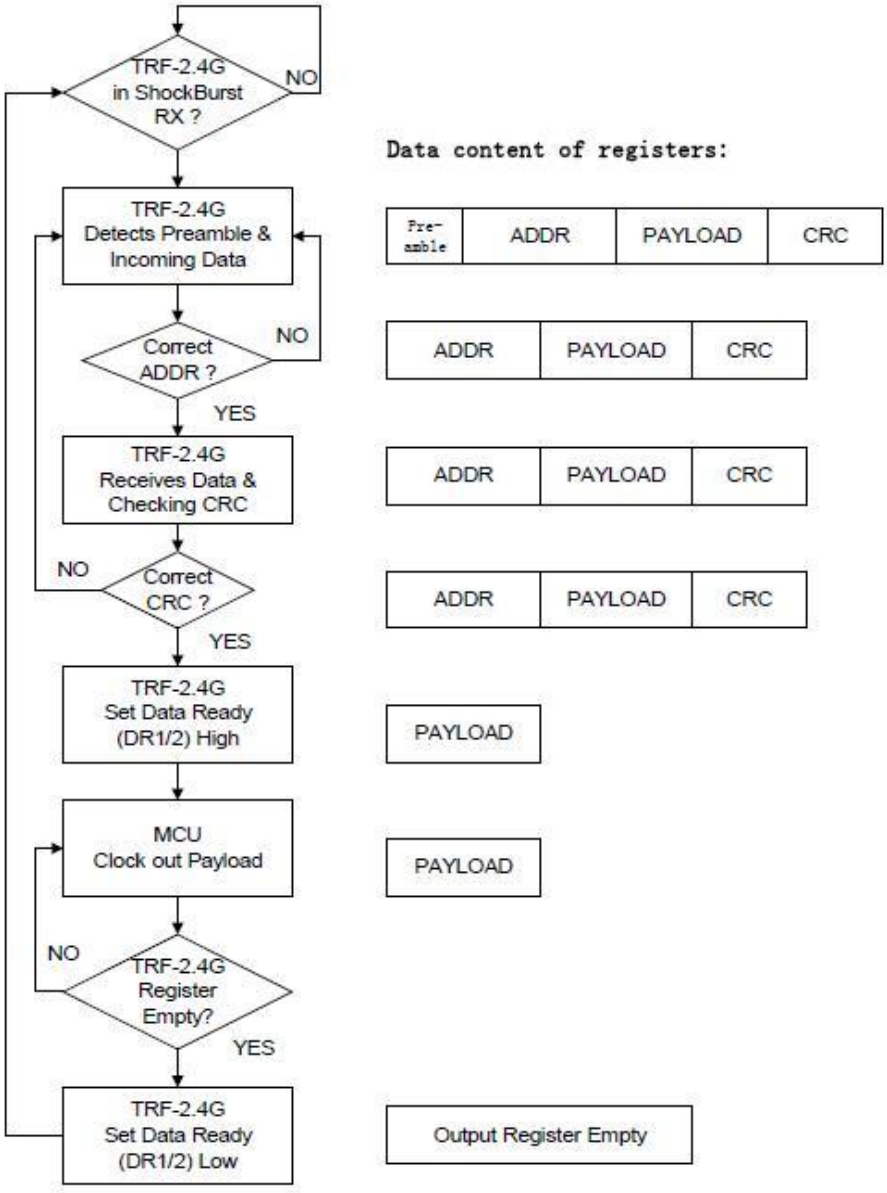


Figura 6: Recepção de Dados Utilizando *ShockBurst* [5]

2.3 Processamento de Dados

Um microcontrolador é um sistema computacional completo, no qual estão incluídos uma CPU (Central Processor Unit), memória de dados e programa, um sistema de *clock*, portas de I/O (*In-*

put/Output), além de outros possíveis periféricos, tais como, módulos de temporização e conversores A/D entre outros, integrados em um mesmo componente. As partes integrantes de qualquer computador, e que também estão presentes, em menor escala, nos microcontroladores são:

- Unidade Central de Processamento (CPU)
- Sistema de *clock* para dar sequência às atividades da CPU
- Memória para armazenamento de instruções e para manipulação de dados
- Entradas para interiorizar na CPU informações do mundo externo
- Saídas para exteriorizar informações processadas pela CPU para o mundo externo
- Programa (*firmware*) para definir um objetivo ao sistema

(DENARDIN, 2011)

O MCU é interessante no âmbito de centralizar as informações, atribuir tarefas aos periféricos e gerenciar suas ações, fazendo com que a plataforma robótica torne-se confiável e controlada, características importantes em um sistema desse porte.

2.3.1 Características Gerais

O MCU usado é o dsPIC33FJ256MC710A, que possui 100 pinos, podendo até 85 deles serem utilizados como E/S, 8 saídas de PWM, com 4 geradores, 9 *Timers* de 16 bits e 32 canais de conversão A/D gerenciados por 2 módulos, além de possuir taxas altas de processamento de dados podendo atingir 40MIPS.

Os pinos, suas nomenclaturas e disposição encontram-se na Figura 7.

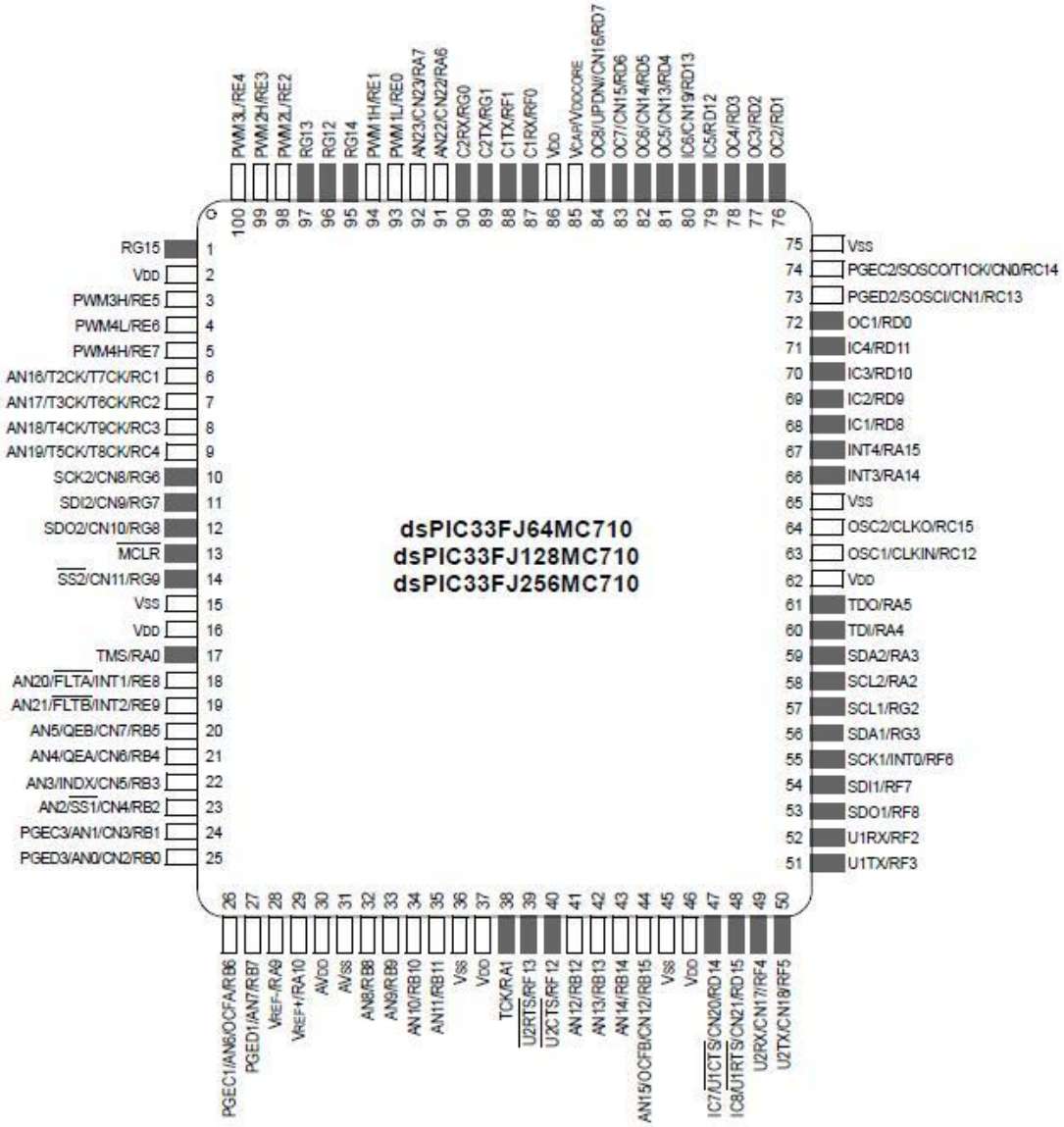


Figura 7: Pinos dsPIC33FJ [8]

Os pinos preenchidos de preto na Figura 7 são capazes de receber tensões de até 5V e fornecer tensões típicas de 3,3V, ao passo que os preenchidos de branco, tanto fornecem quanto recebem tipicamente 3,3V.

As principais características elétricas do dispositivo, juntamente com seus valores mínimos, típicos e máximos, encontram-se na Tabela 4.

Tabela 4: Características Elétricas dsPIC33FJ

Características Elétricas			
Parâmetro	**Mínimo**	**Típico**	**Máximo**
Tensão VDD	-0,3V	3,3V	4,0V
Corrente de Saída do VSS	-	-	300mA
Corrente de Entrada no VDD	-	-	250mA
Corrente fornecida por qualquer pino I/O	-	-	4mA
Corrente drenada por qualquer pino I/O	-	-	4mA
Corrente fornecida por todas as portas	-	-	200mA
Corrente drenada por todas as portas	-	-	200mA

2.3.2 Modo de Operação

O dispositivo dsPIC33FJ256MC710A trata-se de um microcontrolador, dessa forma, possui diversos módulos integrados em seu encapsulamento, bastando configurá-los de maneira correta para obter o funcionamento esperado desses.

Nos tópicos seguintes maiores explicações serão dadas a respeito do funcionamento de cada periférico relevante para a realização do projeto da plataforma robótica.

2.3.2.1 In-Circuit Serial Programming™

Os pinos PGECx e PGEDx são utilizados para realizar a *In-Circuit Serial Programming™*, ICSP™, além do *In-Circuit Debugger*, que será abordado na seção 2.3.2.2 . Para que o procedimento ICSP™ seja realizado com sucesso, também é necessário realizar as conexões dos pinos V_{SS}, V_{DD} e MCLR, sendo a linha de sequência de programação opcional. [8]

Para permtir ao projetista maior flexibilidade durante o projeto do *hardware*, o CI viabiliza a possibilidade da utilização de um dos três pares de linhas para *clock* e dados, denominados: [8]

- PGEC1 e PGED1, pinos 26 e 27, respectivamente;

- PGEC2 e PGED2, pinos, 74 e 73, respectivamente;
- PGEC3 e PGED3, pinos 24 e 25, respectivamente.

2.3.2.2 In-Circuit Debugger

A tecnologia *In-Circuit Debugger* permite que o programador realize a emulação do MCU, sendo que as entradas são processadas fisicamente para então fornecer as saídas, diferentemente das simulações, onde um programa analisa as entradas para então fornecer as saídas. Dessa maneira, caso alguma particularidade de *hardware* não seja programada no simulador, este não irá retornar saídas tão confiáveis quanto as obtidas pela emulação.

Para utilizar essa tecnologia é necessário realizar as conexões dos pinos PGECx, PGEDx, V_{SS}, V_{DD} e MCLR, assim como no processo de ICSP™. Também deve ser informado que certos gravadores são capazes de executar apenas a programação do MCU, portanto, para que seja possível o *In-Circuit Debbuger* é necessário adquirir um gravador/emulador, como o PICKIT3 ou o MPLAB® ICD 2. [8]

O *software* MPLAB IDE gerencia tanto o processo de programação do microcontrolador quanto o de emulação dele, sendo esse último, vital para a verificação do funcionamento das funções programadas no chip e correção de erros.

Assim como na tecnologia ICSP, a *In-Circuit Debugger* permite que o projetista escolha qual dos três pares, PGECx e PGEDx, será utilizado. [8]

2.3.2.3 Configuration Bits

Os bits de configuração podem ser programados ou deixados sem programação (lidos como nível alto), gerando dessa forma, diversas configurações para o dispositivo. Tais bits são mapeados começando da posição 0xF80000 da memória de programa. [8]

Os registradores presentes nessa estrutura definem condições básicas de como o MCU irá operar durante a execução do código presente neste. O mapeamento de tais registradores está apresentado na Figura 8.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0xF80000	FBS	RBS<1:0>		—	—	BSS<2:0>			BWRP
0xF80002	FSS	RSS<1:0>		—	—	SSS<2:0>			SWRP
0xF80004	FGS	—	—	—	—	—	GSS1	GSS0	GWRP
0xF80006	FOSCSEL	IESO	Reserved(2)	—	—	—	FNOSC<2:0>		
0xF80008	FOSC	FCKSM<1:0>		—	—	—	OSCIOFNC	POSCMD<1:0>	
0xF8000A	FWDT	FWDTEN	WINDIS	—	WDTPRE	WDTPOST<3:0>			
0xF8000C	FPOR	PWMPIN	HPOL	LPOL	—	—	FPWRT<2:0>		
0xF8000E	FICD	Reserved(1)		JTAGEN	—	—	—	ICS<1:0>	
0xF80010	FUID0	User Unit ID Byte 0							
0xF80012	FUID1	User Unit ID Byte 1							
0xF80014	FUID2	User Unit ID Byte 2							
0xF80016	FUID3	User Unit ID Byte 3							

Figura 8: Mapeamento dos Registradores de Configuração [8]

Como pode ser observado na Figura 8, o microcontrolador possui tal estrutura distribuida em 12 registradores de configuração, sendo eles FBS, FSS, FGS, FOSCSEL, FOSC, FWDT, FPOR, FICD, FUID0, FUID1, FUID2 e FUID3. Cada um desses divididos em estruturas menores, ao nível de bits, possibilitando diversas configurações distintas do dispositivo.

Para prevenir mudanças indevidas nos bits de configuração durante a execução do programa, tais bits são escritos uma única vez a cada energização do dispositivo, ou seja, para alterar o valor de tais bits é necessário desenergizar o dispositivo e energizá-lo novamente. [9]

Tabela 5: Registradores de Configuração Geral

Registradores de Configuração Geral do Microcontrolador	
Registrador	**Descrição**
FBS	Proteção de escrita no segmento de *boot*
FSS	Proteção de escrita no segmento de segurança
FGS	Proteção de escrita no segmento geral
FOSCSEL	Seleção do oscilador
FOSC	Configuração do oscilador
FWDT	Configuração do *Watchdog Timer*
FPOR	Configurações iniciais do PWM
FICD	Configuração de emulação

2.3.2.4 Oscilador

O dispositivo oferece várias fontes de *clock*, tanto internas ao chip quanto externas: [8]

- Oscilador FRC
- Oscilador FRC com PLL
- Oscilador Primário (XT, HS ou EC)
- Oscilador Primário com PLL
- Oscilador Secundário (LP)
- Oscilador LPRC
- Oscilador FRC com *Postscaler*

O oscilador FRC com PLL foi o escolhido para o projeto, pois, não necessita de circuitos externos, além de permitir que o MCU opere com 40MIPS. O oscilador interno, FRC, possui frequência nominal igual a 7,37MHz. Tal frequência, denominada F_{OSC} é divida por dois para gerar o *clock* de instrução do dispositivo, denominado F_{CY}. [10]

$$F_{CY} = \frac{F_{osc}}{2}$$

O dsPIC33FJ256MC710A é capaz de, utilizando seu oscilador FRC juntamente com a tecnologia de PLL, atingir sua F_{CY} máxima de 40MIPS, característica interessante, pois simplifica o *hardware*, já que nesse caso os cristais somente serão necessários quando a aplicação exigir grande precisão de temporização.

A Figura 9 exibe o diagrama de blocos do funcionamento do PLL, juntamente com as frequências mínimas e máximas aceitas em cada ponto do sistema, tais valores devem ser respeitados para a correta temporização do MCU.

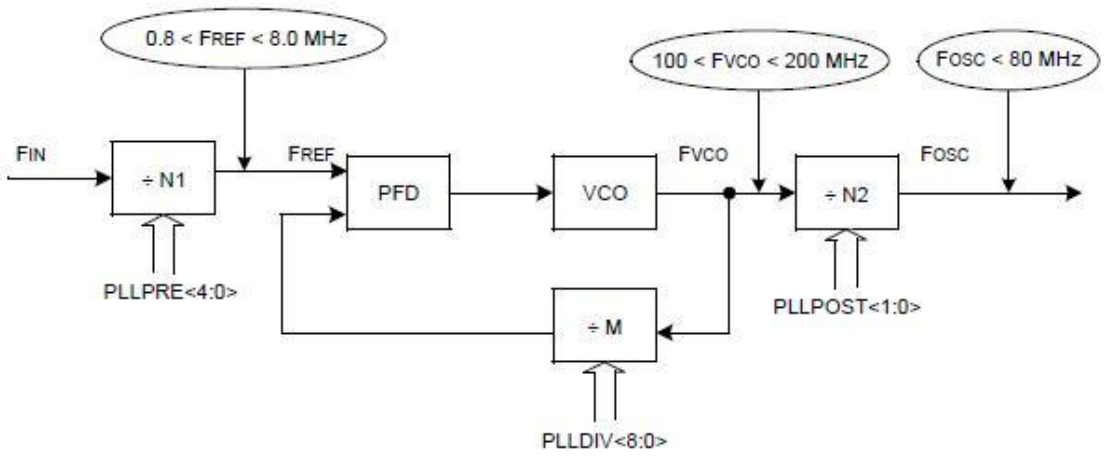


Figura 9: Diagrama de Blocos do PLL [10]

A dedução da fórmula que relaciona FIN com FOSC está apresentada na Figura 10, na qual PLL-PRE, PLLPOST e PLLDIV são registradores.

$$FOSC = FIN \times \left(\frac{M}{N1 \times N2}\right) = FIN \times \left(\frac{(PLLDIV + 2)}{(PLLPRE + 2) \times 2(PLLPOST + 1)}\right)$$

Where:

$N1$ = PLLPRE + 2

$N2$ = 2 x (PLLPOST + 1)

M = PLLDIV + 2

Figura 10: Cálculo de F_{OSC} [10]

2.3.2.5 Portas I/O

Os pinos do dispositivo, com exceção do V_{DD}, V_{SS}, MCLR, e OSC1/CLKIN, são compartilhados entre periféricos e portas paralelas de I/O. Tais portas são todas providas de uma configuração *Schimitt Trigger* garantindo, dessa maneira, melhorias na imunidade ao ruído. [11]

Quando um periférico, associado a um pino, é habilitado, o uso para propósitos gerais de tal pino é desabilitado. Todos os pinos possuem três registradores diretamente relacionados às operações de entrada e de saída digital. [8]

O registrado TRISx determina a direção do fluxo de dados, de tal forma que quando ao bit é atribuído o valor "1", o pino controlado pelo mesmo passa a comportar-se como um pino de entrada, ao passo que caso o valor seja "0", o pino será reconhecido como saída de dados. Após a ação do *Reset*, todos os pinos são definidos como entradas digitais. [8]

Os registradores LATx e PORTx são utilizados para a manipulação dos bits, de tal forma que, ler pelo LATx realiza uma leitura no *latch* e realizar a escrita no registrador LATx, escreve no *latch*, ao

passo que ler do registrador PORTx lê o bit presente na porta e escrever no PORTx escreve no *latch*. [8]

Na Figura 11 está exibida uma estrutura exclusivamente de entrada e saída, sem considerar o periférico, de forma que em tal imagem fica clara as diferenças acima explicadas a respeito da leitura nos registradores LATx e PORTx. [11]

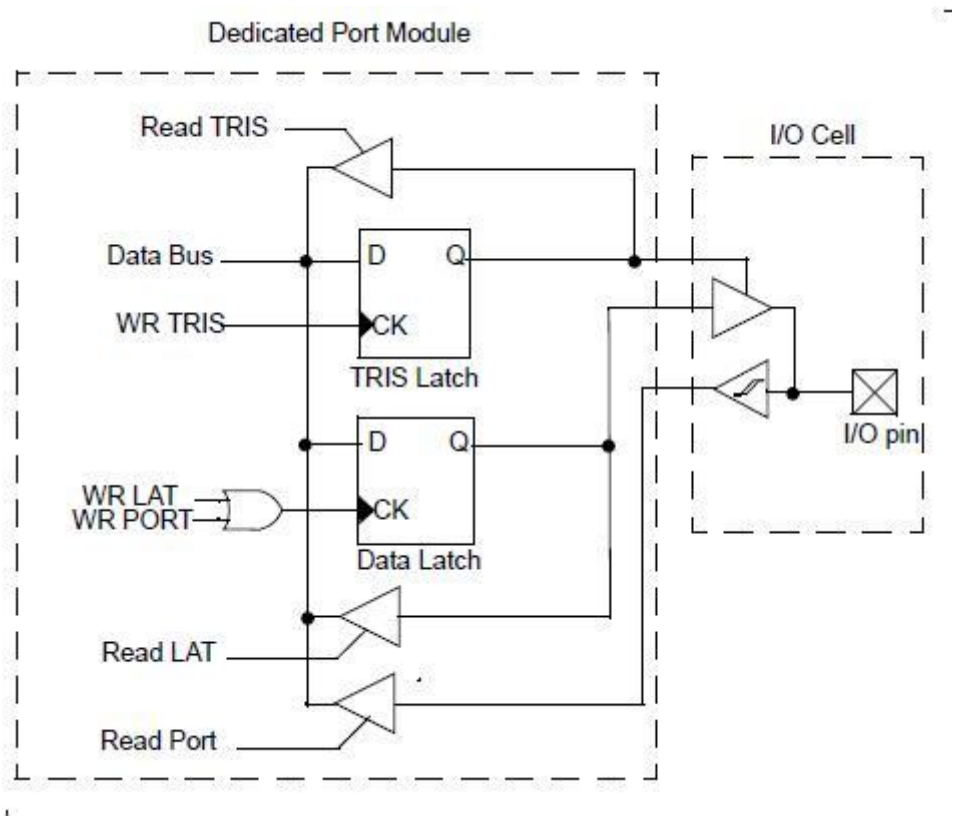


Figura 11: Estrutura exemplo de um pino I/O [11]

2.3.2.6 Timers

O microcontrolador possui nove *timers*, sendo que todos podem ser configurados para funcionar como temporizadores ou contadores. O *timer* 1, tipo A, é independente dos demais, já os *Timers* 2, 4, 6 e 8, tipo B, podem ser associados aos *Timers* 3, 5, 7 e 9, tipo C, respectivamente, para formar temporizadores de 32 bits, já que individualmente, os mesmos possuem 16 bits para realizar a temporização. [12]

Quando o interesse é a configuração de contadores, o dispositivo pode prover até cinco dos mesmos, já que os *Timers* 2 e 7 compartilham do mesmo pino, assim como os *Timers* 3 e 6, 4 e 9 e 5 e 8, como pode ser observado na Figura 7. [12]

Tal periférico é muito importante, já que permite a contagem de tempo ou até mesmo de pulsos externos, sem interferir na execução do programa principal, garantindo dessa maneira exatidão na temporização e contagem.

Os *timers* 6, 7, 8 e 9 foram configurados como contadores externos, ao passo que os *timers* 3 e 5 como temporizadores. Tais configurações são implementadas com base nos registradores apresentados na Tabela 6.

Tabela 6: Configuração dos *Timers*

Registradores de Configuração dos *Timers*	
Registrador	**Descrição**
TxCON (para x=1, tipo A)	Registrador de configuração do *Timer* tipo A
TxCON (para x=2, 4, 6, 8)	Registradores de configuração dos *Timers* tipo B
TxCON (para x=3, 5, 7, 9)	Registradores de configuração dos *Timers* tipo C

2.3.2.7 PWM

A modulação por largura de pulsos pode ser utilizada em aplicações de telecomunicações, regulação de tensão, efeitos de áudio, ou, como nesse caso, na transferência de potência. Tal técnica é muito vantajosa, pois a transferência de potência à carga acontece sem as consideráveis perdas ocorridas devido à queda de tensão por recursos resistivos. [13]

O chaveamento é feito por transistores que ora estão funcionando como chaves fechadas, permitindo a passagem total de corrente e ora bloqueiam a passagem da mesma, funcionando como chaves abertas, porém como tal chaveamento é feito em alta frequência, a estimulação sentida pelo motor DC, carga, é a média entre o tempo em que o transistor conduziu e bloqueou a passagem de corrente. Tal relação é conhecida por *duty cycle* do inglês, ou, razão cíclica, do português, e é dada pela seguinte equação 1: [13]

$$Duty\ Cycle = \frac{T_{ON}}{T_{TOTAL}} \quad (1)$$

O módulo PWM presente no MCU simplifica a tarefa de gerar múltiplas saídas sincronizadas, moduladas por largura de pulsos, já que tal periférico possui em *hardware* quatro geradores de *duty*

cicle independentes, numerados de 1 à 4, além de oito saídas, comandadas por seus respectivos geradores de razão cíclica, porém subdivididas em dois grupos, *High* (H) e *Low* (L). [14]

O periférico em questão pode funcionar de diversas maneiras e em várias aplicações, para tanto o mesmo é configurado por meio da programação de seus registradores. Esses, juntamente com suas funções estão apresentados, de maneira resumida, na Tabela 7.

Tabela 7: Registradores do PWM

Registradores de Configuração do módulo PWM	
Registrador	**Descrição**
PxTCON	Registrador de controle da base de tempo do PWM
PxTMR	Registrador do valor de contagem de tempo do PWM
PxTPER	Registrador do período da base de tempo do PWM
PxSECMP	Registrador de comparação de evento especial
PWMxCON1	Registrador de controle do PWM 1
PWMxCON2	Registrador de controle do PWM 2
PxDTCON1	Registrador de controle do tempo de término 1
PxDTCON2	Registrador de controle do tempo de término 2
PxFLTACON	Configurações relacionadas a falhas ocorridas nos PWMs.
PxFLTBCON	Configurações relacionadas a falhas ocorridas nos PWMs.
PxOVDCON	Utilizado para controle da sobreposição das saídas dos PWMs.
PxDC1	O valor da razão cíclica das duas saídas do PWM 1 é escrito nesse registrador.
PxDC2	O valor da razão cíclica das duas saídas do PWM 2 é escrito nesse registrador.
PxDC3	O valor da razão cíclica das duas saídas do PWM 3 é escrito nesse registrador.
PxDC4	O valor da razão cíclica das duas saídas do PWM 4 é escrito nesse registrador.

2.4 Acionamento dos Motores

O motor escolhido foi o “Série 2342 012 CR” da fabricante Faulhaber, por possuir tensão nominal igual à 12V, a qual é compatível com a bateria, potência de saída de 17W e constante de velocidade de 713rpm/V, além de suportar correntes de até 1,4A. [15] Os componentes do circuito de acionamento são listados nos subitens seguintes.

2.4.1 Ponte H

Tal dispositivo eletrônico tem como função principal possibilitar o acionamento de motores em ambos os sentidos, fornecendo a potência exigida por eles, já que tais mecanismos necessitam de amplitudes de correntes e de tensões as quais um MCU convencional não consegue suprir. O CI L298 possui 15 pinos organizados, como exibido na Figura 12, e um encapsulamento denominado *Multiwatt15*.

Figura 12: Circuito Integrado L298 [16]

2.4.1.1 Características Gerais

O dispositivo L298 é um circuito monolítico integrado com encapsulamento *Multiwatt* de 15 pinos, composto por duas pontes H completas, permitindo dessa forma o acionamento de até dois motores DC para ambos os lados, independentemente. [16]

Na Tabela 8 estão exibidas as características elétricas relevantes do CI.

Tabela 8: Características Elétricas do CI L298 [16]

Características Elétricas			
Parâmetro	**Mínimo**	**Típico**	**Máximo**
Tensão de Alimentação (VS)	2,5V	-	46V

Tensão de Alimentação Lógica (V_{SS})	4,5V	5V	7V
Estado Lógico Baixo de Entrada (V_{iL})	-0,3V	-	1,5V
Estado Lógico Alta de Entrada (V_{iH})	2,3V	-	V_{SS}
Estado Lógico Baixo de Habilitação (V_{enL})	-0,3V	-	1,5V
Estado Lógico Alto de Habilitação (V_{enH})	2,3V	-	V_{SS}
Potência Máxima Dissipada	-	25W	-
Frequência de Comutação	-	25kHz	40kHz

Para melhor análise do dispositivo, a Figura 13 mostra seu diagrama de blocos, onde podem ser feitas observações a respeito da habilitação da base dos transistores BJTs por meio das portas lógicas conectadas às mesmas, além do sentido do fluxo de corrente pelos ramos das pontes-H.

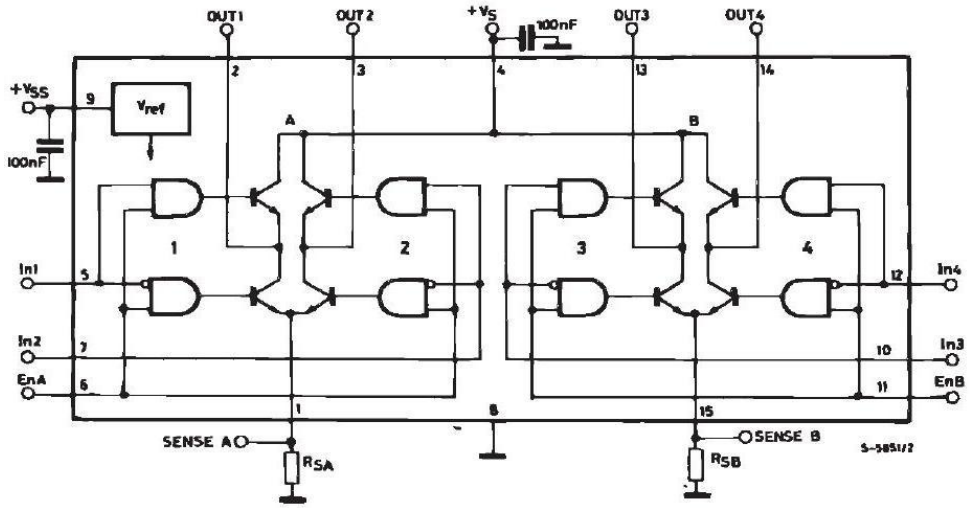


Figura 13 - Diagrama de Blocos do CI L298 [16]

Para filtrar os ruídos fazem-se necessários dois capacitores de 100nF, um conectado ao V_{SS} e outro conectado ao V_S, ambos realizando a ligação entre tais pinos e o GND. Também são indispensáveis dois resistores de potência R_{SA} e R_{SB}, para que possa ser medida a corrente que flui pelos motores conectados ao ramo A e ao ramo B. [16]

Para evitar danos aos transistores BJTs presentes no CI, diodos de rápida recuperação devem ser conectados como mostrado na Figura 14. Esses diodos têm a função de descarregar a corrente residual na bateria que alimenta o circuito, tal corrente surge pelo fato do motor possuir impedâncias relevantes. [16]

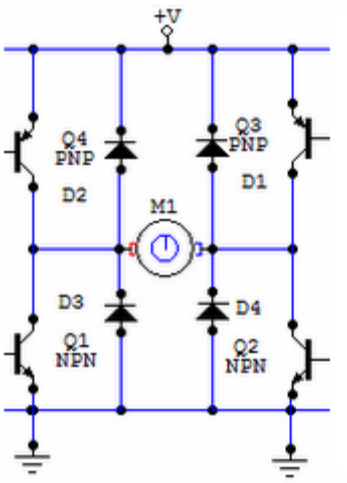


Figura 14: Diodos de Proteção

O manual do CI recomenda a utilização de diodos com tempo de recuperação inferior à 200ns. [16]

2.4.2 Encoders

Acoplado à parte traseira de cada motor da Faulhaber está o dispositivo denominado *encoder* óptico. Tal dispositivo é provido de dois canais, permitindo dessa forma analisar qual o sentido de rotação do motor, 512 linhas por volta, tensão de alimentação podendo variar de 4V à 18V e corrente máxima admitida igual à 15mA. [17]

Para melhorar o tempo de subida do pulso enviado pelo encoder, um resistor externo de *pull-up* pode ser adicionado ao circuito, como exibido na Figura 15, porém deve-se atentar à limitação de corrente de 15mA. [17]

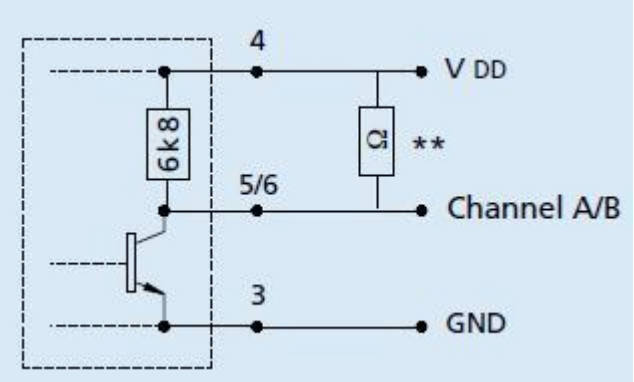


Figura 15: Resistor de *Pull-Up* [17]

O método para determinar o sentido de rotação do motor apesar de simples, não será explicado nesse trabalho, já que o controle aplicado a tais motores é apenas de velocidade, não interessando dessa maneira se o mesmo está funcionando no sentido horário ou anti-horário.

Na Figura 16 está exibida a disposição dos pinos do motor, porém como nesse caso o motor suporta correntes superiores a 1A, atingindo até 1,4A, os pinos 1 e 2 estão presentes em um conector separado. [17]

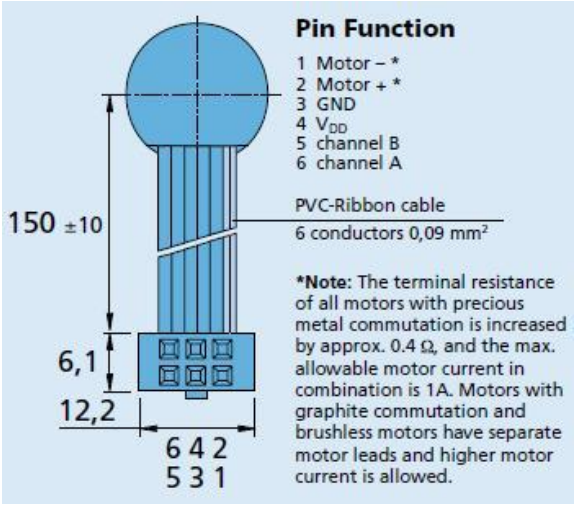


Figura 16: Disposição dos Pinos do Motor Faulhaber [17]

2.4.2.1 Modo de Funcionamento

De maneira simplificada, o *encoder* óptico é composto por um ou mais LEDs emissores de luz além de um ou mais fotorreceptores e um disco, acoplado ao eixo de rotação do motor, com frestas que permitam a passagem de luz. Tal construção está exibida na Figura 17.

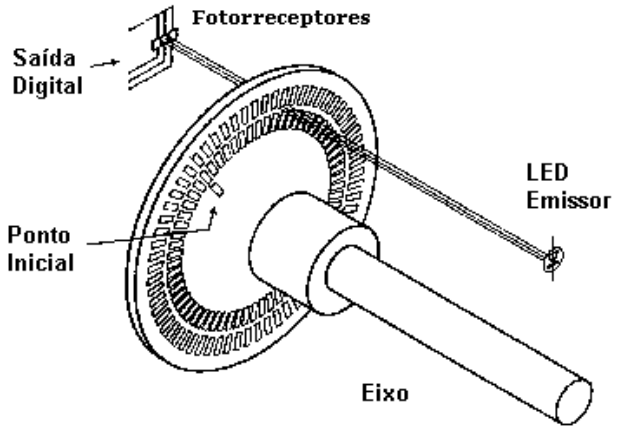


Figura 17: Estrutura do *Encoder* Óptico

Conforme o eixo do motor gira, pulsos quadrados são obtidos na saída digital. Tais pulsos devem então ser enviados a um sistema capaz de realizar o tipo de controle adequado. Para a aplicação em questão apenas o controle de velocidade faz-se necessário.

3 MATERIAIS E MÉTODOS

A placa eletrônica do robô, desenvolvida neste trabalho, tem por função fazer a aquisição dos dados transmitidos por um computador, via RF, convertê-los em sinais elétricos, que são processados por um elemento central e enviados aos respectivos periféricos, garantindo a correta execução dos movimentos solicitados pelo PC ao robô.

Tal placa pode ser dividida em três partes principais, sendo elas, acionamento e controle dos motores, comunicação RF e processamento digital, que serão detalhadas nas seções subsequentes.

A placa eletrônica possui dois módulos transceptores de alta frequência TRF-2.4G, permitindo assim a telemetria da plataforma robótica, sendo que um dos dispositivos RF apenas recebe as informações passadas pelo computador e o outro transmite dados pertinentes do mesmo.

3.1 Comunicação

Como o projeto prevê que a plataforma robótica deve tanto receber quanto transmitir informações ao computador, para facilitar a programação do dispositivo foram utilizados dois módulos transceptores, quando apenas um, desde que programado de forma otimizada, seria capaz de suportar a carga de dados.

Ao observar a Figura 3, apresentada no tópico 2.2.1 , nota-se que a disposição dos 10 pinos do dispositivo não favorece sua utilização. Dessa maneira o dispositivo foi adquirido com uma placa de adaptação de tais pinos para *pin header* de 1x10, como pode ser observado na Figura 18, destacado com um retângulo vermelho.

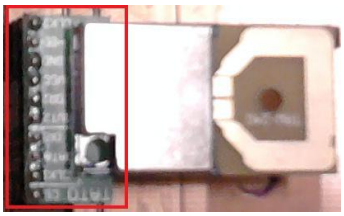


Figura 18: Adaptador do *Transceiver*

Na Figura 19 está exibido o esquemático de conexão dos dois módulos transceptores. Tal projeto é relativamente simples, uma vez que as tensões utilizadas pelo TRF-2.4G e pelo MCU são compatí-

veis, porém deve-se notar que, como é utilizada uma placa de conversão dos pinos para o *layout* de *pin header* 1x10, o posicionamento deles é alterado.

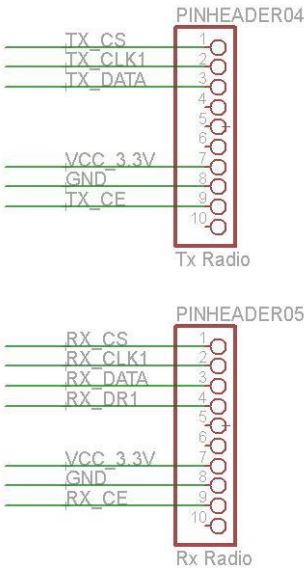


Figura 19: Projeto dos *Transceivers*

No projeto foi utilizado apenas o canal 1 para realizar a comunicação tanto TX como RX. A diferença entre o módulo receptor e o transmissor está no fato do módulo de recepção possuir o pino denominado RX_DR1 conectado ao pino de interrupção externa do MCU, INT0.

Foi utilizada a tecnologia de *ShockBurst*^{TM} para realizar a comunicação, exigindo menor processamento de dados do microcontrolador, já que esse fica encarregado de programar os registradores do *transceiver* uma única vez, sendo necessário então apenas transferir e receber o *payload* desse.

3.2 Processamento de Dados

Todas as informações da placa eletrônica são centralizadas no microcontrolador presente nela, para então serem analisadas e, de acordo com a programação do dispositivo, gerarem ações que serão executadas pelo robô.

O MCU selecionado para realizar o processamento dos dados do robô foi o dsPIC33FJ256MC710A da fabricante Microchip.

O manual do dispositivo recomenda conexões mínimas de capacitores e resistores ao MCU, para que o mesmo funcione de maneira correta. Esta configuração está presente na Figura 20. [8]

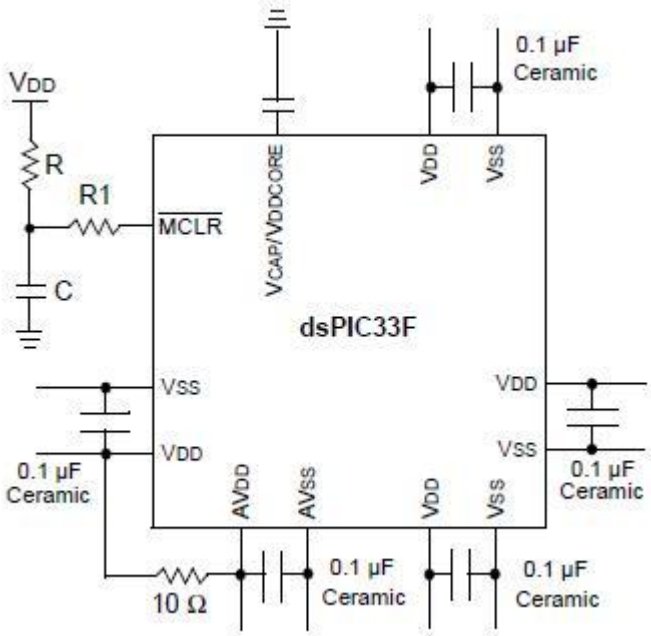


Figura 20: Conexões Mínimas Recomendadas [8]

Como o dispositivo comprado já apresentava tais conexões em sua estrutura, não foi necessário incluí-las no projeto, simplificando-o. Dessa maneira bastou realizar as interligações entre o ds-PIC33FJ256MC710A e seus periféricos, como exibido na Figura 21.

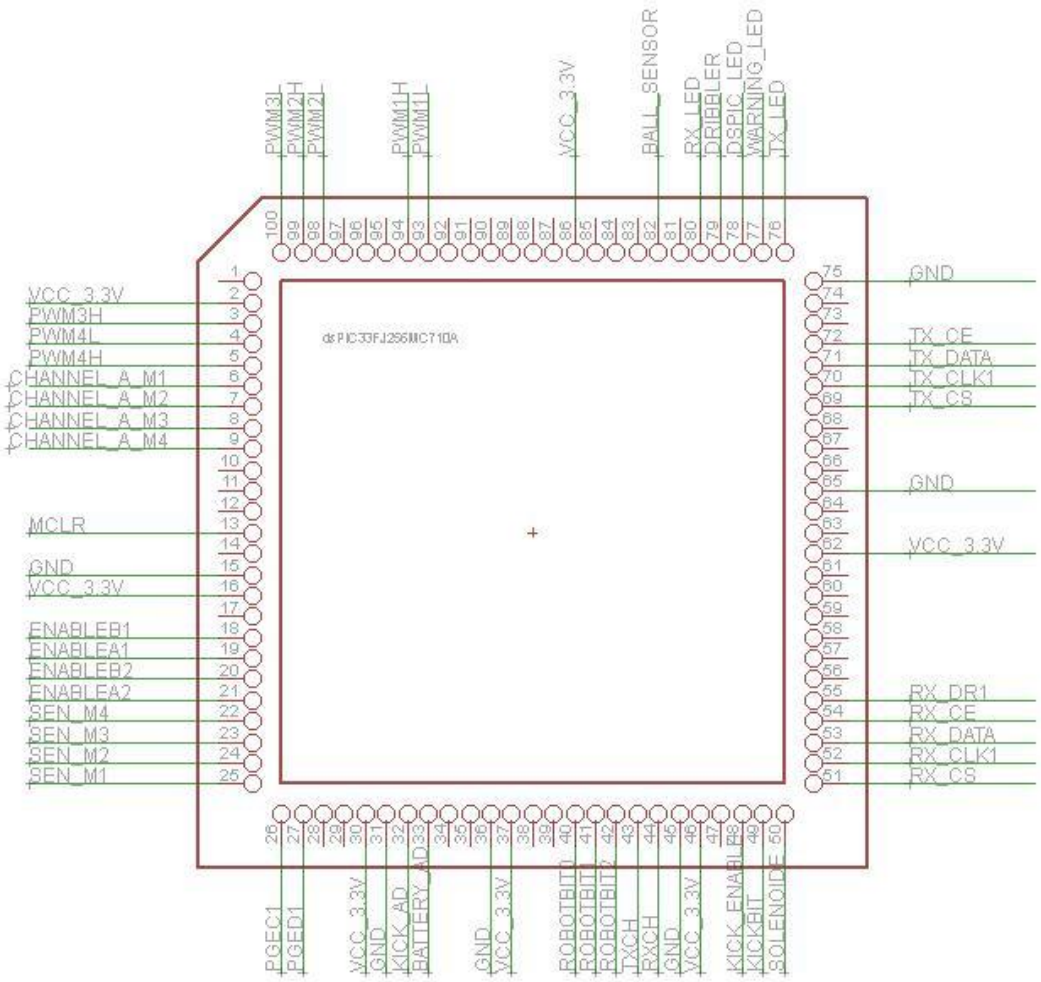


Figura 21: Esquemático dsPIC33FJ

Como tal CI não existia na versão do Eagle 5.4.0, e não foram encontradas versões satisfatórias para *download*, foi necessário o desenvolvimento de uma biblioteca para o mesmo.

3.3 Acionamento dos Motores

O robô é provido de cinco motores, sendo quatro deles para movimentação do sistema completo e um para fornecer torque ao mecanismo auxiliar.

Baseado nos dispositivos apresentados nos itens 2.4.1 e 2.4.2 , considerando também suas cara cterísticas elétricas e modo de funcionamento, foi gerado o seguinte esquemático para possibilitar o acionamento e o controle de tais motores.

Tal esquemático, construído utilizando o software Eagle 5.4.0, está apresentado na Figura 22.

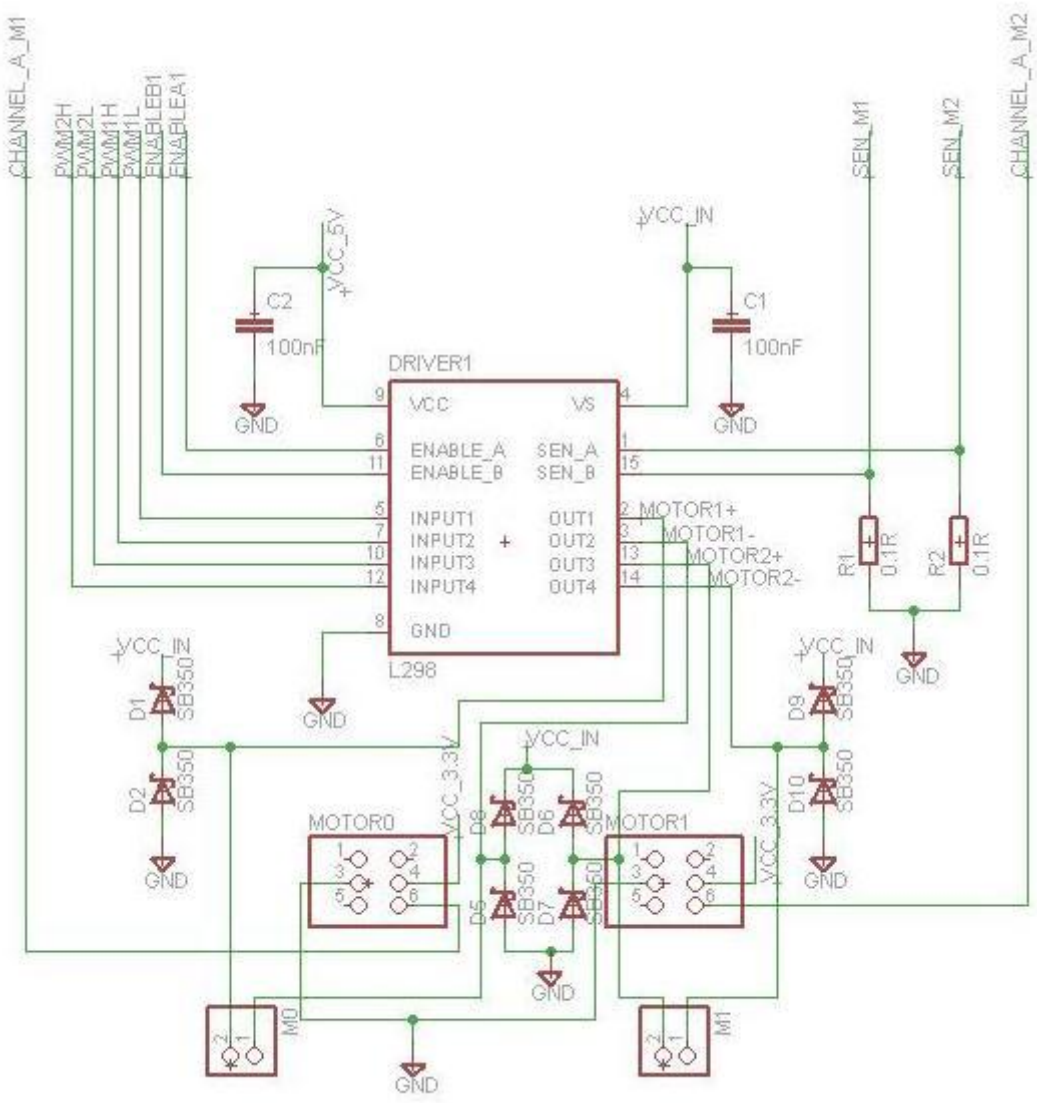
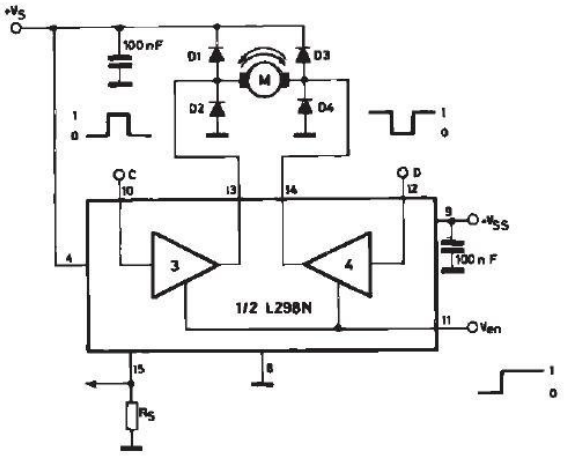


Figura 22: Esquemático do Sistema de Acionamento e Controle dos Motores

Na Figura 22, para melhor visualização, está apresentado apenas o esquemático de acionamento de dois motores, sendo que o esquemático completo contempla o acionamento de quatro motores.

3.3.1 Modo de Operação

O circuito de acionamento de motores DC funciona como exibido na Figura 23.



Entradas		Função
V_{en} = H	C = H ; D = L	Frente
	C = L ; D = H	Trás
	C = D	Parada Rápida
V_{en} = L	C = X ; D = X	Parada com Motor Solto

L = Low H = High X = Don't care

Figura 23: Operação do CI L298

Conforme pôde ser observado na Figura 23 enquanto o pino de habilitação fica em estado lógico baixo, o motor fica livre, ao passo que quando tal pino está em estado lógico alto, existem três possibilidades, sendo elas compostas pelo giro do motor em um sentido, no sentido contrário e travado, quando as duas entradas assumem o mesmo nível lógico.

Foi utilizado o método de modulação em pulsos (PWM) para setar as velocidades desejadas nos motores, a frequência utilizada para tanto foi de 25kHz, como aconselhado no manual do dispositivo.

Inicialmente esses sinais foram pulsados no pino de habilitação, pois dessa forma apenas um pino precisava mudar de estado para aplicar a velocidade desejada ao motor, porém ao levantar a curva de Velocidade x Tensão, foi verificado que a mesma era complexa, ao passo que quando o pino pulsado era uma das entradas, tal relação tornava-se linear, facilitando o cálculo do *duty cicle* do PWM de forma que se conseguisse a velocidade desejada.

Uma grande desvantagem de utilizar o PWM nos pinos de entrada é que são necessários dois PWMs por motor para que o mesmo possa girar em ambos os sentidos, ao passo que utilizando o pino de habilitação é necessário apenas um PWM para executar a mesma função. Como o MCU selecionado é provido de oito PWMs e o projeto prevê quatro motores, optou-se pela utilização dos PWMs nos pinos de entrada e utilização da curva Velocidade x Tensão linear.

As equações 2 e 3 mostram como é calculado o valor que será inserido em cada um do registradores de *duty cicle*, utilizando como parâmetro a velocidade que deseja em cada roda do robô, levando em consideração a caixa de redução utilizada, a tensão da bateria, obtida por um conversor A/D e a queda de tensão aproximada nos BJTs do CI L298 de 1,6V.

$$Tensão_{Rotação} = \frac{Velocidade_{Roda} \times 60 \times Caixa_{Redução}}{2 \times \pi \times Constante_{Rotação}} \quad (2)$$

$$PWM_{DC} = \frac{Tensão_{Rotação} \times PWM_{Máximo}}{Tensão_{Bateria} - 1{,}6} \quad (3)$$

3.3.2 Funcionamento do Acionamento e Controle dos Motores

O valor do *duty cicle* obtido para cada motor é programado em seus respectivos registradores de PWM. Tais PWMs fornecerão os pulsos considerando frequência estabelecida, tempo de trabalho e sentido de rotação, aspectos esses informados pelo MCU.

Essas saídas são aplicadas às entradas das duas pontes-H presentes nos circuitos, fazendo com que as mesmas forneçam a tensão média dos pulsos em suas entradas aos motores e a corrente exigida por eles, desde que não ultrapasse o valor de 1,4A, como recomendado no manual.

O monitoramento da corrente é realizado por meio do pino denominado *Sense*, fornecido na ponte-H, sendo que a corrente que flui pelo motor, também passa por esse pino e pelo resistor de potência conectado entre tal ponto e o GND, possibilitando dessa forma, a leitura da queda de tensão em cima do mesmo e consequentemente a obtenção do valor da corrente de entrada.

Como o piso por onde o robô navega geralmente possui irregularidades e diferentes resistências de contato além de variações nos valores típicos dos componentes, a velocidade determinada pelo MCU difere daquela à qual o motor está realmente funcionando. Para tanto o *encoder* óptico acoplado ao mesmo, fornece ao microcontrolador a velocidade no eixo de saída do motor, realimentando o circuito e possibilitando o ajuste de sua velocidade, por meio de algum algorítimo de controle, como o P, PI ou PID.

3.3.3 Motor Auxiliar

Foi realizada uma análise parcial para a determinação de tal motor, já que tem como finalidade apenas manter a bola do jogo presa ao robô por meio de um dispositivo chamado *dribbler*.

Dessa maneira foram utilizados motores que estavam no laboratório, porém não possuíam números de referências, consequentemente não foi possível encontrar os manuais dos mesmos, apenas sabe-se que tais motores aceitam as tensões de 12V aplicadas a eles. Com isso um simples esquemático foi projetado para realizar o acionamento desses mecanismos. Tal projeto encontra-se na Figura 24.

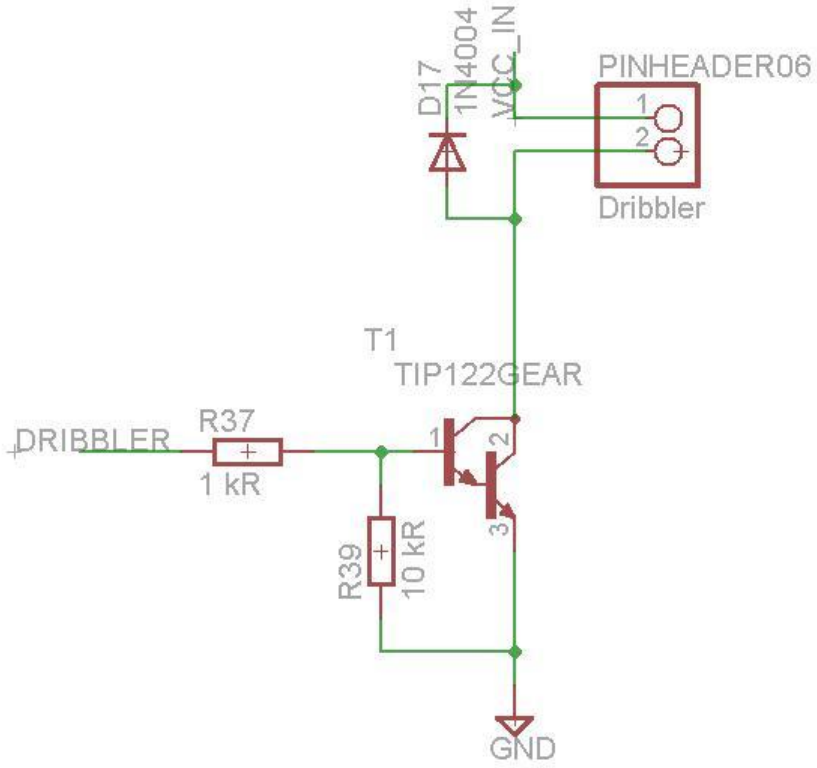


Figura 24: Esquemático *Dribbler*

O resistor R37, com valor de 1kΩ tem como objetivo limitar a corrente que entra na base do BJT Darlington TIP 122. O diodo D17 1N4004 realiza a proteção do TIP 122, jogando a corrente residual do motor de volta no mesmo.

3.4 Módulos Adicionais

3.4.1 DIP *Switch*

A placa eletrônica é provida de um conector DIP *Switch* de 5 chaves, sendo as chaves 1, 2 e 3 responsáveis por informar ao MCU o número do robô, aceitando dessa maneira até 8 numerações distintas. A chave 4 é responsável por selecionar uma das duas frequências de transmissão de dados e, finalmente, a chave 5 é responsável por selecionar uma das duas frequências da recepção de dados. A posição das chaves deve ser ajustada antes de ligar o robô, já que o MCU utilizará tal disposição para realizar as configurações iniciais. A Figura 25 mostra o esquemático de tal bloco.

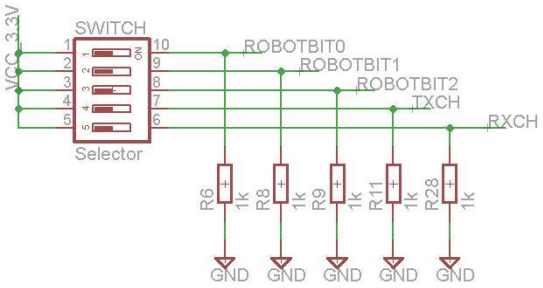


Figura 25: Esquemático DIP *Switch*

3.4.2 Reset

O microcontrolador, segundo seu manual, é reiniciado quando a tensão em seu pino 13, denominado de *Master Clear*, é posta em nível lógico baixo. Esse funcionamento pode ser verificado por meio do esquemático da Figura 26.

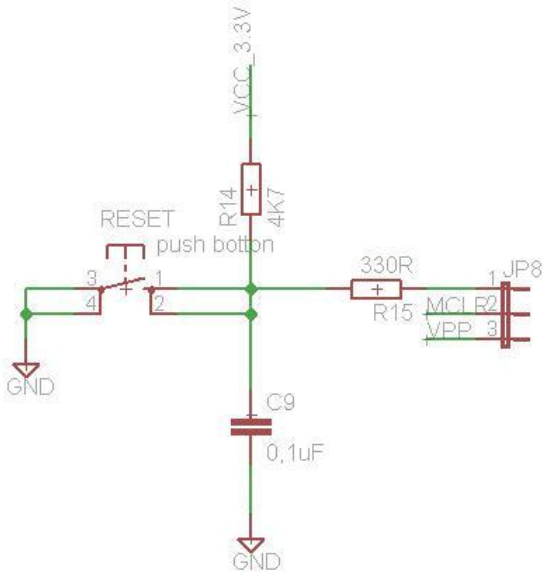


Figura 26: Esquemático *Reset*

Pode-se observar que ao pressionar o *push botton*, o nível lógico presente no pino MCLR será zero, essa tensão será mantida em nível lógico baixo por um curto período de tempo devido ao capacitor C9, garantindo assim o tempo mínimo para executar o *reset* do dispositivo.

Faz-se necessária a utilização de um *jumper* no circuito, pois ao realizar o processo de gravação do MCU, tensões da ordem de 12V podem ser aplicadas ao pino MCLR pelo V_{PP}, podendo danificar outros CIs presentes na placa. Com a inserção do *jumper* a isolação física do sistema é concretizada, fornecendo mais segurança a ele.

3.4.3 ICSP

Como já explicado no capítulo que trata do microcontrolador presente no projeto, tal dispositivo permite a sua programação sem ser necessária a retirada do mesmo do circuito, porém, como podem surgir tensões da ordem de 12V ao realizar a programação de MCU em circuitos, é necessário prover proteções aos periféricos, que podem não suportar tensões de tamanha magnitude.

Para tanto foi incluído o jumper chamado de JP8 no esquemático, como pode ser observado no capítulo 3.4.2 , e o esquemático presente na Figura 27 foi projetado para permitir a realização do ICSP de maneira correta.

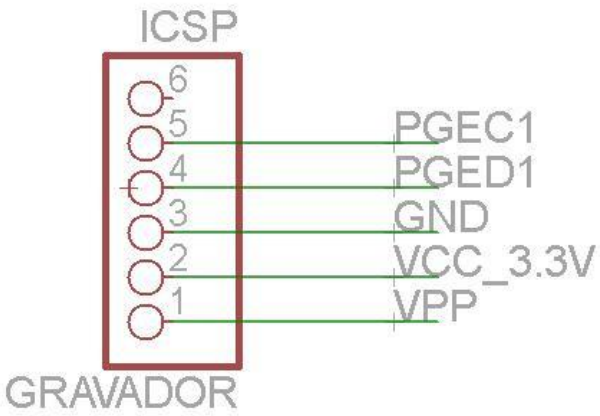


Figura 27: ICSP Esquemático

O MCU, como já dito anteriormente, permite a utilização do *Debug on Chip*, porém o gravador do mesmo, PICKIT 3, não consegue fornecer correntes superiores a 0,5A e para tanto foi interessante colocar o pino 2 do gravador com a saída de 3,3V fornecida pelo regulador de tensão UA78M33CKC. Dessa maneira o circuito consegue ser emulado completamente, pois as correntes exigidas pelo mesmo conseguem ser fornecidas pelo CI.

3.4.4 Reguladores

Para garantir de maneira precisa as tensões de alimentação dos CIs presentes na placa eletrônica foi necessária a utilização de dois reguladores de tensão, sendo um com 5V de tensão de saída e o outro com 3,3V de saída.

Segue apresentado na Figura 28 o esquemático dos reguladores de tensão.

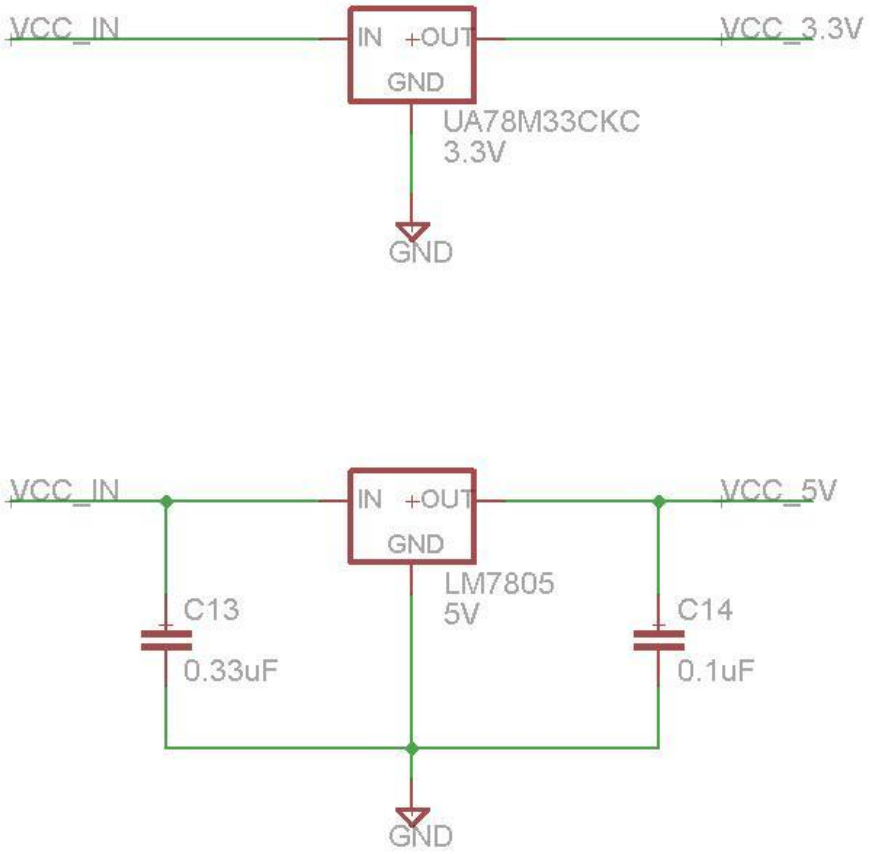


Figura 28: Esquemático Reguladores de Tensão

Nota-se uma singela diferença entre os dois reguladores de tensão. O primeiro, de 3,3V, não necessita de capacitores, pois trata-se de um dispositivo construído para possuir muita precisão em sua tensão de saída, característica exigida pelo dispositivo que será alimentado pelo MCU.

O segundo regulador de tensão, com saída projetada para 5V, necessita do capacitor de saída para garantir maior precisão na mesma, já que trata-se de um dispositivo com menor complexidade, se comparado ao anterior, porém os dispositivos alimentados por ele não são tão sensíveis como o microcontrolador utilizado nesse projeto.

3.4.5 LEDs

Com o intuito de possibilitar uma análise básica e rápida do funcionamento de alguns sistemas da placa de *hardware*, cinco LEDs foram adicionados a ela. O primeiro acende quando a tensão da bateria atinge um valor próximo ao limite, o segundo trata do funcionamento do MCU, o terceiro entra em ação sempre que a tensão de 3,3V está presente, o quarto e o quinto LED piscam quando o robô está transmitindo ou recebendo informações, respectivamente.

A Figura 29 mostra o esquemático do sistema de *debug* visual.

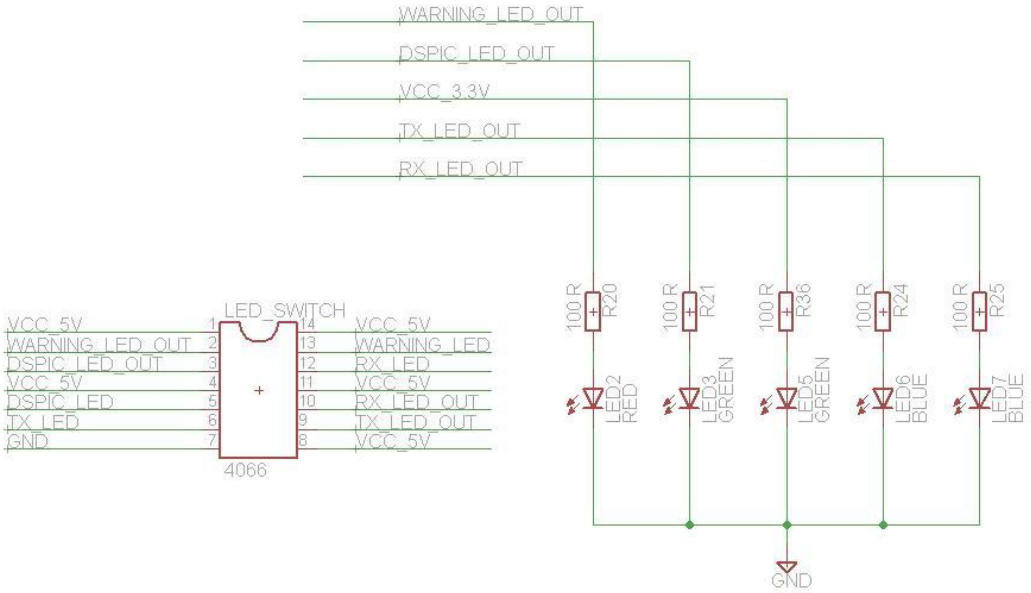


Figura 29: Esquemático LEDs

Como o MCU utilizado não deve fornecer correntes maiores que 4 mA por pino, foi necessário utilizar o CI MC14066, dispositivo aqui chamado de LED *Switch*, que é capaz de fornecer as correntes necessárias para o acionamento dos LEDs, tais correntes podem variar tipicamente entre 20 mA e 35 mA.

3.4.6 Sensor de Presença

Como a aplicação principal de tal robô é jogar futebol, faz-se necessário a utilização de um sensor de presença, para detectar quando o mesmo está de posse da bola.

Para realizar a detecção de objetos com o sensor, utilizou-se um conjunto formado por um LED ultra-violeta e um fotodiodo, acionado quando a luz ultra-violeta entra em contato com sua base.

O projeto deste sistema encontra-se na Figura 30.

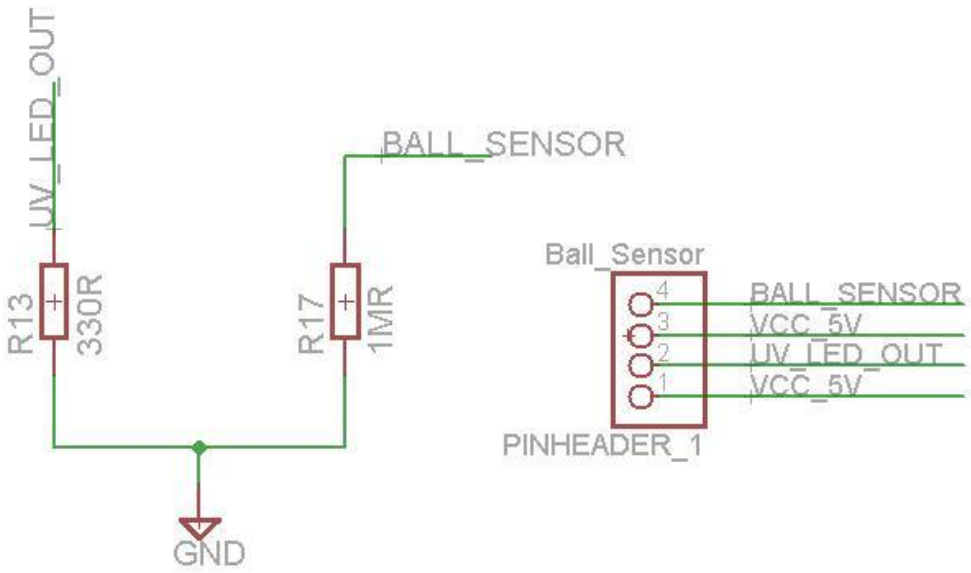


Figura 30: Esquemático do Sensor de Presença

3.4.7 Bateria

Para realizar a alimentação dos robôs foram utilizadas baterias de Lithium Ion de 2.200 mAh com quatro células ligadas em paralelo, gerando tensão nominal de 14,8V, já que cada célula possui tensão nominal igual a 3,7V.

Três características interessantes de tais baterias são que as mesmas não apresentam efeito de memória presente em tantas outras, possuem boa autonomia e sua tensão fica bem estável durante grande parte do tempo de utilização da mesma, como pode ser observado na Figura 31, que exibe a curva de descarga de apenas uma célula de Lithium Ion.

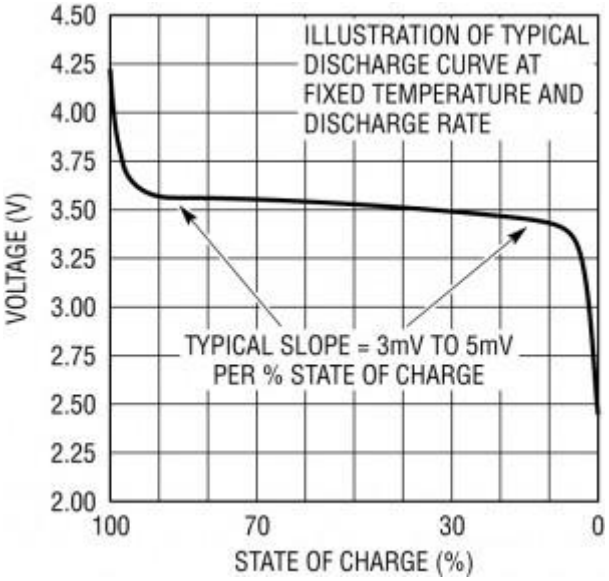


Figura 31: Curva de Descarga de uma Célula [18]

3.5 Confecção da Placa Eletrônica

A lista de materiais utilizada para a confecção da placa eletrônica do robô está apresentada no apêndice A.

Após a construção dos esquemáticos, baseada nas informações coletadas durante o estudo dos componentes, foi gerado um arquivo com extensão ".brd", denominado *board*. A função de tal projeto é traçar as trilhas por onde fluirá a corrente elétrica, além de posicionar os componentes na placa, ou seja, o *board* exibe como realmente ficará a placa, diferentemente do esquemático, onde o intuito principal é apenas realizar a interligação entre componentes.

O projeto da placa pode ser visto na Figura 32.

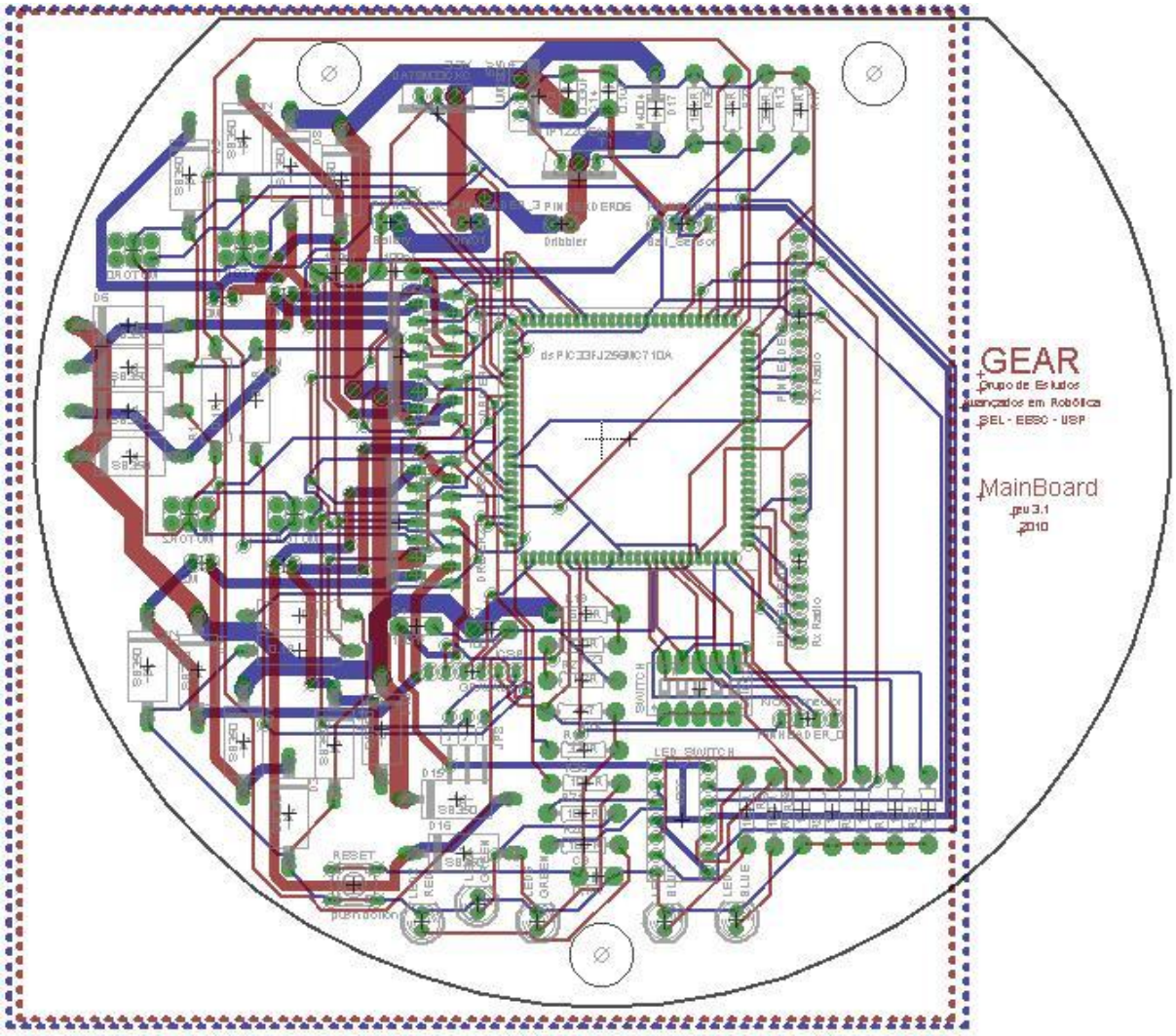


Figura 32: Projeto da Placa Eletrônica

Os componentes são representados pela camada cinza, os *pads* pela camada verde, as trilhas da parte superior de vermelho e as da inferior de azul. Os dois retângulos feitos com linhas tracejadas, nas cores azul e vermelha são para que o *software* faça com que todo espaço não preenchido por trilhas seja conectado ao terra da placa, tanto em sua parte superior como na inferior.

Com o projeto da placa pronto, este deve ser exportado para um formato que a fresa, presente no Departamento de Engenharia Elétrica, compreenda, ou seja, o arquivo com extensão ".brd" deve ser convertido para um arquivo com extensão ".lmd", o qual consegue ser interpretado pelo *software* que gerencia a fresa. Os procedimentos para realizar tal conversão estão descritos no anexo A.

A Figura 33 foi renderizada utilizando o módulo Eagle 3D, possibilitando a melhor visualização de como a placa ficaria quando confeccionada, já a Figura 34 exibe a placa pronta em laboratório.

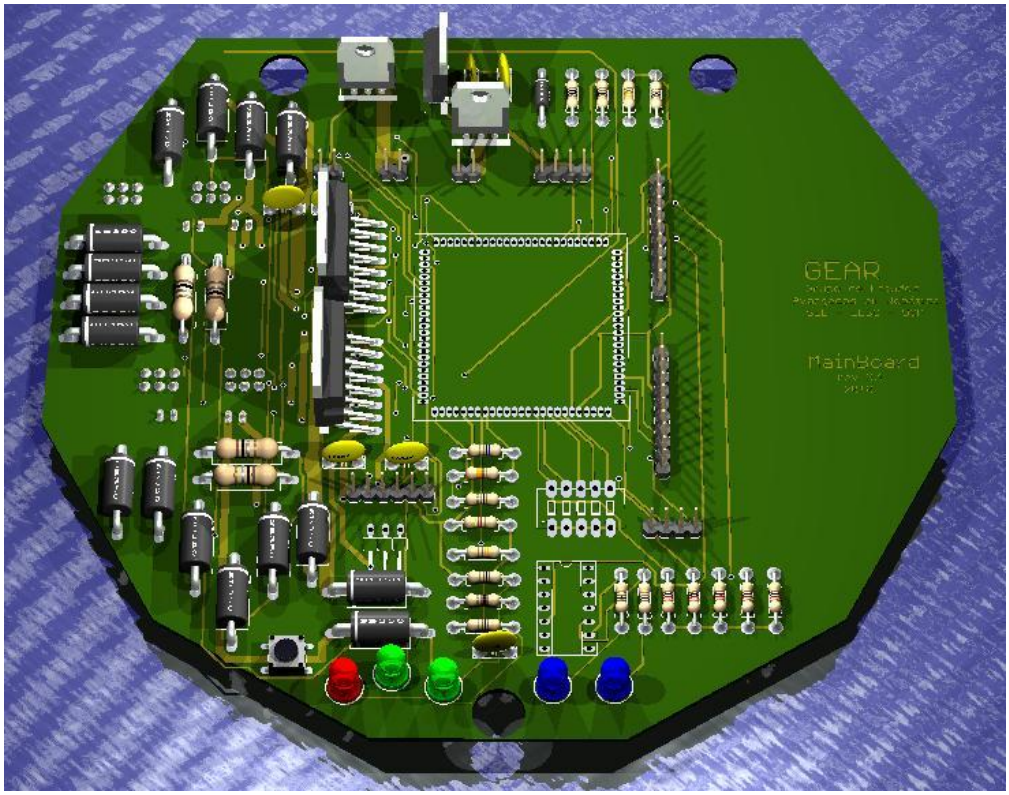


Figura 33: Imagem Renderizada

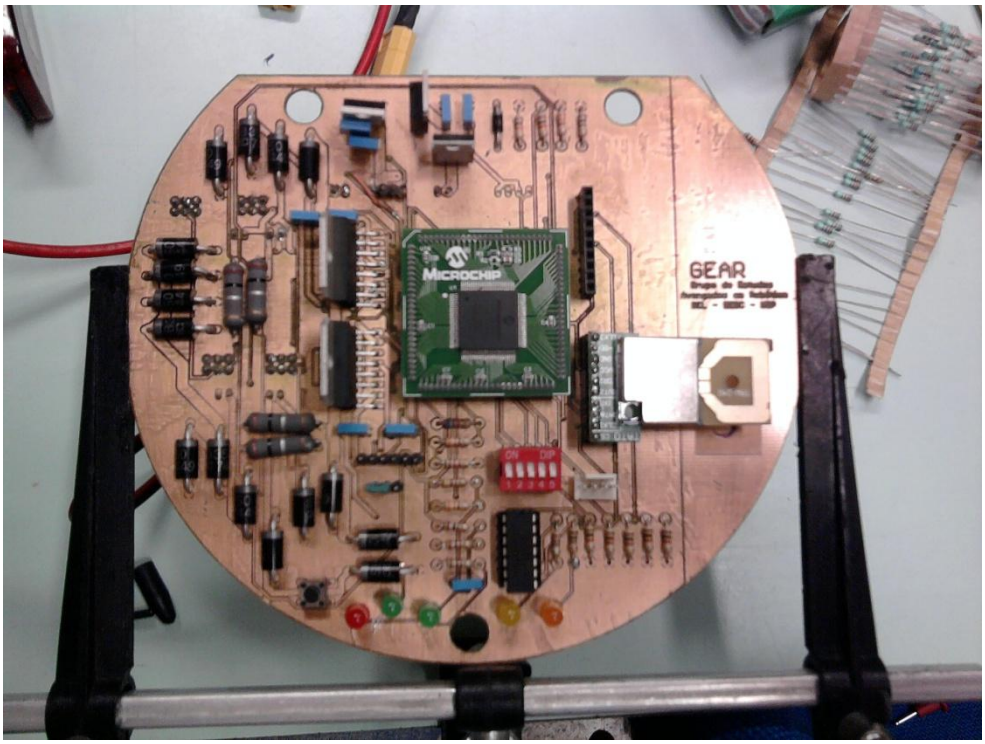


Figura 34: Placa Eletrônica

4 RESULTADOS E DISCUSSÕES

4.1 Resultados

Aplicando-se diferentes valores de *duty cicle*, com frequência igual à 30 kHz, como recomendado em seu manual, no pino denominado *Enable* e no denominado *Input* da ponte-H L298, foram adquiridas as tensões presentes na entrada do motor DC e com elas, foram construídas as curvas presentes na Figura 35, relacionando tais tensões com o *duty cicle* aplicado à entrada do CI L298.

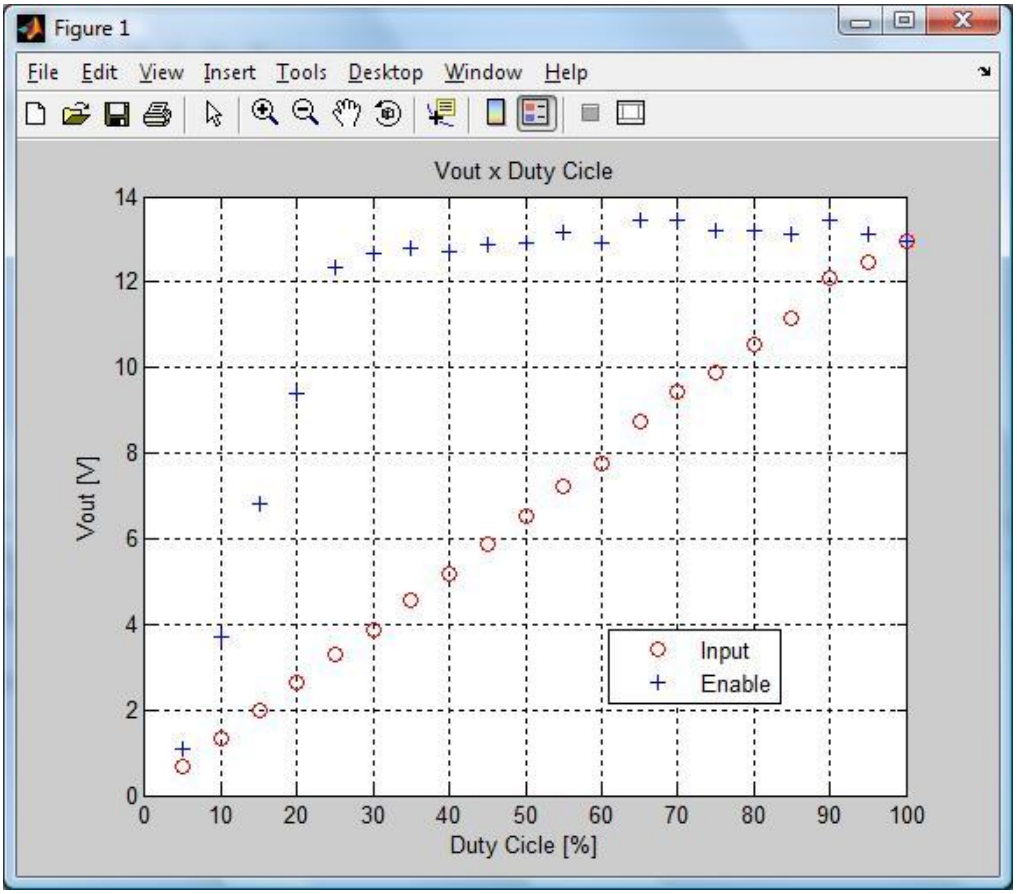


Figura 35: Tensão de Saída x *Duty Cicle*

Assumindo a curva descrita pelo *duty cicle* pulsado no pino *Input* do CI, ao multiplicá-la pela constante de rotação e eficiência do motor, iguais a 713 rpm/V e 81% respectivamente, o gráfico presente na Figura 36 foi gerado, apresentando a velocidade de rotação do eixo do motor contra o *duty cicle* na entrada da ponte-H, antes de ser reduzida pela caixa de redução e sem carga acoplada a ele.

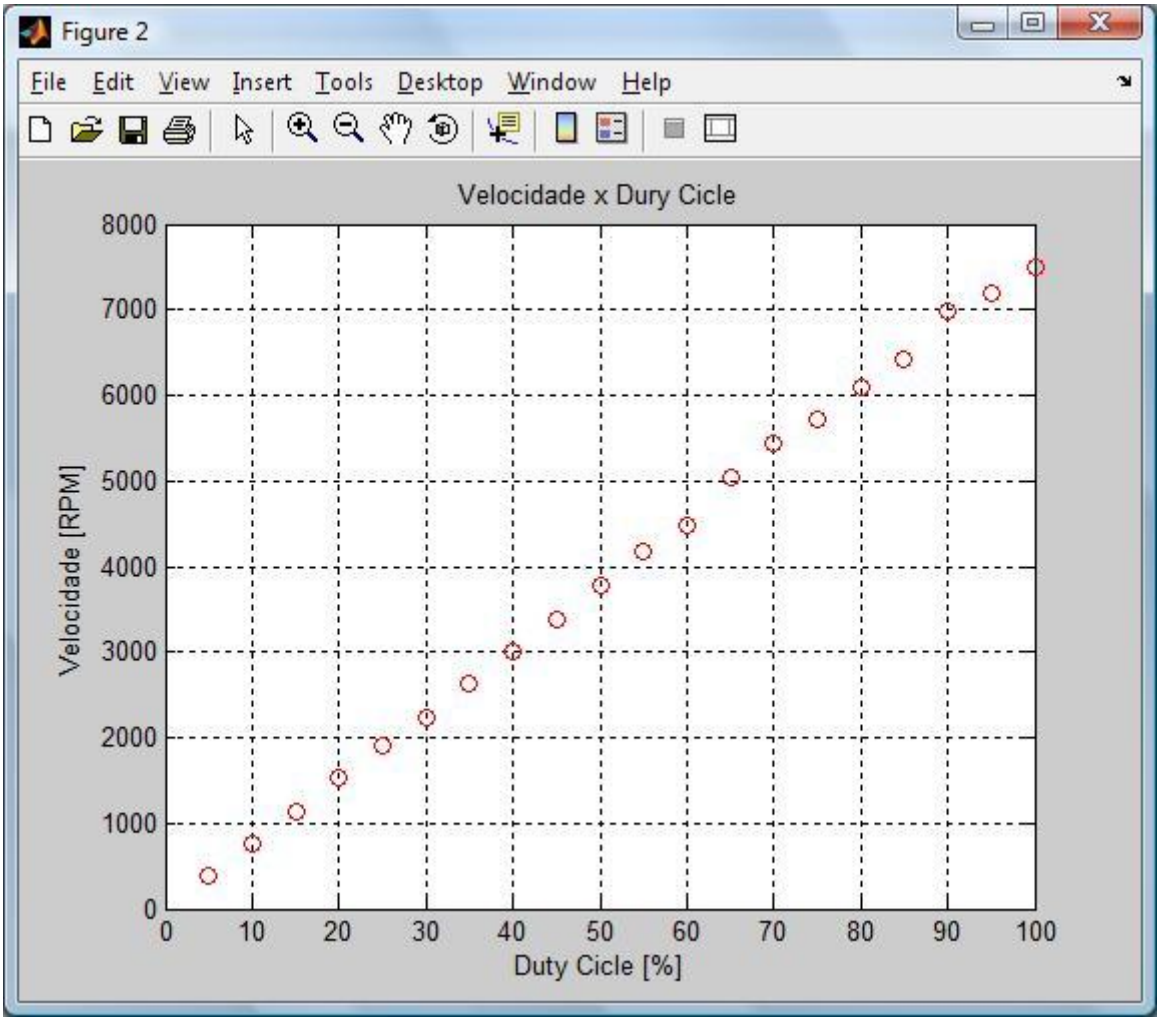


Figura 36: Velocidade do Eixo x *Duty Cicle*

4.2 Discussões

O projeto inicialmente previa a utilização do PWM conectado ao pino *Enable* do CI L298, porém ao observar a curva não linear descrita na Figura 35, tal conexão foi remanejada para o pino *Input*, tornando linear o equacionamento do acionamento dos motores, evitando ter que remodelar as equações 2 e 3. Em contra partida, duas saídas de PWM tiveram que ser utilizadas, ao passo que quando aplicados os pulsos ao pino *Enable*, apenas uma saída seria necessária.

4.3 Melhorias

A primeira melhoria proposta, e mais impactante no projeto, é a contratação de empresas para realizar a confecção das placas eletrônicas e soldagem de seus componentes, melhorando a qualidade do produto final. Muitos pontos de contato entre trilha e terra foram encontrados nas placas, sendo dessa maneira necessário realizar a checagem de todas antes de realizar a alimentação das mesmas. As trilhas muitas vezes vinham com espessuras inferiores àquelas projetadas, fazendo com que as amplitudes de correntes suportadas por elas sejam menores.

Modularizar a placa principal, dividindo-a em duas, uma responsável pela comunicação e processamento de dados e outra somente pelo acionamento dos motores, separando, dessa maneira, a parte digital da parte de potência e tornando mais barata a substituição de um dos dois módulos, caso necessário.

Utilizar motores *brushless*, que duram mais do que os micromotores DC e possuem maior eficiência, além de serem menores que os com buchas.

Absorver o máximo desempenho do transceptor TRF-2.4G utilizando apenas um desses dispositivos para realizar a transmissão e recepção dos dados, diferente de como está o projeto atual, um operando apenas como receptor e outro como transmissor, gerando economia para o projeto na ordem de R$ 70,00 por transceptor.

CONCLUSÃO

Com os ensinamentos obtidos em sala de aula, associados à busca por informações adicionais em livros, na internet e principalmente em manuais dos dispositivos utilizados, juntamente com o apoio e ajuda dos professores e técnicos do Departamento de Engenharia Elétrica, foi possível a construção da placa principal do robô.

Essa placa possui tamanhos compatíveis com as dimensões máximas aceitas pelo robô, além de contar com um desempenho aceitável, seja na trasmissão e recebimento de dados, no processamento do MCU ou na movimentação do robô.

O resultado final foi satisfatório, pois, participando do Grupo de Estudos Avançados em Robótica - GEAR, o aluno pôde aplicar os conhecimentos obtidos em sala de aula, como também aprender novas tecnologias e temas não abordados pelos professores, além de melhorar sua capacidade de trabalho em grupo e conseguir uma maior aproximação de seus professores e dos técnicos do setor.

Esse projeto, mesmo que concluído, ainda possui muitas melhorias a serem buscadas, algumas delas já apresentadas, porém várias ainda não levantadas. O ramo da robótica, em especial o da inteligência artificial, ainda é pouco explorado no Brasil, porém muito avançado em outros países, restando assim aos demais estudantes de uma faculdade de ponta como é a EESC/USP, tomar esse fato como motivacional para caminhar no sentido de tais países.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ROBOCUP. Soccer Small Size League. Disponível em: <http://www.robocup.org/robocup-soccer/small-size/>. Acesso em: 13 maio 2011.

[2] ROBOCUP. Laws of the RoboCup Small Size League. Disponível em: <http://small-size.informatik.uni-bremen.de/_media/rules:ssl-rules-2011.pdf>. Acesso em: 13 maio 2011.

[3] WIKIPEDIA. Radio Frequency. Disponível em: <http://en.wikipedia.org/wiki/Wireless>. Acesso em: 11 maio 2011.

[4] WIKIPEDIA. Transceiver. Disponível em: <http://en.wikipedia.org/wiki/Transceiver>. Acesso em: 11 maio 2011.

[5] LAIPAC TECHNOLOGY INC. (Canada). TRF-2.4G Transceiver: Data Sheet. Disponível em: <http://www.laipac.com/rf-transceivers.htm>. Acesso em: 09 maio 2011.

[6] NORDIC VLSI ASA (Norway). NRF240x ShockBurst™ technology. Disponível em: <http://read.pudn.com/downloads126/sourcecode/embed/535473/APPNOTE/nRF240x-ShockBurst-feb03.pdf>. Acesso em: 11 maio 2011.

[7] GUSTAVO WEBER DENARDIN. Microcontroladores. Disponível em: <http://pessoal.utfpr.edu.br/gustavo/apostila_micro.pdf>. Acesso em: 11 maio 2011.

[8] MICROCHIP (Estados Unidos da América). DsPIC33FJXXXMCX06/X08/X10: Data Sheet. High-Performance, 16-Bit Digital Signal Controllers. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/70594C.pdf>. Acesso em: 09 maio 2011.

[9] MICROCHIP (Estados Unidos da América). Reference Manual: Section 25. Device Configuration. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/70194F.pdf>. Acesso em: 09 maio 2011.

[10] MICROCHIP (Estados Unidos da América). Reference Manual: Section 7. Oscillator. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/70186D.pdf>. Acesso em: 09 maio 2011.

[11] MICROCHIP (Estados Unidos da América). Reference Manual: Section 10. I/O Ports. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/70193C.pdf>. Acesso em: 09 maio 2011.

[12] MICROCHIP (Estados Unidos da América). Reference Manual: Section 11. Timers. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/70205C.pdf>. Acesso em: 09 maio 2011.

[13] WIKIPEDIA. Modulação por Largura de Pulso. Disponível em: < http://pt.wikipedia.org/wiki/Modula%C3%A7%C3%A3o_por_largura_de_pulso>. Acesso em: 12 maio 2011.

[14] MICROCHIP (Estados Unidos da América). Reference Manual: Section 14. Motor Control PWM. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/70187D.pdf>. Acesso em: 09 maio 2011.

[15] FAULHABER (Germany). DC-Micromotors: Graphite Commutation. Series 2342 ... CR. Disponível em: <http://www.faulhaber.com/uploadpk/EN_2342_CR_DFF.pdf>. Acesso em: 13 maio 2011.

[16] STMICROELECTRONICS (Italy). L298: Data Sheet. DUAL FULL-BRIDGE DRIVER. Disponível em: <http://www.datasheetcatalog.org/datasheet2/2/052daje928cw7pc0uqs1ipyryppy.pdf>. Acesso em: 09 maio 2011.

[17] FAULHABER (Germany). Encoders: Magnetic Encoders. Disponível em: <http://www.faulhaber.com/uploadpk/EN_IE2-16_DFF.pdf>. Acesso em: 13 maio 2011.

[18] Automotive Electronics News. Insight for Engineers. Disponível em: http://johndayautomotivelectronics.com/?p=517. Acesso em: 09 maio 2011.

APÊNDICE

A. Lista de Materiais

A Tabela 9 lista todos os componentes necessários na replicação da placa principal e suas quantidades.

Tabela 9: Lista de Materiais Utilizados na Confecção da Placa Eletrônica

Lista de Materiais	
Componente	**Quantidade**
TRF2.4GHz	2
dsPIC33FJ256MC710A	1
L298	2
Diodo SB350	16
Resistor 3W 0,1Ω	4
Diodo 1N4004	1
Transistor TIP122	1
Regulador de Tensão LM7805	1
Regulador de Tensão UA78M33CKC	1
LED Vermelho	1
LED Verde	2

LED Azul	2
Push Botton	1
DIP *Switch*	1
4066	1
***Pin Header* 2x3**	4
Jumper	1
***Pin Header* 1x2**	7
***Pin Header* 1x4**	2
Capacitor 100nF	6
Capacitor 330nF	1
Resistor 1kΩ	6
Resistor 100Ω	5
Resistor 330Ω	3
Resistor 4,7kΩ	1
Resistor 10kΩ	1
Resistor 1MΩ	1
Resistor 6,8kΩ	1
***Pin Header* 1x6**	1
LED Ultra-violeta	1
Fotodiodo	1

B. Códigos

Os códigos apresentados abaixo são responsáveis por realizar a movimentação do robô através do acionamento e controle de seus motores.

main.c

```
#include <p33FJ256MC710A.h>
#include <gtkernel.h>
#include <setup.h>
#include <kick.h>
#include <trf24g.h>
#include <geartucanomotor.h>

/* Configuration Bits */
_FGS(GSS_OFF & GCP_OFF & GWRP_OFF)
_FOSCSEL(FNOSC_FRCPLL & IESO_OFF)
_FOSC(FCKSM_CSDCMD & OSCIOFNC_ON & POSCMD_NONE)
_FWDT(FWDTEN_OFF & WINDIS_OFF & PLLKEN_ON)
_FPOR(PWMPIN_OFF & HPOL_ON & LPOL_ON)
_FICD(JTAGEN_OFF & ICS_PGD1)

/* Variable Declaration */
uint16 adConversion[highestADNumber+1] __attribute__((space(dma),aligned(16)));
uint8 recepFreq;
uint8 transFreq;
uint8 robotName[4] = "TUC";
uint8 stationName[] = "TUCS";
float batteryVoltage;
float kickVoltage;
uint8 adFlag;
uint8 adTypeFlag;
uint16 currentDrivers[4];

int main() {
        uint8 ball;
        float txBuffer[3];

        /* Oscillator Configuration */
        CLKDIVbits.PLLPRE = 0;              /* N1 = 5 */
        CLKDIVbits.PLLPOST = 0;             /* N2 = 2 */
        PLLFBDbits.PLLDIV = 41;             /* M = 16*/

        RCONbits.SWDTEN=0;                  /* Disable Watch Dog Timer */

        /* Clock switching to incorporate PLL */
        __builtin_write_OSCCONH(0x01);          /* Initiate Clock Switch to Primary */
        __builtin_write_OSCCONL(0x01);          /* Start clock switching */
        while(OSCCONbits.COSC != 0x01);         /* Wait for Clock switch to occur */
```

```
	while(OSCCONbits.LOCK != 0x01);			/* Wait for Clock switch to occur */

	setup();		/* General configuration */
	setMotor ();

	dsPICLED = 1;			/* Turn On dsPIC LED */

	robotNumber(robotName);			/* Define the robot number */
	radioFreq(&recepFreq, &transFreq);			/* Define the radio frequencies */

	radioTxConfig(transFreq, robotName);			/* Radio transmission configuration */
	radioRxConfig(recepFreq, robotName);			/* Radio reception configuration */

	while (1) {
		if(adFlag) {
			if(kickVoltage >= kickLEDVoltageTest)		/* Kick voltage check */
				Nop();
//				kickLED = 1;			/* Turn on kicker LED */
			else if(kickVoltage >= kickVoltageTest)
				setKick(255);
			else
				Nop();
//				kickLED = 0;			/* Turn off kicker LED */
			if(batteryVoltage <= batteryVoltageTest)		/* Battery voltage check */
				batteryLED = 1;			/* Turn on battery warning LED */
			else
				batteryLED = 0;			/* Turn off battery warning LED */
			adFlag = 0;
			if(ballSensor == 1)		/* Ball sensor check */
				ball = 1;
			else
				ball = 0;

			txBuffer[0] = kickVoltage;
			txBuffer[1] = batteryVoltage;
			txBuffer[2] = (float) ball;

			TRF24G_setActive(1, 0);			/* Setting radio to active mode */

			TRF24G_putData(1, stationName, sizeof(stationName), (uint8*) txBuffer, sizeof(txBuffer));
		/* Sending data for TUCANO's base */
			TXLED = 1;
			TRF24G_setStandBy(1);			/* Setting radion on stand by mode */
		}
	}
}


void setup(void) {
	ADConfig();			/* AD1 configuration */
```

```
	dmaConfig();			/* DMA configuration */
	timersConfig();			/* Timers configuration */
	portsConfig();			/* Ports configuration */
	extIntConfig();			/* External Interruption configuration */

	enableAD();			/* Enable AD */
	enableDMA();			/* Enable DMA */
	enableTimer3();			/* Enable Timer 3 */
	setInterrupts();		/* Set Interruptions */
}


void robotNumber(uint8* robotName) {
	robotName[3] = 0x30 + robotBit2*4 + robotBit1*2 + robotBit0;
}


void radioFreq(uint8* recepFreq, uint8* transFreq) {
	if(rxFreqBit == 0)
		*recepFreq = 0;			/* reception channel 0 */
	else
		*recepFreq = 1;			/* reception channel 1 */
	if(txFreqBit == 0)
		*transFreq = 0;			/* transmission channel 0 */
	else
		*transFreq = 1;			/* transmission channel 1 */
}
```

geartucanomotor.c:

```
#include <p33FJ256MC710A.h>
#include <math.h>
#include <geartucanomotor.h>

unsigned long pwmMax;
uint16 pwmDC[4], pwmDCRef[4];
uint16 tensaoBat = 13.5;
float finalSpeed[4];

void setSpeed(uint8 motor, float speed) {
//	finalSpeed[motor] = speed;
	finalSpeed[motor] = 200;
}

void motorPinsConfig (void) {
   /* Sets motors enable pins as outputs */
   TRISEbits.TRISE9 = 0;	/* Motor 0 Enable*/
   TRISEbits.TRISE8 = 0;	/* Motor 1 Enable */
   TRISBbits.TRISB4 = 0;	/* Motor 2 Enable */
   TRISBbits.TRISB5 = 0;	/* Motor 3 Enable */
```

```
    /* Sets motors current sensors pins as inputs */
    TRISBbits.TRISB0 = 0;       /* Motor 0 Current */
            TRISBbits.TRISB1 = 0;       /* Motor 1 Current */
    TRISBbits.TRISB2 = 0;       /* Motor 2 Current */
            TRISBbits.TRISB3 = 0;       /* Motor 3 Current */

    /* Sets motors PWM pins as outputs */
    TRISEbits.TRISE0 = 0;       /* PWM0L */
    TRISEbits.TRISE1 = 0;       /* PWM0H */
    TRISEbits.TRISE2 = 0;       /* PWM1L */
    TRISEbits.TRISE3 = 0;       /* PWM1H */
    TRISEbits.TRISE4 = 0;       /* PWM2L */
    TRISEbits.TRISE5 = 0;       /* PWM2H */
    TRISEbits.TRISE6 = 0;       /* PWM3L */
    TRISEbits.TRISE7 = 0;       /* PWM3H */

    /* Sets motors encoders pins as inputs */
    TRISCbits.TRISC1 = 1;       /* Encoder 0 */
    TRISCbits.TRISC2 = 1;       /* Encoder 1 */
    TRISCbits.TRISC3 = 1;       /* Encoder 2 */
    TRISCbits.TRISC4 = 1;       /* Encoder 3 */
}

void pwmConfig (void) {
            uint16 ptperValue;

            /* PWM max value */
            pwmMax = 2*FCY/FPWM;

            /* PTPER value */
            ptperValue = (FCY/(FPWM*PTMRPRESCALER))-1;

            /* PWM TIME BASE CONTROL REGISTER */
            /* PWM time base is off */
            P1TCONbits.PTEN = 0;
            /* PWM time base operates in a Free-Running mode */
            P1TCONbits.PTMOD = 0;
            /* PWM time base input clock period is TCY (1:1 prescale) */
            P1TCONbits.PTCKPS = 0;
            /* 1:1 postscale */
            P1TCONbits.PTOPS = 0;
            /* PWM time base runs in CPU Idle mode */
            P1TCONbits.PTSIDL = 0;

            /* PWM TIMER COUNT VALUE REGISTER -> Not used */
            /* PWM Time Base Register Count Value bits */
            P1TMRbits.PTMR = 0;

            /* PWM TIME BASE PERIOD REGISTER */
```

```
P1TPERbits.PTPER = ptperValue;

/* SPECIAL EVENT COMPARE REGISTER -> Not used */
P1SECMPbits.SEVTCMP = 0;
P1SECMPbits.SEVTDIR = 0;

/* PWM CONTROL REGISTER 1 */
/* PWMxL pin is enabled for PWM output */
PWM1CON1bits.PEN1L = 1;
PWM1CON1bits.PEN2L = 1;
PWM1CON1bits.PEN3L = 1;
PWM1CON1bits.PEN4L = 1;
/* PWMxH pin is disabled; I/O pin becomes general purpose I/O */
PWM1CON1bits.PEN1H = 1;
PWM1CON1bits.PEN2H = 1;
PWM1CON1bits.PEN3H = 1;
PWM1CON1bits.PEN4H = 1;
/* PWM I/O pin pair is in the Independent PWM Output mode */
PWM1CON1bits.PMOD1 = 1;
PWM1CON1bits.PMOD2 = 1;
PWM1CON1bits.PMOD3 = 1;
PWM1CON1bits.PMOD4 = 1;

/* PWM CONTROL REGISTER 2 */
/* Updates from Duty Cycle and Period Buffer registers are enabled */
/* Not used */
PWM1CON2bits.OSYNC = 0;
/* Updates to the active PDC registers are synchronized to the PWM time base */
PWM1CON2bits.IUE = 0;
/* Not used */
PWM1CON2bits.SEVOPS = 0;

/* DEAD-TIME CONTROL REGISTER 1 ->Not used */
P1DTCON1bits.DTA = 0;
P1DTCON1bits.DTAPS = 0;
P1DTCON1bits.DTB = 0;
P1DTCON1bits.DTBPS = 0;

/* DEAD-TIME CONTROL REGISTER 2 -> Not used */
P1DTCON2bits.DTS1I = 0;
P1DTCON2bits.DTS1A = 0;
P1DTCON2bits.DTS2I = 0;
P1DTCON2bits.DTS2A = 0;
P1DTCON2bits.DTS3I = 0;
P1DTCON2bits.DTS3A = 0;
P1DTCON2bits.DTS4I = 0;
P1DTCON2bits.DTS4A = 0;

/* FAULT A CONTROL REGISTER -> Disabled */
P1FLTACONbits.FAEN1 = 0;
```

```
	P1FLTACONbits.FAEN2 = 0;
	P1FLTACONbits.FAEN3 = 0;
	P1FLTACONbits.FAEN4 = 0;
	P1FLTACONbits.FLTAM = 0;
	P1FLTACONbits.FAOV1L = 0;
	P1FLTACONbits.FAOV1H = 0;
	P1FLTACONbits.FAOV2L = 0;
	P1FLTACONbits.FAOV2H = 0;
	P1FLTACONbits.FAOV3L = 0;
	P1FLTACONbits.FAOV3H = 0;
	P1FLTACONbits.FAOV4L = 0;
	P1FLTACONbits.FAOV4H = 0;

	/* FAULT B CONTROL REGISTER -> Disabled */
	P1FLTBCONbits.FBEN1 = 0;
	P1FLTBCONbits.FBEN2 = 0;
	P1FLTBCONbits.FBEN3 = 0;
	P1FLTBCONbits.FBEN4 = 0;
	P1FLTBCONbits.FLTBM = 0;
	P1FLTBCONbits.FBOV1L = 0;
	P1FLTBCONbits.FBOV1H = 0;
	P1FLTBCONbits.FBOV2L = 0;
	P1FLTBCONbits.FBOV2H = 0;
	P1FLTBCONbits.FBOV3L = 0;
	P1FLTBCONbits.FBOV3H = 0;
	P1FLTBCONbits.FBOV4L = 0;
	P1FLTBCONbits.FBOV4H = 0;

	/* OVERRIDE CONTROL REGISTER */
	/* Output on PWMx I/O pin is controlled by the PWM generator */
	P1OVDCONbits.POVD1L = 1;
	P1OVDCONbits.POVD1H = 1;
	P1OVDCONbits.POVD2L = 1;
	P1OVDCONbits.POVD2H = 1;
	P1OVDCONbits.POVD3L = 1;
	P1OVDCONbits.POVD3H = 1;
	P1OVDCONbits.POVD4L = 1;
	P1OVDCONbits.POVD4H = 1;

	PWM1CON2bits.UDIS = 1;
	/* PWM DUTY CYCLE REGISTERS */
	PDC1 = 0;
	PDC2 = 0;
	PDC3 = 0;
	PDC4 = 0;
	PWM1CON2bits.UDIS = 0;
}

void pwmEnable (void) {
	/* PWM time base is on */
```

```
	P1TCONbits.PTEN = 1;
}

void encoderConfig (void) {
	T6CON = 0x0002;
	T7CON = 0x0002;
	T8CON = 0x0002;
	T9CON = 0x0002;

	/* Zera os contadores */
	TMR6 = 0x0000;
	TMR7 = 0x0000;
	TMR8 = 0x0000;
	TMR9 = 0x0000;

	/* Seta os períodos */
	PR6 = 0xFFFF;
	PR7 = 0xFFFF;
	PR8 = 0xFFFF;
	PR9 = 0xFFFF;
}

void encoderEnable (void) {
	/* Liga os timers externos */
	T6CONbits.TON = 1;
	T7CONbits.TON = 1;
	T8CONbits.TON = 1;
	T9CONbits.TON = 1;
}

void sampleTimeConfig (void) {
	unsigned long long perCalc;

	perCalc = (FCY*SAMPLE_TIME)/1000;

	T5CONbits.TON = 0; // Disable Timer
	T5CONbits.TCS = 0; // Select internal instruction cycle clock
	T5CONbits.TGATE = 0; // Disable Gated Timer mode
	T5CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
	TMR5 = 0x00; // Clear timer register
	PR5 = perCalc; // Load the period value

	IPC7bits.T5IP = 0x01; // Set Timer5 Interrupt Priority Level
	IFS1bits.T5IF = 0; // Clear Timer5 Interrupt Flag
	IEC1bits.T5IE = 1; // Enable Timer5 interrupt
}

void initSampling (void) {
	T5CONbits.TON = 1; // Start Timer
}
```

```
void convertVel (void) {
	float tensaoRotacao[4];
	float k;
	uint8 i;

	k = (2.0*M_PI*CONSTANTE_ROTACAO)/(60.0*CAIXA_REDUCAO);

	for (i = 0; i<NUM_MOTORES; i++){
		tensaoRotacao[i] = fabs(finalSpeed[i]/k);
		pwmDC[i] = (tensaoRotacao[i]*pwmMax)/(tensaoBat-1.6);
	}
}

void motorDirection (void) {
	/* Motor 0 */
	if (finalSpeed[0] > 0) {
		P1OVDCONbits.POVD1L = 1;
		P1OVDCONbits.POVD1H = 0;
		P1OVDCONbits.POUT1H = 0;
	}
	else if (finalSpeed[0] < 0) {
		P1OVDCONbits.POVD1L = 0;
		P1OVDCONbits.POVD1H = 1;
		P1OVDCONbits.POUT1L = 0;
	}
	else {
		P1OVDCONbits.POVD1L = 0;
		P1OVDCONbits.POUT1L = 1;
		P1OVDCONbits.POVD1H = 0;
		P1OVDCONbits.POUT1H = 1;
	}

	/* Motor 1 */
	if (finalSpeed[1] > 0) {
		P1OVDCONbits.POVD2L = 1;
		P1OVDCONbits.POVD2H = 0;
		P1OVDCONbits.POUT2H = 0;
	}
	else if (finalSpeed[1] < 0) {
		P1OVDCONbits.POVD2L = 0;
		P1OVDCONbits.POVD2H = 1;
		P1OVDCONbits.POUT2L = 0;
	}
	else {
		P1OVDCONbits.POVD2L = 0;
		P1OVDCONbits.POUT2L = 1;
		P1OVDCONbits.POVD2H = 0;
		P1OVDCONbits.POUT2H = 1;
	}
```

```
		/* Motor 2 */
		if (finalSpeed[2] > 0) {
			P1OVDCONbits.POVD3L = 1;
			P1OVDCONbits.POVD3H = 0;
			P1OVDCONbits.POUT3H = 0;
		}
		else if (finalSpeed[2] < 0) {
			P1OVDCONbits.POVD3L = 0;
			P1OVDCONbits.POVD3H = 1;
			P1OVDCONbits.POUT3L = 0;
		}
		else {
			P1OVDCONbits.POVD3L = 0;
			P1OVDCONbits.POUT3L = 1;
			P1OVDCONbits.POVD3H = 0;
			P1OVDCONbits.POUT3H = 1;
		}

		/* Motor 3 */
		if (finalSpeed[3] > 0) {
			P1OVDCONbits.POVD4L = 1;
			P1OVDCONbits.POVD4H = 0;
			P1OVDCONbits.POUT4H = 0;
		}
		else if (finalSpeed[3] < 0) {
			P1OVDCONbits.POVD4L = 0;
			P1OVDCONbits.POVD4H = 1;
			P1OVDCONbits.POUT4L = 0;
		}
		else {
			P1OVDCONbits.POVD4L = 0;
			P1OVDCONbits.POUT4L = 1;
			P1OVDCONbits.POVD4H = 0;
			P1OVDCONbits.POUT4H = 1;
		}
}

void motorVel (void) {
		int i;
		PWM1CON2bits.UDIS = 1;
		PWMDC0 = pwmDC[0];
		PWMDC1 = pwmDC[1];
		PWMDC2 = pwmDC[2];
		PWMDC3 = pwmDC[3];
		PWM1CON2bits.UDIS = 0;

		for (i=0; i<NUM_MOTORES; i++) {
			pwmDCRef[i] = pwmDC[i];
		}
```

```
}

void motorEnable (void) {
	Nop();
	ENABLE_M0 = 1;
	Nop();
	ENABLE_M1 = 1;
	Nop();
	ENABLE_M2 = 1;
	Nop();
	ENABLE_M3 = 1;
}

void motorDisable (void) {
	ENABLE_M0 = 0;
	ENABLE_M1 = 0;
	ENABLE_M2 = 0;
	ENABLE_M3 = 0;
}

void __attribute__((__interrupt__, no_auto_psv)) _T5Interrupt(void) {
/*
	int i;
	float k;
	float velRad[4], encoderCount[4], tensaoRotacao[4], encoderPwm[4], error[4], outP[4], outI[4];
	float previousOutI[4], previousError[4];

	k = (2.0*M_PI*CONSTANTE_ROTACAO)/(60.0);
	encoderCount[0] = ENCODER_M0;
	encoderCount[1] = ENCODER_M1;
	encoderCount[2] = ENCODER_M2;
	encoderCount[3] = ENCODER_M3;

	for (i=0; i<NUM_MOTORES; i++) {
		velRad[i] = (2.0*M_PI*encoderCount[i]*1000.0)/(SAMPLE_TIME*RESOLUCAO_ENCODER);
		tensaoRotacao[i] = fabs(velRad[i]/k);
		encoderPwm[i] = (tensaoRotacao[i]*pwmMax)/(tensaoBat-1.6);

		error[i] = pwmDCRef[i] - encoderPwm[i];
		outP[i] = KP*error[i];
		outI[i] = previousOutI[i] + KP*KI*previousError[i];
		pwmDC[i] = outP[i] + outI[i];
		previousOutI[i] = outI[i];
		previousError[i] = error[i];
	}

	PWMDC0 = pwmDC[0];
	PWMDC1 = pwmDC[1];
	PWMDC2 = pwmDC[2];
	PWMDC3 = pwmDC[3];
```

```
        ENCODER_M0 = 0;
        ENCODER_M1 = 0;
        ENCODER_M2 = 0;
        ENCODER_M3 = 0;
*/
        IFS1bits.T5IF = 0; // Clear Timer5 Interrupt Flag

}

void setMotor (void) {
        motorPinsConfig ();
        pwmConfig ();
        encoderConfig ();
        sampleTimeConfig ();
        encoderEnable ();
        pwmEnable ();
        initSampling ();
}

void updateMotor (void) {
        convertVel ();
        motorDisable ();
        motorDirection ();
        motorVel ();
        motorEnable ();
}
```

geartucanomotor.h

```
#ifndef _GEARTUCANOMOTOR_H_
  #define _GEARTUCANOMOTOR_H_ 1

        #include <p33FJ256MC710A.h>

        #define uint8       unsigned char
        #define uint16      unsigned int

        /* Constants */
        #define M_PI 3.1415926
        #define CONSTANTE_ROTACAO 713.0
        #define CAIXA_REDUCAO 1.0
        #define NUM_MOTORES 4.0
        #define   ROTACAO_MAX 8100.0
        #define   SAMPLE_TIME 10.0
        #define   RESOLUCAO_ENCODER 512.0
        #define   KP 1.6
        #define   KI 0.5
        #define PTMRPRESCALER 1
        #define FPWM 32500
```

```
#define FCY 39613750

/* Motor Enable */
#define ENABLE_M0 PORTEbits.RE9
#define ENABLE_M1 PORTEbits.RE8
#define ENABLE_M2 PORTBbits.RB4
#define ENABLE_M3 PORTBbits.RB5

/* Motor Current Sensor */
#define CURRENT_M0 PORTBbits.RB0
#define CURRENT_M1 PORTBbits.RB1
#define CURRENT_M2 PORTBbits.RB2
#define CURRENT_M3 PORTBbits.RB3

/* Motor Encoder */
#define ENCODER_M0 TMR7
#define ENCODER_M1 TMR6
#define ENCODER_M2 TMR9
#define ENCODER_M3 TMR8

/* Duty Cicle Registers */
#define PWMDC0 PDC1
#define PWMDC1 PDC2
#define PWMDC2 PDC3
#define PWMDC3 PDC4

/* Functions */
void setSpeed(uint8 motor, float speed);
void motorPinsConfig (void);
void pwmConfig (void);
void pwmEnable (void);
void encoderConfig (void);
void encoderEnable (void);
void sampleTimeConfig (void);
void initSampling (void);
void convertVel (void);
void motorDirection (void);
void motorVel (void);
void motorEnable (void);
void motorDisable (void);
void setMotor (void);
void updateMotor (void);
#endif
```

ANEXOS

A. Descritivo de como exportar arquivos do Eagle PCB para Fresa LPKF

Fonte: http://optics.eee.nottingham.ac.uk/eagle/eagle2lpkf_at_eee.html

Eagle PCB -> LPKF Milling Machine Mini-How-To

Instructions for users in EEE, University of Nottingham

EAGLE 4.09rl (Linux) CircuitCAM 3.0 (99) and BoardMaster 3.0 (45)

Steve D. Sharples

1. SPECIFIC ISSUES TO EEE

1a. Design rules

The Eagle software comes with a set of default physical design rules... such as how close together tracks may be, how close the copper layer can be to the edge of the board etc. Unfortunately, these are just a little bit too good for the milling machine within the School, so a set of design rules suitable for in-house board-making has been designed. **It is important that the "in-house" design rules are used, rather than the "standard" rules, otherwise your circuit may not be milled correctly.** In practice, this means (for each project that you work on) *before* you do any routing (especially auto-routing) the following steps must be taken:

- once you have "created a board from schematic", click on the "DRU" button
- a window will pop up, it will probably say "EAGLE Design Rules"
- click on the "Load" button
- navigate your way to the following directory: /home/share/eagle/local_dru/
- double-click on "steve.dru" (right click to download)
- click on "Ok"

1b. Extra layers (for the milling machine)

As explained above, milling a PCB is a slightly different process to the normal process of making a PCB. in particular:

- the board starts out as blank copper, and the machine mills out ***either:***
  - a thin line *around* the copper tracks/pads (this is the default behaviour) or:
  - all areas *other than* the copper tracks/pads
- there currently no facility to print text onto PCBs with a pen... any text you want must be milled into the copper. You wouldn't normally want to mill around each letter of the text (as with tracks), you would normally just want to mill out the letters of the text.

Because of these differences and extra choices, it is necessary in your design to **use special layers on your board layout** if you want to:

- add any text
- have areas of the board where the tracks are not only isolated, but there is also no extra copper at all

It is important to follow the following conventions, otherwise your boards may not be exported and manufactured correctly

Layer conventions used

Layer	Name	Colour	There by de-fault?	Used for...
41	tRestrict	Diagonal red s-tripes	Yes	Areas on Top layer that you wish to be completely milled out *except for* tracks, pads and vias
42	bRestrict	Diagonal blue stripes	Yes	Areas on Bottom layer that you wish to be completely milled out *except for* tracks, pads and vias
117	tText	Usually white	No	For text on Top layer... will be milled out with 0.2mm cutter
118	bText	Usually white	No	For text on Bottom layer... will be milled out with 0.2mm cutter

Please note the following, which will affect the results given by any Design Rule check:

- Any areas of tRestrict or bRestrict that go over tracks, vias or bads will be picked up as a DRC error. For this reason it is recommended to place these areas once all "genuine" DRC errors have been sorted out
- The tText and bText layers are not checked for DRC errors. Since any objects (text, lines) on these layers will be milled out, make sure that none of your text overlays any tracks on the same side.
- Remember to "mirror" any text you put on the bText layer... otherwise it will appear back-to-front on your completed design.

1c. Shared files in Eagle: /home/share/eagle/

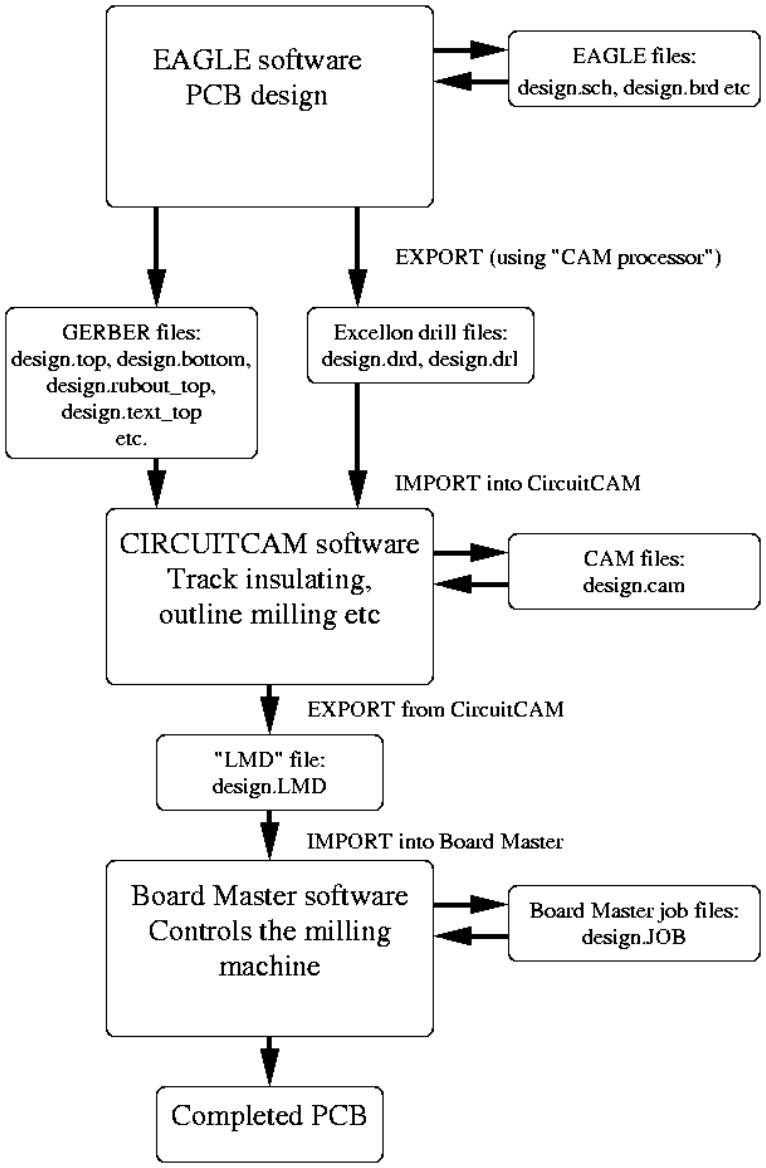
We have set up a system whereby circuits, component libraries, script files (used to perform useful repetetive tasks) developed by one person can be shared amongst everyone else. It involves the use of a *shared directory* which is located at /home/share/eagle/. Within this are sub-directories...

- **local_completed_projects** - any circuits that you've made and completed, you are free to put here so that other people can see them and perhaps learn from them
- **local_dru** - local design rules (see section 1a above)
- **local_lbr** - library elements (ie components) developed "in-house" that didn't come supplied with the Eagle software. In particular, we've developed quite a few R.F. Minicircuits components, added some additional connectors, op-amps, encoders/decoders etc. If you develop any library elements of your own, it would be appreciated if you would place them in this directory. It's possible to change the permissions such that other users may read them (ie use them in their designs) but not alter them (potentially mucking up your beautiful components)... see Steve on how to do this if you don't already know. Any library elements added to this directory will automatically be added to the overall component library (so when you "add" a component, anything in /home/share/eagle/local_lbr/ will be on the list).

2. THE EXPORT/IMPORT PROCEDURE OVERVIEW

You'd think that everyone would use the same kind of file standard, wouldn't you? Well, they kind of do, and they kind of don't. Each PCB design package uses its own file standard, which contains much more informations about the design than just physically where the tracks are laid... since different design packages are capable of different things, this makes sense. Once you have finalised your design process, you just want to tell the board manufacturer where to drill the holes and place the copper. For this you use *Excellon drill files* and *Gerber files*.

All board manufacturers accept these file formats, and they *should* adher to a strict standard. What they do with them and how they process them is down to the individual board maker. In the case of boards made in EEE using the milling machine, the Gerber and Excellon files are imported into a program called **CircuitCam** which will then produce files that consist of routes for the milling machine to mill out (to isolate tracks), places to drill holes, drill sizes etc. Yet another file standard... anyway, it's probably worth looking at the following picture:



Make sense yet? Lots of different stages, involving 2 or 3 different programs on 2 operating systems. You'll be fine....

3. EXPORTING FILES FROM EAGLE

- start Eagle
- open up your project... board and schematic
- in the "Board" window, type: **`run drillcfg`**
- a window pops up, asking for units... default is mm, **select INCH.**
- another window will show you the file it's created... something like:

```
T01 0.024in
T02 0.031in
T03 0.032in
T04 0.040in
```

- click "Ok" it'll suggest "design.drl" as the output filename.
- click "Save"
- in the "Board" window, click on the "CAM" icon (next to the printer icon)
- click File -> Open -> Job...

- double-click "EEE_milling.cam" (right click to download)
- click "Process Job" - it is possible during this process that a warning window saying "No layers active!" will pop up... this means that you have not used all the layers you could do. This is normal, if you (eg) don't have text on either top or bottom side and so have not added the tText or bText layers. Click "Ok" to each of these warnings.

The following GERBER files will have been created (assuming your design is called "design"):

- design.top
- design.bottom
- design.outline
- design.text_bottom
- design.text_top
- design.rubout_top
- design.rubout_bottom

As you can see, there is one Gerber file per layer. The following Excellon drill files will have been created:

- design.drl - list of tools (ie drill sizes)... also known as the "Rack file"
- design.drd - list of positions of holes for each drill size

These are the files you need to import into CircuitCAM (or give to the technicians on the 9th floor to process). So you'll be wanting to put them on a floppy disc....

- open up a "konsole"/"shell" window
- put a floppy disc in the disc drive
- change to the directory where your files are stored, eg:
  `cd ~/eagle/my_current_design/`
- type: `eagle2floppy`
- once told to do so, remove the disc from the drive

4. IMPORTING THE FILES INTO CIRCUITCAM

4a. The Gerber Files (Top, Bottom and BoardOutline etc)

- start CircuitCam
- click File -> Import
- find your files, select "design.bottom" ... click "Ok"
- CircuitCam should recognise it as "Gerber-X"
- in the "Aperture/Tool list" box, type "bottom" (or something)
- Select "Bottom Layer" ... click "Ok"
- Bottom Layer tracks and pads should appear (in green)
- click File -> Import
- select "design.top" ... click "Ok"
- CircuitCam should recognise it as "Gerber-X"
- in the "Aperture/Tool list" box, type "top" (or something)
- Select "Top Layer" ... click "Ok"

- Top Layer tracks and pads should appear (in red)
- import the BoardOutline layer in the same way as the above 2 layers
- the BoardOutline should appear (in yellow)
- if you created any text layers, or any rubout layers, import them into CircuitCAM the same way

4b. The Excellon Drill Data

- click File -> Import
- select "design.drl" ... click "Ok"
- click the "Apertures/Tools select" button
- select "Eagle_Excellon.txt" from the "Aperture/Tool Template" list
- in the "Aperture/Tool list" box, type "drills" (or something)
- click "Ok" - a window should pop up saying "4 apertures were recognised" (or something) ... click "Ok"
- click File -> Import
- select "design.drd" ... click "Ok"
- CircuitCam should recognise it as "Excellon"
- in the "Aperture/Tool list" box, SELECT "drills" FROM THE LIST
- Select "DrillPlated" (or DrillUnplated for single-sided boards) layer... click "Ok"
- The drill holes should appear (in blue) - note that if you have selected the destination layer as "DrillUnplated" you won't be able to see them... change the display order using View -> Layers...

5. INSULATING THE COPPER

- click Edit -> Insulate...
- select "Bottom layer"
- click "Run"
- click Edit -> Insulate...
- select "Top layer"
- click "Run"

6. BOARD CUT-OUT

The aim of this bit is to create a "Cut-out" around the outside of the board outline. A large, 2mm diameter milling bit will mill around your board, to make it the correct size. You first need to run the "contour router," by telling which layer contains the Board Outline. It is, unsurprisingly, the "BoardOutline" layer...:

- click Edit -> Contour Routing...
- select "Source layer" ... "BoardOutline"
- select "Destination layer" ... "CuttingOutside"
- tool should be "Contour Router 2.0mm long"
- click "Run"

You now need to put some gaps in the large gray rectangle, so that the milling machine does not mill out your board completely (it can't do this, otherwise it would fly off the milling machine)...:

- click near the grey CuttingOutside rectangle to select it
- use the "+" and "-" keys on the keypad to move the "*" around the rectangle
- press "Ctrl-g" to make a gap in the middle of one side
- use the "+" and "-" keys to move the "*" around to the other side
- press "Ctrl-g" to make a gap in the middle of the other side
- de-select everything (right-click -> Cancel)

We now need to **ERASE the BoardOutline layer**. (If we don't do this, the milling machine will attempt to mill out this layer.)

- click on the INSIDE of the BoardOutline layer (yellow) to select a part of it
- if you selected the correct object (the outline should go pale yellow) then click the "select layer" button (looks like an arrow pointing into a parallelogram) to select ALL objects in the BoardOutline layer
- once selected, ERASE the layer (using the "X" button) - all the yellow should go

7. SAVING AND EXPORTING

- click File -> Save As...
- save as "design.cam" ... MAXIMUM 8 LETTERS (BoardMaster is very old)
- click File -> Export -> LPKF CircuitBoardPlotter... saves "design.LMD"

The 9th-floor workshop technicians need either the "design.cam" or the "design.LMD" files... the design.cam file is larger (so may not fit on a floppy disc) but it does allow them to check the design in CircuitCAM. The .LMD file can only be checked in BoardMaster.

8. IMPORTING BOARD INTO BOARDMASTER

- start BoardMaster
- click File -> Import -> LMD/LPR...
- select your file "design.LMD" ... click "Ok"
- check everything's there and ok (view both sides, select "Real Tools" etc)

Any comments, or even if you read this Mini-How-To and actually use it, please mail me (remove the no.spams). Also mail me if I've failed to reference a source of knowledge for anything, I'll be happy to correct any omissions.

Hope it's useful!