UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Kevin Borges de Souza Okamura

Desenvolvimento de Interface Gráfica para controle de plataforma didática

São Carlos

2018

Kevin Borges de Souza Okamura

Desenvolvimento de Interface Gráfica para controle de plataforma didática

Monografia apresentada ao Curso de Engenharia Mecânica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Mecânico.

Orientador: Prof. Dr. Thiago Boaventura Cunha

São Carlos

2018

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Borges de Souza Okamura, Kevin
BO41d Desenvolvimento de Interface Gráfica para controle de Plataforma Didática / Kevin Borges de Souza Okamura; orientador Thiago Boaventura Cunha. São Carlos, 2018.

Monografia (Graduação em Engenharia Mecânica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2018.

1. Interface. 2. ROS. 3. Arduino. 4. RTOS. 5. PID. 6. Controle. 7. Plataforma Didática. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE AVALIAÇÃO

Candidato: Kevin Borges de Souza Okemura

Título: Desenvolvimento de Interface Gráfica para controle de plataforma didática

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

BANCA EXAMINADORA

Professor Thiago Boaventura
(Orientador)

Nota atribuída: 9,5 (nove e meio)

(assinatura)

Professor Glauco Caurin

Nota atribuída: 9,5 (nove e meio)

(assinatura)

Professor Daniel Varela Magalhães

Nota atribuída: 9,5 (NOVE E MEIO)

(assinatura)

Média: 9,5 (NOVE E MEIO)

Resultado: APROVADO

Data: 07 / 12 / 2018.

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

SIM ☒ NÃO ☐ Visto do orientador

Dedico este trabalho primeiramente a Deus, por ser essencial em minha vida,
ao meu pai Marcio, a minha mãe Marcia, as minhas irmãs Kira e Keeva
e ao professor Thiago, pela paciência na orientação e incentivo que tornaram possível
a conclusão desta monografia.

AGRADECIMENTOS

A Deus, por ter me dado saúde e força para superar as dificuldades.

Ao Prof. Dr. Thiago, pelo suporte e orientação dentro deste projeto que me proporcionou o desenvolvimento de conhecimentos adicionais a minha formação.

Ao Grupo SEMEAR EESC USP e seus membros, por todo o suporte fornecido ao longo desta jornada, sendo através de equipamentos, materiais e conhecimento.

Aos técnicos Jair Diego Antonietti e Sergio Donizete Carvalho Ferreira do Laboratório de Dinâmica da Escola de Engenharia de São Carlos, que contribuíram com sua atenção, força e conhecimento no desenvolvimento deste projeto.

À funcionária Jiuliana Pizzaia Perandré do Prédio Administrativo do Departamento de Engenharia Mecânica, que se esforçou para conseguirmos realizar as compras dos equipamentos.

Aos meus pais, pelo amor, incentivo e orientação em todos os momentos.

RESUMO

Okamura, K. B. S. **Desenvolvimento de Interface Gráfica para controle de plataforma didática**. 2018. 101p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

Motores elétricos, principalmente de indução, são muito utilizados dentro da área industrial, entretanto, um uso inadequado e ineficiente destes aparelhos pode trazer prejuízos estruturais e econômicos. Para isso, o uso de inversores de frequência tem sido cada vez mais usado para tornar o processo mais viável. Com o intuito de familiarizar os estudantes dos cursos de Engenharia Mecânica e Engenharia Mecatrônica da Escola de Engenharia de São Carlos (USP) com tais práticas industriais, este trabalho de conclusão de curso tem como objetivo o desenvolvimento de uma interface gráfica para uma futura bancada didática para fins experimentais, onde os estudantes terão a possibilidade de controlar um inversor de frequência e, consequentemente, um motor de indução através de comandos enviados pelo computador. Para isso, este projeto partiu da continuidade do TCC desenvolvido pelo Marcio Luis (SILVA, 2017), com foco no desenvolvimento da parte de controle através de uma interface gráfica de fácil acesso ao usuário.

Palavras-chave: Interface Gráfica. ROS. Arduíno. Controle. RTOS. PID. Malha-fechada. RS-485. Modbus RTU. Frequência. Velocidade. Inversor de Frequência. Encoder. Motor de Indução. Bancada Didática. Plataforma linear.

ABSTRACT

Okamura, K. B. S. **Graphic Interface Development to control didactic platform**. 2018. 101p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

Electric motors, mainly induction motors, are widely used in the industrial area, however, an inadequate and inefficient use of these devices can bring structural and economic damages. For this, the use of variable-frequency drives has been increasingly used to make the process more feasible. In order to familiarize the students of the Mechanical Engineering and Mechatronics Engineering courses at University of Sao Paulo (Sao Carlos campus) with such industrial practices, this final paper has the objective of developing a graphical interface for a future didactic bench for experimental purposes, where the students will be able to control a variable-frequency drive and, consequently, an induction motor hrough commands sent by the computer. For this, the project started from the continuity of the TCC developed by Marcio Luis (SILVA, 2017), focused on the development of the control part through a graphical interface of easy access for the user.

Keywords: Graphic Interface. ROS. Arduino. Control. RTOS. PID. Closed Mesh. RS-485. Modbus RTU. Frequency. Velocity. Variable-frequency drive. Frequency Inverter. Encoder. Induction Motor. Didatic Bench. Linear platform.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

EESC	Escola de Engenharia de São Carlos
TCC	Trabalho de Conclusão de Curso
USP	Universidade de São Paulo
CA	Corrente alternada
MME	Ministério de Minas e Energia
IHM	Interface Homem-Máquina
CPU	*Central Processing Unit* ou Unidade Central de Processamento
CLP	Controlador lógico programável
CC	Corrente Contínua
cv	Cavalo-vapor (unidade de potência)
GUI	Interface Gráfica do Utilizador ou *Graphical User Interface*
RTU	*Remote Terminal Unit*
TTL	*Transistor-Transistor Logic*
RTOS	*Real Time Operating Systems*
UART	*Universal asynchronous receiver/transmitter*
RTOS	*Real Time Operating Systems*
PID	Proporcional Integral Derivativo
IDE	Ambiente de Desenvolvimento Integrado ou *Integrated Development Environment*
LED	Diodo Emissor de Luz ou *Light Emitting Diode*
PWM	Modulação de Largura de Pulso ou *Pulse Width Modulation*

SUMÁRIO

1 INTRODUÇÃO E REVISÃO BIBLIOGRÁFICA

A tecnologia de corrente alternada (CA) surgiu à partir de experimentos desenvolvidos por Michael Faraday e Joseph Henry, por volta de 1830-1831, onde foi descoberto que a mudança de campo magnético pode induzir corrente elétrica em um circuito (BEN-MENAHEM, 2009). E, após diversas mutações e avanços nesta área, os motores elétricos de corrente alternada têm sido cada vez mais utilizados em aplicações industriais.

No Brasil, de acordo com Ministério de Minas e Energia (MME), a indústria consome 43,7% de toda energia elétrica nacional e a força motriz em operação usa 68% dessa energia elétrica. Sendo assim, contata-se que aproximadamente 30% de toda a energia elétrica do Brasil é consumida por motores elétricos, sendo grande parte, motores de indução trifásicos (GOVERNO DO BRASIL, 2015). Diante desta constatação, o Brasil, a exemplo de outras nações, estabeleceu legislações e programas específicos, de modo a regular e incentivar ações que promovam o aumento de eficiência de processos, motores, máquinas e equipamentos; podendo citar, como exemplo, a troca de motores de baixa eficiência por motores de indução trifásicos.

É importante ressaltar também o fato de que um motor de indução transforma em energia mecânica, em média, 85% de toda a energia elétrica que recebe e que os 15% restantes são desperdiçados, sendo assim o acionamento elétrico de máquinas um assunto de extraordinária importância no que se refere a economia de energia. Outro fator agravante que impacta em maiores custos é o baixo fator de proteção contra picos na rede elétrica e a elevada taxa de manutenção dos equipamentos mecânicos devido a partidas bruscas desses motores.

Com o intuito de promover eficiência e segurança para a proteção de motores,indústrias desde o setor sucroalcooleiro até o setor de bens de consumo têm escolhido cada vez mais usar inversores de frequência em seus sistemas de potência (CONTROLE & INSTRUMENTAÇÃO, 2001). As principais vantagens de seu uso corresponde à redução de energia elétrica através do controle da velocidade do motor, eliminação de acionamentos bruscos, aumento da vida útil do sistema através do oferecimento de maior proteção aos motores, baixo custo de manutenção do próprio inversor e automatização de processos através da possibilidade de programação de eventos automáticos.

1.1 Revisão da bibliográfica

Esta monografia tem como intuito a retomada do trabalho proposto pelo ex-aluno Marcio da Silva (SILVA, 2017), com ênfase na implementação do controle do sistema à partir da integração eletrônica com a interface gráfica. Nas seguintes sub-seções, serão abordadas informações gerais sobre inversor de frequência e motor de indução, tópicos os quais foram ênfase do trabalho apresentado pelo Marcio. O intuito é trazer informações relevantes, já apresentadas, que

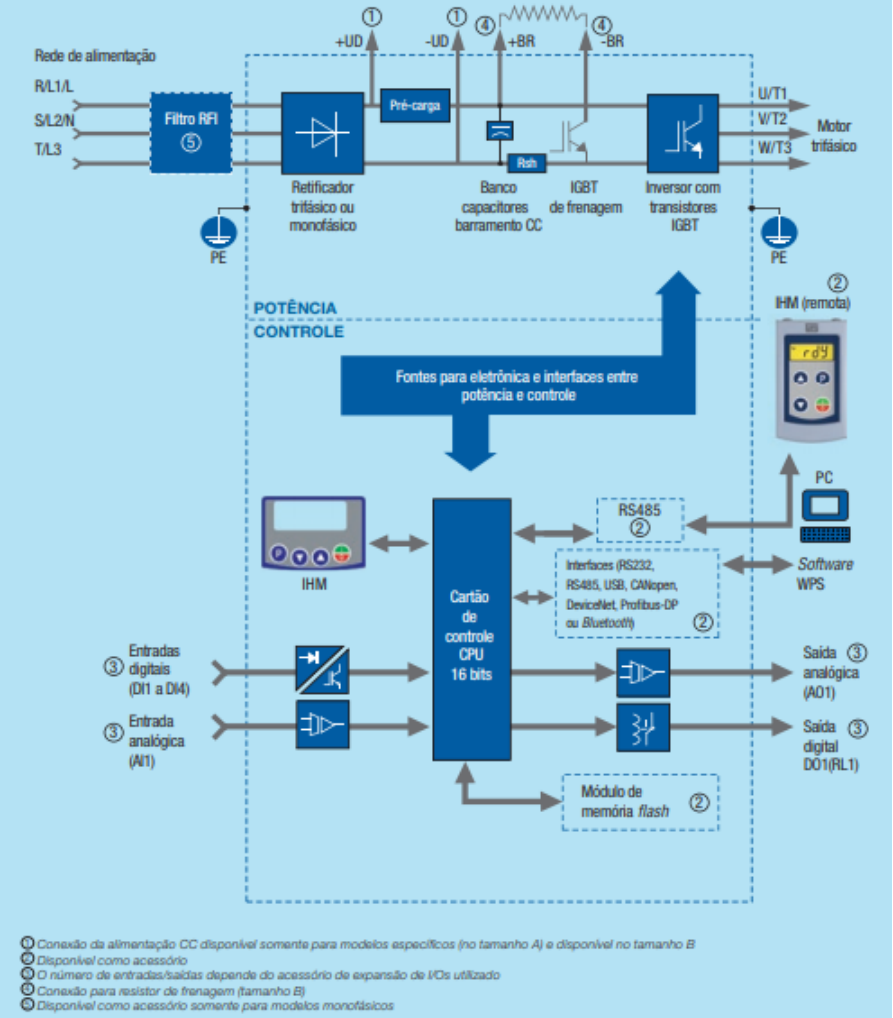
auxiliaram no desenvolvimento do projeto.

1.1.1 Inversor de Frequência

Um inversor de frequência é um tipo de controlador de velocidade ajustável usado em sistemas de acionamento eletromecânicos para controlar a velocidade e o torque do motor CA variando a frequência e a tensão de entrada do motor (SISKIND, 1963).

Resumidamente, o inversor de frequência apresenta os seguintes blocos internos: bloco de conversão CA-CC-CA, bloco de interfaces (comunicação serial e sinais analógicos e digitais), o bloco IHM (Interface Homem-Máquina) e o bloco da CPU (*Central Process Unit*), apresentados na Figura 1.

Figura 1 – Bloco-diagrama do Inversor de Frequência WEG CFW300 contendo sistema de potência e de controle

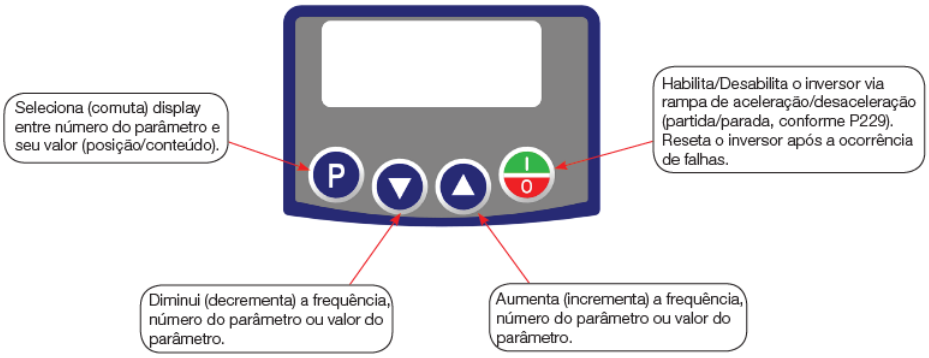


Fonte: WEG (2016a)

Através da IHM é possível o comando do inversor, a visualização e o ajuste de todos os

parâmetros. A IHM, especificamente para o caso do inversor previamente selecionado WEG CFW300, apresenta as seguintes funções:

Figura 2 – Teclas da IHM



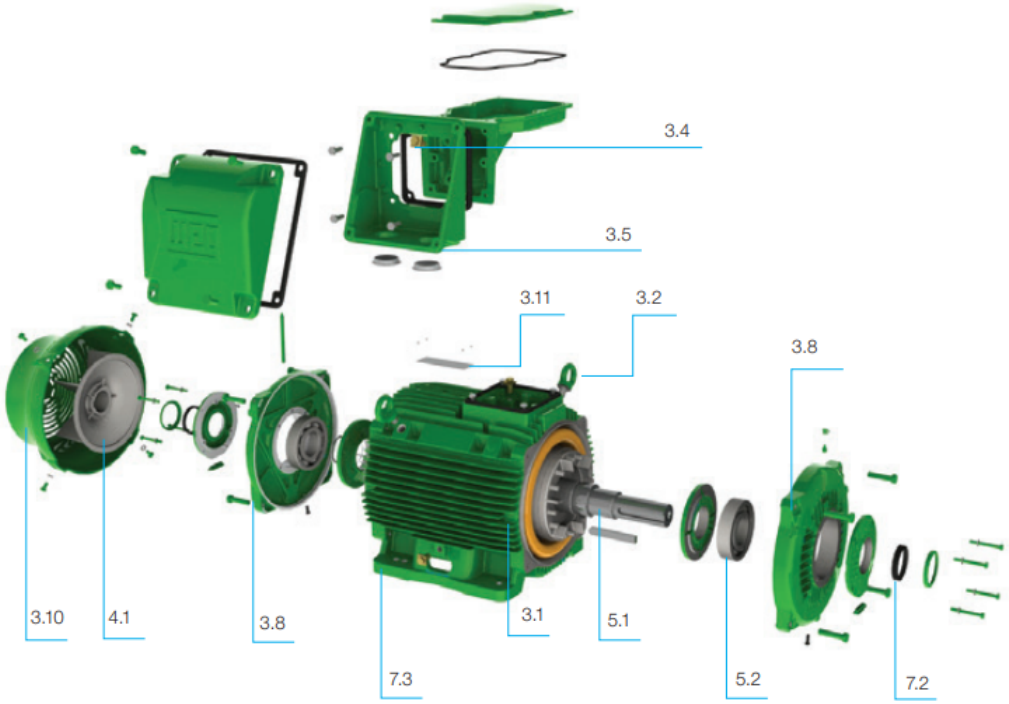
Fonte: WEG (2016a)

1.1.2 Motor de indução

Um motor de indução ou motor assíncrono é um motor elétrico CA no qual a corrente elétrica no rotor necessária para produzir torque é obtida por indução eletromagnética do campo magnético do entreferro (IEC 60050, 1990-10). Um rotor do motor de indução pode ser do tipo rotor bobinado ou do tipo gaiola de esquilo (rotor curto-circuitado).

A grande maioria dos motores de indução trabalha com rotores tipo gaiola de esquilo, que possui este nome em função do formato do rotor. O motor do tipo gaiola de esquilo é montado sobre um eixo que gira dentro do campo magnético girante suportado por rolamentos instalados nas extremidades do eixo (CARVALHO, 2006). Instalada no eixo, na parte traseira do motor, geralmente encontram-se uma ventoinha, que direciona ar entre as aletas na carcaça do motor, ajudando a resfriá-lo. A Figura 3 mostra o aspecto construtivo especificamente para o motor WEG W22, modelo utilizado na bancada didática em questão.

Figura 3 – Aspecto construtivo do motor WEG W22



Fonte: WEG (2017)

E a Tabela correspondente é apresentada na Figura 4.

Figura 4 – Tabela referente à Figura 3

Item	Componente	Página
3.1	Carcaça	9
3.2	Olhais	10
3.4	Terminais de aterramento	10
3.5	Caixa de ligação	10
3.8	Tampas	12
3.10	Tampa defletora	12
3.11	Placa de identificação	12
4.1	Sistema de ventilação	13
5.1	Eixo	14
5.2	Rolamentos	15
7.2	Vedação	19
7.3	Pintura	19

Fonte: WEG (2017)

No motor de indução a corrente alternada é fornecida diretamente ao estator, ao passo que o rotor recebe a corrente por indução, como em um transformador, a partir do estator. Quando o motor é alimentado por uma fonte trifásica equilibrada, um campo magnético é produzido no entreferro girando na velocidade síncrona. Esta velocidade síncrona depende do número de polos do estator e da frequência imposta no estator do motor de indução (JR; UMANS; FITZGERALD, 2006).

Quando o motor está parado, a frequência das correntes que surgem nos enrolamentos do rotor é idêntica à frequência das correntes do estator. A partir do momento que o motor acelera, a frequência das correntes do rotor diminui, de tal forma que sob condições de carga nominal ela é de apenas uma pequena parcela do estator (tipicamente de 2 a 10%) (JR; UMANS; FITZGERALD, 2006). Por outro lado, a rotação mecânica é muito próxima da velocidade com que o campo magnético do estator gira, chamada de velocidade síncrona n_s [*rpm*], que é dada por:

$$n_s = \frac{120 \times f_e}{p} \tag{1.1}$$

onde,

f_e frequência de operação imposta no estator do motor de indução [*Hz*]

p número de polos da máquina [-]

A velocidade do rotor em *rpm* pode ser expressa em termos da velocidade síncrona e do escorregamento como segue:

$$n = (1 - s)n_s \tag{1.2}$$

A diferença entre a velocidade síncrona e a do rotor é referida comumente como o escorregamento do rotor. Neste caso, o escorregamento do rotor é n_s - n, medido em rotações por minuto (*rpm*). O escorregamento é expresso mais usualmente como sendo uma fração da velocidade síncrona e é denominado de escorregamento fracionário, denotado por s e dado por:

$$s = \frac{n_s - n}{n_s} \tag{1.3}$$

O movimento relativo entre o fluxo do estator e os condutores do rotor induz tensões de frequência f_r, denominada de frequência de escorregamento e dada por:

$$f_r = s \times f_e \tag{1.4}$$

Assim, o motor de indução se comporta similarmente como um transformador, mas apresenta a característica adicional da transformação de frequência produzida pelo movimento relativo entre os rolamentos do estator e do rotor.

1.2 Objetivo

Tendo em vista a importância mundial das condições de uso de motores de indução trifásicos e do uso de inversores de frequência para melhoria da eficiência destes, assim como a forte presença destes equipamentos nas indústrias e tecnologias modernas, torna-se necessário um contato mais relevante dos alunos da Escola de Engenharia de São Carlos (EESC-USP) com estas tendências com o objetivo de melhor prepará-los para futuras aplicações profissionais.

Um outro fator importante para a escolha deste tipo de motor em aplicação de precisão é o estudo de possíveis extensões para seu uso, além dos industriais e domésticos. Para tais aplicações, são indicados outros tipos de motores, como motores de passo, que já são utilizados comumente na atualidade. Mas estes não serão abordados neste trabalho.

Conforme foi mencionado anteriormente, esta monografia tem como intuito a retomada do trabalho proposto pelo ex-aluno Marcio (SILVA, 2017), porém com um foco voltado para a produção de uma interface gráfica, que permitirá o usuário controlar os parâmetros de entrada para o sistema assim como analisar os dados obtidos.

1.3 Trabalhos semelhantes já desenvolvidos

O projeto global em questão não é o primeiro que é voltado para a construção de uma bancada didática que envolve o controle de um motor trifásico através de um inversor de frequências. No mercado e em projetos acadêmicos foram encontradas diversas outras bancadas relacionadas que propõem os mesmos objetivos: proporcionar ao estudante um maior contato com tecnologias utilizadas na indústria, isto é, motores CA e dispositivos para acionamento e controle de motores trifásicos, como inversores de frequência.

Foram encontrados diversos trabalhos em cima deste assunto, entretanto, foram escolhidos os três projetos mais relevantes para comparação com a bancada didática proposta nesta monografia.

1.3.1 Bancada Posicionamento Linear

Esta bancada ((AUTTOM, 2016)) é formada por uma estrutura metálica que aloja um sistema de fuso para exercer o posicionamento linear de um disco através da rotação de três motores em questão: motor assíncrono trifásico (de indução, no caso), motor de passo e servomotor. Seu controle é realizado por um inversor de frequências, que pode ser programado via IHM ou através de um Controlador Lógico Programável (CLP). Além disso, o projeto conta

com dispositivos e softwares para parametrizações adicionais, assim como um indicador digital de posição que fecha malha com o restante do sistema para melhorar o posicionamento linear.

Segundo o fabricante Auttom, a bancada ainda permite a criação de redes de comunicação entre os equipamentos presentes na bancada. A Figura 5 mostra o arranjo da bancada disponibilizada no site do fabricante (AUTTOM, 2016).

Figura 5 – Bancada Posicionamento Linear Auttom

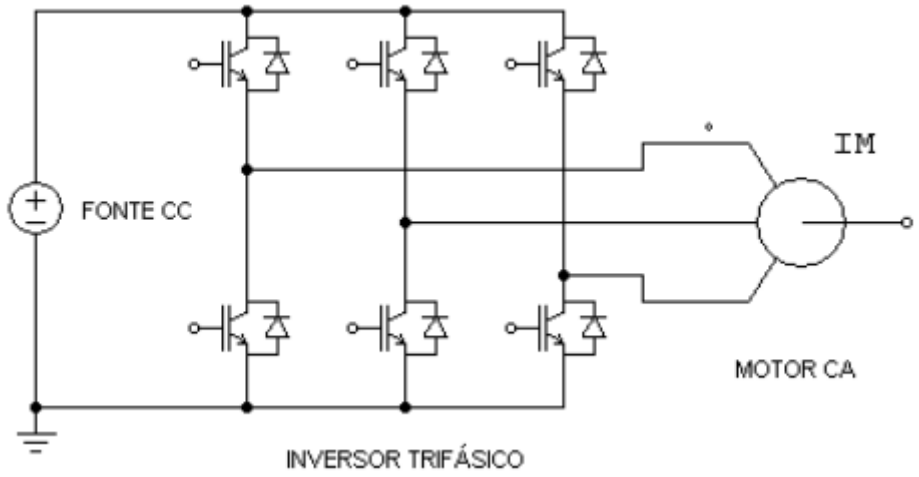
Fonte: Auttom (2016)

1.3.2 Bancada didática para acionamento e controle de motores CA controlada por DSC programado em ambiente *MATLAB/Simulink*

Neste trabalho (SILVA NEWTON ; GAINO (2014)) é apresentado uma bancada didática para acionamento e controle de motores CA (motor de indução, no caso) onde é possível desenvolver, implementar e estudar técnicas de controle e acionamento de máquinas através da programação do processador de um inversor trifásico comandado por sinais gerados por este, além da realização de aulas praticas de laboratório.

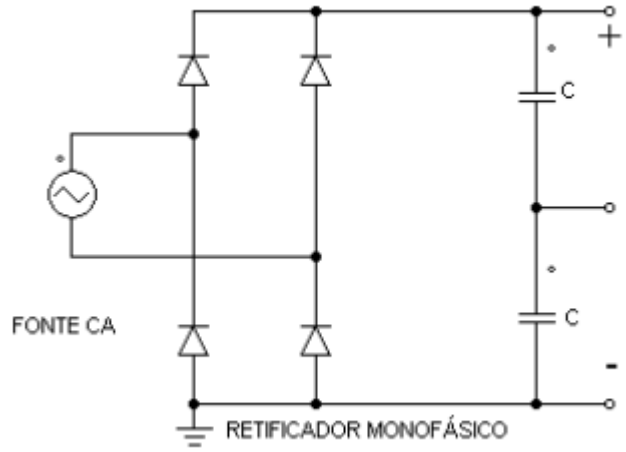
A alimentação do motor trifásico é realizada através de um inversor trifásico, cujo esquemático é apresentado na Figura 6, que por sua vez, é alimentado por uma fonte Corrente Contínua (CC), obtida através de um circuito de retificação do sinal da rede, cujo esquemático é dado na Figura 7.

Figura 6 – Esquemático de um circuito de inversor trifásico



Fonte: SILVA NEWTON ; GAINO (2014)

Figura 7 – Esquemático de um Retificador Monofásico

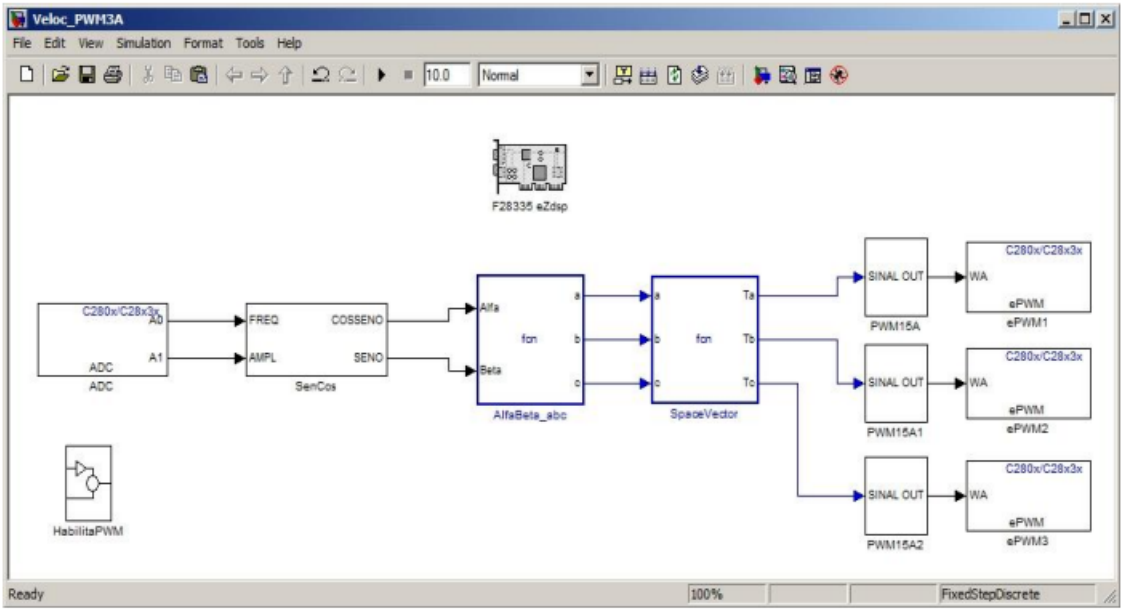


Fonte: SILVA NEWTON ; GAINO (2014)

É importante ressaltar também que o processador (DSC) é o responsável pela síntese dos sinais de comando para o acionamento da chaves do inversor. Através da programação dele, é possível implementar diferentes técnicas de controle para aplicação no motor CA. E, no caso do processador utilizado neste projeto, este possui uma biblioteca no ambiente do Simulink para configuração de ajustes do seu *hardware* permitindo sua programação por esse ambiente do *Matlab*. A Figura 8 mostra um exemplo de programação do projeto.

Por fim, a Figura 9 e Figura 10 apresentam a visão geral da bancada didática produzida,

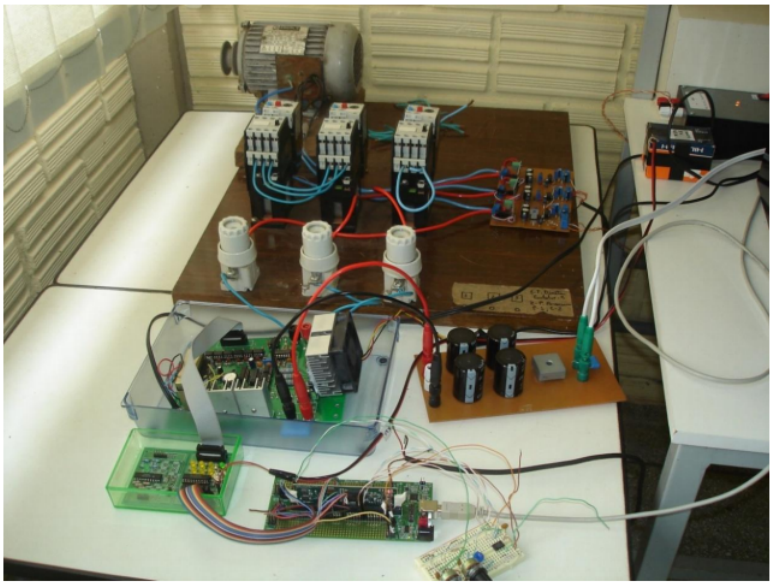
Figura 8 – Exemplo de programação no ambiente Simulink do Matlab



Fonte: SILVA NEWTON ; GAINO (2014)

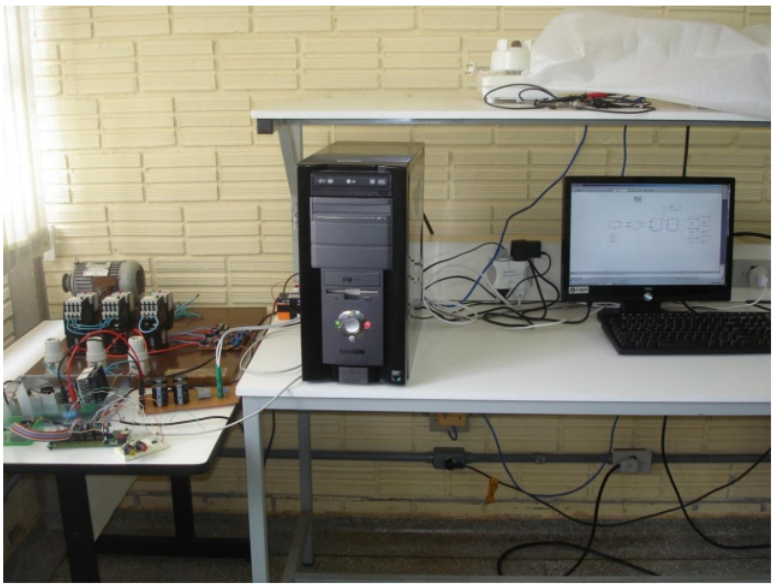
com todos os elementos físicos citados anteriormente.

Figura 9 – Estrutura física da bancada didática



Fonte: SILVA NEWTON ; GAINO (2014)

Figura 10 – Visão geral da bancada didática



Fonte: SILVA NEWTON ; GAINO (2014)

1.3.3 Bancada Didática Inversor e Motor de Indução

Já no caso da bancada ((SOMA, 2016)) referente a este projeto, assim como no caso anterior, também não temos um sistema de movimentação linear para averiguar posição da plata-

forma assim como não há a implementação de um sistema de malha fechada para comparação de posição. A bancada é composta por um motor trifásico de indução com rotor tipo gaiola de esquilo e potência de 0,25 cv. Seu controle é feito por um inversor de frequência com módulo de controle via IHM ou remoto, onde é feito a configuração dos parâmetros do motor via interface de programação no computador.

Os experimentos propostos pela bancada (Figura 11) propõem o estudo do comportamento do motor de indução trifásico em diferentes situações de velocidade, estudo da aplicação dos variadores de velocidade para motores trifásicos de indução e propõem a integração do inversor em rede industrial.

Figura 11 – Bancada Didática Inversor Motor de Indução SOMA

Fonte: Soma (2016)

1.3.4 Comparações

Com relação aos outros projetos, pode-se estabelecer elementos de comparação entre os apresentados acima e ao qual se refere nesta monografia. O parâmetros de comparação estão relacionados com os dispositivos presentes na bancada e as formas de controle propostas.

1.3.4.1 Dispositivos de potência

Esta bancada didática apresenta apenas um dispositivo de potência, sendo o motor de indução trifásico com rotor tipo gaiola de esquilo e potência de 0,25 cv. Observa-se que este é o mesmo tipo de motor proposto pela SOMA (SOMA, 2016) e, a princípio, pela tese (SILVA NEWTON ; GAINO, 2014), onde foi especificado motor CA trifásico de indução.

Já comparando com o dispositivo da bancada da Auttom (AUTTOM, 2016), essa bancada permite o acoplamento de três tipos de motores, sendo um deles um motor assíncrono trifásico; enquanto que para esta, em tese, há apenas a possibilidade de acoplar um motor.

1.3.4.2 Controle do Motor

No projeto atual, temos que o controle do motor é feito através do Inversor de frequência da WEG própria para o motor do mesmo fabricante. Já o motor dos outras bancadas são controladas, respectivamente, por: inversor de frequência, placa de inversor trifásico e também inversor de frequência. Os modelos dos inversores não são especificados exatamente no site do fabricante, por isso o modelo não foi levado em consideração para comparação.

1.3.4.3 Parametrização dos dados

Dentre todas as bancadas apresentadas, todas apresentam um sistema de *input* e *output*, isto é, meios de definição e escolha dos parâmetros de controle do motor. No caso da primeira bancada (AUTTOM, 2016), a parametrização é feita através da IHM externa ao inversor de frequência por comandos digitais e analógicos, possibilitando também o uso de cabos de programação para CLP e *drivers* via software. Já para o caso da segunda bancada (SILVA NEWTON ; GAINO, 2014), o processador que controla a placa de inversor trifásico é programado via diagrama de blocos pelo ambiente do *Matlab/Simulink*, no computador. Finalmente, para o caso da terceira bancada (SOMA, 2016) permite a programação via IHM do próprio inversor ou, através de um cabo, programar o inversor por uma interface de programação via computador.

Comparando com as informações propostas acima, o projeto atual conta com a programação via IHM do inversor, não sendo este o foco principal, e, principalmente, pela programação via interface gráfica do utilizador (GUI) disponibilizada na *Raspberry* (utilizada como computador), o qual manda os comandos e informações para o intermediário Arduíno que, por sua vez, envia os dados para o inversor de frequência.

1.3.4.4 Equipamentos adicionais

Além dos elementos apontados anteriormente, o atual projeto conta com um sistema mecânico de patins e guias lineares instalados sobre perfis extrudados de alumínio (aspecto melhor abordado no Capítulo 2) com o objetivo de analisar a movimentação de uma plataforma. Além disso, o sistema consta com um *encoder* absoluto acoplado ao fuso que visa gerar uma malha fechada para controle de posição.

Em comparação com as outras bancadas, aponta-se abaixo os elementos que diferenciam-nas deste projeto:

Bancada Auttom (AUTTOM, 2016) Apresenta um painel IHM acoplado na estrutura, apresenta espaço para uso de CLP e apresenta um indicador digital de posição, não sendo indicado qual;

Bancada da Tese (SILVA NEWTON ; GAINO, 2014) Apresenta placa de inversor trifásico e um processador próprio para seu controle com comunicação via ambiente computacional Matlab; diferentemente da *Raspberry*, é utilizado um computador com sistema operacional *Windows*; e não existe um sistema linear para movimento.

Bancada da Soma (SOMA, 2016) Sua estrutura de alojamento é compacta, não possuindo um elemento próprio de sustentação, sendo necessário apoiá-la numa mesa ou superfície. Além disso, apresenta um painel de botões para proteção contra choques elétricos e contatos acidentais.

Ainda abordando um pouco sobre a revisão do trabalho já apresentado, o Capítulo 2 abordará sobre os aspectos mecânicos da bancada e sobre os dois dispositivos elétricos já definidos: motor CA de indução e inversor de frequência.

2 PROJETO MECÂNICO

O projeto mecânico, assim como a escolha dos dispositivos fundamentais presentes no projeto (motor elétrico de indução e inversor de frequências), foram previamente definidos pela primeira parte de desenvolvimento desta bancada didática realizada pelo Marcio (SILVA, 2017). Nos tópicos à seguir, serão abordados aspectos gerais sobre tais elementos já previamente definidos e montados.

2.1 Estrutura mecânica

O projeto mecânico e design foram baseados em uma plataforma já existe no Laboratório de Dinâmica da EESC. A diferença é que esta possui orientação vertical, enquanto que a bancada ao qual esta monografia se refere possui orientação horizontal (SILVA, 2017).

Na sequência, serão feitos alguns repasses quanto ao trabalho já realizado. A Tabela 1 apresenta uma lista dos materiais e componentes utilizados para a construção da estrutura puramente mecânica do projeto. Observe que, nos próximos capítulos, serão informados itens acrescentados no projeto.

Tabela 1 – *Bill of Materials* (BOM) parcial

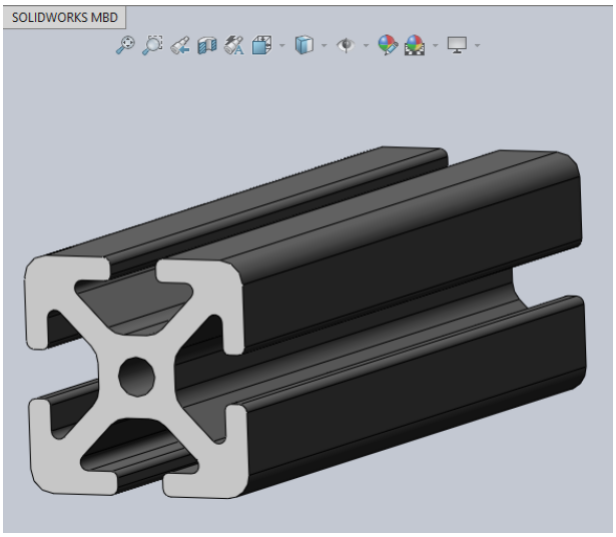
Nº Item	**Descrição**	**Quantidade**
1	Acoplamento elástico	1
2	Alumínio extrudado 30 x 30 mm	6,23m
3	B18.2.3.2M - Formed hex screw, M6 x 1.0 x 16 – 16WN	4
4	B18.3.1M - 10 x 1.5 x 20 Hex SHCS – 20NHX	6
5	B18.3.1M - 6 x 1.0 x 12 Hex SHCS – 12NHX	24
6	B18.3.1M - 6 x 1.0 x 16 Hex SHCS – 16NHX	24
7	B18.3.1M - 6 x 1.0 x 20 Hex SHCS – 20NHX	4
8	B18.3.1M - 8 x 1.25 x 16 Hex SHCS – 16NHX	14
9	B18.3.1M - 8 x 1.25 x 30 Hex SHCS – 30NHX	8
10	Chapa em alumínio 1060 111 x 560 x 5/8"mm	1
11	Chapa em alumínio 1060 140 x 222 x 10 mm	1
12	Chapa em alumínio 1060 150 x 60 x 10 mm	4
13	Chapa em alumínio 1060 180 x 95 x 5/8"mm	1
14	Chapa em alumínio 1060 182 x 124 x 5/8"mm	1
15	Chapa em alumínio 1060 86 x 150 x 10 mm	1
16	Flange triangular em alumínio 1060 90º 124 x 124 x 5/8"mm	2
17	Flange triangular em alumínio 90º 95 x 95 x 5/8"mm	2
18	Inversor de frequência WEG CFW300	1
19	Motor WEG NBR 7094 AL63 0.25CV	1
20	NSK Ball screw support bearings WBK25DF-31	1
21	NSK Ball screw W3207SA-6Z-C5Z10-NSK	1
22	NSK LH Series L1H35-Z 920mm	2
23	Patins NSK H35	2
24	Porca trava	1
25	Redutor Cestari N1/N2 = 1/60	1
26	Suporte em alumínio 1060 110 x 67 x 34 mm	4

Fonte: Silva (2017)

A modelagem dos aspectos mecânicos da bancada e as imagens consequentes apresentadas nesta seção serão provindas do desenvolvimento realizado em ambiente computacional através do *software Solidworks* 2014 x64.

A escolha do perfil modular de alumínio de 30x30mm (Figura 12) como base para a plataforma foi devido às facilidades que tal componente proporciona, como leveza, durabilidade e montagens práticas, rápidas e versáteis através de parafusos e conexões.

Figura 12 – Perfil Modular em Alumínio 30x30mm



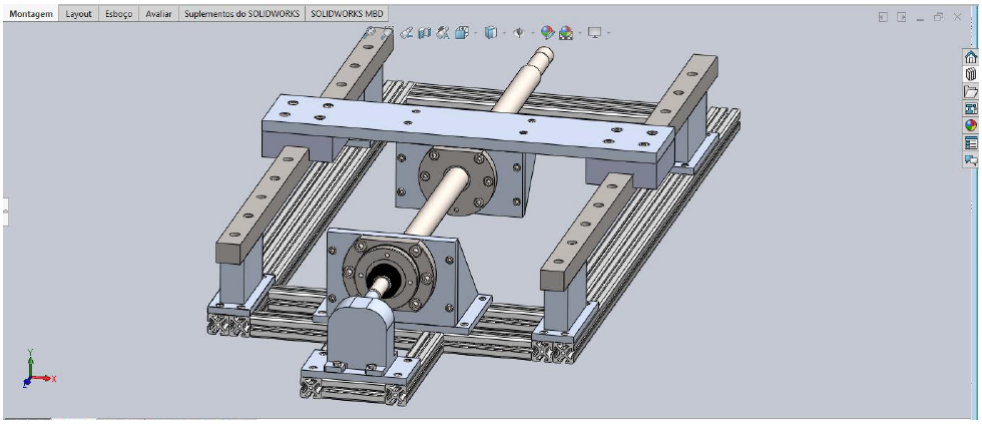
Fonte: Silva (2017)

Partindo das fases de modelagem mecânica e analisando os possíveis erros cabíveis a cada ideia proposta, foram modelados três modelos para a estrutura e arranjo mecânico da bancada. Como o foco deste trabalho não é retomar profundamente o projeto anterior, será apresentado somente o modelo final adotado. As medidas finais do modelo são: 1371,80 mm de comprimento, 450,00 mm de largura e uma altura dos perfis de alumínio de, aproximadamente, 74,00 mm.

O modelo escolhido teve como objetivo economizar material utilizado para confecção da base da plataforma assim como eliminar os erros de construção apresentados nos modos apresentados anteriormente. Com as adequações, segundo Marcio (SILVA, 2017), foram economizados 17,30m de perfil extrudado de alumínio. As imagens computacionais do modelo e suas respectivas vistas são apresentadas na sequência.

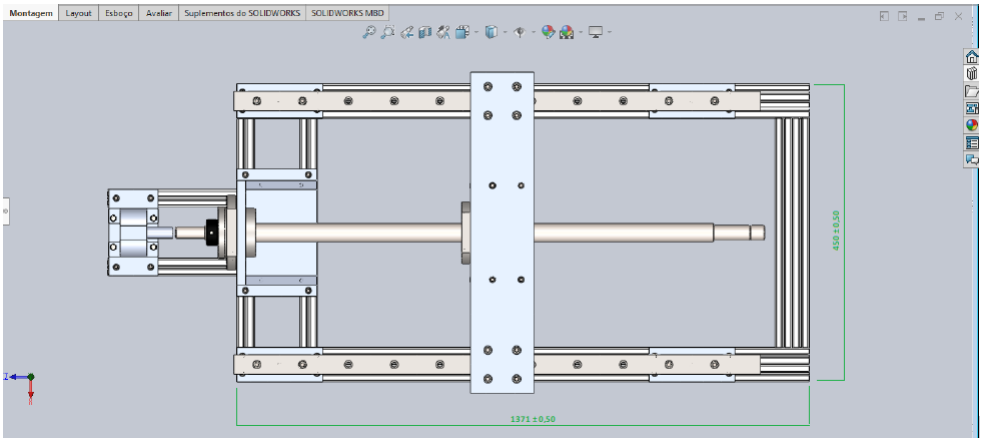
O projeto mecânico foi enviado para usinagem na oficina da universidade junto dos materiais necessários para sua fabricação e atualmente já se encontram fabricadas e instaladas na bancada. Novas compras foram necessárias serem feitas para a finalização do projeto da bancada. Os elementos necessários são para o controle de alimentação de todos os componentes elétricos presentes com um sistema de segurança já previsto e para finalizar a estrutura geral da bancada. Além disso, há os elementos-chaves para a bancada propriamente dita: Arduíno, *Raspberry* Pi 3, *Encoder* e Módulo de Comunicação CFW300-CRS45.

Figura 13 – Modelo final da estrutura mecânica da Bancada apresentada em *Solidworks*



Fonte: Silva (2017)

Figura 14 – Vista superior do modelo final



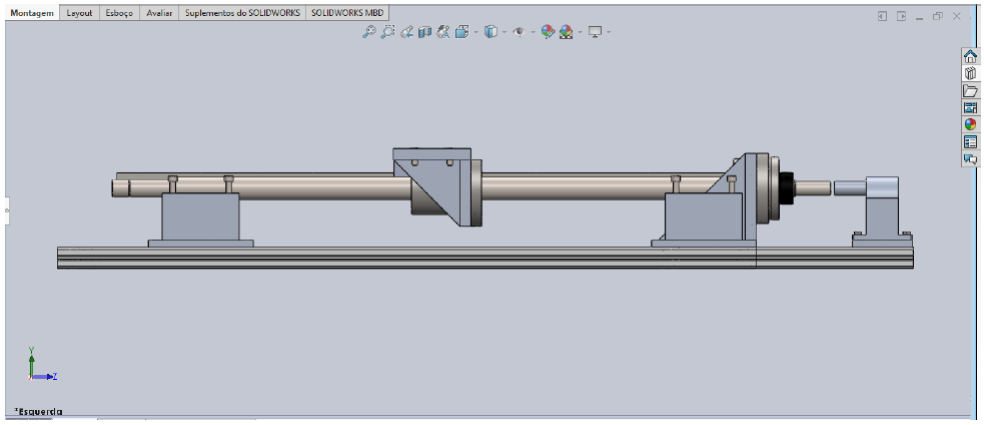
Fonte: Silva (2017)

Tais produtos, conforme citado acima, foram orçadas e encaminhadas para pedido de compra dentro do ambiente universitário. Entretanto, devido a alguns fatores internos e externos, os componentes não puderam ser utilizados na construção real da bancada didática em questão. Por isso, o foco deste trabalho se baseia na produção da Interface Gráfica que será usada futuramente.

2.2 Detalhes técnicos dos componentes

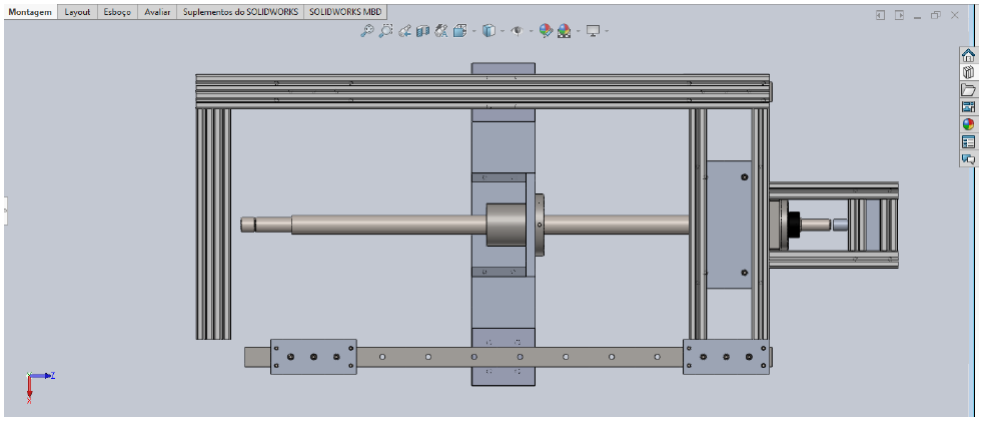
Nesta seção, será abordado alguns elementos estruturais adicionados para compor a plataforma. São eles: redutor de velocidade, módulo de comunicação do inversor de frequências e *encoder*.

Figura 15 – Vista lateral do modelo final



Fonte: Silva (2017)

Figura 16 – Vista inferior do modelo final



Fonte: Silva (2017)

2.3 Redutor de Velocidade

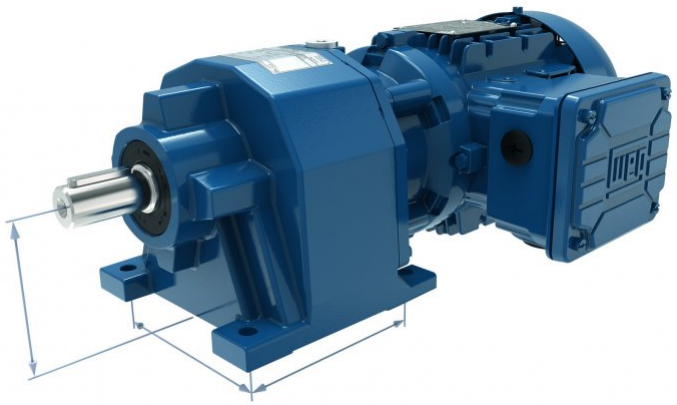
Um redutor de velocidade é dispositivo mecânico que reduz a velocidade de rotação de um acionador que, no caso em questão, é um motor de indução. Consequentemente, entrando com uma elevada rotação no eixo de entrada, obtém-se uma rotação reduzida no eixo de saída e também um torque maior. Seu uso é necessário para se possuir uma maior precisão do deslocamento do fuso e exigir do motor um torque menor do que sem o acoplamento do redutor.

O redutor utilizado na bancada foi o WEG Cestari com número serial 772309. A sua relação de transmissão fixa é de $i = 1/60$, ou seja, a velocidade de saída é 60 vezes menor do que a entrada, em rpm. A velocidade máxima de entrada permita de 1750 rpm produz uma velocidade de saída de, aproximadamente, 29,2 rpm. E o óleo utilizado internamente apresenta

codificação ISO VG 460 CLP PAO.

A Figura 17 apresenta o modelo de redutor utilizado, já com um motor acoplado ao seu eixo de entrada.

Figura 17 – Modelo de redutor WEG com motor acoplado



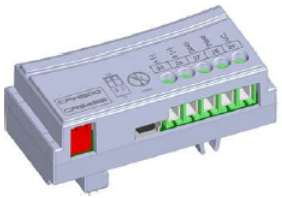
Fonte: WEG (2017)

2.4 Módulo de Comunicação do Inversor

Este equipamento é utilizado dentro da bancada proposta como uma possibilidade de implementar o uso de protocolo Modbus RTU, que será explicado no próximo capítulo (3.3.2), para fazer a comunicação e controle entre o módulo Arduíno e o inversor de frequências. Este, por sua vez, não apresenta as portas necessárias para realizar tal comunicação, por isso há a necessidade do acoplamento do módulo.

O módulo escolhido particularmente para esta aplicação foi o CFW300-CRS-485, que permite a comunicação serial RS-485 entre os dispositivos envolvidos. A Figura 18 apresenta o módulo de comunicação e suas pinagens pode ser visto na Tabela 2 seguinte.

Figura 18 – Módulo com conexão RS-485



Fonte: WEG (2016b)

Mais detalhes à respeito da interface RS-485 serão apresentados no Capítulo 3.

Tabela 2 – Pinagem do conector RS-485 para o módulo CFW300-CRS-485

Conector		Descrição
25	RS-485 - A(-)	RS-485 (Terminal A)
26	RS-485 - B(+)	RS-485 (Terminal B)
27	GND	Referência 0V
28	Shield (PE)	Blindagem do cabo
29	N.C.	Sem conexão

Fonte: WEG (2016b)

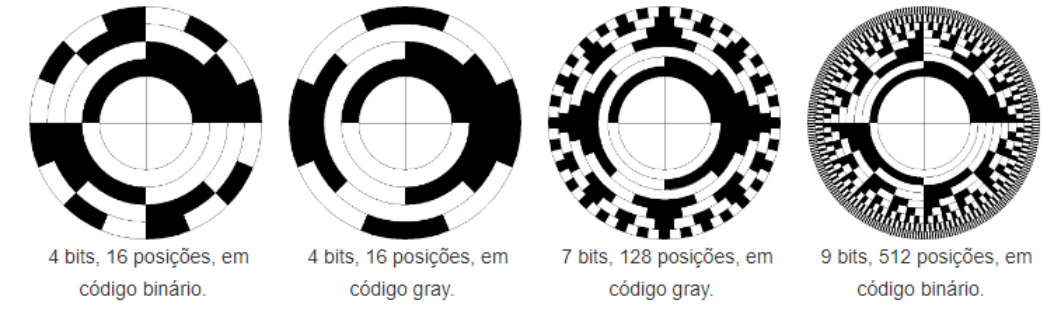
2.5 *Encoder*

Em automação industrial, um *encoder* é um dispositivo eletromecânico que conta ou reproduz pulsos elétricos a partir do movimento rotacional de seu eixo. Isto é, ele transforma posição em sinal elétrico digital. Por isso, com um *encoder* é possível quantizar distâncias, controlar velocidades, medir ângulos, número de rotações e realizar posicionamentos, por exemplo.

Neste projeto será utilizado um *encoder* absoluto. É um *encoder* mais complexo que o *encoder* incremental já que indica a posição absoluta através de saídas digitais codificadas em um valor binário ou via comunicação. Dentre suas vantagens, pode-se citar também a capacidade de armazenamento e distinção do número da volta em que o elemento acoplado ao seu eixo se encontra.

A resolução é representada em bits que correspondem ao número de faixas disponíveis no sensor, por exemplo como um *encoder* absoluto de 4 bits irá gerar uma representação de posição absoluta de 16 valores. A grande vantagem do *encoder* absoluto é que, em qualquer momento, independente por exemplo de uma queda de energia, é possível saber a posição do dispositivo sem necessidade de re-sincronizar a posição, pois este utiliza um sistema mecânico de pinhão e coroa para localizar sua posição. A Figura 19 abaixo mostra discos de *encoders* absolutos conforme sua capacidade de identificação.

Figura 19 – Discos de *Encoders* Absolutos



Fonte: Almeida (2017)

O *encoder* absoluto utilizado é o *encoder* da marca *hohner* da série 67 - *Encoder* Absoluto Serial. Foi especificado como *Multiturn*, isto é, ele irá indicar a quantidade de voltas realizadas dentro do seu intervalo permitido.

Ele apresenta Interface Serial RS-485 e comunicação Modbus RTU, para que seja possível conectar em rede com o inversor de frequências e que seja utilizada apenas um protocolo de comunicação. A Figura 20 mostra o *encoder* da marca *hohner* utilizado.

Figura 20 – *Encoder* Absoluto *hohner* serie 67

Fonte: hohner (2016)

A Tabela 3 mostra a pinagem do *encoder* utilizado. É importante ressaltar ainda que, dentro das especificações do produto, sua resolução é de 12 bits de posições por volta e 4096 voltas permitidas de reconhecimento. Ou seja, ele pode armazenar e identificar 4096 posições dentro de uma única volta e um total de 4096 voltas (sendo, portanto, classificado com 1212 bits), sendo necessário, portanto, dois discos para identificação.

Tabela 3 – Ligações com cabo ou conector para Identidade Fixa

Preto	Pino 1	= 0 Volts (-)
Vermelho	Pino 2	= +Vcc (24V)
Azul	Pino 3	= RS-485- (Low)
Branco	Pino 4	= RS-485+ (High)

Fonte: hohner (2016)

3 ELEMENTOS ELETRÔNICOS E ASPECTOS DA INTERFACE GRÁFICA

Nos capítulos anteriores, foram apresentadas informações relevantes ao projeto da bancada didática que é a inspiração para este trabalho. Além de uma repasse geral sobre a construção atual realizada por (SILVA, 2017), foi ainda acrescentado dados sobre os novos equipamentos que irão fazer parte futuramente da bancada.

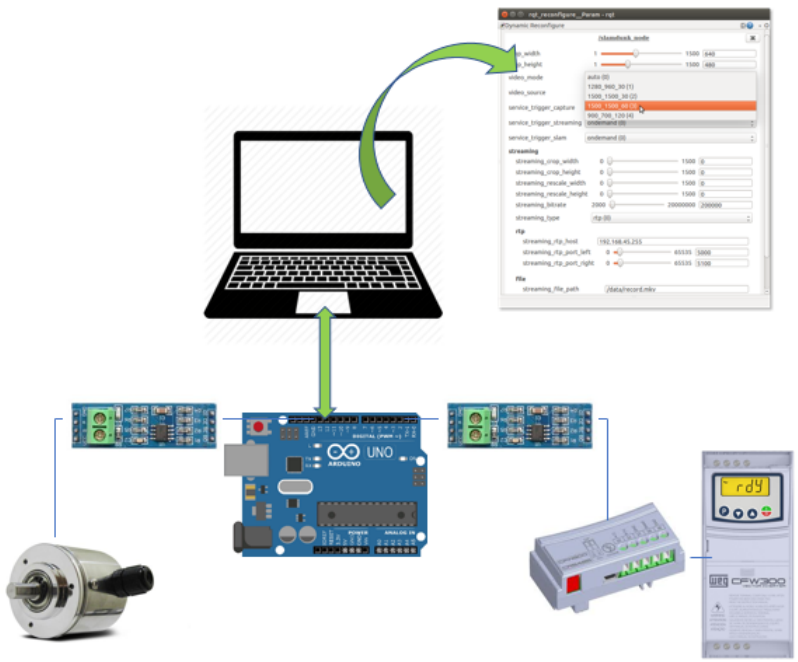
O intuito deste e dos próximos capítulos é apresentar a interface gráfica propriamente dita que é o foco do deste trabalho.

3.1 Visão Geral

A finalidade principal da criação de uma interface gráfica para a bancada didática de motivação é fornecer ao usuário um meio mais amigável e simples para realizar o controle e, consequentemente, os experimentos propostos na bancada através da escolha dos parâmetros em uma única janela.

A Figura 21 apresenta um esquemático geral de como deve funcionar a interface gráfica atrelada ao sistema de controle da bancada didática.

Figura 21 – Esquema completo do controle e comunicação dos equipamentos



Fonte: Própria

Os elementos presentes são: um dispositivo de processamento que permite operar com

Ubuntu (computador ou *Raspberry* Pi 3); Arduíno Mega (foi verificado que o UNO não teve capacidade suficiente para armazenar e processar a programação); Transceptor de baixo-poder MAX485, responsável pela conversão RS-485 para TTL (Transistor-Transistor Logic), com a qual o Arduíno trabalha; *encoder hohner*; Módulo de comunicação CFW300 CRS-485; e, por fim, o inversor de frequências *WEG* CFW300.

Com as conexões físicas apresentadas, é importante ressaltar as conexões internas de comunicação entre os equipamentos:

Ubuntu Dentro do dispositivo de processamento de dados (Computador ou *Raspberry* Pi 3), ocorre a execução dos arquivos em ROS, que permitem uma vasta quantidade de funções e processos dentro e fora do próprio dispositivo;

USB Para conectar fisicamente o computador, por exemplo, e o Arduíno utiliza-se um cabo USB por onde os dados são enviados e recebidos;

Arduíno Dentro do módulo Arduíno um nó de ROS é iniciado e é posto para executar ao longo de toda a operação da interface gráfica, permitindo a comunicação entre os dois dispositivos (foco deste projeto);

Nas seções seguintes serão abordados com maior profundidade os termos e as construções realizadas neste trabalho.

3.2 Módulo Arduíno

O módulo Arduíno é uma plataforma de prototipagem eletrônica de hardware livre, com software de código aberto (*open source*) e de placa única, contendo um microprocessador *Atmel AVR* embutido para controle. Apresenta suporte de entrada/saída embutido e uma linguagem de programação padrão da própria empresa (sendo, essencialmente, C/C++).

Como o Arduíno apresenta baixo custo, acessível, flexível e de fácil manipulação para iniciantes e profissionais, esta placa com microcontrolador foi o escolhido para gerir, enviar e receber dados do ROS no Ubuntu. Assim como também irá realizar a comunicação e o controle como mestre dos possíveis equipamentos para a bancada de inspiração: *encoder* e inversor de frequências.

3.2.1 Arduíno UNO

A placa UNO é uma ótima placa para quem está começando com eletrônica e programação de códigos. Com o intuito de mostrar que além de sua facilidade de uso ainda há o fator capacidade de processamento e grande leque de possibilidades, esta placa foi escolhida inicialmente para o desenvolvimento do projeto.

A Figura 22 mostra a placa UNO propriamente dita.

Figura 22 – Placa com módulo Arduíno UNO

Fonte: Arduino (2016b)

O seu uso foi extremamente eficaz ao longo do período inicial, onde estava sendo implementado apenas o sistema de comunicação de ROS através da biblioteca *rosserial*. Foi possível estabelecer a comunicação entre os dois dispositivos onde se enviava os dados do computador e recebia-os no Arduíno.

Entretanto, com o acréscimo das funções de Sistemas Operacionais de Tempo Real (RTOS), o tamanho de memória que o novo programa ficou foi muito maior que o permitido pelo UNO, cuja Memória Flash é de 32 KB. Nesta etapa, o código a ser submetido no microcontrolador já ultrapassava os 70 KB.

Portanto, a próxima alternativa foi utilizar o Arduíno Mega 2560, cujo microcontrolador embutido é o ATmega2560.

3.2.2 Arduíno Mega 2560

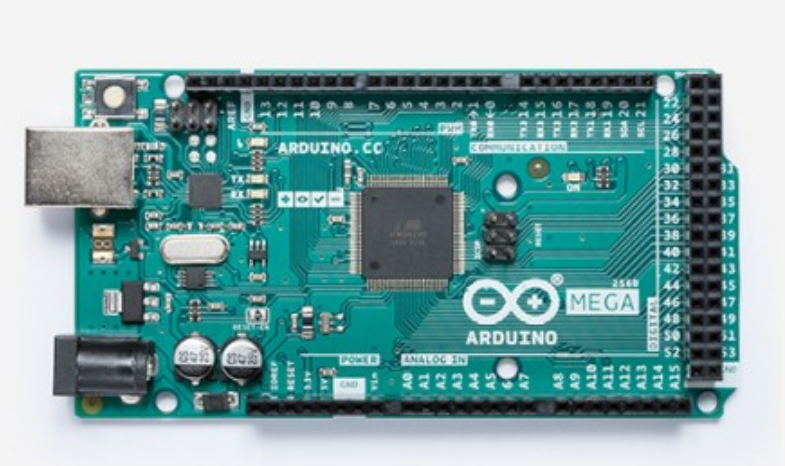
Comparado ao UNO, a placa Mega 2560 possui quantidade de portas lógicas (digital e analógica) muito maior: enquanto que o primeiro possui 14 portas digitais e 6 portas analógicas, o segundo apresenta 54 portas digitais e 16 portas analógicas.

Além disso, sendo o fator crucial, o Mega 2560 apresenta 256 KB de Memória Flash, sendo mais que o suficiente para alojar o programa e suas bibliotecas necessárias. Também pode-se somar o fato que este modelo é fácil de ser adquirido e é comumente utilizado além do UNO.

É importante também ressaltar que as duas placas apresentam o mesmo *clock* de 16MHz (ARDUINO, 2016c), mesma tensão de alimentação de 5V e mesma interface de programação USB via ATMega 16U2.

A Figura 23 apresenta a placa Mega 2560.

Figura 23 – Módulo Arduíno Mega 2560



Fonte: Arduino (2016a)

3.3 Comunicação entre Arduíno e Equipamentos

Apesar desta comunicação não ter sido implementado de fato dentro do projeto desta monografia, ainda é importante introduzir este assunto para os futuros trabalhos visto que os equipamentos já foram determinados e encaminhados para compra.

A comunicação do módulo Arduíno com os outros equipamentos (inversor de frequências e *encoder* absoluto) é realizado através da comunicação serial. A comunicação serial consta com um processo de enviar *bit* a *bit* sequencialmente em um canal de comunicação ou barramento.

Por parte do Arduíno, o protocolo de comunicação é o UART, isto é, o pino de transmissão (T_x) do protocolo envia um pacote de *bits* que será interpretado *bit* a *bit* pelo pino receptor (R_x). Cada pacote enviado possui um *header* ou *start bit* que indica o início da mensagem, 5 a 9 bits de informação, 1 bit de paridade para enviar a recepção de erros e 1 ou 2 stop bits para indicar o final da mensagem. A Figura 24 apresenta a ligação dos pinos na comunicação UART.

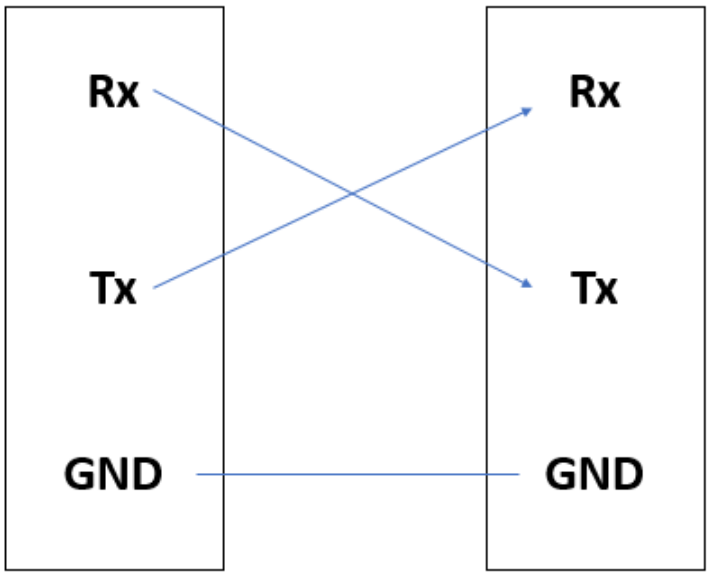
Enquanto isso, por parte dos dispositivos foi escolhido a comunicação serial Modbus RTU com barramento RS-485, que serão detalhados as sub-seções seguintes.

E para conectar estes equipamentos à placa Arduíno é necessário o uso de elementos de conversão de um protocolo para outro. Para isso, foi escolhido um módulo conversor MAX485, que é um transceptor de baixa energia para comunicação RS-485. A Figura 25 mostra seus detalhes físicos.

3.3.1 RS-485

Também conhecido como TIA-485, é um padrão (meio físico) que define as características elétricas de transmissores e receptores para uso em sistemas de comunicação serial. A

Figura 24 – Comunicação assíncrona do protocolo UART que depende da taxa de transmissão como referência.



Fonte: Própria

sinalização elétrica é balanceada e os sistemas multiponto são suportados.

As redes de comunicação digital que implementam o padrão podem ser usadas de maneira eficaz em longas distâncias e em ambientes eletricamente ruidosos. Vários receptores podem ser conectados a tal rede em um barramento multiponto linear.

Sobre as características da interface RS-485, pode-se listar as mais importantes (WEG (2016b)) para o módulo de comunicação do inversor de frequências e para o *encoder* absoluto:

a) Interface segue o padrão EIA/TIA-485;

b) Possibilita comunicação utilizando taxas de 9600 até 38400 Kbit/s;

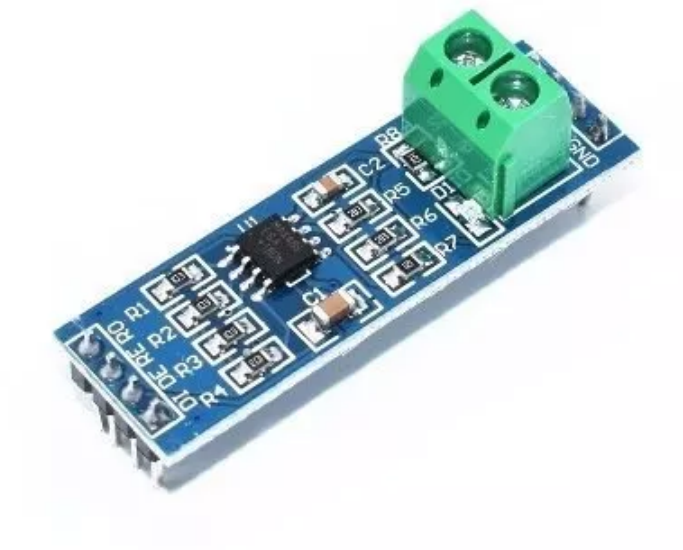
c) Interface isolada galvanicamente e com sinal diferencial, conferindo maior robustez contra interferência eletromagnética;

d) Permite a conexão de até 32 dispositivos no mesmo segmento;

e) Comprimento máximo do barramento de 1000 metros;

Para o caso do projeto da bancada, será utilizado o arranjo mestre-escrevo, onde o mestre será o Arduíno Mega e os escravos serão os dispositivos Inversor e *encoder*. Este método de comando é centralizado onde apenas o dispositivo mestre pode iniciar uma comunicação, enviando comandos, controlando a taxa de comunicação, etc.

Figura 25 – Módulo conversor MAX485



Fonte: Products (2014)

3.3.2 Modbus RTU

O protocolo Modbus foi inicialmente desenvolvido em 1979. Atualmente, é um protocolo aberto amplamente difundido, utilizado por vários fabricantes em diversos equipamentos.

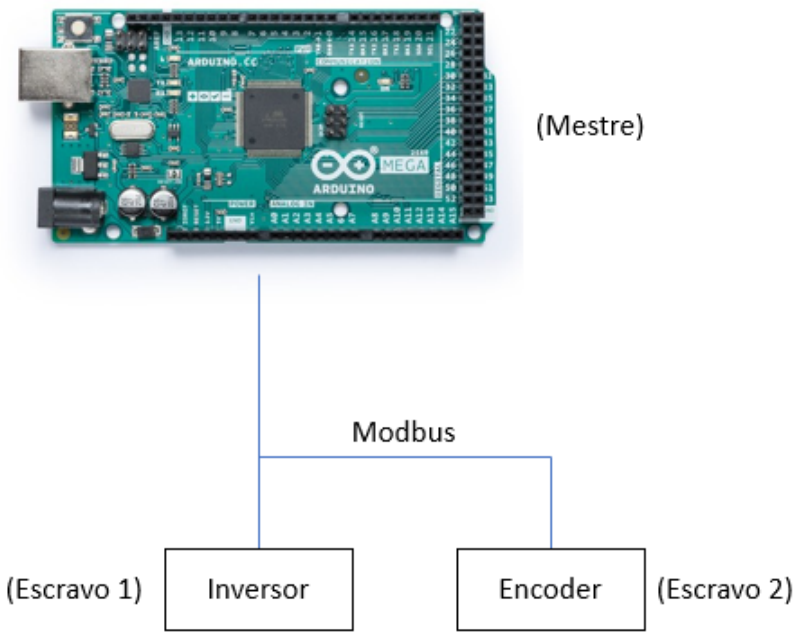
Este protocolo especifica que o modelo de comunicação é do tipo mestre-escravo, como foi comentando anteriormente. Assim, um escravo não deve iniciar nenhum tipo de comunicação no meio físico enquanto não tiver sido requisitado pelo mestre. A subseção 3.3.2 mostra o exemplo aplicado de rede Modbus com um mestre (Arduíno Mega) e dois escravos (Inversor de Frequências e *encoder* Absoluto).

A rede Modbus RTU permite até 247 escravos, mas somente um mestre (mas fica-se limitado ao meio físico adotado). Toda comunicação inicia com o mestre fazendo uma solicitação a um escravo, e este responde ao mestre o que foi solicitado (WEG (2016b)). Em ambos os telegramas (pergunta e reposta), a estrutura utilizada é a mesma: Endereço, Código da Função, Dados e CRC. Apenas o campo de dados poderá ser tamanho variável, dependendo do que está sendo solicitado. E a linguagem é, no caso dos dispositivos da bancada didática, em hexadecimal.

Na sequência, aborda-se com mais detalhes os tópicos da estrutura da mensagem Modbus RTU (WEG (2016b)):

Endereço O mestre inicia a comunicação enviando um byte com o endereço do escravo para o qual se destina a mensagem (caso seja igual a zero, então destina-se a todos os escravos da rede sem esperar resposta destes). Ao enviar a resposta, o escravo também inicia o telegrama com o seu próprio endereço;

Figura 26 – Rede Modbus com um mestre e dois escravos



Fonte: Própria

Figura 27 – Formato do telegrama de requisição enviado pelo Mestre

Endereço (1 byte)	Função (1 byte)	Dados da requisição (n bytes)	CRC (2 bytes)

Fonte: WEG (2016b)

Figura 28 – Formato do telegrama de resposta enviado pelo Escravo ao Mestre

Endereço (1 byte)	Função (1 byte)	Dados da resposta (n bytes)	CRC (2 bytes)

Fonte: WEG (2016b)

Código da Função Este campo também contém um único byte, onde o mestre especifica o tipo de serviço ou função solicitada ao escravo (leitura, escrita, etc.). Pode-se consultar tabelas de manuais para saber qual função utilizar;

Campo de Dados O formato e conteúdo deste campo dependem da função utilizada e dos valores transmitidos. Novamente, parte-se da particularidade de cada equipamento em questão;

CRC A última parte do telegrama é o campo para checagem de erros de transmissão. O método utilizado é o CRC-16 (Cycling Redundancy Check). Este campo é formado por dois bytes, onde primeiro é transmitido o byte menos significativo (CRC-), e depois o mais significativo (CRC+);

3.4 Interface Gráfica

Nesta seção será detalhado o foco principal deste projeto: o desenvolvimento de uma interface gráfica com aspectos voltados para o controle de uma bancada didática que apresenta uma plataforma sob guias lineares movimentada por um motor de indução controlado por um inversor de frequência, em malha fechada com um *encoder* absoluto.

Uma interface gráfica (em inglês, *Graphical User Interface* - GUI) permite que o usuário tenha interação com dispositivos digitais por meio de elementos gráficos como ícones, barras e outros indicadores visuais. O usuário pode interagir com um mouse ou teclado, podendo selecionar, mover uma barra ou simplesmente colocar o valor da variável que deseje.

O desenvolvimento foi realizado em duas vertentes: uma para disponibilizar ao usuário um *software* onde se pode colocar os *inputs* (entradas) e visualizar os*outputs* (saídas) provindos do sistema de análise; e outra para realizar toda a coleta de dados e controle do sistema que se deseja estudar que, no caso de estudo, é a bancada didática citada anteriormente. A primeira vertente foi desenvolvida em ROS (*Robot Operating System*) e a segunda em C/C++ para Arduíno.

Nas próximas seções serão abordados tópicos importantes para compreensão do trabalho desenvolvido, que será apresentado definitivamente no Capítulo 4.

3.4.1 *Robot Operating System* (ROS)

Em português, Sistema Operacional de Robôs, o ROS é uma coleção de *frameworks* de *software* para desenvolvimento de robôs que disponibiliza bibliotecas e ferramentas para auxiliar desenvolvedores de *software* a criar aplicações robóticas. Ele fornece abstração de *hardware*, *device drivers*, bibliotecas, visualizadores, transmissão de mensagens e gerenciamento de pacotes (Robotics (2018)). Além de todas essas funcionalidades e qualidades, o ROS é licenciado sob uma licença livre (*open source*) que permite a todos utilizá-lo, como neste caso, para fins acadêmicos.

Grande parte do desenvolvimento da interface tomou-se como base os tutoriais e funcionalidades semelhantes disponibilizadas no *Wiki ROS*, um site do próprio desenvolver *Open Robotics* que contém diversas informações para uso e tutoriais de fácil entendimento e implementação. Dentre as informações obtidas é importante salientar elementos importantes da construção.

3.4.1.1 *Workspace* ou Espaço de Trabalho

Um *workspace* em ROS é uma pasta onde se pode modificar, construir e instalar pacotes do *catkin* (ferramenta do ROS utilizada para compilar os arquivos dentro da estrutura). Caso não existe o *workspace*, cada pacote teria que ser executado separadamente e um de cada vez como um projeto *standalone* Robotics (2018).

As três principais pastas dentro da construção do *workspace* são: *build*, *devel* e *src*. Mas também podem haver as pastas *Install* e *Result*, as quais não são utilizadas neste trabalho.

Source Space Contém o código fonte de todos os pacotes do *catkin* presentes no *workspace*. É onde se pode extrair, modificar e clonar os códigos fontes dos pacotes que se queira construir, os quais estão localizados separados e independentes em cada pasta própria;

Build Space É onde o arquivo *CMake* é invocado para construir os pacotes do catkin no *source space*. *CMake* e *catkin* mantêm suas informações ocultas e outros arquivos intermediários aqui.;

Development (devel) Space É onde os alvos da construção são colocados antes de serem instalados. O modo como estes alvos são organizados dentro do *devel space* é do mesmo jeito em que são organizados quando instalados;

3.4.1.2 *Node*

É basicamente um executável. Isto é, deverá exercer alguma ou várias funções dentro do ambiente do ROS. Portanto, é possível ter nós rodando em qualquer uma das máquinas da rede. Cada nó pode publicar e se inscrever em tópicos, os quais funcionam como uma variável. Ou seja, um nó pode tanto escrever valores nessa variável ou ler os valores dessa variável. Neste intuito é que foi construído o sistema de *server* em que a interface atua.

Na sequência, portanto, é importante ressaltar algumas características sobre os nós das quais se extraíram inspirações para a construção do projeto:

a) Qualquer nó pode publicar uma mensagem em qualquer tópico;

b) Qualquer nó pode se inscrever em qualquer tópico;

c) Múltiplos nós podem publicar no mesmo tópico;

d) Múltiplos nós podem se inscreverem no mesmo;

e) Um nó pode publicar e se assinar em vários tópicos, que é o caso do nó *server* presente neste trabalho;

3.4.1.3 ROS *Message*

Uma mensagem, dentro do contexto de ROS, é uma serialização formada por dados estruturados, isto é, é um arquivo de extensão *.msg* com várias variáveis inclusas podendo ser de

qualquer tipo (*int, float, double, bool, char,* etc.). Ela permite que os nós escrevam em C++ ou Python para se comunicar entre eles.

3.4.1.4 ROS *Master*

É um servidor que faz a conexão de endereços entre todos os nós, identificando cada um deles, e, portanto, é extremamente necessário que em um terminal do Ubuntu este esteja sendo executado. Além disso, informa os *subscribers* sobre nós publicando no mesmo tópico.

Para executá-lo, basta abrir um novo terminal do Ubuntu, neste caso, através do comando $CTRL + Alt + T$ ou através do ícone do terminal disponível. Na sequência, digitar o código para executá-lo:

roscore

Ou, no caso deste trabalho, pode-se acoplar diversos nós presentes dentro de uma ferramenta presente na pasta *launch*, dentro do *source space*: ROS *Launch*. E, ao executar o comando referente a este atalho, o ROS *Master* é executado automaticamente. Portanto, a estrutura do comando fica:

roslaunch nome-do-pacote arquivo.launch

3.4.1.5 ROS *Subscriber*

É um nó (executável) ou simplesmente uma ferramenta dentro de um nó que "escuta"/recebe dados de um tópico através do comando $subscribe()$ de um manipulador de nó (*node handle*), que é um movimentador que possui e fornece acesso ao valor armazenado no nó.

3.4.1.6 ROS *Publisher*

Assim como o ROS *Subscriber*, o *publisher* é um nó ou ferramenta de um nó, entretanto, é responsável pela criação dos dados presentes na variável e por enviar/publicar estes dados através de tópicos para algum possível *subscriber* ou *listener*.

3.4.2 Controle dos processos pelo Arduíno

O Arduíno Mega é responsável por receber os dados da interface gráfica, mais especificamente, da ferramenta *dynamic reconfigure* e, portanto, ele atua como um *subscriber* neste caso. Além disso, ele também é responsável por enviar constantemente dados coletados e processados por ele para os aplicativos do ROS (através de um cabo USB físico), que devem estar em funcionamento ao realizar os experimentos. Logo, ele atua tanto como um *subscriber* quanto um *publisher* através de uma biblioteca chamada *rosserial* (Robotics (2018)).

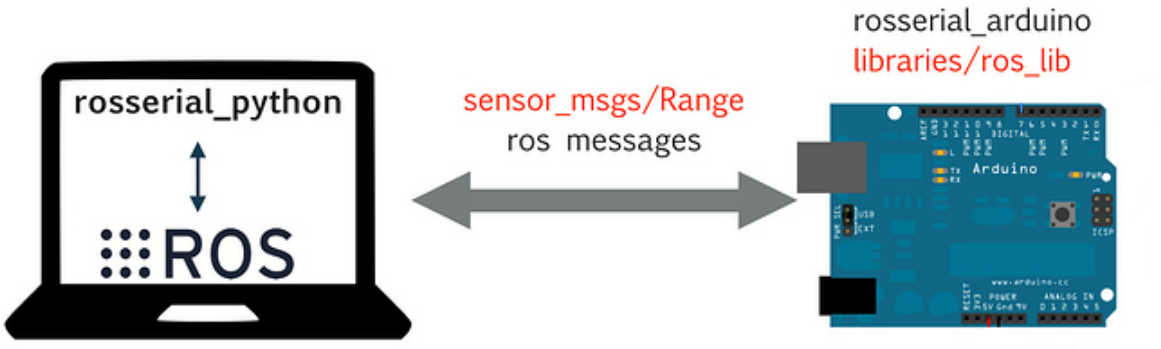
Além dele atuar como um intermediário de comunicação entre os dispositivos principais da bancada (inversor de frequências e *encoder* absoluto), ele também atua como um processador e um mestre para os dispositivos. Isto é, o Mega irá realizar o controle da velocidade e posição através de PID para verificar seu comportamento de sinal e resposta conforme os parâmetros selecionados pelos usuário; e o Mega irá enviar ordens para o inversor de frequências e irá requisitar dados para os dois dispositivos. E, para realizar todas estas funções simultaneamente, foi utilizado uma biblioteca de Sistema Operacional de Tempo Real ou, da sigla em inglês, RTOS.

3.4.2.1 *rosserial*

O *rosserial* é um protocolo para agrupar mensagens serializadas padrão do ROS e multiplexar vários tópicos e serviços em um dispositivo de caractere, como uma porta serial ou um soquete de rede. Para este trabalho, foi utilizado a biblioteca deste protocolo voltado especialmente para o Arduíno.

Nela contém bibliotecas de cliente que permite que o microcontrolador obtenha facilmente os nós do ROS e rode vários subsistemas. Para isso, é necessário que tenha um nó na máquina de hospedagem para se criar uma ponte de conexão entre o protocolo serial e a rede ROS. A Figura 29 mostra uma representação ilustrativa para mostrar a conexão entre o ROS e o microcontrolador presente no módulo Arduíno.

Figura 29 – Representação ilustrativa da comunicação por *rosserial*



Fonte: Robotics (2018)

Algumas limitações devem ser mencionadas: o número de *publishers* e *subscribers* é limitado a 25 e o tamanho dos *buffers* de serialização e de-serialização é limitado em 512 bytes por padrão, entretanto, o processador do Arduíno Mega é capaz e foi programado para suportar até 1536 bytes.

3.4.2.2 Controle PID

O controlador proporcional integral derivativo (PID) é uma técnica de controle de processos que une as ações derivativa, integral e proporcional, fazendo assim com que o sinal de erro seja minimizado pela ação proporcional, zerado pela ação integral e obtido com uma velocidade antecipativa pela ação derivativa (Wescott (2000)). É baseado na resposta da modelagem matemática de uma malha de processo a ser controlada.

A saída de um controlador PID, que é igual à entrada de controle para o centro, é calculada no domínio do tempo a partir do erro de *feedback* de acordo com a Equação 3.1 abaixo:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{3.1}$$

Para este trabalho, no intuito de representar o possível circuito real presente na bancada, os valores de K_p, K_i e K_d são escritos pelo próprio usuário na interface do *dynamic reconfigure*. E, partindo do uso de uma biblioteca de PID para Arduíno, consegue-se fazer um possível controle da velocidade e da posição da plataforma da guia linear (no Capítulo 4 será apresentado e explicado o teste realizado para simular e comprovar o funcionamento do controle PID realizado pelo Arduíno Mega).

3.4.2.3 RTOS

É um sistema operacional (conjunto de ferramentas (*softwares*)) destinado à execução de múltiplas tarefas onde o tempo de resposta a um evento (externo ou interno) é pré-definido, ou seja, o programador seleciona a frequência de ocorrência de cada tarefa (*tasks*). O tempo de resposta é chamado de prazo da tarefa e o não cumprimento de uma tarefa dentro do prazo esperado caracteriza uma falha do sistema.

Um RTOS é mais eficaz e é mais valorizado pela forma previsível e rápida na resposta a um evento, do que pela quantidade de dados que processa. Os fatores chave são, então, fornecer latências de interrupções e de alternância de tarefas mínimas.

Para tanto, o uso deste sistema operacional é necessário visto que o Arduíno executará diversas tarefas simultâneas e necessita-se que elas aconteçam em frequências determinadas para êxito do sistema. Tais tarefas podem ser listadas abaixo:

a) Receber e enviar dados para o ROS;

b) Enviar e receber dados para os outros dispositivos (inversor e *encoder*);

c) Controlar a posição e a velocidade da plataforma através de PID;

d) Gerar modos referência ou malha-fechada;

Para implementar este sistema dentro do Arduíno, foi utilizado uma biblioteca própria chamada "FreeRTOS", que contém toda a estrutura já preparada para a criação, configuração e

execução de *tasks* conforme a frequência e o nível de prioridade de acontecimento de cada uma. Mais detalhes de como foi implementado no microcontrolador serão apresentados no Capítulo 4.

4 PROJETO CONCLUÍDO E EXPERIMENTOS

O Capítulo 3 apresentou alguns elementos gerais utilizados no trabalho e alguns conceitos fundamentais para o seu entendimento. Neste capítulo, de fato, será apresentado a construção do projeto em si, mostrando todo o procedimento de verificações e *debugs*; e será apresentado os experimentos realizados simulando a sua aplicação na bancada didática proposta.

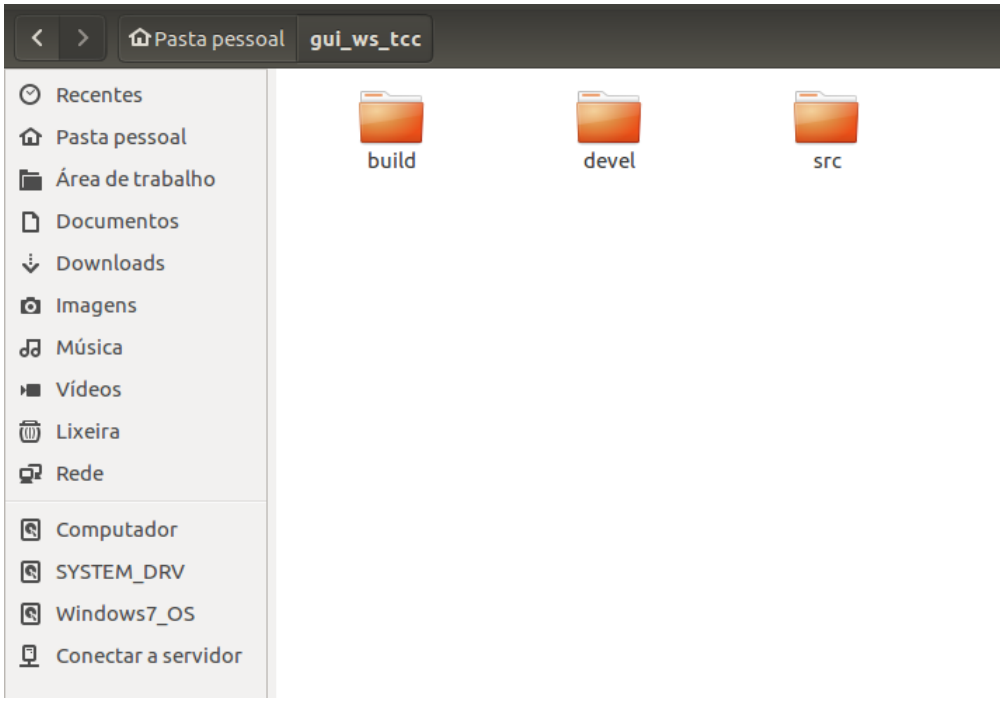
4.1 Construção da Interface

Conforme apresentado anteriormente, a construção da interface foi realizada em ROS. Partiu-se da elaboração de uma tela em que o usuário pudesse escrever alguns valores e escolher opções simples, como ligar e desligar o motor. A sua construção foi realizada em um pacote denominado dynamic_interface criado dentro de um **workspace** padrão do ROS.

O pacote foi gerado com três dependências (ferramentas e extensões prontas disponibilizadas pelo próprio ROS): *rospy*, *roscpp* e *dynamic_reconfigure*. As duas primeiras extensões correspondem a bibliotecas de cliente (isto é, uma coleção de códigos que facilita o trabalho de programadores em ROS) que promovem aos programadores de, respectivamente, Python e C++ um acesso fácil aos *topics*, serviços e parâmetros do ROS (Robotics (2018)). E a terceira extensão corresponde à própria biblioteca da interface.

A Figura 30 mostra as pastas presentes dentro do *workspace* utilizado para a criação dos pacotes utilizados neste trabalho.

Figura 30 – Pastas presente no *workspace* do ROS utilizado neste projeto



Fonte: Própria

4.1.1 *dynamic_reconfigure*

Para a criação do pacote de dynamic_reconfigure foi necessário criar uma pasta, cujo nome foi dynamic_interface, dentro do *source space* e dentro dela foi criada o arquivo de configuração, onde parâmetros presentes na interface foram geradas e onde foi criado um *topic* para transmitir todos os parâmetros como uma *message* para o nó *server*, que para este função atua como *subscriber*. O comando utilizado no terminal dentro diretório **src**, visto que o *ROS distribution* utilizado é o *Kinetic*, foi:

catkin_create_pkg –rosdistro kinetic dynamic_interface rospy roscpp dynamic_reconfigure

E as opções disponíveis na interface, inicialmente, eram:

Estado do motor Selecionar para Ligar ou Desligar o Motor;

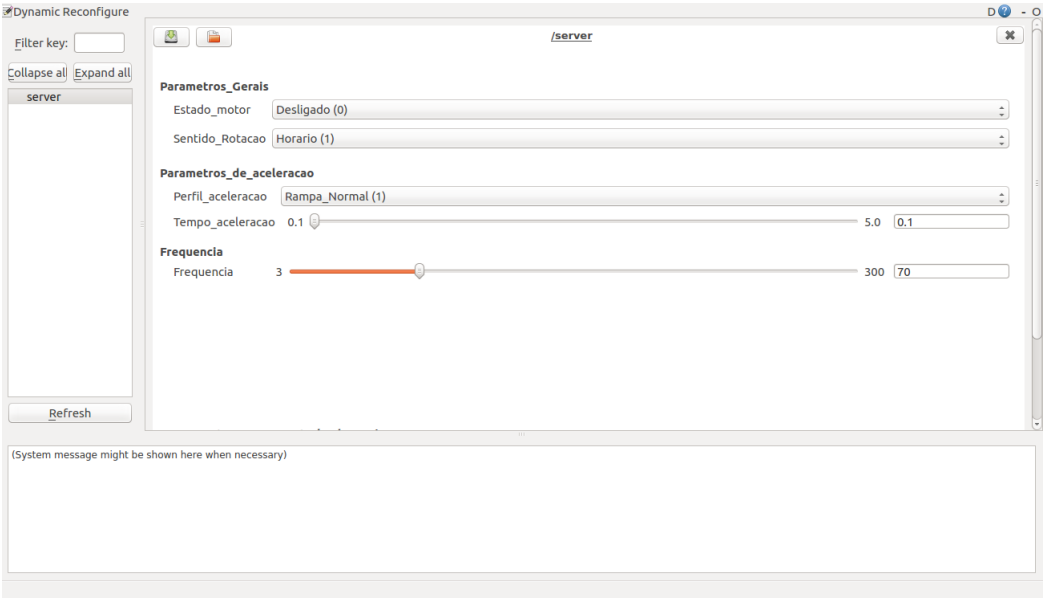
Sentido de rotação Selecionar entre Horário e Anti-Horário;

Perfil de aceleração Selecionar entre Curva S ou Rampa normal;

Tempo de aceleração Arrastar a barra ou escrever um valor decimal entre 0.1 e 5.0 segundos;

Frequência Arrastar a barra ou escrever um valor inteiro entre 3 e 300 Hz;

Figura 31 – Primeira versão das opções disponíveis na interface



Fonte: Própria

É importante ressaltar que: alguns comandos foram adicionados no arquivo *CMakeLists.txt* para que os nós ligados à interface do *dynamic* consigam ler os dados; após cada alteração dentro do *workspace* foi necessário executar o comando ***catkin_make*** no terminal, dentro da própria pasta do *workspace*; e, por fim, para efeito de consultas futuras, o alicerce da estrutura foi feita à partir dos tutorais disponíveis no *Wiki ROS* (Robotics (2018)).

Conforme a comunicação com os outros nós, que serão descritos em subseção 4.1.2 e subseção 4.1.4, foi se estabelecendo, os testes comprovavam o funcionamento da estrutura produzido até então e pequenos experimentos resultaram em resultados positivos, a interface passou a agregar mais parâmetros para alteração e envio. Novas opções foram acrescentadas:

Botão de enviar os dados para o sistema Ao clicar na caixa de seleção, os dados são enviados dentro de um tópico e recebidos pelos *subscribers*. Nesta versão, o usuário pode organizar todos os parâmetros conforme o desejado antes que estes sejam enviados para ação, diferentemente da versão anterior que bastava uma mudança de parâmetro que todos os dados já eram enviados;

Referência: Senoidal É composto por uma caixa de seleção que, ao ser selecionada, disponibiliza três parâmetros para envio: amplitude, período de ocorrência do movimento senoidal da plataforma e um valor de *offset*, que representa o patamar onde irá se encontrar a onda senoidal;

Referência: Degrau É composto por uma caixa de seleção que, ao ser selecionada, disponibiliza apenas um parâmetro para envio: módulo do degrau, isto é, o valor em que se encontrará o patamar;

Malha Fechada: Controle de posição É composto por uma caixa de seleção que, ao ser selecionada, disponibiliza três parâmetros de envio para que o Arduíno consiga executar o controle do movimento da plataforma próximo do sinal de referência com malha fechada: os ganhos K_p (proporcional), K_i (integral) e K_p (derivativo) para a posição;

Malha Fechada: Controle de velocidade É análogo ao caso de Controle de posição, porém é aplicado sob a velocidade. Os parâmetros de envios também são os três ganhos apresentados anteriormente porém para a velocidade.

A interface finalizada é representada na Figura 41 e na Figura 42, sua execução deve ser feita pelo terminal do Ubuntu e pode-se escolher entre duas opções disponíveis. A primeira consiste em abrir três terminais e executar os seguintes códigos na sequência em cada terminal:

roscore

para executar o ROS *Master*, obrigatório em toda operação envolvendo ROS.

rosrun dynamic_interface server

para poder executar o nó *server* que recebe todos os parâmetros provindo da interface e, portanto, contém as opções disponibilizadas nela.

rosrun rqt_reconfigure rqt_reconfigure

para abrir a própria ferramenta do *dynamic_reconfigure* já com o nó *server* disponibilizado na lateral esquerda da interface (por causa do código anterior em execução).

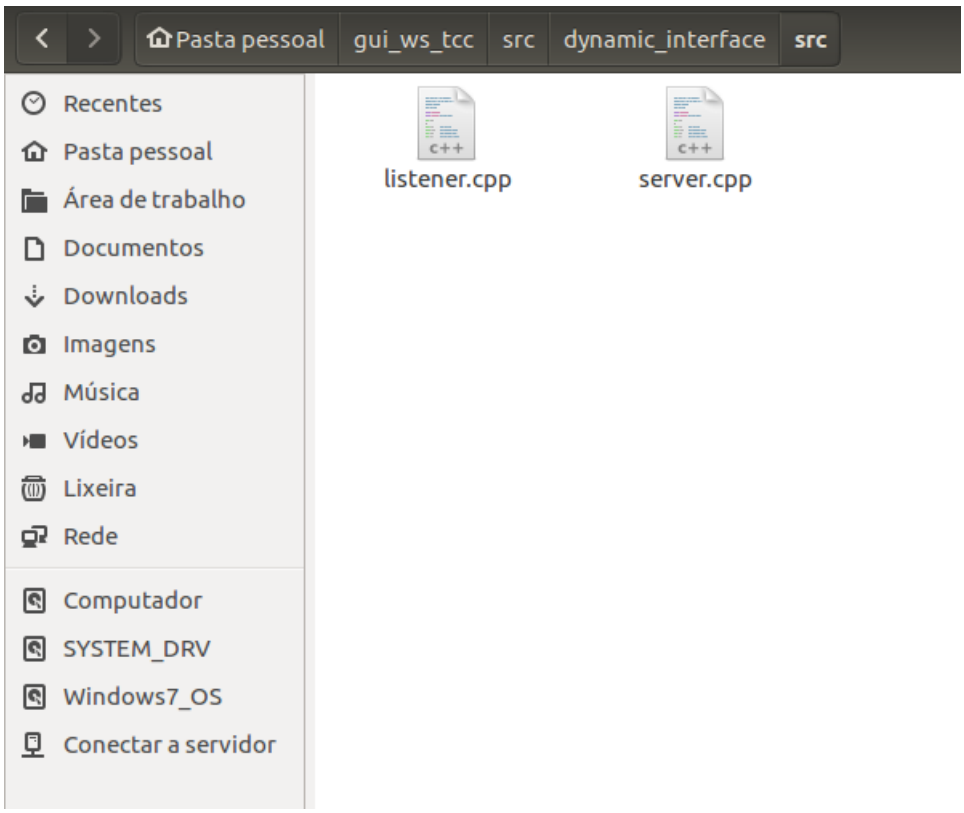
E a segunda opção, sendo a mais viável, consiste em criar uma pasta dentro do pacote chamada *launch* e criar um arquivo *.launch* (nomeado como *server.launch* para este trabalho) com conteúdos referentes ao ROS *launch* para executar os três códigos acima (e outros que serão apresentados posteriormente) com um único terminal e um único código:

roslaunch dynamic_interface server.launch

4.1.2 Nós do Pacote

Dentro da pasta do pacote em questão, foram criados dois nós, com um deles já comentado anteriormente: *server.cpp* e *listener.cpp*. O primeiro é responsável por receber a *message* enviada através de um tópico pela interface, descompactar essa mensagem em variáveis e compactar a mensagem em uma nova *message* e propagá-la em um tópico global, onde tanto o *rosserial* do Arduíno pode receber quanto o nó auxilar *listener.cpp*. Além disso, também é responsável por ler os dados da mensagem e escrever/*"printar"* no terminal onde está sendo executado o ROS *launch*. Já o último nó foi utilizado apenas para fins de testes e ele não se encontra declarado dentro do arquivo *CMakeLists.txt* (arquivo fundamental onde se declara todas os executáveis, nós, dependências, mensagens etc).

Figura 32 – Nós presentes dentro do *workspace* utilizado no pacote *dynamic_interface*



Fonte: Própria

O tipo de variável da mensagem lido pelo nó *server.cpp*, dentro de uma função de retorno *callback*, é *dynamic_interface::dyn_interfaceConfig* (este nome é formado primeiramente pelo nome do pacote e pelo nome do arquivo cfg) e ela foi criada à partir da extensão *dynamic_reconfigure*. Já a mensagem que este nó envia (portanto, ele atua como um *publisher*) foi criada à partir de um arquivo *.msg* e colocada dentro do pacote *dynamic_interface* e seu objetivo é repassar todas os parâmetros desmembrados para que facilite leitura. A Figura 34 mostra as variáveis e o formato da mensagem.

Figura 33 – Nó *server.cpp* escrevendo as mensagens lidas no terminal

Fonte: Própria

Observe na Figura 34 que cada parâmetro tem seu tipo de valor agregado, isto é, o dado armazenado pode ser do tipo int, float64 ou bool.

Um outro ponto importante a ser citado é o uso das estruturas *if*, *else if* e *else*, padrões da programação em C++, dentro do nó *server.cpp*. Seu uso é fundamental para que o botão de enviar dados seja retirado de seleção após ser clicado; e para que os botões dentro do modo Referência e do modo Malha-fechada não sejam selecionados simultaneamente. ou seja, ao selecionar uma opção, a outra fica automaticamente bloqueada para seleção e vice-versa.

Estes comandos funcionam, respectivamente, zerando o parâmetro *enviar_dados* após ter se tornado verdadeiro com a seleção do botão pelo usuário; e, caso um parâmetro de qualquer um dos dois modos disponíveis seja verdadeiro, os parâmetros referentes aos outros modos se tornam zero, por definição do *if*. Tais estruturas podem ser visualizadas na Figura 35.

4.1.3 Traço de Gráficos

Para o *plot* de gráficos, é necessário que o Arduíno envie dados constantemente através de tópicos para o ROS do Ubuntu, tornando o Arduíno um *subscriber*. Os dados enviados (mais detalhes na subseção 4.1.4) são utilizados pela extensão **rqt_plot** do ROS para mostrar em tempo real a variação destes parâmetros.

Para a execução dessa ferramenta, é necessário a utilização do código em um novo terminal:

Figura 34 – Arquivo *mensagem.msg*, que representa as variáveis e o formato da mensagem enviada do nó *server.cpp* para a rede

```
 1 uint8 Estado_motor
 2 uint8 Sentido_Rotacao
 3 uint8 Perfil_aceleracao
 4 uint8 Frequencia
 5 bool Senoidal
 6 bool Degrau
 7 bool Controle_posicao
 8 bool Controle_velocidade
 9 float32 Tempo_aceleracao
10 float32 Amplitude
11 float32 Periodo
12 float32 Offset
13 float32 Modulo
14 float32 Pp
15 float32 Ip
16 float32 Dp
17 float32 Pv
18 float32 Iv
19 float32 Dv
```

Fonte: Própria

rqt_plot

Após sua inicialização, dentro da aba referente a esta ferramenta, é necessário selecionar os tópicos em que os dados estão sendo transmitidos. Pode-se ver todos os dados em uma única janela, para efeitos de comparação, ou pode-se visualizar os dados em janelas diferentes abrindo diversos terminais e executando o código acima. A Figura 36 mostra a janela do *rqt_plot* sem nenhum tópico selecionado, sendo possível selecioná-los através do espaço *Topic* e clicando no "+"verde.

Para facilitar a execução automática desta ferramenta, o código acima foi adicionado dentro do arquivo *server.launch* colocando todos os tópicos enviados pelos Arduíno como argumentos da função. Assim, duas janelas serão abertas já com os tópicos selecionados.

4.1.4 Código Arduino

Toda a escrita e compilação do código presente no módulo Arduíno foram desenvolvidos e manipulados dentro do editor de código-fonte / IDE *Visual Studio Code*, da Microsoft. Dentro dele, foi utilizada a extensão PlatformIO IDE, necessária para a organização das bibliotecas e do código principal (*main*) no formato aceito pelo Arduíno. Assim, as pastas para código-fonte

Figura 35 – Parte do código do nó *server.cpp*, com ênfase na estrutura condicional *if*, *else if* e *else*

```
48 if(config.Senoidal == true)
49         {
50         config.Degrau = false;
51         config.Controle_posicao = false;
52         config.Controle_velocidade = false;
53 
54         if(config.Enviar_dados)
55         {
56           ROS_INFO("Dados enviados: %s %s %s %f %s %s %s %s %f %f %f %d",
57         config.Estado_motor?"Motor_ON":"Motor_OFF",
58             config.Sentido_Rotacao?"Sentido_Horario":"Sentido_AntiHorario",
59             config.Perfil_aceleracao?"Rampa_Normal":"Curva_S",
60             config.Tempo_aceleracao,
61             config.Senoidal?"Senoidal_selecionado":"-",
62             config.Degrau?"Degrau_selecionado":"-",
63             config.Controle_posicao?"Posicao_controlada":"-",
64             config.Controle_velocidade?"Velocidade_controlada":"-",
65             config.Amplitude,
66             config.Periodo,
67             config.Offset,
68             config.Frequencia);
69 
70                 config.Enviar_dados = false;
71 
72                 chatter_pub.publish(msg);
73 
74         }
75 
76         }
77 
78 else if(config.Degrau == true)
79         {
80         config.Senoidal = false;
81         config.Controle_posicao = false;
82         config.Controle_velocidade = false;
83 
84         if(config.Enviar_dados)
85         {
```

Fonte: Própria

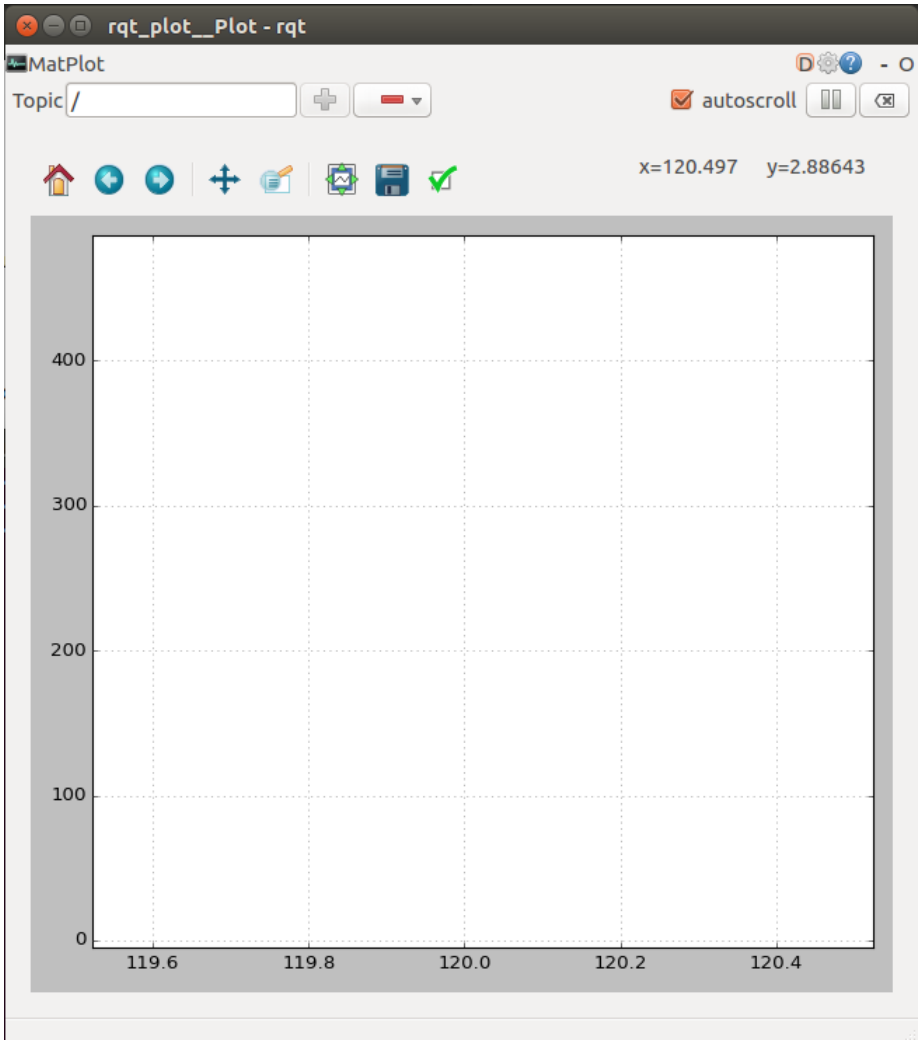
(*src*) e bibliotecas (*lib*) ficam organizadas dentro de uma pasta "projeto", que foi intitulado como *Gui_Arduino_Interface*.

Uma ação necessária para haver a comunicação via USB entre o ROS do Ubuntu e a extensão do ROS presente no microcontrolador é a instalação do pacote *rosserial* (mesma biblioteca utilizada no Arduíno) dentro do *workspace* em que se encontra o pacote *dynamic_reconfigure*. Nele são estabelecidos os tipos de mensagens enviadas e recebidas assim como a identificação e a configuração da porta USB utilizada.

Após a instalação através do comando ***catkin_make***, toda vez que se queira fazer a comunicação entre o Mega e Ubuntu é necessário executar um comando no terminal para que a porta USB seja aberta para a troca de mensagens do ROS:

rosrun rosserial_python serial_node.py /dev/ttyUSB0

Figura 36 – Estrutura do *rqt_plot* sem nenhum tópico selecionado



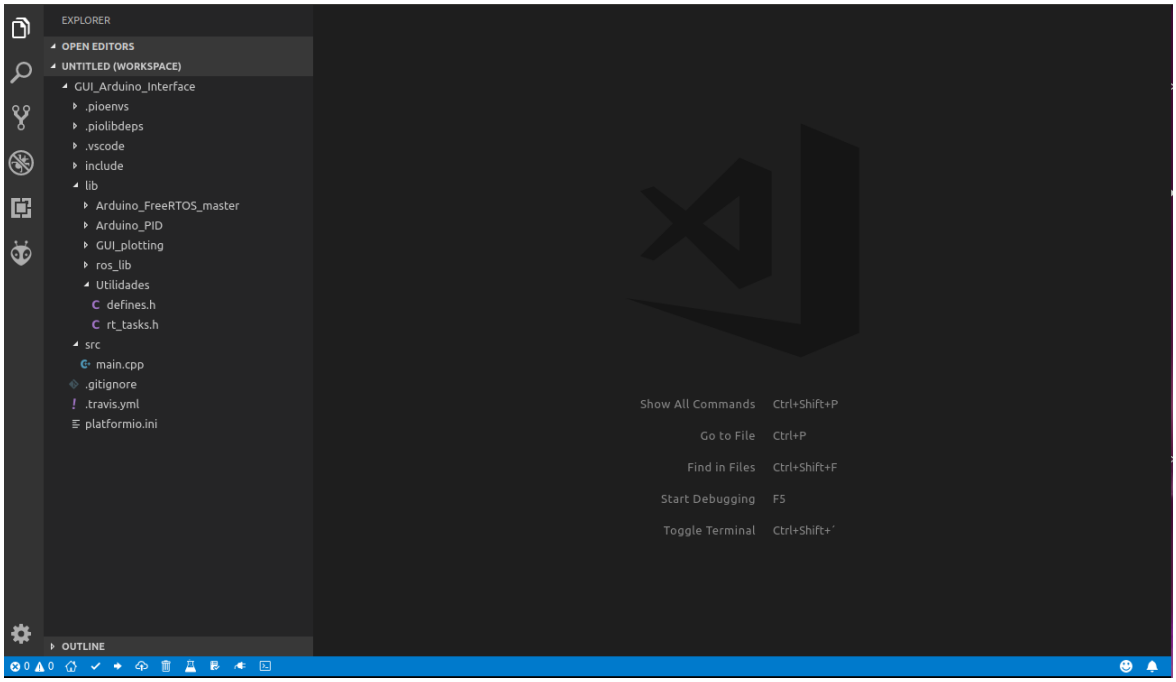
Fonte: Própria

onde ttyUSB0 deve ser trocado pelo nome da porta USB onde o Arduíno está conectado. No caso do computador utilizado para a construção do projeto, a porta utilizada era **ttyACM0**. E, para evitar que este código seja aberto sempre em um terminal a parte, ele foi acrescentado dentro do arquivo *server.launch*, conforme apresentado anteriormente.

4.1.4.1 Bibliotecas *rosserial* e *ros_lib*

Conforme já citado anteriormente, esta biblioteca é necessária para que o Arduíno possa utilizar as ferramentas do ROS dentro de sua programação. Após ter sido colocada na pasta *lib* do *Gui_Arduino_Interface*, foi necessário fazer a compilação análoga ao realizado no *rosserial* presente no *workspace* do Ubuntu para que o formato do arquivo *message.msg* pudesse ser lido pelo Arduíno. O código seguinte foi utilizado dentro da pasta *ros_lib* do *Gui_Arduino_Interface* para que a mensagem fosse compilada (analogamente, foi feito antes para o pacote do *workspace*):

Figura 37 – Árvore de pastas do projeto Arduíno_interface dentro do *software* Visual Studio Code



Fonte: Própria

rosrun rosserial_client make_libraries path_to_libraries

Com o Arduíno conseguindo receber e desmembrar a *message* através de uma função do tipo *callback*, foi possível torná-lo um *subscriber*, onde cada parâmetro recebido e que estava presente no *buffer* é armazenado em uma variável local para que seus dados possam ser analisados e direcionadas para as devidas ações.

Além disso, o Arduíno foi transformado também em um *publisher*, ao publicar tópicos contendo dados importantes para a análise do sistema, divididos por funções exercidas:

Traçar gráficos de velocidade e posição pelo tempo Para tais funções foi necessário enviar a velocidade e posição lido pelo sensor e a velocidade e posição de referência (que corresponde aos valores enviados do *dynamic_reconfigure*);

Análise do controle PID Foram necessários ser enviados os valores de *input*, *output* e *setpoint* que correspondem, respectivamente, ao valor lido do sensor de luminosidade (que representa o *encoder*), o valor gerado pela malha-fechada para servir de ajuste (que definirá a luminosidade do LED) e o valor de luminosidade requisitada pelo usuário a ser controlada pela iluminação do LED.

Tabela 4 – *Tasks* e as funções que o Arduíno Mega desempenha simultaneamente

Nome da Tarefa	**Função**
vUSBTask	Recebe e envia dados do ROS no Ubuntu
vControlTask	Controle de malha-fechada da posição e velocidade
vReadingSensorTask	Comanda e lê os dados provindos do Inversor e Encoder

Fonte: Elaborado pelo autor.

Tabela 5 – *Tasks* e suas respectivasinformações de construção

Nome da Tarefa	**Frequência (Hz)**	**Prioridade**	**Tamanho de Alocação (Bytes)**
vUSBTask	50	Alta	700
vControlTask	1000	Máxima	100
vReadingSensorTask	200	Alta	700

Fonte: Elaborado pelo autor.

Tais dados são enviados separadamente em cada tópico e são "escutados"pelo *rqt_plot*, conforme mencionado anteriormente. Cada tópico só é enviado mediante a seleção do modo de atuação (Referência ou Malha-fechada) e estes se encontram dentro de *tasks* do RTOS.

4.1.4.2 Biblioteca *FreeRTOS*

Conforme foi citado anteriormente, o uso da biblioteca de RTOS é de extrema importância neste trabalho, visto que permite que o microcontrolador realize diversas tarefas com a frequência que o programador escolher e sem que uma *task* atrapalhe a outra (caso não ocorra uso excessivo da capacidade do microcontrolador).

Para isso, foi instalado a biblioteca chamada *FreeRTOS* (ou Arduino_FreeRTOS) para que todas estas funcionalidades estejam disponíveis. Assim, foi possível criar três principais *tasks* com graus de prioridades diferentes, que executam ações diferentes uma da outra e possuem frequências de funcionamento diferentes. A Tabela 4 mostra o nome e a função de cada *task* implementada dentro do código do Arduíno. Já a Tabela 5 mostra dados construtivos destas *tasks*.

Um ponto importante a ser revelado é que estas funções não puderam ser exercidas, de fato, visto que os dispositivos não foram comprados a tempo para sua implementação. Para isso, algumas ações foram implementadas como uma forma de representar estas funções. A Tabela 6 mostra as reais ações implementadas dentro de cada *task* presente no Arduíno. Nas próximas seções será explicado com mais detalhes o funcionamento de cada umas das funções dentro do RTOS.

Dentro deste assunto, é importante ainda ressaltar a implementação de uma ferramenta para verificação de excesso de frequência requisitada, ou seja, verificar qual a máxima frequência que a *task* suporta sem apresentar *overrun* ou não-cumprimento da ação dentro do tempo previsto.

Tabela 6 – *Tasks* e as reais ações implementadas

Nome da Tarefa	Ação Implementada
vUSBTask	Recebe tópicos do Ubuntu , define e envia novos tópicos de volta
vControlTask	Realiza o controle de PID da luminosidade captada pelo sensor através de LED
vReadingSensorTask	Acende um outro LED. Sua interrupção indica erro no sistema

Fonte: Elaborado pelo autor.

A Figura 38 mostra a estrutura citada para o caso aplicado da *task* vUSBTask. Para as frequências adotadas, as *tasks* aconteceram sem nenhum problema e, portanto, esta ferramenta não acusou erro durante os experimentos.

Figura 38 – Ferramenta auxiliar para verificação de frequência das *tasks*

```
//Auxiliary variable to check task frequency
overruns = tickTime - ((long)xLastWakeTime + (long)taskPeriod_ticks_USB);
if(overruns > 0) //If task did not manage to finish in time
{
 usb_task_overruns += overruns;
}
else
{
 vTaskDelay(taskPeriod_ticks_USB);
}
```

Fonte: Própria

4.1.4.3 Biblioteca *PID*

Para o uso desta função, foi necessário instalar a última biblioteca que compõem o arsenal do Arduíno: *Arduino_PID*. Novamente, é um biblioteca disponibilizada *open source* para usos voltados para controle por PID de sistemas de malha-fechada.

Para seu uso, inicialmente, é necessário a criação de variáveis do tipo *double* que irão receber os dados: *input*, *output* e *setpoint*, já citados anteriormente. Também é necessário criar variáveis do tipo *double* que irão alojar os valores dos ganhos: para este caso, foi dado os nomes Kp_double, Ki_double e Kp_double para o ganho proporcional, integral e derivativo, respectivamente.

Na sequência, é necessário criar uma instância local do PID utilizando o formato *PID* padrão da biblioteca onde são enviados os endereços das três primeiras variáveis apresentadas

acima, os valores dos três ganhos e um comando padrão da função. E, dentro da *task* vControlTask, é realizado o cálculo do PID através da função *Compute()* e, então, é disponibilizado a saída (*output*) para controle.

4.1.4.4 Arquivo *defines.h*

Finalmente, mas não menos importante, foi criado um arquivo de extensão *.h* que aloja diversas variáveis globais que são utilizadas em mais de uma biblioteca e/ou arquivo dentro do projeto *Gui_Arduino_Interface*. Dentro dela estão os valores das prioridades utilizadas nas *tasks*, o valor dos tamanhos para os *stacks sizes*, entre outras propriedades.

A Figura 39 e a Figura 40 mostram parte do código utilizado, com ênfase nos parâmetros usados dentro do RTOS do código principal do Arduíno.

Figura 39 – Frequências das *tasks* para o RTOS

```
/*************************************************************/
/*               Real time Tasks Frequencies                 */
/*************************************************************/

//Main system frequency (default = 30 kHz)
#define MainSysFreq             30000


//Loadcell oversampling frequency (default = main system frequency)
#define OversamplingFreq        MainSysFreq


//Sensor reading frequency (default = 200 Hz)
#define SensorFreq                  200

//Control loop frequency (default = 1 kHz)
//It measures sensors (apart from load cell, which is measured faster), calculates control commands, and sends them.
#define ControlLoopFreq         1000
#define ControlLoopFreqInterface        50
#define ControlLoopFreqSensor           200

//Oversampling/ControlLoop frequency ratio (default = 30 kHz / 5 kHz = 6)
#define OversamplingFreqRatio   (OversamplingFreq / ControlLoopFreq)


//Communication with PC frequency (default = 50 Hz)
//It sends data to the GUI, and receive commands and control desired values
//TODO: in case this freq changes, it needs to be changed also in publisher.cpp
#define CommunicationWithPcFreq 50


//Status LEDs blinking frequency
#define LEDBlinkingFreq         1
```

Fonte: Própria

4.2 Projeto Concluído

Após diversos testes realizados de etapa em etapa para averiguar e comprovar o funcionamento de cada um dos subsistemas dos trabalho, foi possível finalizar o projeto dentro do prazo previsto e com todas as características desejadas já comentadas anteriormente. A Figura 41 e Figura 42 mostram a interface final do *dynamic_reconfigure* com os novos parâmetros embutidos.

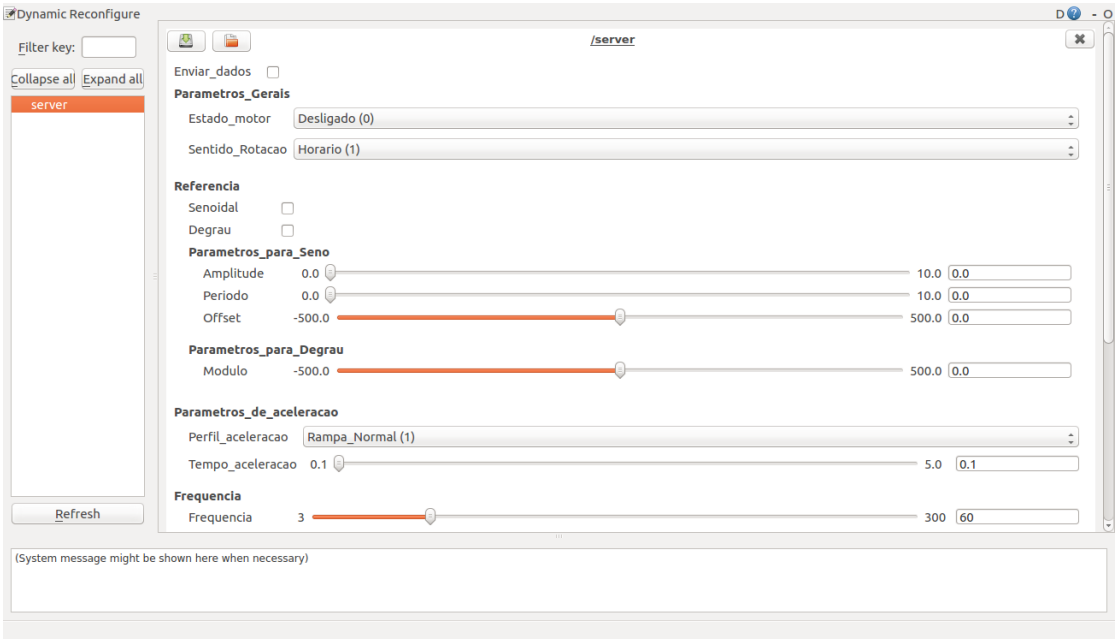
Figura 40 – Prioridades e *stack Sizes* das *tasks* para o RTOS

```
/*********************************************************************/
/*          Real time Tasks Priorities and Stack Sizes               */
/*********************************************************************/

#define MINIMUM_PRIORITY    0
#define LOW_PRIORITY        1
#define MEDIUM_PRIORITY     2
#define HIGH_PRIORITY       3
#define MAXIMUM_PRIORITY    4

#define STACK_MINIMUM_SIZE  100
#define STACK_NORMAL_SIZE   200
#define STACK_LARGE_SIZE    300
#define STACK_MEGA_SIZE     500
```

Fonte: Própria

A interface inteira não coube dentro da tela do computador, por isso foi apresentada em duas imagens separadas.

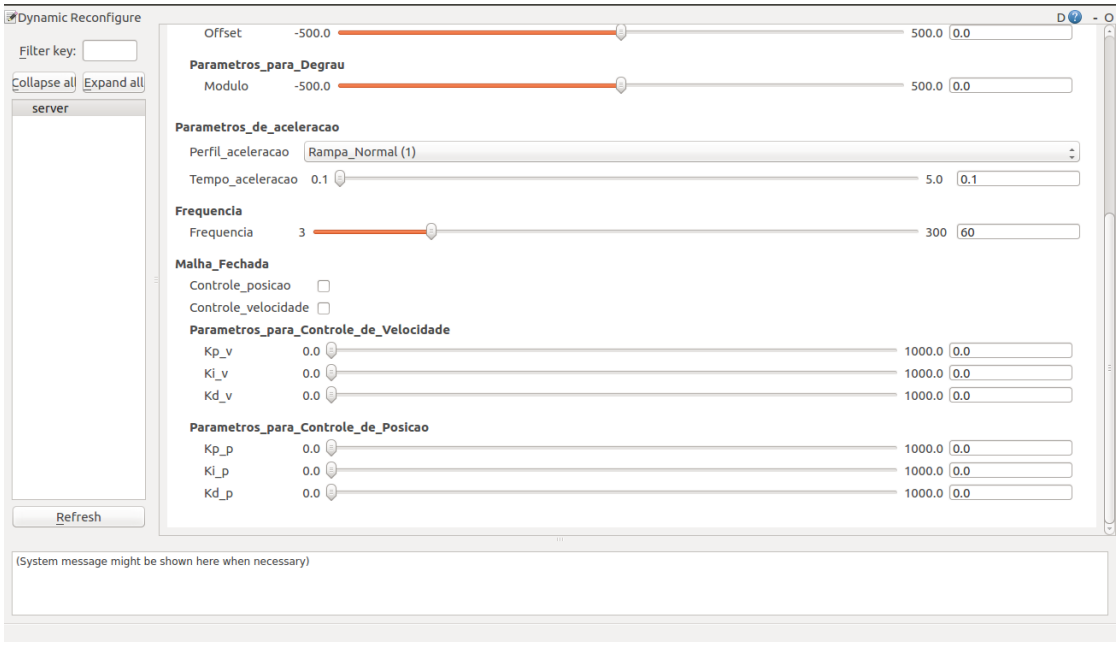
Figura 41 – Versão Final da interface em *dynamic_reconfigure*, parte 1



Fonte: Própria

Utilizando o ROS *launch* é possível, com apenas uma linha de código no terminal, executar todos os nós disponíveis e extensões necessários para a visualização completa da interface. O código citado é:

Figura 42 – Versão Final da interface em *dynamic_reconfigure*, parte 2



Fonte: Própria

roslaunch dynamic_interface server.launch

sendo a primeira estrutura padrão do ROS *launch*, a segunda é o nome do pacote principal e a terceira é o nome do arquivo *.launch*. A Figura 43 mostra o arquivo *server.launch* e todos os nós executados.

Figura 43 – Arquivo *server.launch* do ROS *launch*, que é executado pelo comando rápido

```
<launch>
        <node name="server" pkg="dynamic_interface" type="server" output="screen"/>
        <node name="rqt_reconfigure" pkg="rqt_reconfigure" type="rqt_reconfigure"/>
        <node name="serial_node" pkg="rosserial_python" type="serial_node.py" output="screen" args="/dev/ttyACM0"/>
        <node name="velocidade_plot" pkg="rqt_plot" type="rqt_plot" args="/arduinochattervelocidade /arduinochattervelocidaderef"/>
        <node name="pid_plot" pkg="rqt_plot" type="rqt_plot" args="/arduinochatterinput /arduinochatteroutput /arduinochattersetpoint"/>
</launch>
```

Fonte: Própria

À partir da Figura 43 é possível perceber que, ao todo, são executados cinco nós/extensões disponíveis. São elas:

a) O nó *server.cpp*, que atua como *publisher()* e *subscriber()*;

b) A extensão correspondente à interface do *dynamic_reconfigure*;

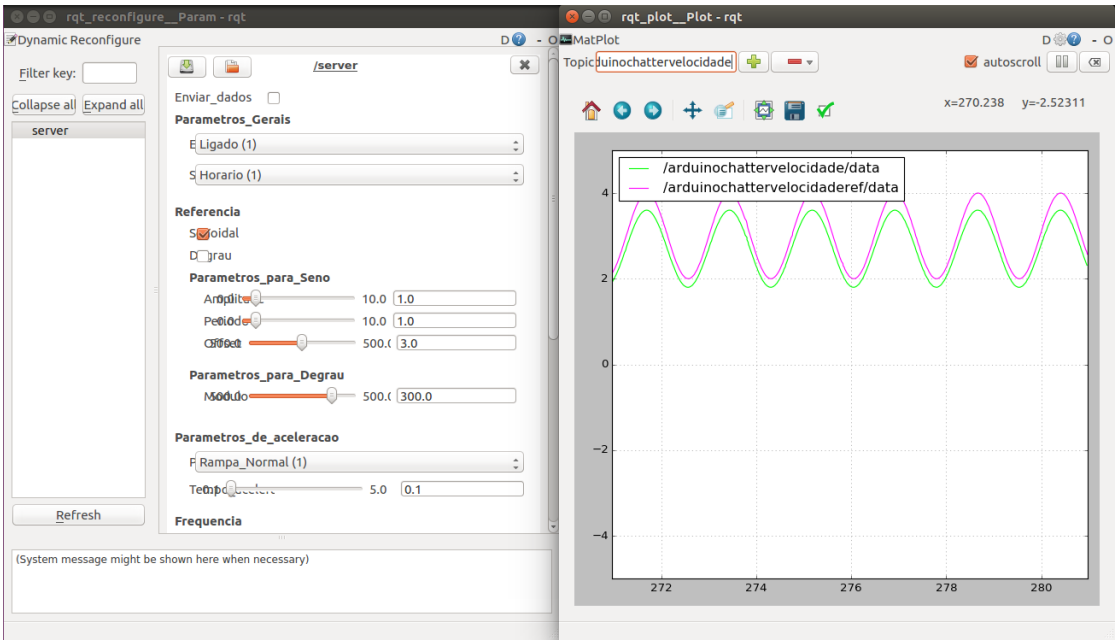
c) O nó responsável pela abertura e configuração da porta USB para o Arduíno Mega;

d) A extensão *rqt_plot* com dois tópicos como entrada: referentes à velocidade real e à velocidade de referência, respectivamente;

e) A extensão *rqt_plot* com três tópicos como entrada: referentes ao valor do *input*, *output* e *setpoint* do controle por PID;

Dentro da interface final do *dynamic_reconfigure* visto em Figura 41 e Figura 42, ao selecionar qualquer uma das quatro opções de modos (duas para Referência e duas para Malha-Fechada), uma de cada vez, obtém-se os gráficos plotados para cada. A Figura 44 mostra o gráfico da velocidade real e da velocidade de referência (aquela determinada pelo usuário) variando de forma senoidal e Figura 45 mostra para os mesmos parâmetros porém contínuos buscando representar o efeito degrau. Já a Figura 46 mostra os parâmetros do controle da posição/velocidade por PID, variando conforme ocorre distúrbios ou variações no sistema. Na seção 4.3 será detalhado o que cada um desses três parâmetros significam em um caso aplicado.

Figura 44 – Divisão de telas entre a seleção de parâmetros (amplitude, período e *offset*) dentro do *dynamic_reconfigure* e o *plot* do gráfico da velocidade real e a de referência variando de forma senoidal

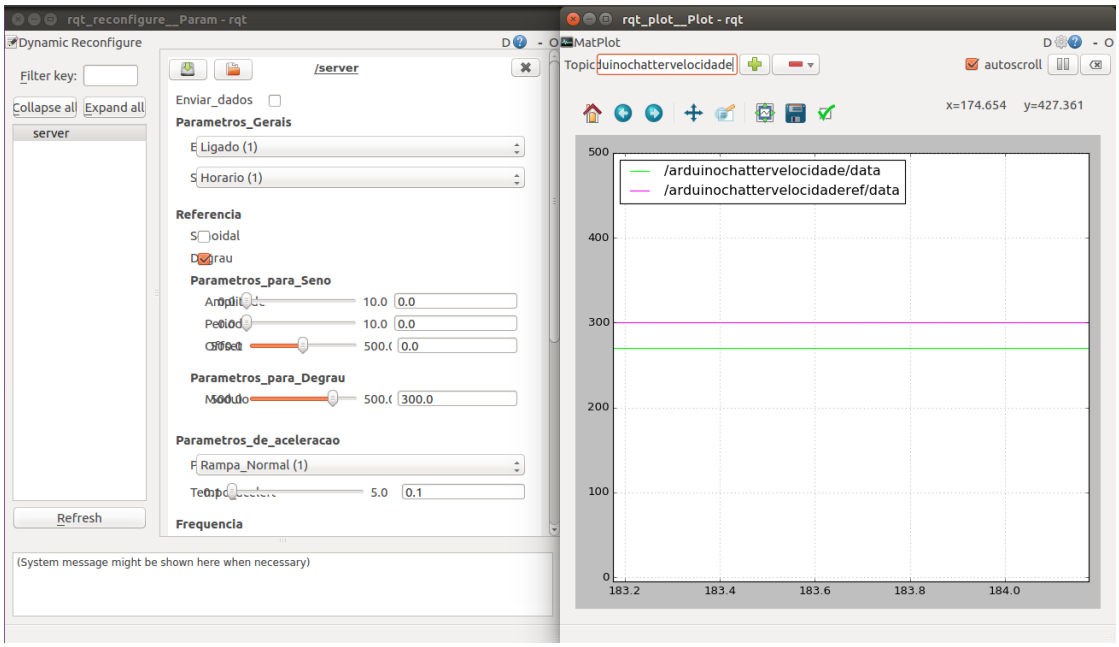


Fonte: Própria

Para configurar os eixos *x* e *y*, o intervalo apresentado na tela, a curva e outros fatores, é disponível alguns ícones dentro do *rqt_plot* para o usuário customizar conforme a necessidade. A Figura 47 mostra o *range* dos eixos sendo customizado na aba *Figure Options*.

Finalmente, é importante ressaltar que ainda dentro da interface do *rqt_plot* há algumas opções a serem comentadas: o símbolo da casa indica o retorno à visão sem *zoom* do gráfico; as setas indicam o retorno para a vista anterior e o avanço para a próxima; a cruz é um acessório

Figura 45 – Divisão de telas entre a seleção do parâmetro (módulo) dentro do *dynamic_reconfigure* e o *plot* do gráfico da velocidade real e a de referência variando de forma "degrau"



Fonte: Própria

para movimentar o gráfico com o mouse do computador para a seção desejada; a caixa de seleção é para arrastar com o mouse e selecionar a seção de *zoom* desejada; por fim, os três últimos símbolos são, respectivamente, para configurar os eixos (Figura 47), salvar a imagem instantânea do gráfico e selecionar o modo de plotagem. Também pode-se selecionar ou não a opção de *autoscroll* no canto superior direito.

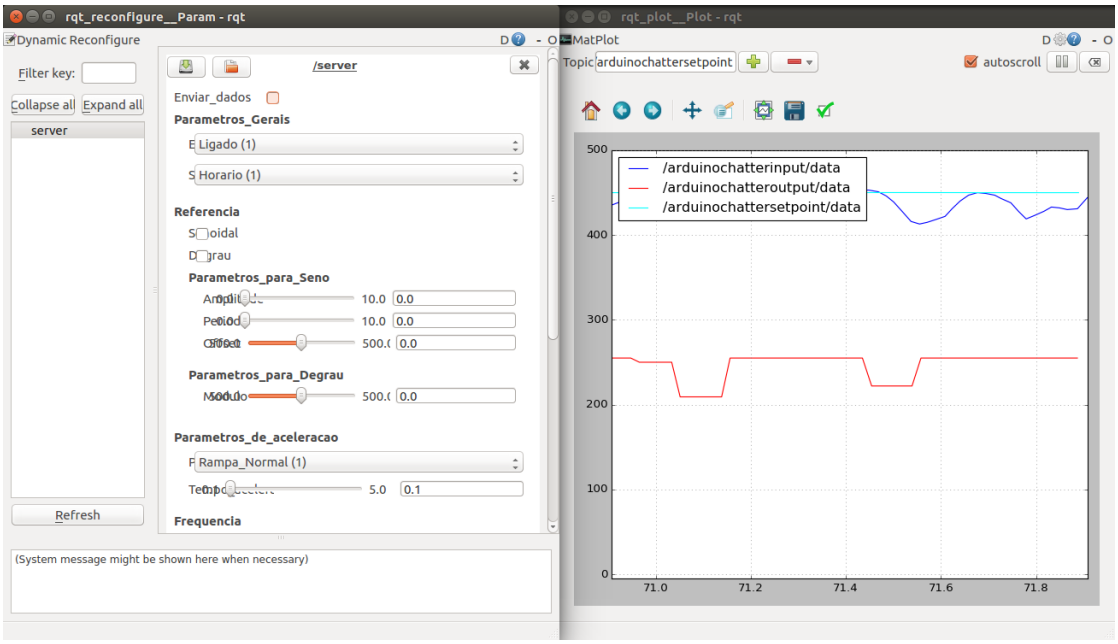
4.3 Experimentos aplicados

Ao todo, foram aplicados três experimentos demonstrativos para comprovar e ilustrar o funcionamento de todo o sistema/projeto como um todo. Estes foram construídos e elaborados de forma a abranger o três pontos principais: recebimento de informações provindas da interface do *dynamic_reconfigure*, envio de dados em forma de tópicos sobre o perfil senoidal da velocidade (já abrangendo o caso do Degrau) e envio de dados em forma de tópicos sobre o controle de PID realizado com circuito em *protoboard*.

4.3.1 Comprovação de Recebimento de dados

Para comprovar que o Arduíno Mega está recebendo dados provindo da interface do *dynamic_reconfigure* selecionados pelo usuário, ao clicar na caixa de enviar dados, foi configurado dentro da programação do Arduíno para que ele acenda o LED embutido em sua placa (porta

Figura 46 – Divisão de telas entre a seleção de parâmetros (Kp, Ki e Kd) dentro do *dynamic_reconfigure* e o *plot* do gráfico do *input*, *output* e *setpoint*, tanto para o controle de posição quanto para o de velocidade



Fonte: Própria

digital 13) caso o valor do tempo de aceleração seja igual a 1.0 (um) segundo.

Para realizar esta função, dentro da função *Callback* (chamada de chatterCallback) foi colocado uma estrutura *if* cujo argumento de análise é lacuna referente ao tempo de aceleração ser igual a um segundo. E, caso seja verdadeiro, utilizando a linguagem própria do Arduíno, o LED 13 é ligado (digitalWrite(13, HIGH)), coloca-se um tempo de atraso (*delay*) de 1000 milissegundos ou 1 segundo e depois ele é desligado (digitalWrite(13, LOW)).

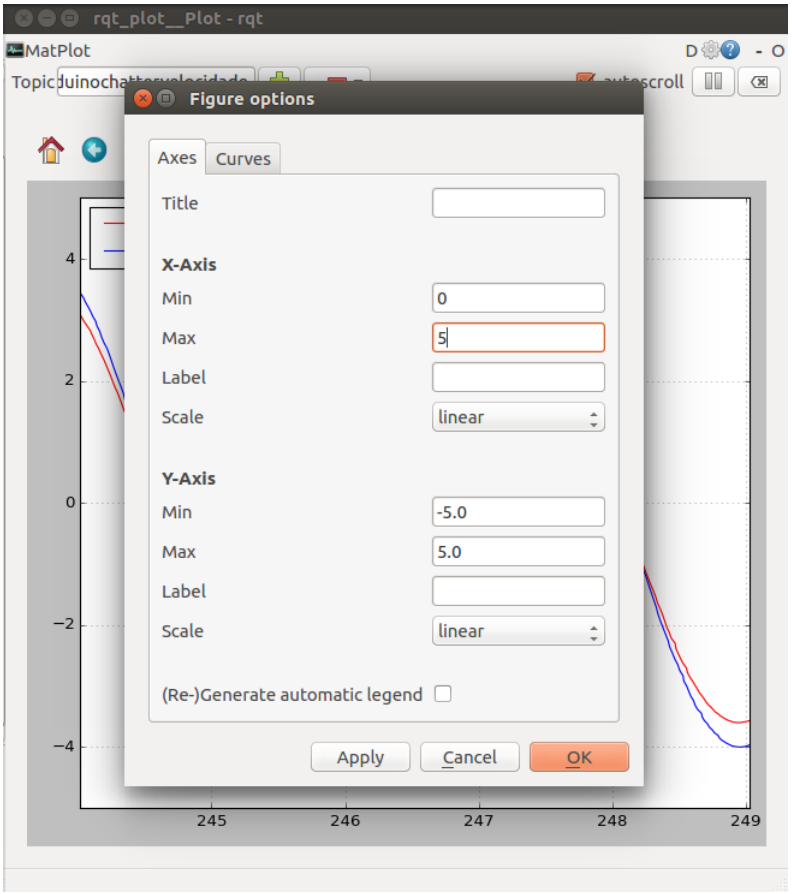
Observe a LED laranja acesa na Figura 48, indicando o recebimento a mensagem. O LED verde também presente na Figura indica que o Arduíno está *online* e conectado na alimentação.

4.3.2 Envio de tópicos para perfil senoidal de velocidade

Para este experimento, não foi necessário nenhuma montagem elaborada fisicamente: foi necessário apenas utilizar a função *sin()* da biblioteca padrão do Arduíno para que os tópicos fossem plotados em formato senoidal dentro do *rqt_plot*. Esta função é calculada para amplitude igual a 1, como padrão; foi somado o valor do *offset* para que o gráfico oscile em torno de outras posições além do zero; e foi necessário variar o ângulo (em radianos) em um *loop* onde o ângulo é incrementado de 0.01 em 0.01 radianos, obtendo-se um gráfico contínuo.

E para demonstrar a diferença entre a velocidade real e a velocidade de referência (ou seja, o atraso da resposta em relação ao sinal), optou-se por colocar a velocidade real como 90% da

Figura 47 – Aba *Figure Options* do *rqt_plot* sendo customizada



Fonte: Própria

velocidade de referência configurado pelo usuário através da interface do *dynamic_reconfigure*.

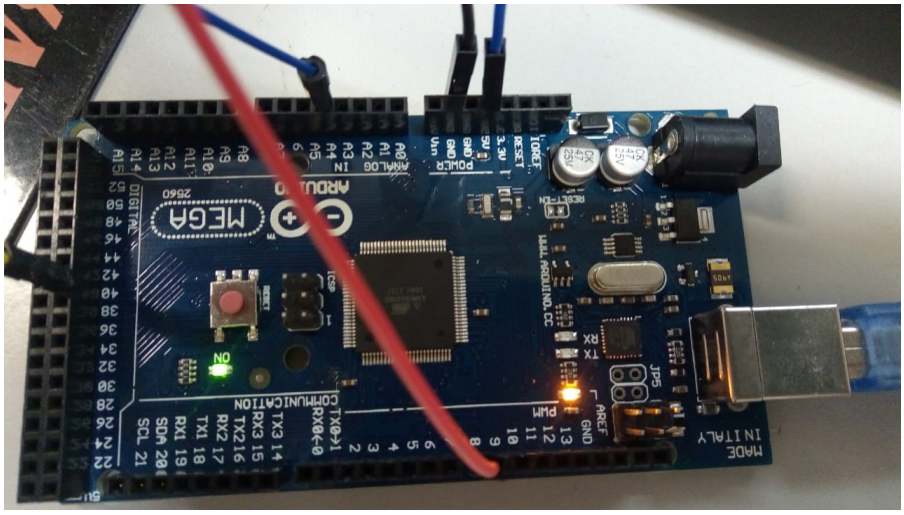
A Figura 49 mostra o LED Tx laranja aceso, mostrando que está ocorrendo transmissão de dados do Arduíno para outros dispositivos (no caso, por USB para o Ubuntu). Os dados enviados, para este experimento, são a velocidade, posição, velocidade de referência e posição de referência.

4.3.3 Controle de PID com circuito em *protoboard*

A ideia principal deste experimento é testar o funcionamento da biblioteca PID específica para o Arduíno através do uso de um sensor de luminosidade e um LED vermelho; e enviar os dados gerados como tópicos para o ROS do Ubuntu, que são plotados pelo *rqt_plot*. A visão geral do experimento pode ser vista pela Figura 50.

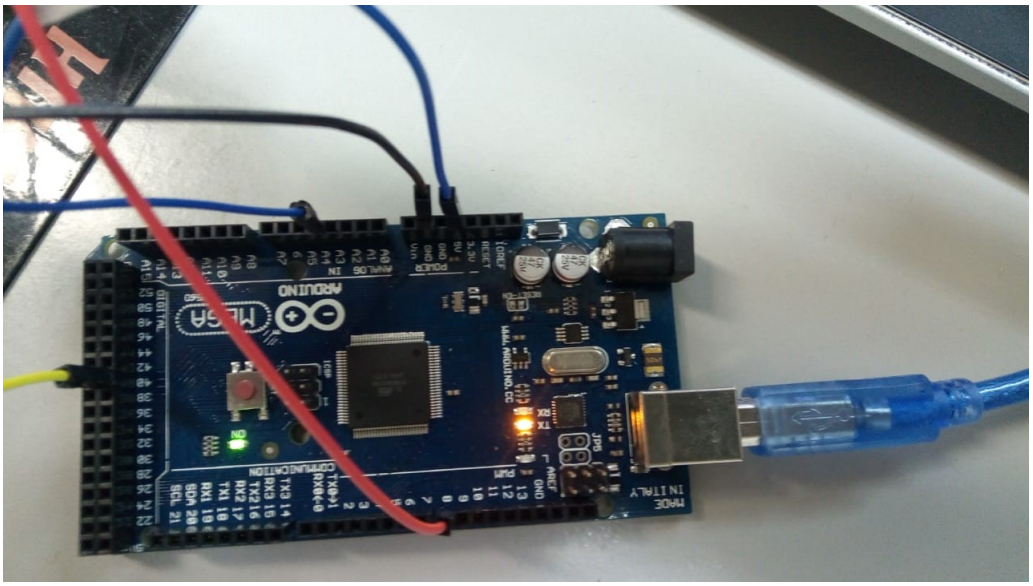
Todo o circuito eletrônico foi montado em uma *protoboard*: uma placa de com furos e conexões condutoras para montagem própria e prática para circuitos simples. Nela, foram utilizados fios condutores ("*jumpers*") para fazer a conexão entre os elementos, um sensor de

Figura 48 – LED 13, embutido na placa Arduíno Mega, acendendo conforme é enviado o parâmetro de tempo de aceleração com valor igual a 1.0 segundo



Fonte: Própria

Figura 49 – LED Tx aceso mostrando que o Arduíno está transmitindo dados

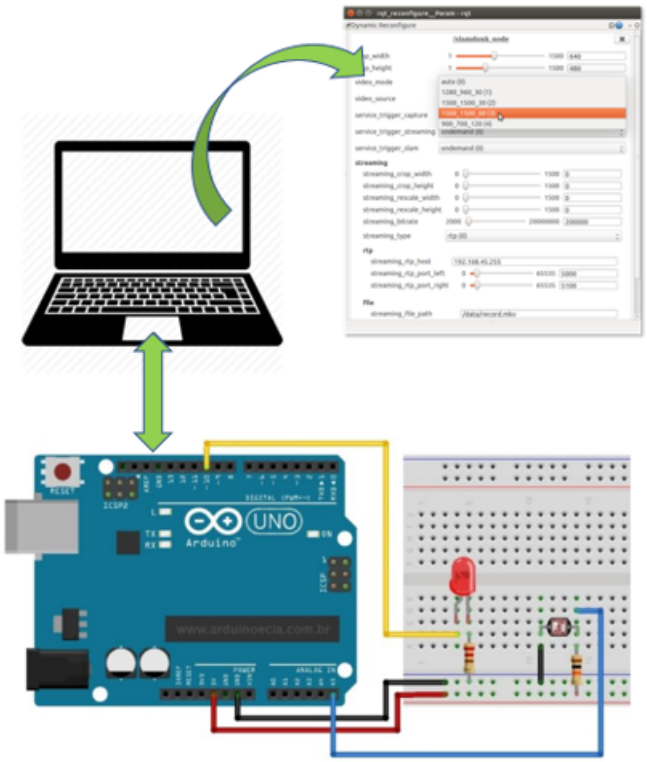


Fonte: Própria

luminosidade, um LED vermelho e um resistor de 10 quiloohm. A Figura 51 mostra a imagem real do circuito montado.

Ao selecionar a opção de controle de posição, o usuário tem que selecionar os valores dos ganhos proporcional, integral e derivativo, logo abaixo na interface. Além disso, por efeito de experimento, o valor do *setpoint* é igual ao dobro do valor da frequência selecionada pelo

Figura 50 – Visão Geral do experimento de controle de PID. Onde está escrito UNO entende-se por Mega 2560



Fonte: Própria

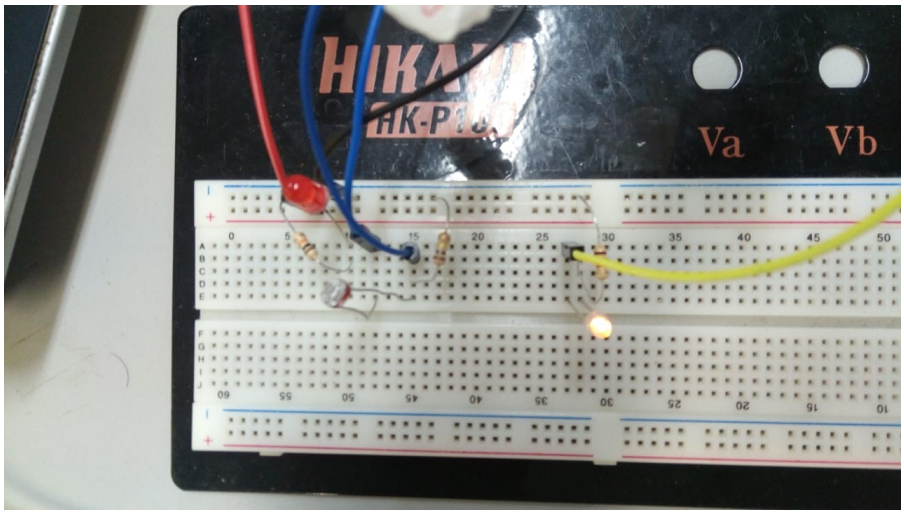
usuário. Portanto, $setpoint$ = 2 x $Frequencia$.

O motivo para este fato é que, ao selecionar o valor de frequência igual a 225, obtém-se o valor de 450 para o *setpoint*. Este valor foi testado em diversos ambientes iluminados por lâmpadas fluorescentes padrões da USP e percebeu-se que, com este valor, era possível apenas tampar o sensor de luminosidade parcialmente com as próprias mãos para se observar o LED vermelho acendendo (*output*) para que a luminosidade captada pelo sensor (*input*) se tornasse igual ao desejado (*setpoint*). O LED tem sua luminosidade controlada através do uso de PWM (*Pulse Width Modulation*), portanto acende conforme a intensidade da corrente de alimentação.

A Figura 52 mostra o gráfico de controle onde o *input* alcança o *setpoint* através da variação do *output* e a Figura 53 mostra este efeito na prática.

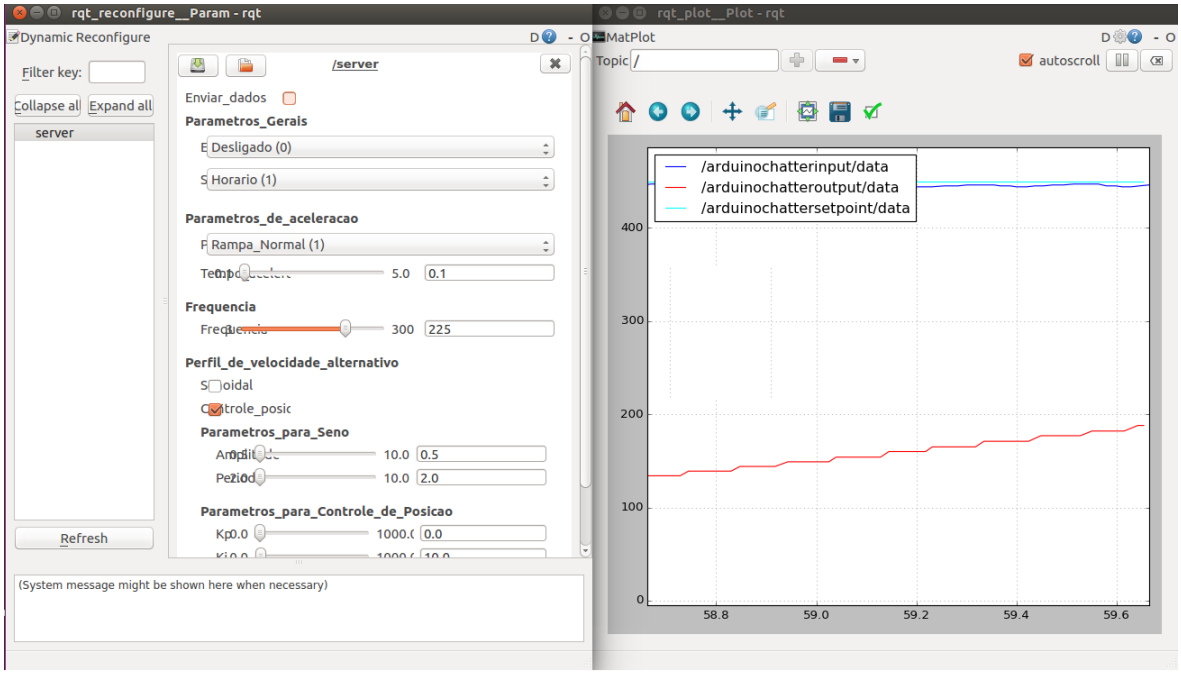
Por fim, é importante também comentar a adição de um LED branco adicional aos experimentos, porém independente a eles, para mostrar que o Arduíno está operando sem erros. Portanto, caso algum erro relacionado com o *FreeRTOS* esteja ocorrendo ou ocorra alguma interrupção ou variação na comunicação do sistema, o LED sofrerá variações. No caso da opção de controle PID estiver acionada, o LED passará a brilhar mais forte, por exemplo. Ele aceso pode ser percebido na Figura 51.

Figura 51 – Circuito eletrônico montado em *protoboard* para demonstração de funcionamento da interface



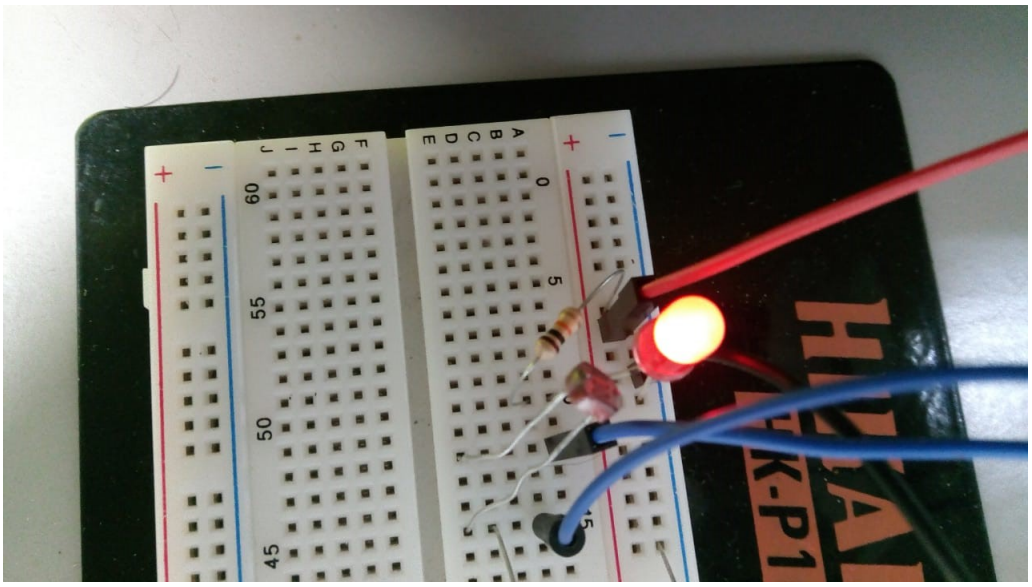
Fonte: Própria

Figura 52 – A variável *input* alcança o valor do *setpoint* selecionado pelo usuário, através da variação do *output*



Fonte: Própria

Figura 53 – LED vermelho acendendo para que o sistema entre equilíbrio, ou seja, para que a luminosidade captada pelo sensor seja igual ao valor de luminosidade selecionado pelo usuário



Fonte: Própria

5 CONCLUSÃO E TRABALHOS FUTUROS

Apesar de todos os equipamentos previstos para este projeto, como um todo, não terem sidos comprados a tempo para a montagem completa da bancada didática, um foco maior no desenvolvimento da interface gráfica para o controle do sistema foi de extrema importância para solidificar mais um sub-sistema do projeto e poderá contribuir para futuros estudos do próprio autor e para projetos de alunos e profissionais do âmbito acadêmico e profissional.

O assunto, apesar de não ser puramente mecânico conforme a formação do autor, contribuiu para a aquisição de novas ferramentas de programação, de desenvolvimento de *softwares* e de controle. Tais ferramentas são fundamentais nos dias atuais onde os métodos estão cada vez mais automatizados e digitais.

5.1 Pontos analisados

Para ter uma visão geral do projeto, é válido analisá-lo por pontos positivos e negativos, na busca de sugerir ao leitor o melhor uso do projeto assim como possíveis sugestões para melhorias. Os pontos principais de análise são: microcontrolador escolhido, código, *layout* da interface, troca de mensagens, experimentos e práticas.

5.1.1 Pontos positivos

a) O módulo definitivo para controle, o Arduíno Mega 2560 Rev3, foi capaz de desempenhar com êxito todas as tarefas propostas, isto é, referentes ao ROS, RTOS e controle por PID, dentro do prazo/frequência de execução;

b) O mesmo microcontrolador ATmega2560 possui memória mais que suficiente para alojar todas as bibliotecas e códigos dentro de sua memória interna;

c) A escolha de um microcontrolador presente na família Arduíno também facilitou a programação das funções visto que muitas bibliotecas já haviam sidas desenvolvidas por outros programadores;

d) O uso do editor *Visual Studio Code* facilitou a escrita do código durante todo o desenvolvimento do projeto, visto que é possível diferenciar por cores as classes de palavras presentes e é possível verificar informações extras sobre a palavra, como declaração e valor;

e) A interface disponibiliza ao usuário múltiplas opções de escolha assim como alto nível de controle dos parâmetros referentes à bancada didática;

f) O uso da biblioteca de RTOS (*FreeRTOS*) se apresentou uma boa alternativa para a execução de tarefas com diferentes frequências de acontecimento dentro do escopo

de tarefas em tempo real. A sua estrutura construtiva mostrou-se flexível, eficaz e de fácil manipulação ao longo do desenvolvimento e também apresentou-se compatível com os aspectos do próprio módulo Arduíno;

g) A simplificação da execução de diversos nós através de um único código, utilizando ROS *launch*, foi muito benéfica para a facilitação do uso da interface para o usuário;

h) Ao utilizar o ROS como ferramenta de construção da interface e para a troca de mensagens, o trabalho foi realizado com segurança e com fácil acesso para correções, visto que diversas bibliotecas, tutoriais e exemplos são disponibilizados abertamente na internet;

i) Os experimentos realizados foram de suma importância para comprovar o funcionamento de todo o projeto e seus materiais assim como sua implementação são de fácil acesso para todos os interessados;

j) As práticas propostas no Apêndice B são de fácil acesso para todos os usuários que possuam um dispositivo com Ubuntu (computador ou *Raspberry*);

5.1.2 Pontos negativos

a) Para implementações de mais funções dentro do código, como a Comunicação Serial entre o Arduíno e os dispositivos nele conectados (Inversor de Frequências e *Encoder* Absoluto), talvez seja necessário maior memória interna e maior velocidade de processamento. Portanto, a escolha de outro microcontrolador passa a ser uma opção;

b) Os três pontos principais do projeto (ROS, RTOS e PID) foram implementados em um único arquivo C++, o que dificulta sua visualização como um todo e a reparo de eventuais erros;

c) O *layout* da interface apresentou alguns nomes irregulares (separados por *underline* ou "_") e esteticamente não muito agradável quando não está em tela cheia;

d) As práticas são simples com relação ao funcionamento real de dispositivos acoplados ao microcontrolador e fica limitado apenas aos usuários que possuam o sistema operacional Ubuntu instalado em sua máquina, tornando um obstáculo em sua execução;

5.2 Possíveis melhorias

Partindo dos pontos positivos e negativos analisados e buscando sempre melhorar os procedimentos utilizados, sugere-se possíveis melhorias em todo o âmbito. Parte-se, inicialmente, para a escolha de um microcontrolador com maiores atributos de memória e velocidade de processamento: ao invés de utilizar um Arduíno Mega 2560 Rev3, sugere-se o uso de um Arduíno Due ou um Arduíno Yùn. Na sequência, sugere-se para o desmembramento do código principal presente no Arduíno (*main.cpp*) em códigos parciais que forma este principal, para que o trabalho se torne mais organizado e mais fácil para reparos de erros.

Já com relação ao *layout* da interface, sugere-se a simplificação ou um uso de legendas para os nomes das variáveis presentes no *dynamic_reconfigure* de forma que torne-a mais esteticamente agradável. Fica também há cargo do programador que queria utilizar os códigos disponíveis deste projeto (com as referências devidamente apresentadas) acrescentar mais opções de configuração e parametrização dentro da interface.

Por fim, após a chegada dos equipamentos, sugere-se o seu estudo completo e sobre a comunicação serial em rede entre eles para que se possa implementá-los dentro do sistema. Assim, será possível desenvolver práticas mais avançadas e mais reais com relação às tendências industriais modernas.

REFERÊNCIAS

ALMEIDA, F. **O que é Encoder? Para que serve? Como escolher? Como interfacear?** 2017. Disponível em: <https://www.hitecnologia.com.br/blog/o-que-\%C3\%A9-encoder-para-que-serve-como-escolher-como-interfacear/>.

ARDUINO (Ed.). **Arduino Mega 2560 Rev3**. 2016. Disponível em: <https://store.arduino.cc/usa/arduino-mega-2560-rev3>.

____. **Arduino Uno Rev3**. 2016. Disponível em: <https://store.arduino.cc/usa/arduino-uno-rev3>.

____. **Compare board specs**. 2016. Disponível em: <https://www.arduino.cc/en/products.compare>.

AUTTOM. **Bancada Posicionamento Linear**. 2016. Disponível em: <https://auttom.com.br/produtos/bancada-posicionamento-linear/>.

BEN-MENAHEM, A. **Historical encyclopedia of natural and mathematical sciences**. [S.l.]: Springer Science & Business Media, 2009. v. 6.

CARVALHO, G. Máquinas elétricas: teoria e ensaios. **Editora Érica Ltda. São Paulo**, 2006.

CONTROLE & INSTRUMENTAÇÃO. **Inversores de frequência: vendas crescem 15%, mas a participação no parque industrial é inexpressiva**. 2001. Disponível em: <http://www.controleinstrumentacao.com.br/arquivo/ed_56/ed_56a.html>.

GOVERNO DO BRASIL. **Aneel estimula troca de motores elétricos para promover eficiência energética**. 2015. Disponível em: <http://www.brasil.gov.br/noticias/infraestrutura/2015/11/aneel-estimula-troca-de-motores-eletricos-para-promover-eficiencia-energetica>.

HOHNER. **SERIE 67 ENCODER ABSOLUTO SERIAL**. [S.l.], 2016.

IEC 60050. **Induction Machine - an asynchronous machine of which only one winding is energized**. 1990–10.

JR, C. K.; UMANS, S. D.; FITZGERALD, A. E. **Máquinas Elétricas-: Com Introdução à Eletrônica de Potência**. [S.l.]: Bookman, 2006.

PRODUCTS, I. M. I. (Ed.). **MAX481/MAX483/MAX485/MAX487–MAX491/MAX1487**. 2014. Disponível em: <https://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf>.

ROBOTICS, O. (Ed.). **ROS Wiki**. 2018. Disponível em: <http://wiki.ros.org/pt_BR>.

SILVA, M. L. H. da. **Desenvolvimento de plataforma para transporte de cargas usando inversor de frequência e motor de indução**. 2017. Monografia (Trabalho de Conclusão de Curso) — Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

SILVA NEWTON ; GAINO, R. . C. M. R. D. **BANCADA DIDÁTICA PARA ACIONAMENTO E CONTROLE DE MOTORES CA CONTROLADA POR DSC PROGRAMADO EM AMBIENTE MATLAB / SIMULINK**. 2014 — Universidade Estadual de Londrina – Centro de Tecnologia e Urbanismo – Departamento de Engenharia Elétrica, 2014.

SISKIND, C. S. **Electrical Control Systems in Industry**. [S.l.]: Glencoe/McGraw-Hill School Pub Co, 1963. ISBN 0-07-057746-3.

SOMA. **Bancada Didática Inversor Motor de Indução**. 2016. Disponível em: <https://www.soma.eng.br/portfolio-items/bancada-didatica-inversor-motor-de-inducao-bdimi-01/>.

WEG. **Catálogo Inversor de Frequência CFW300**. [S.l.], 2016.

____. **Manual do Usuário Modbus RTU CFW300**. [S.l.], 2016.

____. **Catálogo Técnico Motor Elétrico Trifásico W22**. [S.l.], 2017.

WESCOTT, T. **PID without a PhD**. 2000 — University of Michigan, 2000.

Apêndices

APÊNDICE A – INSTRUÇÕES DE USO

Os requisitos mínimos para o uso do projeto elaborado nesta monografia é uma máquina (computador ou Raspberry, por exemplo) que possua o sistema operacional Ubuntu instalado, alguma versão/distribuição do ROS instalado (no desenvolvimento, foi utilizado o ROS Kinetic Kame) e os principais pacotes do ROS instalado: *dynamic_reconfigure*, *rqt_plot*, *rviz* etc.

Com todos esses recursos e requisitos, deve-se verificar se o Arduino está conectado pela porta USB da máquina e se ele possui o código desenvolvido compilado internamente.O próximo passo, então, é abrir o terminal de comando através do ícone disponibilizado no próprio Ubuntu ou utilizando o atalho do teclado **CTRL** + **ALT** + **T**.

Com o terminal aberto, deve-se escrever o seguinte comando para que seja executado o ROS *launch* com todos os nós importantes para o projeto, dentro do pacote *dynamic_interface*:

roslaunch dynamic_interface server.launch

Ao todo, serão abertos cinco nós: um executável que contém todos os parâmetros do *server*, a interface do *dynamic_reconfigure* que irá alojar os parâmetros do executável, um nó que irá abrir e configurar a porta USB de conexão do Arduino (caso esteja dando problema, verifique o documento *server.launch* e altere o nome da porta no *node* do meio) e dois nós que irão fornecer os tópicos principais para duas janelas do *rqt_plot*. Após abrir as janelas, **o terminal em execução deve permanecer aberto**. Portanto, basta minimizá-lo e prosseguir com as operações.

Caso os tópicos não tenham aparecido diretamente nas janelas do *rqt_plot*, é necessário colocá-los manualmente: no espaço escrito *Topic* que contém uma barra (/), apague esta barra e escreva uma nova para aparecer as opções de tópicos. A ideia é colocar em uma janela os tópcios:

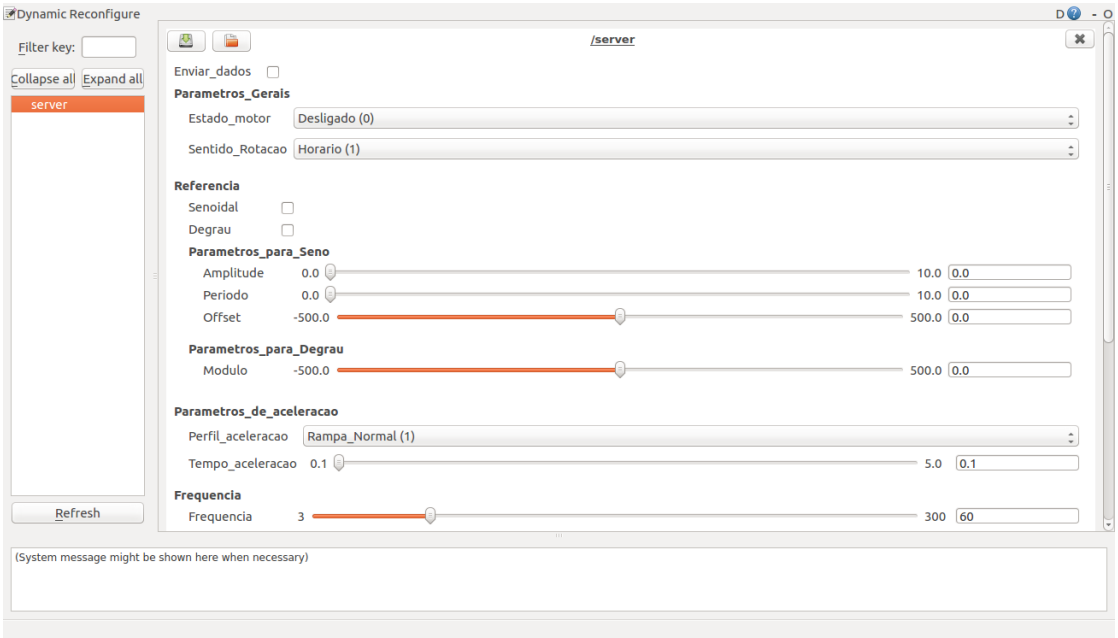
/arduinochattervelocidade
/arduinochattervelocidaderef

e em uma outra janela do *rqt_plot*, colocar os três tópicos:

/arduinochatterinput
/arduinochatteroutput
/arduinochattersetpoint

Com o *rqt_plot* para plotar os dados recebidos do Arduino, abra a interface do *dynamic_reconfigure*, selecione no canto superior esquerdo o executável *server* e observe que no espaço à direita serão abertas diversas opções de configuração do sistema.

Figura 54 – Interface do *dynamic_reconfigure* com todos os parâmetros para configuração e escolha - parte 1

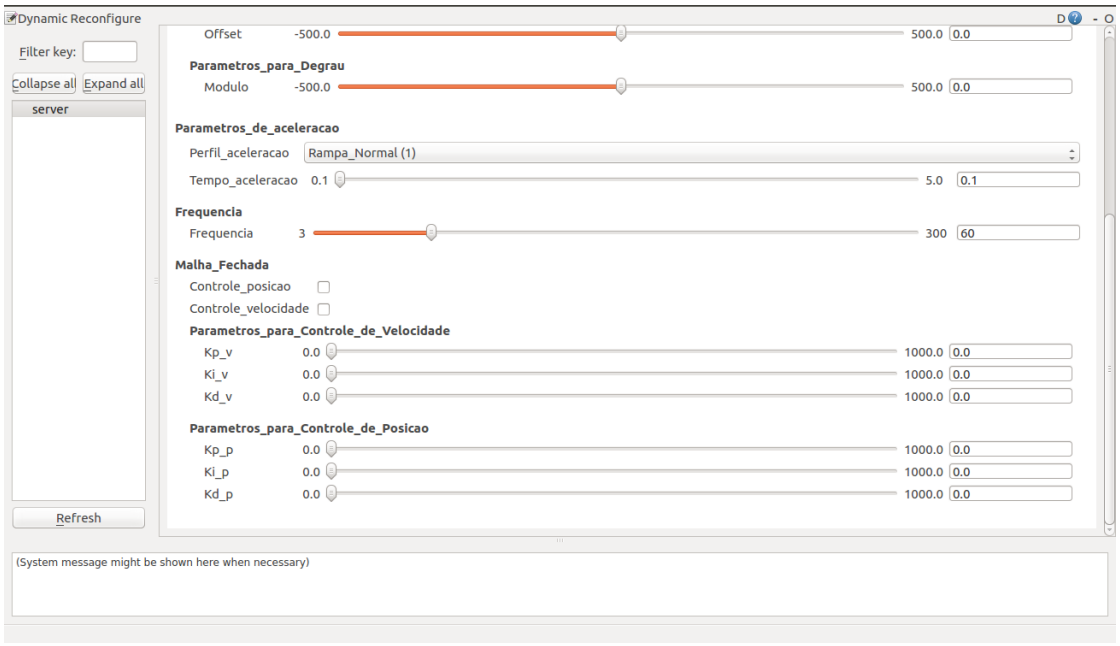


Fonte: Própria

Após a selecionar as opções e os valores desejados, é necessário clicar na opção de enviar dados (**Enviar_dados**) para que todos as variáveis sejam atualizados e o sistema sofra o efeito das escolhas do usuário. Caso contrário, o sistema não irá responder às mudanças. É importante também salientar que, para se evitar conflitos entre os modos de atuação da interface, ao selecionar modos conflitantes (senoidal e degrau, por exemplo) ao mesmo tempo, apenas uma ficará selecionada.

Para fechar todos as janelas em uma única etapa, basta abrir novamente a janela do terminal em que todo o processo está sendo executado e utilizar o atalho **CTRL** + **C** para encerrar o ROS *launch* e, portanto, todos os nós.

Figura 55 – Interface do *dynamic_reconfigure* com todos os parâmetros para configuração e escolha - parte 2



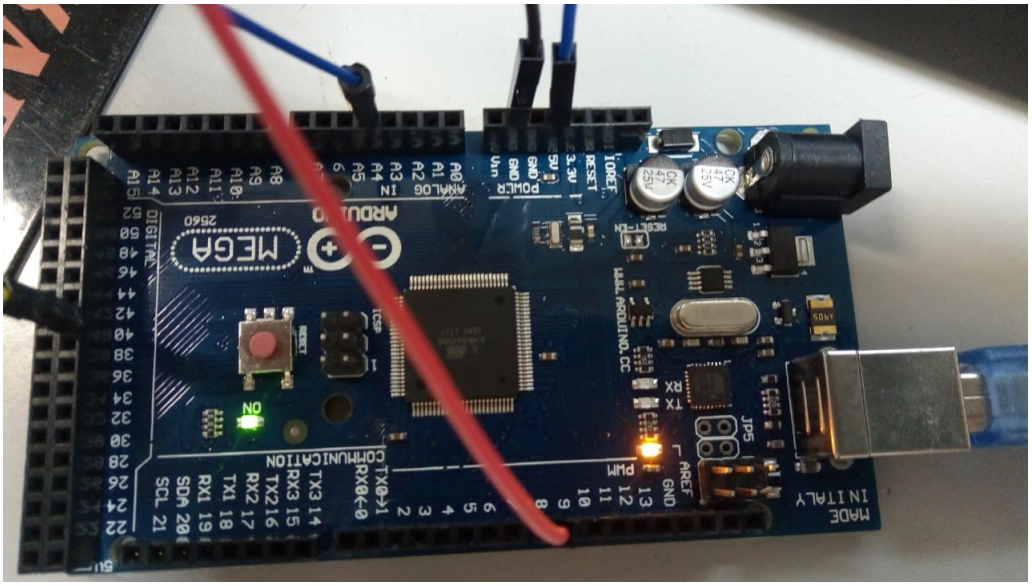
Fonte: Própria

APÊNDICE B – SUGESTÕES DE PRÁTICAS

Com as instruções de uso em mãos e com toda a interface aberta, sugere-se ao usuário realizar pequenos experimentos para verificar o funcionamento do projeto assim como para inspirá-lo a buscar novas práticas e otimizações.

A primeira sugestão de prática consiste em acender o LED 13 embutido na própria placa do Arduino Mega: selecione ou escreva o valor 1.0 para o parâmetro *Tempo_aceleracao*, envie a mensagem e observe que o LED em questão irá acender por 1 segundo e depois irá apagar (isto ocorre somente para o valor igual a 1.0). O objetivo é mostrar que o Arduino está recebendo a mensagem e os seus respectivos valores da interface produzida.

Figura 56 – LED 13 acendendo conforme o Tempo de aceleração é configurado com valor igual a 1.0



Fonte: Própria

A próxima prática consiste em verificar, no terminal que está executando o ROS *launch*, a impressão dos valores das variáveis que o usuário está enviando naquele momento. Para cada modo de uso da interface, os seus respectivos valores irão aparecer.

Na sequência, sugere-se o uso das ferramentas de plotagem de gráficos, provindos do processamento do próprio microcontrolador. Pode-se optar tanto pelas opções de **Referência** quanto pelas opções de **Malha fechada**. Os tópicos referentes a estes modos são, respectivamente:

/arduinochattervelocidade
/arduinochattervelocidaderef

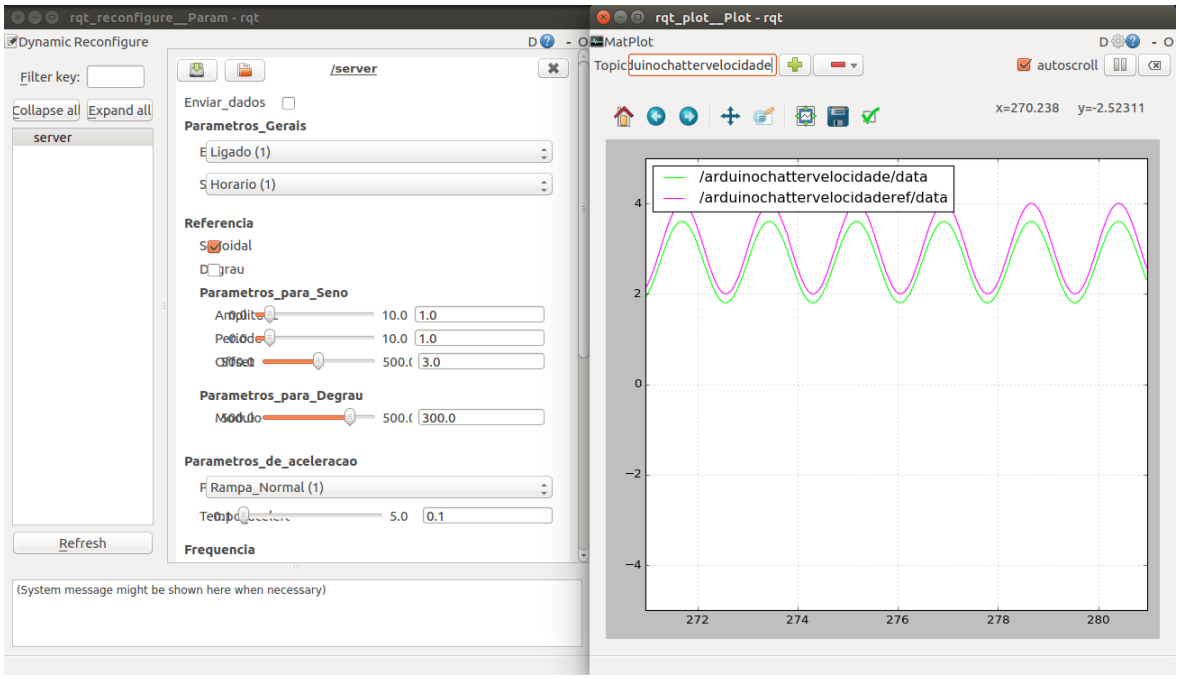
/arduinochatterinput
/arduinochatteroutput
/arduinochattersetpoint

Figura 57 – Terminal apresentando os dados enviados pelo usuário a partir da interface do *dynamic_reconfigure*

Fonte: Própria

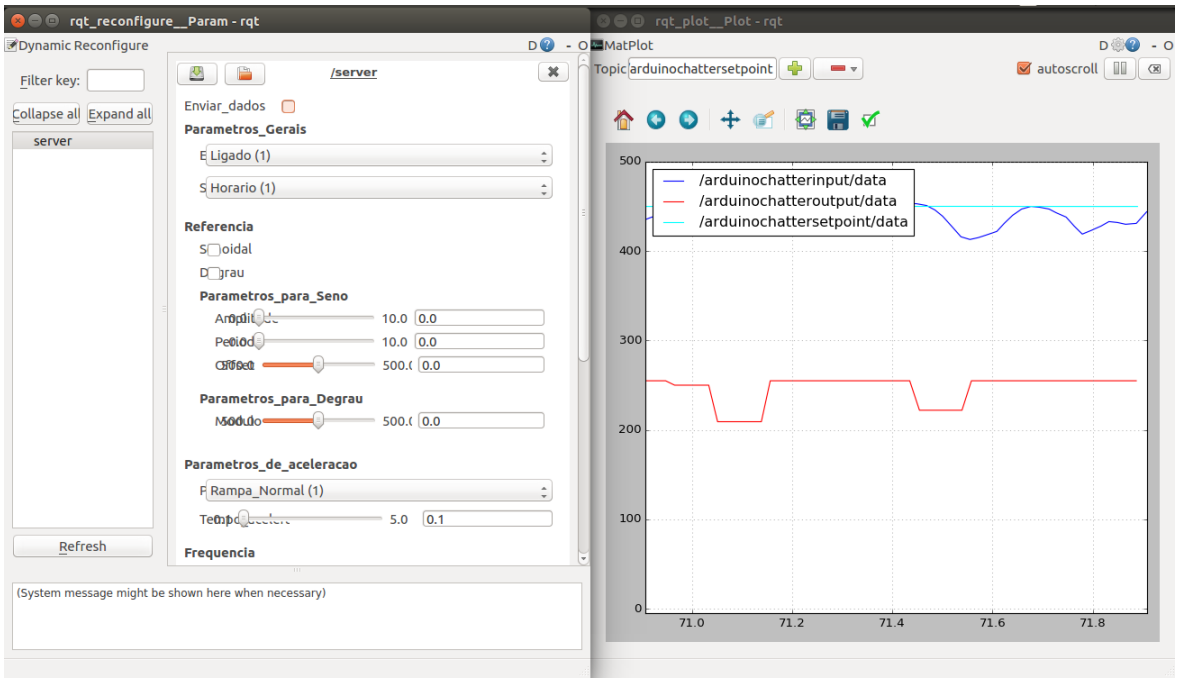
Observe que para cada dado ou tópico, a própria ferramenta do ROS configura cores diferentes para que seja possível visualizar com maior facilidade o que está acontecendo. Além disso, é possível configurar os eixos conforme a necessidade das opções disponibilizadas pelos símbolos na aba superior da janela.

Figura 58 – Plotagem de Referência com dados da interface do *dynamic_reconfigure*, para o caso Senoidal



Fonte: Própria

Figura 59 – Plotagem de Malha Fechada com dados da interface do *dynamic_reconfigure*, para o caso de Controle de Posição



Fonte: Própria

APÊNDICE C – DISPONIBILIDADE DOS CÓDIGOS

Com a intenção de contribuir para futuras pesquisas e ensino de estudantes universitários nas disciplinas associadas a elementos de automação e sistemas de controle, o autor desta monografia (Kevin) disponibiliza abertamente os códigos utilizados neste trabalho desde que o uso deles em projetos e pesquisas sejam referenciado corretamente a esta monografia.

O código se encontra na plataforma de hospedagem de projetos controlados *Bitbucket*, cujo link do repositório é dado abaixo:

https://bitbucket.org/kevinokamura/dynamic_interface/src

Neste repositório se encontram duas pastas: uma contendo os códigos utilizados via ROS dentro do ambiente Ubuntu, por exemplo; e a outra, contém códigos que são inseridos dentro do Arduino Mega. Por fim, está disponível um arquivo de texto que fornece instruções simples de uso dos arquivos no repositório.

Lembre-se de referenciar este projeto em todo material que ele foi utilizado como embasamento.