UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

Pedro Henrique Santos de Medeiros

Controle de uma maquete de ponte rolante com interface gráfica no MATLAB utilizando o microcontrolador ATMEGA 2560 embarcado em uma plataforma Arduino

Vitória-ES

Julho/2017

Pedro Henrique Santos de Medeiros

Controle de uma maquete de ponte rolante com interface gráfica no MATLAB utilizando o microcontrolador ATMEGA 2560 embarcado em uma plataforma Arduino

Parte manuscrita do Projeto de Graduação do aluno Pedro Henrique Santos de Medeiros, apresentada ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para a obtenção do grau de Engenheiro Eletricista.
Orientador: Prof. Dr. José Leandro Félix Salles

Vitória-ES

Julho/2017

Pedro Henrique Santos de Medeiros

Controle de uma maquete de ponte rolante com interface gráfica no MATLAB utilizando o microcontrolador ATMEGA 2560 embarcado em uma plataforma Arduino

Trabalho aprovado em 31 de Julho de 2017:

Prof. Dr. José Leandro Félix Salles
Orientador

Prof. Dr. Anselmo Frizera Neto
Examinador

Eng. Felipe Machado Lobo
Examinador

Vitória-ES

Julho/2017

AGRADECIMENTOS

Agradeço em primeiro lugar a DEUS, por ter me concedido a oportunidade de participar deste curso e poder ter chegado ao fim realizando o trabalho aqui apresentado.

Em segundo lugar agradeço à minha família, por sempre ter me incentivado a continuar buscando aprender cada vez mais.

Ao meu orientador José Leandro Félix Salles e aos professores a quem procurei nos momentos de dificuldades, que me forneceram a base teórica necessária para que este trabalho pudesse ser concluído. Ao departamento e ao colegiado do curso de Engenharia Elétrica da Ufes, assim como a própria Ufes em si, que proporcionaram toda a base estrutural para que minha formação pudesse ser realizada de forma gratuita e de qualidade.

Agradeço também a todos os meus amigos, do curso, do intercâmbio e da vida, que fizeram com que esta trajetória fosse inesquecível.

RESUMO

A ponte rolante é um equipamento bastante utilizado na indústria para o transporte de cargas, possuindo restrições de funcionamento que se transformam em fatores motivantes para que sejam desenvolvidos sistemas de controle capazes de regular a magnitude e a duração do balanço das cargas durante o transporte, visando evitar possíveis danos às cargas e até mesmo acidentes envolvendo pessoas. Este projeto tem como objetivo o desenvolvimento de um sistema de controle em malha fechada de uma maquete de ponte rolante existente no Laboratório de Controle e Instrumentação do Departamento de Engenharia Elétrica da UFES, utilizando o *software* MATLAB e o microcontrolador Arduino Mega 2560. O sistema foi implementado de forma a realizar o controle de posição de uma carga suspensa na ponte rolante, a qual se movimentará nas direções horizontal e vertical. Para tal, inicialmente foram analisados métodos de controle baseados em heurística devido às não-linearidades e atritos consideráveis presentes no sistema. Além disso, foi desenvolvido um sistema supervisório com interface gráfica a fim de tornar o processo de controle mais didático, visando um possível aproveitamento deste projeto em disciplinas de Controle e Automação do curso de Engenharia Elétrica da UFES.

Palavras-chave: Ponte Rolante; Controle Automático; Interface com o usuário; Aquisição de dados; MATLAB.

ABSTRACT

An overhead crane is a device commonly used for transportation of heavy loads in the industry. They may have operational restrictions that become motivating factors for the development of control systems capable of regulating magnitude and duration of the balance of the load during transportation, in order to avoid possible damage to the loads and even accidents involving people. This project aims to develop a closed loop control system of a crane model in the Control Systems Laboratory of the Electrical Engineering Department at UFES using the software MATLAB and the Arduino Mega 2560 microcontroller. The system was implemented in order to perform position control of a suspended load on the crane, which will move in the horizontal and vertical axis. In order to achieve that, initially control methods based on heuristics were taken into consideration due to nonlinearities and substantial friction present in the system. In addition, a supervisory system with a graphical user interface was developed in order to make the control process more didactic, aiming at a possible use of the project in Control subjects of the Electrical Engineering Course at UFES.

Keywords: Overhead crane; Automatic control; User Graphical Interface; Data acquisition; MATLAB.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

$UFES$	Universidade Federal do Espírito Santo
LCI	Laboratório de Controle e Instrumetação
I^2C	Inter-Integrated Circuit
PWM	Pulse Width Modulation
CI	Circuito Integrado
IDE	Integrated Development Environment
$SRAM$	Static Random Access Memory
$EEPROM$	Electrically-Erasable Programmable Read-Only Memory
TTL	Transistor–transistor logic
SPI	Serial Peripheral Interface
SDA	Serial Data Line
SCL	Serial Clock Line
$UART$	Universal asynchronous receiver/transmitter
$FTDI$	Future Technology Devices International
USB	Universal Serial Bus
COM	Communication port
FET	Field Effect Transistor
DMP	Digital Motion Processor
DOF	Degrees of Freedom
$GUIDE$	Graphical User Interface Development Environment

LISTA DE SÍMBOLOS

$\hat{\boldsymbol{x}}_{k-1\|k-1}$	Estado anterior
$\hat{\boldsymbol{x}}_{k\|k-1}$	Estado a priori
$\hat{\boldsymbol{x}}_{k\|k}$	Estado a posteriori
z_k	Medição do estado
$\boldsymbol{x}_k$	Estado no tempo k atual
θ	Ângulo de oscilação da carga no eixo Y
$\dot{\theta}_b$	Desvio de medição proveniente do giroscópio
$\boldsymbol{F}$	Matriz de transição de estados
Δt	Intervalo de tempo entre leituras dos sensores
$\dot{\theta}$	Velocidade angular medida pelo giroscópio
$\boldsymbol{B}$	Modelo de controle de entrada
$\boldsymbol{w}_k$	Ruído de processo
$\boldsymbol{Q}_k$	Matriz de covariância do ruído de processo
Q_θ	Variância do acelerômetro
$Q_{\dot{\theta}_b}$	Variância do bias
$\boldsymbol{H}$	Modelo de observação
v_k	Ruído de medição
R	Matriz de covariância do ruído de medição
$\boldsymbol{P}$	Matriz de covariância do erro
$\tilde{\boldsymbol{y}}_k$	Vetor de inovação
$\boldsymbol{S}$	Matriz de covariância da inovação
$\boldsymbol{K}$	Ganho de Kalman no tempo atual k
$\boldsymbol{I}$	Matriz identidade

SUMÁRIO

1 APRESENTAÇÃO E OBJETO DE PESQUISA

1.1 A ponte rolante

O processo de automação e controle é indispensável quando o objetivo é fornecer um produto ou serviço que seja o mais confiável e eficiente possível. Com o avanço da tecnologia, diversas atividades que antes eram realizadas manualmente estão cada vez mais passando a ser feitas por máquinas e equipamentos eletrônicos. Hoje em dia pode-se observar inúmeras aplicações de automação nas mais variadas áreas, como por exemplo em residências, com o avanço da domótica, na agricultura, com a automação de processos de irrigação, além claro da indústria, que vem cada vez mais aperfeiçoando a automação e controle de processos de fabricação.

Aprofundando-se no ramo industrial, dentre os diversos equipamentos utilizados para o transporte e movimentação de cargas estão as pontes rolantes. Uma ponte rolante é uma máquina do tipo guindaste para elevação e transporte de cargas, constituída de um carro que se desloca sobre vigas, permitindo deslocamentos verticais e horizontais independentes (BRASIL, 1987). Apesar de seu custo elevado quando para sua aplicação em portos ou empresas de logística que lidam com cargas muito pesadas, as pontes rolantes possuem um ótimo custo-benefício devido à sua fácil manutenção e considerável robustez.

O sistema da ponte rolante possui um número maior de parâmetros a serem controlados do que entradas de controle (MARCOS, 2014). É importante que os comandos sejam feitos de forma correta e precisa ao se projetar o controle de sistemas desta espécie, pois caso não seja feito dessa forma, a carga pode apresentar oscilações que comprometam a agilidade do transporte e até mesmo a segurança da operação (CHWA, 2009). Uma proposta de automação e controle deste tipo de ponte se torna importante quando leva-se em conta a dificuldade de determinar padrões quando estas pontes são operadas manualmente (YANG, 2009).

Figura 1 – Ponte Rolante Industrial.

Fonte:Grupo Incatep. Disponível em: <https://www.grupoincatep.com.br/40-ponte-rolante-industrial>. Acesso em 09/11/2016.

Durante a operação manual, a ponte é controlada de acordo com a experiência e características normalmente usadas por seu operador, fazendo com que o tempo de operação entre o içamento e a descida da carga seja diferente e dependa de quem a está operando, não possibilitando assim estabelecer com precisão uma média de tempo necessário para a conclusão da atividade (MARCOS, 2014). Deste modo, para que este transporte seja eficiente, é necessário que o mesmo aconteça de forma rápida, reduzindo o tempo de ciclo de trabalho na movimentação das cargas (CHWA, 2009).

Entretanto, durante a operação da ponte rolante, a carga transportada pode apresentar um movimento pendular indesejado, devido à inércia causada pelas acelerações e desacelerações tanto dos sistemas de translação quanto de suspensão (FURTADO, 2006). Tal movimento é indesejado pois além de aumentar o tempo de operação, uma vez que dificulta o posicionamento final da carga ao seu local de destino, aumentando assim os custos com a movimentação de carga, pode também causar danos à carga, ao ambiente de operação, à ponte propriamente dita, além da possibilidade de acidentes envolvendo pessoas (COSTA, 2010).

Dito isto, fica evidente a necessidade de haver um sistema de controle automático que garanta rapidez, confiabilidade, e eficiência no transporte da carga durante a operação de uma ponte rolante, tendo em vista a maximização da produtividade destas atividades no setor industrial. Assim sendo, este projeto propõe o desenvolvimento de um sistema supervisório de controle para a posição do conjunto *trolley*/carga da maquete de ponte rolante presente no LCI (Laboratório de Controle e Instrumentação) do curso de Engenharia Elétrica da UFES.

1.2 Objetivo geral

O objetivo geral deste projeto é o desenvolvimento de um sistema de controle de posição em malha fechada de uma maquete de ponte rolante utilizando o *software* MATLAB para a realização da interface gráfica, e o Arduino Mega 2560 para realizar o acionamento da Planta e a coleta de dados dos sensores presentes na mesma.

1.3 Objetivos específicos

- Elaborar a interface gráfica para o controle da planta utilizando a ferramenta GUIDE do MATLAB.
- Elaborar a lógica para a comunicação entre o MATLAB e o sistema de aquisição de dados implementado no Arduino.
- Desenvolver um sistema de obtenção e filtragem do ângulo de balanço da carga durante sua movimentação.
- Implementar a lógica de controle em conjunto com a interface gráfica de modo a realizar o controle de posição da carga na Planta.

1.4 Organização do texto

O presente trabalho está dividido em 5 capítulos:

O capítulo 1 apresenta de forma resumida o projeto que foi desenvolvido, sua motivação, os objetivos gerais e específicos que foram alcançados com o mesmo, assim como a descrição da estrutura do texto aqui presente.

O capítulo 2 descreve os equipamentos e técnicas utilizados para realizar o acionamento da Planta, assim como para coletar os dados dos sensores presentes na mesma.

O capítulo 3 apresenta os algoritmos de controle desenvolvidos para o MATLAB e para o Arduino, descrevendo os pontos principais de seu funcionamento. Posteriormente é apresentado o método de filtragem implementado para a obtenção do ângulo de balanço da carga com maior precisão.

O capítulo 4 apresenta os resultados obtidos com os testes realizados na Planta, analisando o comportamento dos sistemas de posicionamento horizontal e vertical da carga, verificando também a oscilação da mesma durante este acionamento.

O capítulo 5 apresenta as conclusões obtidas com este projeto, verificando os objetivos alcançados com o mesmo, realizando ao final uma análise crítica e sugerindo pontos de melhorias para trabalhos futuros a serem realizados na Planta.

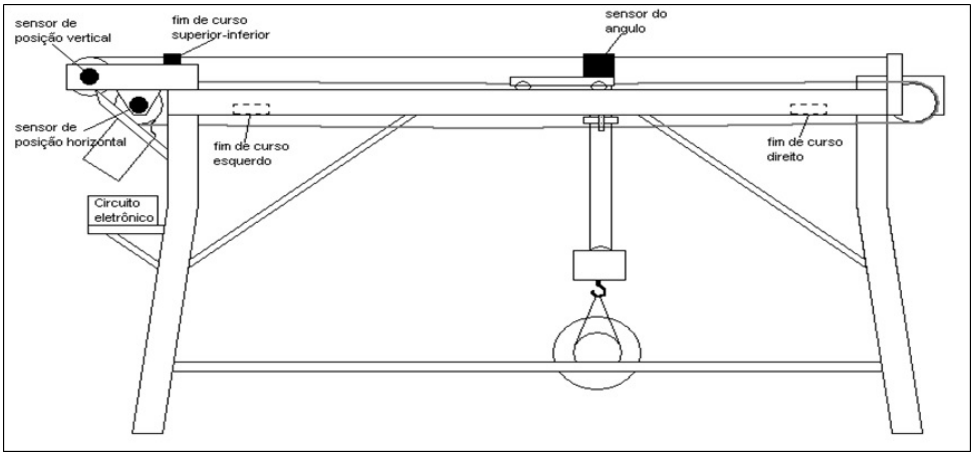
2 DESENVOLVIMENTO DO *HARDWARE* DA PONTE ROLANTE

2.1 Estrutura da ponte rolante

A estrutura da ponte rolante que será controlada, mostrada na Figura 2, possui 1,8 m de comprimento por 1,5 m de altura (OLIVEIRA, 2015). Nela estão adaptados dois motores de corrente contínua, um sendo responsável pelo movimento horizontal e outro pelo movimento vertical da carga. Além disso, após os trabalhos realizados por Oliveira (2015) a mesma já possui sensores de posição através de *encoders*, tanto para o motor que comanda o movimento vertical, quanto para o horizontal.

O circuito de acionamento dos motores é composto de duas fontes de alimentação, uma para cada motor, dois módulos driver motores do tipo Ponte H para fornecer a corrente necessária e realizar controle de sentido de giro dos mesmos, assim como quatro chaves fim de curso para garantir a segurança do sistema durante o seu funcionamento. Além disso, foi adicionado um módulo acelerômetro e giroscópio GY-521 para a medição do ângulo de oscilação da carga durante seu movimento. Todos estes componentes são acionados e monitorados a partir de um microcontrolador ATMEGA 2560 embarcado em uma plataforma Arduino (Arduino MEGA 2560) (ARDUINO, 2017), que além de monitorar os sinais enviados pelos *encoders* e pelo sensor de ângulo, gera o sinal PWM (*Pulse Width Modulation*, ou modulação por largura de pulso) necessário para acionar os motores, controlando a velocidade de transporte e a elevação da carga.

Figura 2 – Esquemático da maquete da ponte rolante.

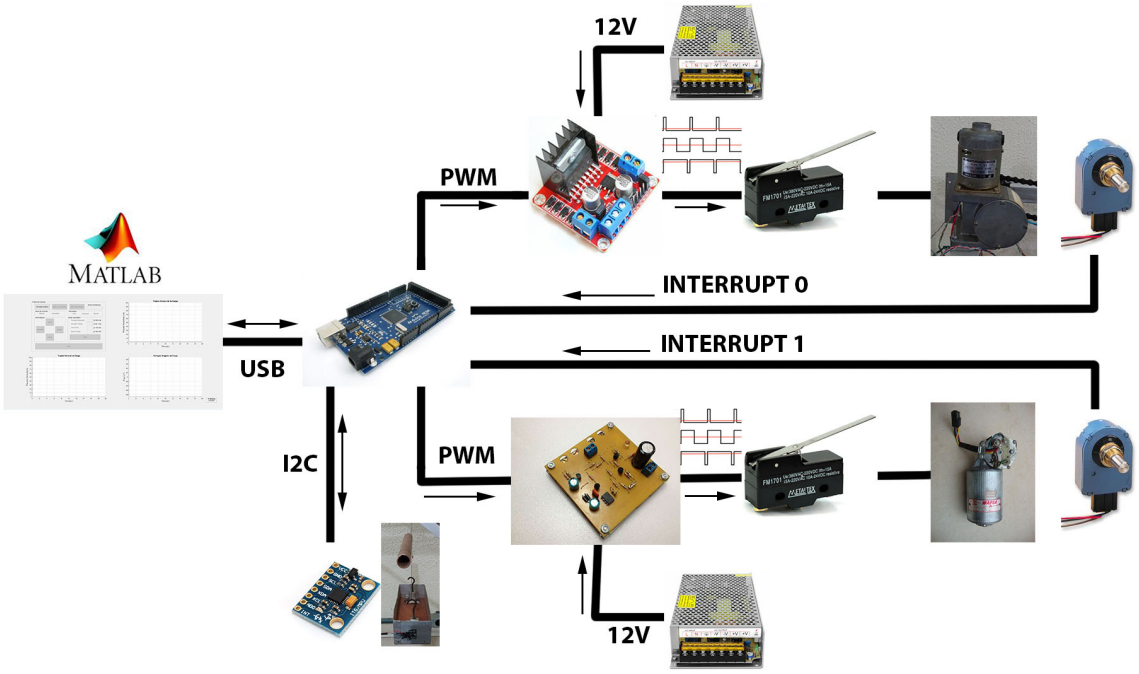


Fonte: (OLIVEIRA, 2015)

2.2 Diagrama Geral do *Hardware*

A seguir pode-se ter uma visão geral do *Hardware*, em conjunto com suas conexões e tipos de comunicação envolvidos no sistema, onde pode-se notar a interconexão entre todos os aspectos mencionados neste capítulo. Deste modo, fazendo uso das interfaces de comunicação e de acionamento presentes na Figura 3 pode-se realizar o acionamento e controle da planta em questão.

Figura 3 – Diagrama Geral do Hardware.



Fonte: Elaborado pelo Autor.

2.3 Arduino MEGA 2560

O Arduino é uma plataforma de prototipagem eletrônica de hardware livre, com um microcontrolador ATMEGA em sua placa única. Seu objetivo é criar ferramentas acessíveis, de forma prática e fácil, para amadores e profissionais (MULTILóGICA, 2016). Inicialmente neste projeto, o Arduino seria utilizado somente como atuador para o acionamento dos motores através do envio de sinais PWM, sendo que todo o código de controle e programação do sistema seria implementado através do MATLAB. Porém, realizando-se testes notou-se que esta configuração gerava uma taxa de amostragem dos sinais dos sensores muito baixa, devido ao alto custo computacional imposto pelo MATLAB. Deste modo, optou-se por realizar toda a implementação do código diretamente no Arduino utilizando o seu próprio *Software* Arduino IDE, utilizando o MATLAB somente para enviar comandos e receber

dados dos sensores via comunicação *Serial*, e realizar a interface com o usuário através de sua interface gráfica GUIDE. Esta nova configuração apresentou uma capacidade de amostragem 20 vezes maior que a anterior, mostrando-se capaz de gerar dados mais precisos e confiáveis para um controle e monitoramento mais eficiente do sistema.

Figura 4 – Arduino MEGA 2560.

Fonte: (ARDUINO, 2017)

2.3.1 Alimentação

O Arduino MEGA 2560 pode ser alimentado pela conexão USB ou com uma fonte externa, sendo que a entrada de alimentação é selecionada automaticamente. A placa pode operar com alimentação externa entre 6 e 20V. No entanto, se menos de 7V forem fornecidos externamente, o pino de 5V pode fornecer menos de 5V e a placa pode ficar instável. Com mais de 12V o regulador de tensão pode superaquecer e danificar a placa. A faixa recomedável é de 7 a 12V (ARDUINO, 2017).

Os pinos de alimentação são os seguintes:

- VIN: É a entrada de tensão da placa Arduino quando utiliza-se uma fonte de alimentação externa (em oposicão aos 5 V fornecidos pela conexão USB ou outra fonte de alimentação regulada). É possível fornecer alimentação através deste pino, ou acessá-la a partir dele, caso a placa esteja sendo alimentada pelo conector de alimentação.
- 5V: Trata-se do fornecimento de alimentacão regulada para o microcontrolador, e para periféricos que possam ser utilizados em conjunto.
- 3.3V: Trata-se de uma saída de tensão de 3.3 V gerada pelo regulador de tensão. A corrente máxima para este pino é de 50 mA.

- GND: Pinos terra.

2.3.2 Memória

O microcontrolador ATmega2560 tem 256 *KBytes* de memória *FLASH* para armazenamento de código, 8 *KBytes* de memória SRAM e 4 *KBytes* de memória EEPROM, que podem ser lidos e escritos com a biblioteca EEPROM (ARDUINO, 2017).

2.3.3 Entrada e Saída

Cada um dos 54 pinos digitais do Arduino MEGA 2560 pode ser usado como entrada ou saída digitais, utilizando-se as funções de *pinMode()*, *digitalWrite()*, e *digitalRead()*. Eles operam a 5V, sendo que cada pino pode fornecer ou receber uma corrente máxima de 40 mA e possui um resistor interno, desconectado por padrão, de 20 a 50KΩ. Além disso, alguns pinos possuem funções especializadas (ARDUINO, 2017):

- Serial: pinos 0 (RX) e 1 (TX); Serial 1: pinos 19 (RX) e 18 (TX); Serial 2: pinos 17 (RX) e 16 (TX); Serial 3: pinos 15 (RX) e 14 (TX). Usados para receber (RX) e transmitir (TX) dados via serial padrão TTL.
- Interrupções Externas: pinos 2 (interruptor 0), 3 (interruptor 1), 18 (interruptor 5), 19 (interruptor 4), 20 (interruptor 3), e 21 (interruptor 2). Estes pinos podem ser configurados para disparar uma interrupção por um valor baixo, uma borda de subida ou descida, ou uma mudança de valor. A função *attachInterrupt()* é a responsável pelo tratamento das interrupções.
- PWM: pinos 0 a 13. Fornecem saída PWM de 8-bits com a utilização da função *analogWrite()*.
- SPI: pinos 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). Estes pinos dão suporte à comunicação SPI que, embora suportada pelo *hardware*, ainda não está incluida na linguagem Arduino.
- LED: pino 13. Há um LED conectado ao pino digital 13. Quando o pino está em nível lógico alto, o led se acende.
- I^2C: pinos 20 (SDA) e 21 (SCL). Fornecem suporte à comunicação I^2C utilizando a biblioteca *Wire*.

2.3.4 Comunicação

O Arduino MEGA 2560 possui facilidades para comunicar-se com um computador, com outro Arduino ou outros microcontroladores. O ATmega2560 fornece quatro portas de comunicação *serial* UARTs para TTL (5V). Um *chip* FTDI FT232RL direciona uma destas portas para a conexão USB e os drivers FTDI, que acompanham o *software* do Arduino, fornecem uma porta COM virtual para *softwares* no computador (ARDUINO, 2017). A biblioteca *Serial* presente no *software* Arduino IDE é a responsável por realizar tal comunicação. O *software* também inclui um monitor *serial* que permite que dados de texto sejam enviados e recebidos pela placa Arduino. Os LEDs RX e TX piscarão enquanto dados estiverem sendo transmitidos pelo *chip* FTDI e pela conexão USB ao computador. Além disso, o ATmega2560 também fornece suporte para comunicação I^2C e SPI, como mencionado anteriormente.

2.4 Motores CC e seus acionamentos

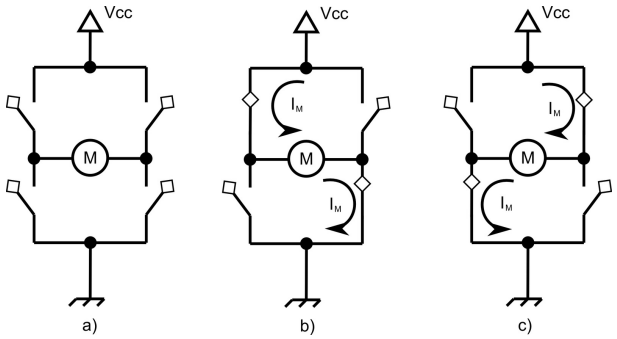
Os motores de corrente contínua (CC) são dispositivos que operam através das forças de atração e repulsão geradas por eletroímãs e ímãs permanentes, convertendo energia elétrica em energia mecânica. Um motor CC consiste de um estator, um enrolamento de armadura, um rotor e um comutador com escovas, onde polaridades opostas entre os dois campos magnéticos formados dentro do motor fazem com que o rotor gire. Se uma corrente elétrica percorre os enrolamentos do motor em um sentido, o eixo do motor gira para um lado. Se a corrente elétrica é invertida, o eixo gira para o outro lado. Os dois motores utilizados no projeto são de corrente contínua e já se encontravam afixados à maquete. Ambos os motores trabalham com tensão de 12V devido às limitações das pontes H, sendo o mais robusto, de movimentação vertical da carga, com corrente de operação nominal de 6A, e o menor, para movimentação horizontal do carro, de 1A (OLIVEIRA, 2015).

Ao contrário das pontes rolantes convencionais, onde o motor responsável pela movimentação vertical da carga encontra-se afixado ao carro (*trolley*), e, deste modo, movimenta-se junto com o conjunto *trolley*/carga, na maquete de ponte rolante deste projeto o motor responsável pela movimentação vertical da carga encontra-se afixado à própria estrutura da ponte. Sendo assim, é então necessário que seu acionamento seja feito mesmo quando deseja-se movimentar a carga apenas horizontalmente, tendo em vista que se houver rotação apenas do motor responsável pelo movimento horizontal, a carga irá subir ou descer em relação à sua posição vertical inicial, dependendo da direção horizontal desejada.

2.4.1 A ponte H

Como visto na seção anterior, pode-se inverter o sentido de rotação do eixo do motor CC simplesmente invertendo a polaridade de seus terminais elétricos, e por consequência, invertendo a corrente que circula por seus enrolamentos. Um dos métodos mais utilizados para isso é o circuito envolvendo transistores funcionando como chaves, apelidado de ponte H, pela forma como o circuito é montado. A Figura 5 ilustra o funcionamento da ponte H para inverter o sentido de giro do motor, onde na situação (a), com todas as chaves abertas, o motor está parado, na situação (b), com um par de chaves fechadas, a corrente percorre os enrolamentos do motor fazendo-o girar em um sentido, e na situação (c), com o outro par de chaves fechadas, a corrente percorre os enrolamentos no sentido oposto, fazendo com que o motor inverta seu sentido de rotação.

Figura 5 – Funcionamento do circuito Ponte H



Fonte: AECX - Robótica. Disponível em: <http://aecxrobot.blogspot.com.br/p/aula-10-ponte-h.html>. Acesso em 09/11/2016.

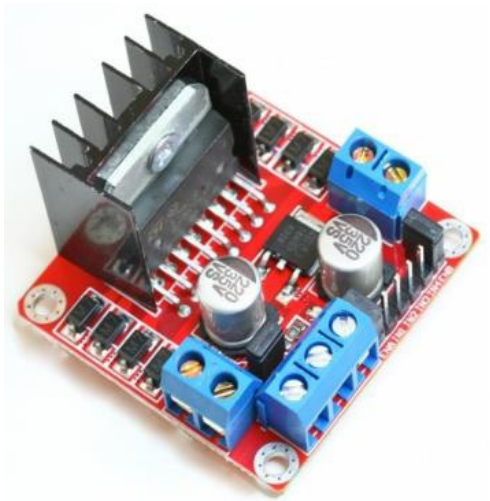
2.4.2 Acionamento do motor responsável pelo movimento horizontal - Ponte H L298N

Para o controle de velocidade e de rotação dos motores através de Ponte H, é utilizado o módulo L298N, como mostrado na Figura 6, para o motor menor, responsável pela movimentação horizontal. Este módulo é projetado para controlar cargas indutivas como relés, solenóides, motores CC e motores de passo, permitindo o controle não só do sentido de rotação do motor, como também da sua velocidade, utilizando os pinos PWM do Arduino. A seguir tem-se as especificações técnicas do modelo utilizado (THOMSEN, 2013):

Especificações técnicas da Ponte H L298N:

- Tensão de Operação: 4∼35 V.
- Chip: ST L298N.
- Controle de 2 motores CC ou 1 motor de passo.
- Corrente de Operação máxima: 2A por canal ou 4A max.
- Tensão lógica: 5 V.
- Corrente lógica: 0∼36 mA.
- Limites de Temperatura: -20 a +135°C.
- Potência Máxima: 25W.
- Dimensões: 43 x 43 x 27 mm.
- Peso: 30g.

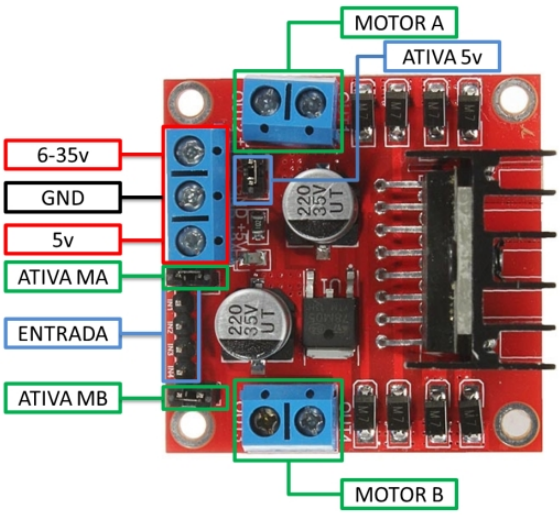
Figura 6 – Ponte H embarcada L298N para a movimentação horizontal



Fonte: Driver Motor ponte-H - L298N. Disponível em: <https://www.baudaeletronica.com.br/driver-motor-ponte-h-l298n.html>. Acesso em 09/11/2016.

A figura 7 ilustra em detalhes as conexões do módulo L298N. A utilização de cada pino encontra-se detalhada a seguir:

Figura 7 – Ponte H embarcada L298N em detalhes



Fonte: (THOMSEN, 2013).

Funcionamento da Ponte H L298N (THOMSEN, 2013):

- (Motor A) e (Motor B) referem-se aos conectores para ligação de 2 motores CC ou 1 motor de passo.
- (Ativa MA) e (Ativa MB) são os pinos responsáveis pelo controle PWM dos motores A e B. Se estiver com jumper, não haverá controle de velocidade, pois os pinos estarão ligados aos 5 V. Estes pinos podem ser utilizados em conjunto com os pinos PWM do Arduino.
- (Ativa 5v) e (5v) – Este *driver* Ponte H L298N possui um regulador de tensão integrado. Quando o *driver* está operando entre 6 e 35V, este regulador disponibiliza uma saída regulada de +5 V no pino (5 V) para um uso externo (com jumper), podendo alimentar por exemplo outro componente eletrônico. Portanto não deve-se alimentar este pino (5v) com +5 V do Arduino se estiver controlando um motor de 6 a 35 V e jumper conectado, pois isto danificará a placa. O pino (5v) somente se tornará uma entrada caso esteja controlando um motor de 4 a 5.5 V (sem jumper), podendo assim usar a saída +5 V do Arduino.
- (6-35v) e (GND) - Aqui será conectada a fonte de alimentação externa quando o driver estiver controlando um motor que opere entre 6 e 35 V. Por exemplo, ao utilizar um motor CC 12 V, basta conectar a fonte externa de 12 V neste pino e (GND).
- (Entrada) - Este barramento é composto por IN1, IN2, IN3 e IN4. Sendo estes pinos os responsáveis pela rotação do Motor A (IN1 e IN2) e Motor B (IN3 e IN4).

A Figura 8 mostra a ordem de ativação do motor CC através dos pinos IN1 e IN2:

Figura 8 – Ativação do motor CC através dos pinos IN1 e IN2

MOTOR	IN1	IN2
HORÁRIO	5v	GND
ANTI-HORÁRIO	GND	5v
PONTO MORTO	GND	GND
FREIO	5v	5v

Fonte: (THOMSEN, 2013).

2.4.3 Acionamento do motor responsável pelo movimento vertical - Ponte H 12V 16A

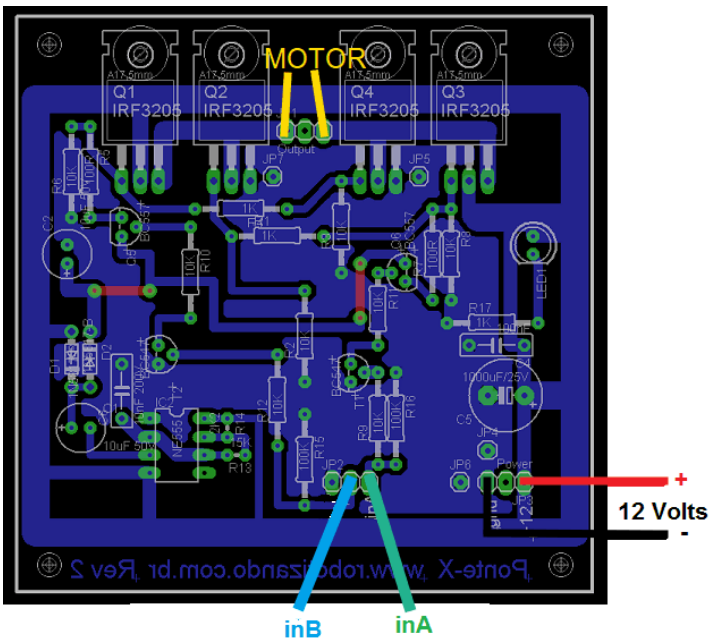
Devido ao fato do motor mais robusto, responsável pela movimentação vertical, drenar uma corrente superior à que a L298N suporta, foi confeccionada por Oliveira (2015) uma ponte H mais potente, baseando-se na ponte H desenvolvida por (FILHO, 2012), ilustrada na Figura 9. Assim como a ponte H tomada como base, esta também possui alimentação de 12V, mas suporta uma corrente de até 16A para garantir a corrente necessária para o acionamento do motor de movimentação vertical. Esta ponte H utiliza-se de transistores de efeito de campo (FET) para realizar o chaveamento, já que estes conseguem chavear correntes muito superiores em relação aos transistores bipolares, podendo chegar até a 1000 Ampéres em um único transistor (FILHO, 2012).

Figura 9 – Ponte H mais robusta desenvolvida para a movimentação vertical

Fonte: (FILHO, 2012).

Os pinos e conexões da ponte H para o controle vertical podem ser visualizados a seguir:

Figura 10 – Detalhe de conexões da Ponte H para a movimentação vertical



Fonte: (FILHO, 2012).

Funcionamento da Ponte H para a movimentação vertical (FILHO, 2012):

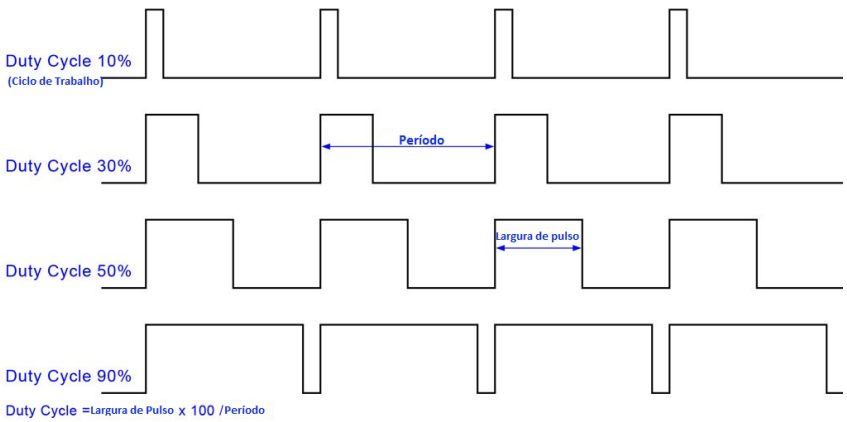
- (+/- 12 V) - Nestes pinos será conectada a fonte de alimentação de 12 V para fornecer a corrente necessária para que o motor CC gire.
- (Motor) - Nestes pinos serão conectados os terminais do motor CC a ser acionado.
- (inA) e (inB) - São os pinos responsáveis pelo controle PWM do motor CC. Para acionar a ponte H, deve-se entrar com um sinal de 5V no pino inA ou inB, para girar o motor no sentido horário ou anti-horário, respectivamente.
- Observação importante: Se as entradas inA e inB forem acionadas simultaneamente, uma situação de curto circuito ocorrerá e a ponte H poderá pegar fogo. Portanto deve-se ter este cuidado durante seu acionamento.

2.4.4 Controle de velocidade dos motores CC - Modulação PWM

O controle de velocidade dos motores CC é realizado por meio da tensão de alimentação aplicada em seus enrolamentos, isto é, quanto maior a tensão aplicada, mais rápido o motor CC gira. Este controle será feito através da aplicação de um sinal PWM à ponte H gerado pelo Arduino, já que este não possui saídas analógicas, mas somente digitais (0 ou 5V). O Arduino faz a geração deste sinal através da função *analogWrite*, que irá variar o ciclo de trabalho de acordo com a entrada, seja ela uma tensão nos pinos de entrada, ou um valor de entrada através da comunicação *Serial*, ou ainda um valor pré-definido. PWM significa *"Pulse Width Modulation"* ou Modulação de Largura de Pulso, ou seja, através da largura do pulso de uma onda quadrada é possível o controle de potência ou velocidade fornecida ao motor CC.

Para ilustrar melhor este conceito pode-se imaginar uma chave simples liga e desliga, que quando ligada, 100% da tensão e da potencia é aplicada à carga, e quando desligada, a tensão é nula e assim a potência é zero. Controlando-se o tempo que a chave permanece ligada, e, consequentemente, o tempo dela desligada, pode-se controlar a potencia média entregue a carga. Como exemplo, se a chave fica ligada durante 50% do ciclo e desligada durante os outros 50%, significa dizer que em média tem-se 50% do tempo com corrente e 50% sem. Portanto a potência média aplicada na carga é proporcional à tensão média, ou seja, 50%. Desse modo, quanto maior for o tempo que o pulso se mantém em nível lógico alto, ou seja, ligado, maior é a potência entregue à carga, assim como quanto menor for o tempo em nível lógico alto, menor a entrega de potência à carga. A Figura 11 a seguir ilustra um sinal PWM para diversos ciclos de trabalho distintos.

Figura 11 – Sinal PWM característico para diferentes ciclos de trabalho



Fonte: ATmega168A Pulse Width Modulation. Disponível em: <http://www.protostack.com/blog/2011/06/atmega168a-pulse-width-modulation-pwm/>. Acesso em 09/11/2016. Traduzido pelo Autor.

2.5 Sensoriamento da Planta

A fim de monitorar as grandezas necessárias para um controle eficiente da ponte rolante, estão presentes na planta diversos tipos de sensores. Para o controle e monitoramento da posição da carga, tanto horizontalmente, quanto verticalmente, estão presentes os *encoders* HEDS-5700. Já para ter acesso à variação angular da carga durante sua movimentação, foi utilizado o módulo acelerômetro e giroscópio GY-521, baseado no circuito integrado MPU-6050, da *InvenSense* (INVENSENSE, 2013). Finalmente, por questões de segurança foram inseridas chaves fim de curso com o intuito de desligar os motores CC em situações de emergência. Cada um destes sensores e seus princípios de funcionamento estão detalhados ao longo desta seção.

2.5.1 Controle e Monitoramento das posições horizontal e vertical da carga - *Encoders*

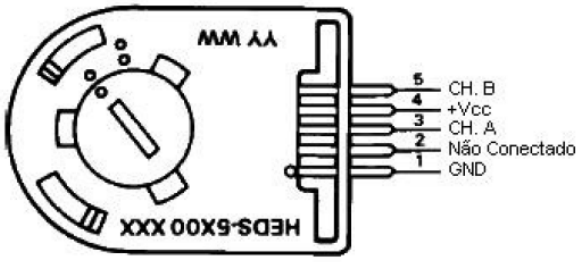
Para realizar a medição de posição e velocidade da carga, a planta possui instalados nos eixos dos motores dois *encoders* incrementais rotativos. O *encoder* é um transdutor que converte um movimento angular ou linear em uma série de pulsos digitais elétricos (SALDANHA, 2016). Os pulsos gerados podem ser utilizados para determinar a velocidade, a taxa de aceleração, distância, rotação, posição ou direção. Na planta em questão, o *encoder* presente é o HEDS-5700, um *encoder* óptico incremental de alta precisão, sendo possivel então obter uma alta resolução de pulsos por deslocamento angular, aumentando a qualidade dos dados coletados. O *encoder* utilizado neste projeto pode ser visualizado na Figura 12.

Figura 12 – *Encoder* HEDS-5700.

Fonte: Octopart. Disponível em: <https://octopart.com/heds-5701%23g00-avago-2516017>. Acesso em 09/11/2016.

O HEDS-5700 é alimentado por uma faixa de tensão de +4.5V a +5.5V, consumindo uma corrente típica de 17mA (AGILENT, 2016). Possui uma resolução de 256 ciclos por revolução e dois pinos de saída (canal A e canal B), juntamente com os pinos de alimentação, como pode ser visualizado na Figura 13.

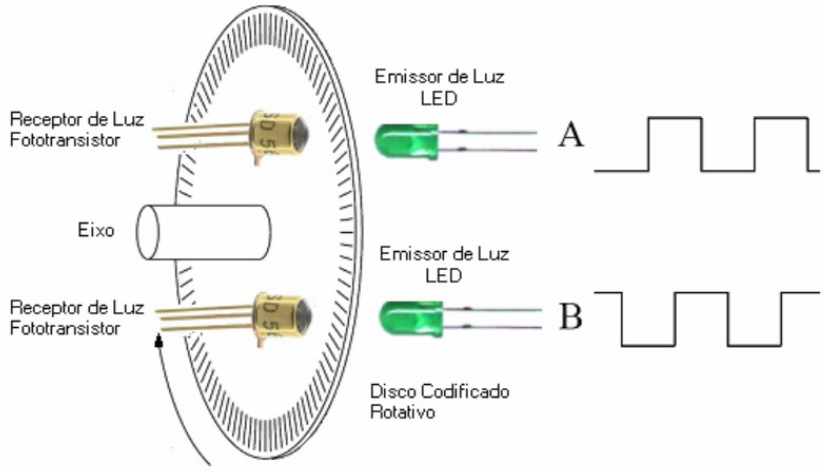
Figura 13 – Esquema de pinos do *Encoder* HEDS-5700.



Fonte: (AGILENT, 2016).

A Figura 14 ilustra o esquema de leitura realizada pelo *encoder*. A leitura é feita através de emissores infravermelho posicionados em uma extremidade, em conjunto com sensores, como fotodiodos ou fototransistores, posicionados na extremidade oposta. De acordo com a rotação do disco de plástico, partes claras ou escuras do disco passam diante dos sensores, acarretando na geração de um sinal pulsante que é coletado e enviado para o sistema supervisório. Cada pulso representará uma distância percorrida, possibilitando assim o cálculo da posição da carga e da velocidade dos motores de acionamento.

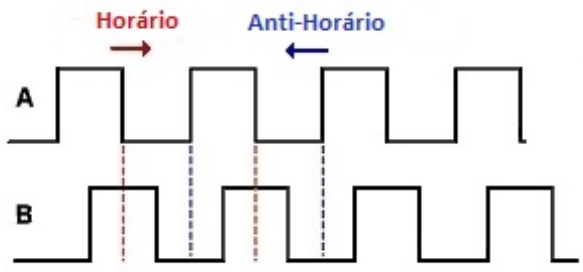
Figura 14 – Esquema de leitura de dados no encoder



Fonte:Como se Constituem e Operam os Motores CC Sem Escovas. Disponível em: <http://automoveiseletricos.blogspot.com.br/2015/05/como-se-constituem-e-operam-os-motores.html>. Acesso em 09/11/2016.

Para determinar o sentido de rotação, compara-se dois sinais gerados pelo *encoder* que estão em quadratura, ou seja, defasados entre si de 90 graus. Esta comparação é feita detectando-se as bordas de subida ou descida de um sinal, e comparando-a com o outro sinal. Dessa forma, o sistema supervisório incrementa ou decrementa a distância, de acordo com a referência previamente definida. A Figura 15 ilustra este funcionamento, em que é lido o sinal do canal B sempre que o sinal do canal A varia de positivo a zero. Caso o sentido de rotação seja horário, nota-se que o sinal B estará sempre em nível lógico alto no momento de leitura, enquanto que se a rotação estiver ocorrendo no sentido anti-horário, o sinal B estará sempre em nível lógico baixo no momento da leitura.

Figura 15 – Sinais em quadratura gerados pelo encoder.

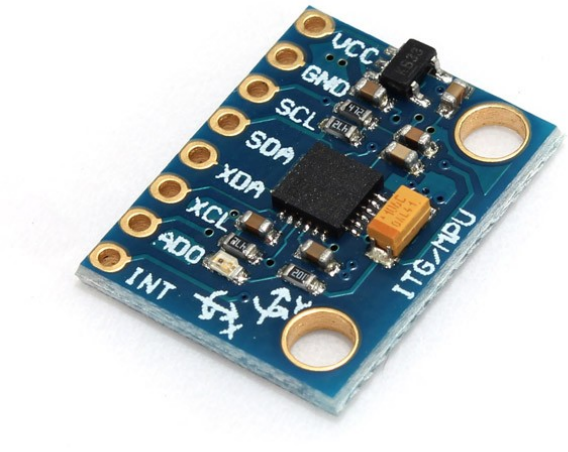


Fonte:Tutorial de *Encoder* rotatório. Disponível em: <http://www.hobbytronics.co.uk/rotary-encoder-tutorial>. Acesso em 09/11/2016. Traduzido pelo Autor.

2.5.2 Controle e Monitoramento da oscilação da carga - Módulo GY-521

Para determinar a variação angular da carga durante a sua movimentação foi utilizado o módulo GY-521, como mostrado na Figura 16, onde em uma mesma placa tem-se um acelerômetro e um giroscópio, que por sua vez são controlados pelo circuito integrado (CI) MPU-6050. Este CI possui 6 eixos (6 graus de liberdade), sendo 3 eixos para o acelerômetro e 3 eixos para o giroscópio (INVENSENSE, 2013). Além dos dois sensores, o CI tem embutido um recurso chamado DMP (*Digital Motion Processor*), responsável por realizar cálculos a partir dos dados gerados pelos sensores que podem ser usados para sistemas de reconhecimento de gestos, determinação de ângulos de inclinação, navegação (GPS), jogos e diversas outras aplicações. Outro recurso adicional presente, mas que não foi utilizado neste projeto, é o sensor de temperatura embutido no CI, cujas especificações podem ser encontradas em (INVENSENSE, 2013).

Figura 16 – Módulo Acelerômetro e Giroscópio GY-521.



Fonte: Eletrodex Eletrônica. Disponível em: <http://www.eletrodex.com.br/mpu-6050-modulo-giroscopio-e-acelerometro-digital-6-dof.html>. Acesso em 07/07/2017.

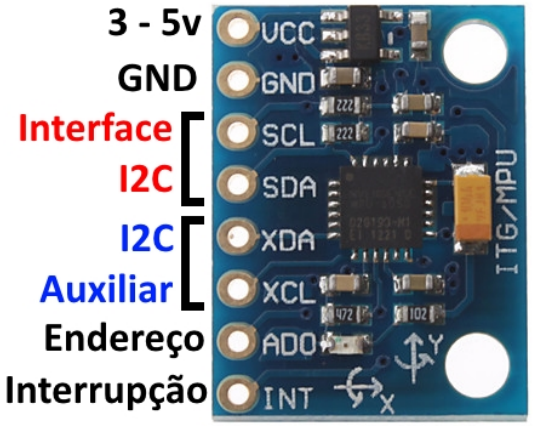
Especificações técnicas do módulo GY-521 (INVENSENSE, 2013):

- Controlador: MPU-6050.
- Tensão de Operação: 3 a 5V DC.
- Resolução do ADC (Conversor Analógico-Digital): 16 bits.
- Interface de comunicação: I^2C.
- Faixa de leitura do Giroscópio: ±250, ±500, ±1000, ±2000°/s.
- Faixa de leitura do Acelerômetro: ±2, ±4, ±8, ±16g.
- Leitura do sensor de temperatura: -40 a 85°C.
- Dimensões: 17mm(L) X 3mm(A) X 21mm(C).
- Peso: 2g.

2.5.2.1 Pinagem e endereçamento do GY-521

A comunicação do MPU-6050 com o Arduino utiliza a interface I^2C, por meio dos pinos SCL e SDA do sensor, como pode ser visto na Figura 17. Nos pinos XDA e XCL pode-se conectar outros dispositivos I^2C, possibilitando a criação de um sistema de orientação mais completo. A alimentação do módulo pode variar entre 3 e 5 V, sendo recomendado a utilização de 5 V para a obtenção de melhores resultados (INVENSENSE, 2013). O pino AD0 desconectado define que o endereço I^2C do sensor é 0x68. Caso o pino AD0 seja conectado ao pino 3.3V do Arduino, seu endereço será alterado para 0x69, permitindo a utilização de dois módulos MPU-6050 em um mesmo circuito. O pino INT pode ser conectado ao pino de interrupção do Arduino para auxiliar o microcontrolador, sinalizando quando ocorrem novas leituras pelos sensores.

Figura 17 – Pinagem do Módulo Acelerômetro e Giroscópio GY-521.

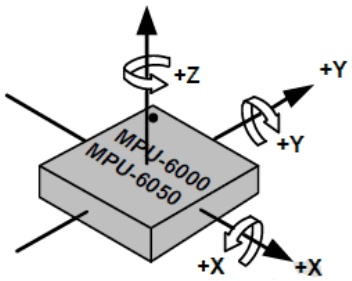


Fonte: Tutorial: Acelerômetro MPU6050 com Arduino. Disponível em: <http://blog.filipeflop.com/sensores/tutorial-acelerometro-mpu6050-arduino.html>. Acesso em 07/07/2017.

2.5.2.2 Princípio de Funcionamento do GY-521

Como mencionado anteriormente, o MPU-6050 possui 6 DOF (*Degrees of Freedom*, ou graus de liberdade), o que significa que ele provê 6 valores como saída, sendo três desses provenientes do acelerômetro (um valor para cada um dos três eixos), e três do giroscópio, obtendo-se também um valor para cada um de seus eixos, como ilustrado na Figura 18. Para obter a oscilação angular do módulo, e consequentemente da carga, já que este estará acoplado à mesma, combinam-se as leituras advindas destes dois sensores, como detalhado na Seção 3.6.

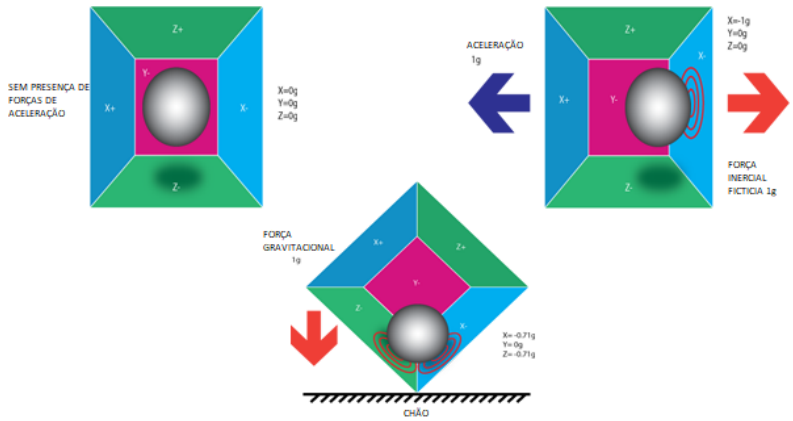
Figura 18 – Eixos de detecção de rotação do GY-521.



Fonte: Arduino - Interface com acelerômetro e giroscópio. Disponível em: <https://www.embarcados.com.br/arduino-acelerometro-giroscopio/>. Acesso em 07/07/2017.

O acelerômetro funciona a partir do princípio do efeito piezoelétrico. Para ilustrar seu funcionamento, pode-se imaginar uma caixa em forma de cubo com uma pequena esfera em seu interior, como na Figura 19. Suponha-se que as paredes desta caixa são feitas de cristais piezoelétricos. Sempre que houver uma inclinação da caixa, a esfera será empurrada na direção da inclinação devido à força da gravidade. Ao ocorrer a colisão entre a esfera e a parede, pequenas correntes piezoelétricas serão formadas nesta parede. Existem três pares de paredes opostas em um cubo como o da figura, onde cada par corresponde à um eixo no espaço 3D: os eixos X, Y, e Z. Dependendo da corrente produzida pelas paredes piezoelétricas pode-se determinar a direção da inclinação, assim como a sua magnitude.

Figura 19 – Funcionamento do Acelerômetro.

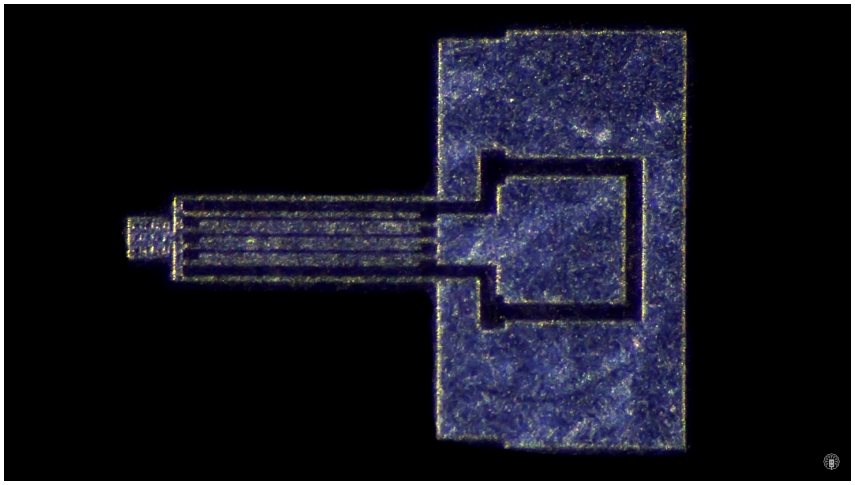


Fonte: IMU Interfacing Tutorial: Get started with Arduino and the MPU 6050 Sensor. Disponível em: <https://diyhacking.com/arduino-mpu-6050-imu-sensor-tutorial/>. Acesso em 07/07/2017. Traduzido pelo Autor.

Na prática, o acelerômetro deste módulo trabalha com um arranjo de semicondutores microscopicamente espaçados entre si, onde quando uma força é aplicada sobre um destes arranjos, a distância entre estes semicondutores varia, variando assim a capacitância que é monitorada pelo circuito integrado, sendo convertido então a um valor correspondente de

aceleração naquele eixo (INVENSENSE, 2013). A Figura 20 ilustra um destes arranjos.

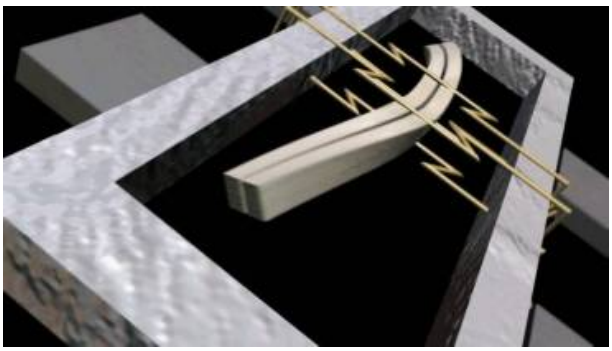
Figura 20 – Arranjo de semicondutores do Acelerômetro.



Fonte: How an accelerometer works. Disponível em: <http://www.afrotechmods.com/>. Acesso em 07/07/2017.

O giroscópio, por sua vez, funciona a partir do princípio da força inercial de Coriolis (INVENSENSE, 2013). Suponha-se que exista uma estrutura microscópica em forma de garfo que está em constante oscilação para frente e para trás, que está sendo mantida no lugar por cristais piezoelétricos, como ilustrado na Figura 21. Ao ocorrer uma torção neste sistema, os cristais sofrerão a ação de uma força na direção da inclinação, causada pela inércia da estrutura oscilatória, que tende a permanecer oscilando na mesma direção inicial. Esta força aplicada aos cristais causam então o efeito piezoelétrico, gerando uma corrente elétrica que será tratada pelo microcontrolador, gerando o sinal de velocidade angular. Da mesma forma que o acelerômetro, porém, no caso do sensor deste projeto utiliza-se o mesmo princípio de Coriolis, porém coletando-se a capacitância entre semicondutores (INVENSENSE, 2013).

Figura 21 – Funcionamento do Giroscópio.



Fonte: IMU Interfacing Tutorial: Get started with Arduino and the MPU 6050 Sensor. Disponível em: <https://diyhacking.com/arduino-mpu-6050-imu-sensor-tutorial/>. Acesso em 07/07/2017.

Como os sinais provenientes do acelerômetro e do giroscópio são fornecidos separadamente, cabe a implementação de técnicas de filtragem para combinar estes dois sinais a fim de

obter uma leitura mais precisa da inclinação da carga. Dentre os filtros mais utilizados estão o Filtro Complementar e o Filtro de Kalman (AGUIRRE, 2007), sendo o segundo o escolhido para implementação neste projeto, devido à sua maior precisão de leitura. Detalhes de sua implementação podem ser obtidas na Seção 3.6.2.

2.5.3 Chaves fim de curso

A fim de garantir a proteção do sistema, fazendo com que o carro permaneça no intervalo de espaço delimitado da ponte rolante quando movendo-se horizontalmente, e também que a carga não suba além do limite permitido, forçando desta forma o motor, são utilizadas chaves fins de curso que interromperão a rotação do motor quando acionadas. Ocorrendo seu acionamento, automaticamente a tensão de controle dos motores será cortada, prevenindo assim qualquer ação indesejada no sistema, protegendo-o. Na planta em questão existem 4 chaves fim de curso. Uma em cada extremidade do caminho do carro, protegendo o sistema de controle horizontal, e duas abaixo do mesmo, que são acionadas quando a carga é tracionada para cima além do limite, impedindo que ambos os motores forcem a carga para cima. A chave mecânica fim de curso pode ser visualizada na Figura 22 a seguir.

Figura 22 – Chave mecânica fim de curso.

Fonte: Chave Fim de Curso Metaltex FM-1701. Disponível em: <https://www.eletropecas.com/Produto/chave-fim-de-curso-metaltex-fm-1701>. Acesso em 09/11/2016.

3 DESENVOLVIMENTO DO *SOFTWARE* DA PONTE ROLANTE

Os centros de processamento do sistema desenvolvido para controle e monitoramento da maquete de ponte rolante deste projeto giram em torno do Arduino e do MATLAB. No Arduino está implementado todo o *software* de configuração, acionamento e monitoramento dos elementos de *hardware* descritos no capítulo anterior. Já no MATLAB, funcionando em um computador no qual o Arduino está conectado via USB, está implementado o *software* de desenvolvimento da interface gráfica e da comunicação *Serial* entre o MATLAB e o Arduino. A comunicação *Serial* entre estes dois elementos é fundamental para que seja possível tanto enviar comandos que foram gerados pelo usuário a partir da interface gráfica do MATLAB para o Arduino, quanto receber os dados dos sensores captados pelo Arduino durante a execução do programa, para que os mesmos sejam disponibilizados por meio de gráficos na interface gráfica.

A ideia inicial do projeto era de que todo o *software* fosse implementado somente no MATLAB, utilizando-se de *toolboxes* próprias do MATLAB para Arduino. Porém após a realização de testes de obtenção dos dados do sensores, notou-se que o alto custo computacional imposto pelo MATLAB acarretava em uma taxa de amostragem muito baixa dos dados coletados, gerando valores não muito confiáveis, principalmente em relação à variação angular da carga. Desse modo, foi escolhido implementar o *software* da maneira descrita no parágrafo anterior, já que deste modo o cálculo e a obtenção das grandezas a serem monitoradas não dependem mais do processamento do MATLAB no computador, mas somente do próprio *hardware* do Arduino, onde conseguiu-se obter uma taxa de amostragem cerca de 20 vezes maior em relação à obtida com a configuração utilizando somente o MATLAB.

O *software* responsável pela implementação do código de programação desenvolvido para o Arduino é o *Arduino Integrated Development Environment*, ou Arduino IDE. Este *software* utiliza a chamada linguagem de programação Arduino, que é uma linguagem de programação baseada em C/C++, com pequenas modificações (ARDUINO, 2016). Para realizar a comunicação entre o Arduino e dispositivos periféricos, a linguagem dispõe de diversas bibliotecas próprias, que serão detalhadas no decorrer deste capítulo.

O MATLAB (de *MATrix LABoratory*) é um programa produzido pela *Mathworks, Inc.*, que consiste de uma ferramenta de computação numérica de análise e visualização de dados. Possui uma linguagem de programação de alto nível que proporciona um ambiente fácil de usar, já que os problemas e soluções são expressos em linguagem matemática,

e não na linguagem de programação tradicional. A ferramenta MATLAB tem como principais objetivos a construção de gráficos, compilação de funções, manipulação de funções específicas de cálculo e variáveis simbólicas. O *software* fornece também um conjunto de subprogramas (*toolboxes*) que solucionam problemas diversos tais como: álgebra matricial, aritmética com complexos, sistemas de equações lineares, determinação de autovalores e autovetores, solução de equações diferenciais, solução de equações não lineares, além de representar e de dar subsídios para a análise e síntese de sistemas lineares e não lineares (MATHWORKS, 2016).

3.1 Funcionamento do sistema

A execução do sistema de movimentação da carga na maquete da ponte rolante consiste de uma série de etapas a serem realizadas. Na primeira delas, estabelece-se a comunicação entre o MATLAB e o Arduino através do comando dado pelo usuário a partir da interface gráfica. Com a conexão estabelecida, o Arduino fica ocioso até que seja dado algum comando de movimentação por parte do usuário. O usuário então configura os parâmetros de movimentação desejada através da interface, e solicita o início do movimento. Neste momento, o MATLAB emite via *Serial* um comando consistindo de um conjunto de caracteres contendo todos os parâmetros de movimentação definidos pelo usuário, e passa a monitorar a porta *Serial* para posteriormente receber informações de posição e variação angular da carga durante a execução da tarefa. Este conjunto de caracteres que foi enviado é então lido e interpretado pelo Arduino, que realiza as instruções de acionamento correspondentes, recolhendo e enviando para o MATLAB via *Serial* as informações mencionadas, que serão disponibilizadas via gráficos para o usuário através da interface gráfica.

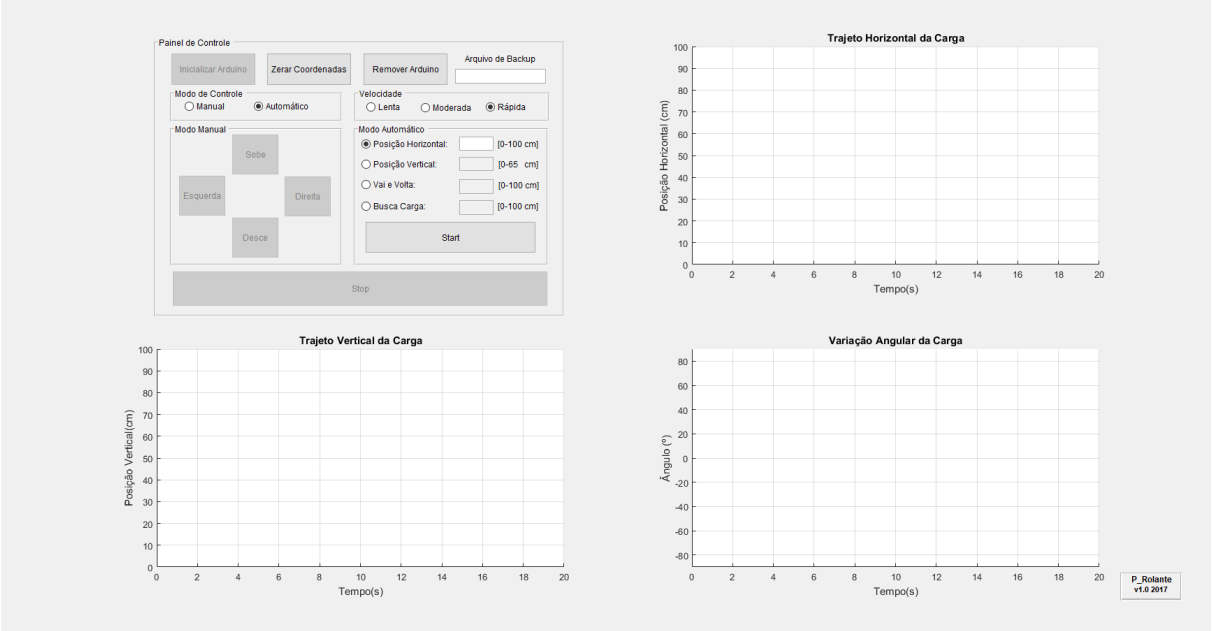
Nas seções subsequentes deste capítulo serão detalhados os procedimentos realizados em cada uma das duas plataformas em questão, responsáveis por realizar o controle e monitoramento da planta de forma sincronizada e eficiente.

3.2 A Interface gráfica

A interface gráfica para controlar a planta deste projeto foi desenvolvida utilizando a ferramenta GUIDE do MATLAB. O GUIDE é um instrumento projetado para que o usuário construa interfaces gráficas com maior facilidade e rapidez. A *toolbox* GUIDE contém ferramentas para criar, instalar, alinhar e alterar o tamanhos de objetos *uicontrol*,

que são objetos gráficos adicionados à interface que tem como função realizar a comunicação entre o usuário e o programa criado. Mais informações sobre a utilização do GUIDE podem ser obtidos em (TEIXEIRA, 2008). A figura a seguir ilustra a interface gráfica desenvolvida para este projeto.

Figura 23 – Interface Gráfica desenvolvida no GUIDE para o controle da ponte rolante.



Fonte: Elaborado pelo Autor.

A interface conta com um painel de controle, por onde são enviados os comandos de movimentação, além de gráficos onde serão disponibilizados os dados obtidos após a execução do programa. O painel de controle conta com duas seções principais, onde pode-se realizar um controle manual e instantâneo de movimentação da carga, ou um controle automático com quatro modos diferentes de percurso pré-definidos. Antes de começar qualquer movimentação da carga, deve-se estabelecer a comunicação entre o MATLAB e o Arduino, através do botão 'Inicializar Arduino'. Da mesma maneira, ao finalizar a utilização do programa, deve-se encerrar a comunicação entre os elementos mencionados através do botão 'Remover Arduino'.

No primeiro modo de controle, o controle manual, ao pressionar um dos quatro botões de direção disponíveis para o movimento da carga (sobe, desce, para a esquerda ou para a direita), os motores serão acionados para mover a carga na direção selecionada de acordo com uma das três velocidades pré-definidas (lenta, moderada ou rápida), até que seja pressionado o botão *'Stop'*, que fará com que os motores desliguem, finalizando assim a movimentação da carga.

No segundo modo de controle, o controle automático, há a opção de escolher entre

quatro diferentes tipos de percurso para a carga, que serão executados com uma das três velocidades selecionadas pelo usuário. No primeiro deles, onde define-se somente a posição horizontal, ao configurar a posição desejada em centímetros, os motores serão acionados para movimentar a carga horizontalmente desde a posição atual até a posição desejada. Já no segundo tipo define-se a posição vertical na qual a carga será posicionada, seguindo o mesmo plano de coordenadas da movimentação anterior.

O plano de coordenadas de posição horizontal e vertical tem como referência a posição inicial em que encontra-se a carga no momento em que o sistema é inicializado. Desse modo, caso seja necessária uma calibração da posição inicial da carga, deve-se utilizar o modo de controle manual para posicionar a carga na origem do plano de coordenadas, e utilizar o botao 'Zerar Coordenadas', presente na interface, para que aquele se torne o ponto de referência na qual as coordenadas de posição inseridas na interface serão baseadas.

A terceira opção de controle automático consiste de um movimento horizontal puro 'Vai e Volta', onde o *trolley* se desloca até a posição horizontal definida na interface, e retorna à origem do sistema de coordenadas, sem haver deslocamento vertical da carga. O movimento é desenvolvido seguindo a velocidade selecionada pelo usuário.

Na quarta e última opção de controle automático tem-se uma simulação de recolhimento de carga na posição horizontal desejada, nomeado 'Busca Carga'. Neste modo, também obedecendo a velocidade selecionada pelo usuário, o *trolley* movimenta-se horizontalmente até a posição definida na interface, realizando a seguir um movimento de descida simulando a coleta de uma carga no solo, retornando então à posição vertical inicial, com posterior movimentação horizontal de volta para a posição inicial do plano de coordenadas. O botão *'Stop'* também pode ser utilizado para interromper a movimentação do conjunto *trolley*/carga durante qualquer um dos quatro tipos de controle do modo automático.

Ao final de cada comando mencionado anteriormente, são gerados gráficos demonstrando o comportamento de deslocamento vertical e horizontal da carga, além também da oscilação angular da carga em relação ao eixo vertical, medida durante a realização do percurso. Quando na utilização do modo automático, pode-se obter os dados provenientes das medições também através de um arquivo de texto, cujo nome deve ser especificado no campo denominado 'Arquivo de Backup', como por exemplo 'dados.txt'. Neste arquivo, cada grandeza medida ocupa uma das quatro colunas de dados, na seguinte ordem: 'tempo decorrido','coordenada de posição horizontal do *trolley*','coordenada de posição vertical da carga', e 'variação angular da carga em relação ao eixo vertical durante o movimento'.

A lógica de programação de intertravamento dos botões da interface gráfica, assim como

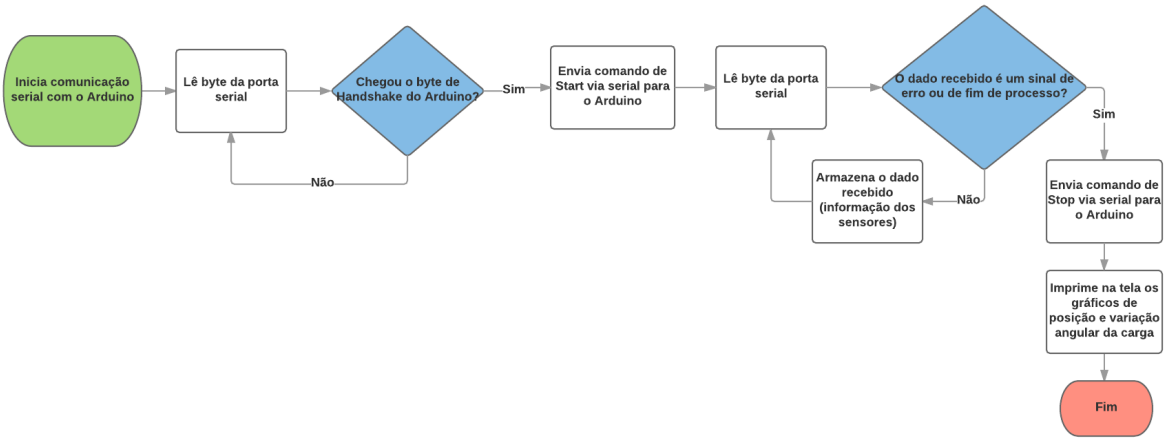
as lógicas de comunicação, controle e monitoramento desenvolvidas para cada tipo de configuração escolhida nesta mesma interface serão detalhadas nas próximas seções deste capítulo.

3.3 Implementação do *software* da interface gráfica, de envio de comandos e recebimento de informações no MATLAB

No MATLAB foi desenvolvida a porção do *software* que trata da interface gráfica e faz a comunicação com o Arduino para enviar comandos e receber as grandezas desejadas via *Serial*. A programação da lógica de ativação e desativação dos botões da interface gráfica é de suma importância para o correto funcionamento do sistema, já que todas as ações são tomadas com base em quais botões estão pressionados ou ativados no momento em que o MATLAB faz a comunicação com o Arduino e envia o comando de movimentação.

O fluxograma a seguir sintetiza o funcionamento do algoritmo de movimentação automática implementado para o MATLAB, a partir do momento em que o usuário inicializa o Arduino através da interface gráfica. O algoritmo de movimentação manual segue o mesmo princípio, porém realiza o movimento indefinidamente até que o usuário pressione o botão 'Stop' ou que as chaves fim de curso sejam acionadas. Os diversos trechos presentes no fluxograma serão detalhados através da demonstração do código desenvolvido para a realização das tarefas apresentadas na figura.

Figura 24 – Fluxograma de funcionamento do *software* implementado no MATLAB.



Fonte: Elaborado pelo Autor.

3.3.1 Intertravamento dos botões da interface gráfica

Devido à dependência dos comandos a serem realizados com os botões da interface que estarão ativos naquele momento, foi necessário realizar ao longo de todo o código lógicas de intertravamento destes botões para que não ocorressem conflitos de ações a serem tomadas. Como exemplo, ao acionar o botão de modo de controle manual, desabilita-se todos os botões do modo de controle automático de modo a evita-se uma situação onde um movimento automático poderia ser solicitado enquanto ocorre uma movimentação manual da carga, por exemplo. O algoritmo completo implementado no MATLAB pode ser visualizado com os comentários pertinentes no Apêndice A.

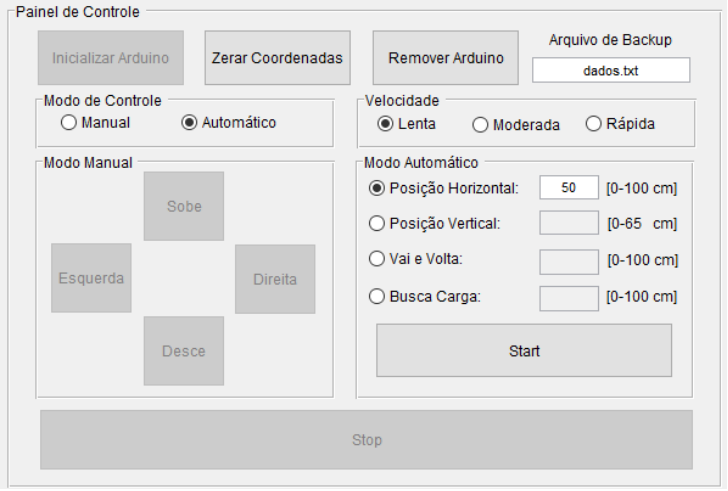
3.3.2 Inicialização da comunicação entre MATLAB e Arduino

Suponha-se para fins ilustrativos que o desejo do usuário seja de movimentar a carga apenas horizontalmente, até a posição de 50 cm, com velocidade lenta, partindo da origem. Na primeira parte do fluxograma da Figura 24 é estabelecida a comunicação entre o MATLAB e o Arduino. Tal comunicação é iniciada quando o usuário pressiona o botão 'Inicializar Arduino', através da interface gráfica, já que a mesma funciona a partir do princípio em que cada botão executa uma função pré-definida ao ser pressionado pelo usuário. Deste modo, a função atrelada ao botao 'Inicializar Arduino', além das configurações de ativação e desativação de botões similares às já mencionados anteriormente e da declaração de variáveis a serem utilizadas durante a execução do programa, realiza também a abertura de um canal de comunicação *Serial* com o Arduino, utilizando a taxa de transferência pré-definida (*baud*), que deve ser idêntica à utilizada pelo Arduino para que a conexão seja estabelecida com sucesso.

3.3.3 Configuração da interface gráfica e envio de comandos

Após a etapa de inicialização, instruções que serão detalhadas posteriormente na Seção 3.4 são realizadas por parte do Arduino, que ficará enviando periodicamente um sinal, aguardando o comando de movimentação por parte do MATLAB. Para a movimentação exemplo proposta nesta seção, o usuário então selecionará a opção de modo de controle automático, de posição horizontal, com velocidade lenta, preenchendo o campo com a posição desejada (50 cm), como ilustrado na Figura 25 a seguir.

Figura 25 – Interface gráfica com as opções do exemplo de movimentação selecionadas.



Fonte: Elaborado pelo Autor.

Deste modo, ao pressionar o botão 'Start', executa-se uma nova função, onde será enviado via *Serial* ao Arduino o comando de movimentação desejado através da função *fprintf*. Durante a execução deste algoritmo, o MATLAB trata de aceitar o "primeiro contato"solicitado pelo Arduino e enviar via *Serial* para o mesmo o *string* contendo as informações da movimentação desejada pelo usuário, onde no caso tomado como exemplo será *'<start,automatico,horizontal ,lenta,setpoint_h>'*. Este *string* será interpretado pelo *software* implementado no Arduino, que iniciará o processo de movimentação da carga, enviando em seguida para o MATLAB as informações dos sensores, contendo as posições horizontal e vertical do conjunto *trolley*/carga, além também da oscilação angular da carga durante o processo.

3.3.4 Obtenção das variáveis monitoradas

O MATLAB então ficará monitorando a porta *Serial*, recebendo do Arduino as medições das grandezas a serem monitoradas até que o movimento seja finalizado, ou até que o usuário pressione o botão 'Stop' para parar o processo. Nesta operação foi implementado um sistema em que as grandezas a serem monitoradas são enviadas do Arduino para o MATLAB em "blocos"contendo três elementos, sendo cada um destes uma amostra de cada uma das três variáveis fornecidas pelos sensores (posição horizontal, posição vertical e inclinação da carga). Ao fim do recebimento de cada bloco, cada elemento é então alocado em seu vetor correspondente, sendo este procedimento repetido até que o Arduino envie um sinal indicando que o processo foi concluído (flag_fim).

Cogitou-se a possibilidade de geração dos gráficos enquanto o processo de movimentação

estava em andamento. Porém o alto custo computacional imposto pela tarefa de geração de gráficos ao mesmo tempo em que se obtém dados via *Serial* acarretou em uma dessincronização na comunicação entre o MATLAB e o Arduino, resultando em uma leitura incorreta dos dados enviados para o MATLAB e impossibilitando assim tal implementação.

3.3.5 Tratamento dos dados obtidos via *Serial*

Devido à impossibilidade de geração de gráficos durante a operação, os dados obtidos só serão tratados ao final do processo, quando são escritos no arquivo 'dados.txt', e os gráficos são gerados na tela da interface gráfica. Os dados no arquivo de texto são armazenados em quatro colunas, em que cada uma destas contém as medições de tempo, posição horizontal, posição vertical e ângulo medido, respectivamente.

Assim como o exemplo aqui descrito, todos os outros tipos de movimentação, seja ela manual ou automática, são baseados na mesma rotina principal explicitada anteriormente. As mudanças ocorrerão nas condições a serem satisfeitas durante o código, que irão variar de acordo com os botões que estão selecionados no momento em que se dá o comando de início de movimento. Ao final da análise de condições, o MATLAB envia então para o Arduino o *string* contendo as informações pertinentes para o tipo de movimento desejado.

3.3.6 Comando de interrupção de movimento

Supondo que o usuário deseje interromper a movimentação durante a execução da tarefa, deverá ser pressionado o botão 'Stop', que ao mudar de estado, ativará uma condição na rotina principal que enviará para o Arduino um *string* contendo o comando de parada. O Arduino então lê este *string* contendo a palavra 'stop', faz seu tratamento e interrompe a tarefa que estava sendo executada. Caso haja algum erro de leitura dos sensores, o Arduino enviará para o MATLAB uma sinalização e interromperá a tarefa.

3.3.7 Comando de realocação da origem do sistema de coordenadas

Com a planta parada, o usuário pode utilizar o botão 'Zerar Coordenadas' para transformar a posição atual do conjunto *trolley*/carga no ponto de origem do plano em que a movimentação será baseada. Ao ser pressionado, o botão mudará de estado e executará a

função que realizará esta operação. Desta vez será enviado um *string* contendo a palavra 'zera', que será tratada pelo Arduino para reposicionar a origem do sistema de coordenadas.

3.3.8 Comando de encerramento de comunicação entre MATLAB e Arduino

Ao final da utilização da interface gráfica, deve-se encerrar a comunicação *Serial* entre o MATLAB e o Arduino através da utilização do botão 'Remover Arduino'. O pressionamento deste botão ativará uma caixa de texto que, por questões de segurança, informará ao usuário para desligar as fontes de alimentação antes de encerrar a comunicação, já que podem ocorrer situações onde ao encerrar a comunicação com as fontes ligadas, o Arduino poderá gerar saídas altas em ambas as entradas da ponte H responsável pela movimentação vertical, fato que pode gerar reações adversas, como mencionado na Seção 2.4.3.

O algoritmo completo, comentado, contendo todos os tipos de movimentação e configurações de intertravamento de botões que foi desenvolvido para o MATLAB pode ser visualizado no Apêndice A.

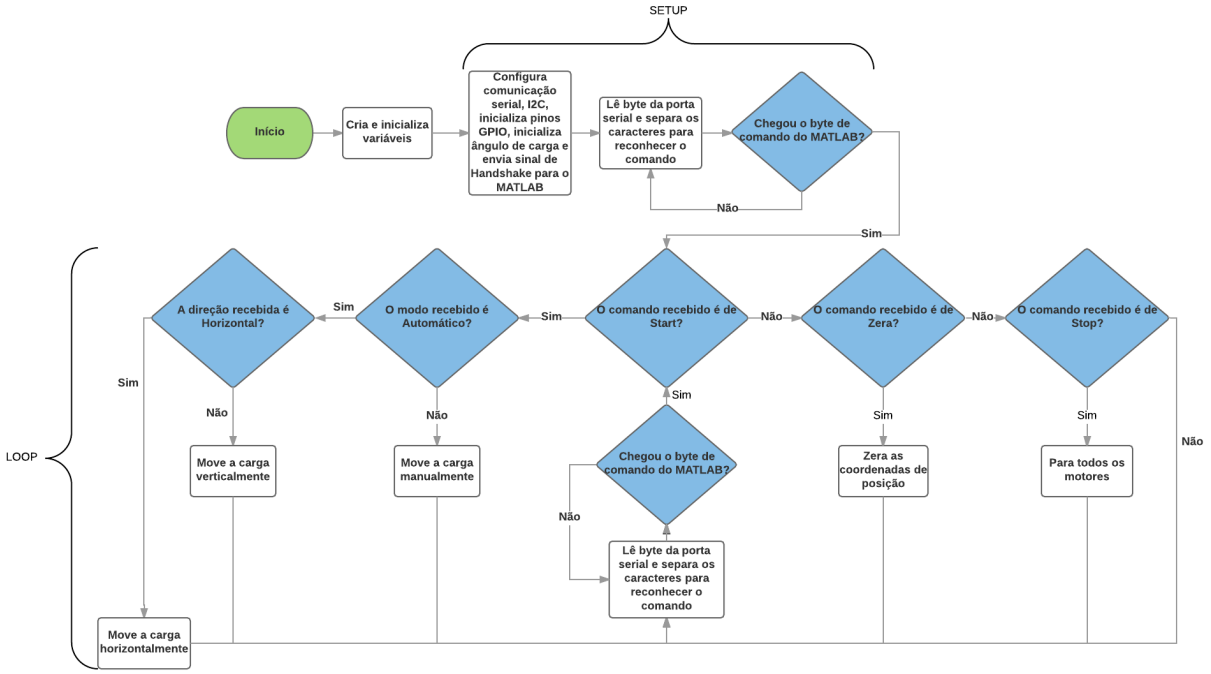
3.4 Implementação do *software* de acionamento da planta e de aquisição de dados no Arduino - Parte 1: Inicialização e obtenção dos comandos

No Arduino, utilizando as linguagens de programação C/C++ através do Arduino IDE foi implementada a porção do *software* que faz o acionamento dos motores na planta, assim como a obtenção das grandezas a serem monitoradas através da comunicação com os sensores mencionados na Seção 2.5. O algoritmo desenvolvido possui basicamente duas partes principais, sendo a primeira, de nível mais alto, o gerenciamento e tratamento dos comandos que chegam via *Serial* do MATLAB, e a segunda, de nível mais baixo, as execuções das ações definidas por tais comandos, ou seja, a realização da movimentação da carga ao longo da planta.

Serão abordadas inicialmente as funções que tratam os comandos advindos do MATLAB, consistindo então da inicialização e configuração do sistema no Arduino (*"Setup"*), em conjunto com as funções que serão executadas em repetição (*"loop"*), para que o Arduino esteja sempre à disposição do MATLAB para executar as operações requeridas pelo usuário. A seguir tem-se então o fluxograma principal de funcionamento do Arduino, com as etapas de *"Setup"* e *"loop"* devidamente identificadas. Os diversos trechos presentes no fluxograma serão detalhados através da demonstração do código desenvolvido para a realização das

tarefas apresentadas na figura.

Figura 26 – Fluxograma principal de funcionamento do algoritmo implementado no Arduino.



Fonte: Elaborado pelo Autor.

3.4.1 Inicialização do Arduino

No momento em que é aberta a comunicação *Serial* entre o MATLAB e o Arduino, através do pressionamento do botão 'Inicializar Arduino' pelo usuário na interface gráfica, começa a ser executada a função *setup*, para definir as configurações iniciais do Arduino. Neste processo, são inicializados os pinos de entrada e saída, as variáveis a serem utilizadas no processo, além também da inicialização das bibliotecas de comunicações *Serial* e I^2C, para a interação com o MATLAB e com o módulo GY-521, responsável pela medição do ângulo de carga, respectivamente. São também realizadas as configurações iniciais para o módulo sensor de ângulo mencionado, assim como a calibração do mesmo para uma medição de ângulo mais precisa, sendo posteriormente executada a função *estabelece_contato*(), onde o Arduino fica ocioso aguardando o contato do MATLAB.

3.4.2 Estabelecimento da comunicação entre Arduino e MATLAB

Como mencionado anteriormente, após a inicialização do Arduino, enquanto não houver um comando fornecido por parte do MATLAB, o Arduino permanece na execução da função

estabelece_contato(), que envia repetidamente via *Serial* para o MATLAB a sinalização para estabelecer contato e receber os comandos. Assim que houver o primeiro envio de comando do MATLAB para o Arduino, a função mencionada será terminada, e o programa entrará no *loop* principal, onde serão executados os comandos dados pelo MATLAB. Ao entrar no *loop*, a primeira tarefa a ser realizada é coletar os dados enviados pelo MATLAB e tratá-los para realizar as funções de acordo com o comando correspondente.

3.4.3 Leitura e interpretação do comando enviado via *Serial* pelo MATLAB

Será utilizado como exemplo ilustrativo a mesma situação exemplificada na seção 3.3, onde foi dado pelo usuário um comando de modo de controle automático, posição horizontal, velocidade lenta, e posição desejada de 50 cm partindo da origem, na interface do MATLAB. O *string* contendo o movimento requerido (*'<start,automatico,horizontal ,lenta,setpoint_h>'*) foi então enviado do MATLAB para o Arduino, e será tratado pela função *le_serial*().

A função *le_serial*(), por sua vez, executa duas outras funções, cada uma com um objetivo específico. Primeiramente, é chamada a função *recebe_string*(), que possui o papel de armazenar em um vetor todos os caracteres enviados pelo MATLAB. Para realizar esta tarefa com eficiência, foi necessário definir marcadores para o início ('<') e o fim ('>') dos dados enviados pelo MATLAB, para que o programa consiga identificar com precisão onde começa e onde termina a informação útil que deve ser coletada no *buffer serial*. Assim, esta função lê *byte* por *byte* do *buffer*, onde quando encontrar um marcador de início ('<'), começa a armazenar os próximos *bytes* para serem tratados, até que encontre o marcador de fim ('>').

Neste ponto o programa já possui armazenadas no vetor *string_recebido* as informações de comando a serem executadas, tendo como conteúdo o *string start,automatico,horizontal ,lenta,setpoint_h*. Porém ainda é necessário separar cada um destes comandos em uma variável própria, para que seja possível executar as funções de acordo com cada parâmetro desejado. Esta tarefa é realizada pela função *separa_palavras*(), que assim como a função anterior, também utiliza um marcador para identificar o fim de cada palavra, desta vez utilizando a vírgula (",") como marcador para separar as palavras. Após esta etapa, tem-se armazenados cada um dos parâmetros de comando em suas respectivas variáveis para que a ação de movimentação possa ser tomada nas próximas etapas do *loop*.

3.4.4 Identificação do movimento solicitado

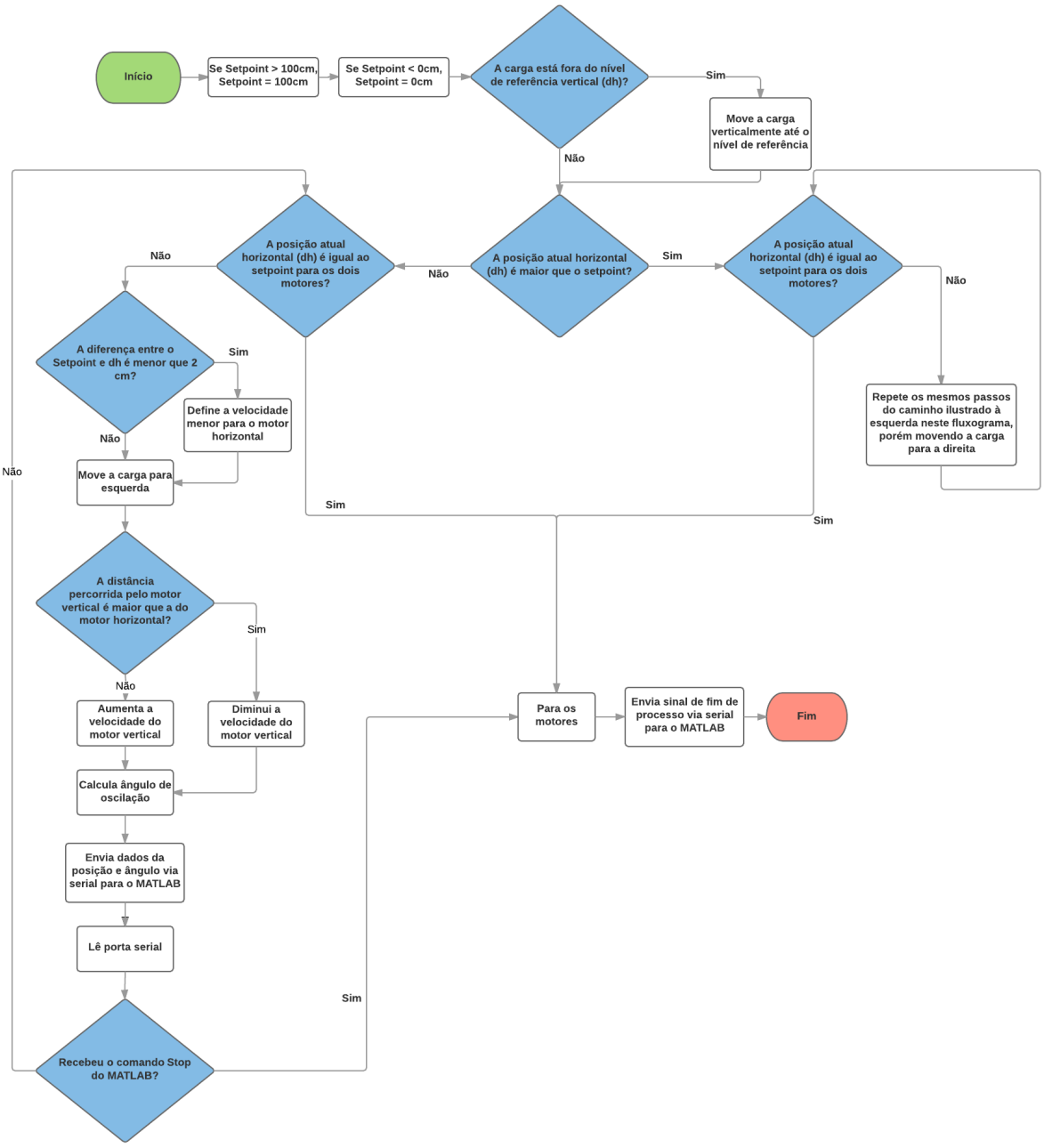
O próximo passo é verificar as condições de cada um dos parâmetros obtidos e com isto executar a função pertinente que realizará a movimentação desejada pelo usuário. Assim, satisfeitas as condições para a movimentação desejada, são chamadas as funções de movimentação propriamente ditas, onde é realizada toda a segunda parte mencionada no início desta seção, que trata do acionamento dos motores, e da coleta e envio de dados dos sensores para o MATLAB.

Ao final da execução da função de movimentação, é chamada novamente a função *estabelece_contato*(), que mantém o Arduino ocioso até que um novo comando seja enviado pelo MATLAB, finalizando assim o ciclo demonstrado no fluxograma da Figura 26. Todos os outros tipos de movimentação seguem este mesmo princípio, variando apenas os parâmetros, e por consequência, as funções de acionamento a serem chamadas.

3.5 Implementação do *software* de acionamento da planta e de aquisição de dados no Arduino - Parte 2: As funções de movimentação

Na segunda parte da execução do código, que trata das funções de movimentação, pode-se visualizar na Figura 27 o diagrama de fluxo o qual estas funções obedecem. O fluxograma da figura em questão detalha o caminho para um movimento horizontal para a esquerda, porém os outros tipos de movimentação seguem o mesmo padrão, variando apenas o sentido de rotação dos motores.

Figura 27 – Fluxograma de acionamento e coleta de dados do algoritmo implementado no Arduino.



Fonte: Elaborado pelo Autor.

3.5.1 Leitura das posições horizontal e vertical pelos *encoders*

De acordo com as configurações de medição de distância pelos *encoders*, um movimento horizontal para a esquerda implica em um aumento no valor da posição horizontal em relação à origem. Da mesma maneira, um movimento vertical para baixo da carga implica em um aumento do valor da posição vertical em relação à origem. Os *encoders* estão conectados ao Arduino por meios dos pinos de interrupção, configurados para gerar uma interrupção em todas as bordas de subida de um de seus canais, como explicado nas Seções 2.5.1 e 3.4.1.

Desta maneira, conforme o eixo do motor gira, criando assim um trem de pulsos no canal de comunicação do *encoder*, será chamada uma função atrelada à interrupção que incrementa ou decrementa a variável de contagem de posição correspondente, todas as vezes em que houver uma borda de subida no canal monitorado. Como a contagem realizada é da quantidade de pulsos gerados pelos *encoders*, foi necessário obter uma relação entre o número de pulsos gerados e a distância percorrida a fim de obter as coordenadas de posição em centímetros do conjunto *trolley*/carga.

3.5.2 Implementação das funções de movimentação

De acordo com as condições satisfeitas demonstradas na Seção 3.4.4, a fim de realizar o movimento solicitado pelo usuário, que é de zero a 50cm horizontalmente, é chamada então a função *move_esquerda*(). Nesta função, primeiramente verifica-se qual foi a velocidade enviada pelo MATLAB ao Arduino. A partir daí, configura-se a tensão PWM que será aplicada a cada um dos dois motores responsáveis pela movimentação do conjunto *trolley*/carga para gerar a velocidade desejada, que no presente exemplo trata-se da velocidade lenta.

Em seguida, é iniciada a realização do movimento pertinente, através das funções *digitalWrite*() e *analogWrite*(), que definem qual será o sentido e a velocidade de rotação dos motores, respectivamente. Como já mencionado na Seção 2.4, mesmo se tratando de uma movimentação horizontal pura, há a necessidade de que ambos os motores sejam acionados para que a carga mantenha-se nivelada na posição vertical de referência. Dito isto, para cada uma das três velocidades foram definidos valores PWM para os motores responsáveis pelos movimentos horizontal e vertical tais que resultam em uma velocidade de deslocamento sincronizada em ambas as direções. Devido ao atrito presente durante a movimentação do *trolley*, atrito este que varia durante o percurso, é necessário realizar

ajustes extras na velocidade do motor responsável pelo movimento vertical para manter o nível da carga constante, já que a velocidade horizontal irá variar durante estes trechos com diferentes coeficientes de atrito.

Com o conjunto *trolley*/carga se movimentando em direção à posição desejada, o código verifica se a posição atual, tanto horizontal, quanto vertical, está se aproximando da posição desejada. Quando a diferença entre a posição atual e a desejada for menor do que dois centímetros, automaticamente as velocidades de movimentação são modificadas para a velocidade lenta. Este procedimento é realizado para que o posicionamento do conjunto *trolley*/carga seja feito com maior precisão, já que caso o conjunto esteja se movimentando em alta velocidade, tanto a inércia do sistema quando a velocidade limitada de tomadas de decisão por parte do *software* fazem com que o conjunto não se posicione corretamente na posição desejada.

Caso o usuário envie um sinal para interrupção de movimento, o mesmo deve ser tratado pelo Arduino para realizar a frenagem dos motores. Deste modo, enquanto é realizado o *loop* de movimentação, é também lido o canal *Serial* para verificar se houve algum comando de parada dado por parte do MATLAB. Caso o comando chegue, uma condição dentro do *loop* é ativada, chamando a função que interrompe a rotação dos motores e encerra a operação.

Da mesma forma, caso o usuário envie um comando para reiniciar o sistema de coordenadas, o mesmo será lido pela função *le_serial* no *loop* e será ativada então a condição que realiza o procedimento de zerar a contagem dos *encoders* e o sistema de coordenadas.

3.5.3 Implementação da função de atualização e envio dos dados de saída

Enquanto há a movimentação do conjunto *trolley*/carga até a posição desejada, o código coleta as informações dos sensores, realiza os cálculos necessários e envia para o MATLAB via *Serial* os dados referentes aos percursos horizontal e vertical, assim como a variação angular da carga ocorrida durante este percurso. Este procedimento é feito a cada iteração do *loop* de movimentação da carga.

Devido à incompatibilidade entre a função de escrita *Serial* do Arduino (*Serial.write*()) e o tipo dos dados a serem enviados, foi necessário realizar uma adaptação para que os dados pudessem ser enviados de forma correta. A incompatibilidade acontece devido aos dados que são armazenados e calculados no código serem do tipo *float*, que possuem tamanho de 4 *bytes*, enquanto a função *Serial.write*() envia somente dados binários pela porta *Serial*.

Sendo assim, foi necessário declarar estas variáveis com um "tipo"personalizado, onde fosse possível tanto obtê-las na forma de *float*, quanto na forma binária. Deste modo, pode-se trabalhar com a mesma variável em forma de *float* na hora de obtê-las e realizar cálculos, como por exemplo utilizando *posicao_h.floatingPoint*, e em forma binária para enviá-las via *Serial*, utilizando *posicao_h.binary*.

A obtenção do deslocamento do conjunto *trolley*/carga é realizado pelas funções já descritas na Seção 3.5.1, sendo necessário somente enviar o valor instantâneo armazenado nas variáveis de posição. Já para obter o valor do ângulo de oscilação da carga, medições e cálculos mais complexos são necessários. Para isto foi implementada a função *calcula_angulo*(), que realiza a comunicação I^2C com o CI MPU-6050 no módulo GY-521 para obter as medições do acelerômetro e do giroscópio, podendo então tratá-las e filtrá-las utilizando a implementação em C++ do filtro de Kalman, para uma obtenção do ângulo final com maior precisão. A implementação desta função encontra-se detalhada na Seção 3.6.

Neste algoritmo, as informações são enviadas para o MATLAB em "blocos"de três elementos a cada iteração, estando assim sincronizado com o modo de leitura por parte do MATLAB, como detalhado na Seção 3.3.4. Como cada variável é do tipo *float*, que ocupa 4 *bytes*, cada um destes bytes que forma a variável em seu formato binário é enviado sequencialmente, sendo lido pelo MATLAB e convertido de volta para o tipo *float* automaticamente. Caso haja algum erro de leitura dos sensores, o Arduino interrompe o movimento e envia um sinal de erro para posterior tratamento por parte do MATLAB.

3.6 Implementação do *software* de acionamento da planta e de aquisição de dados no Arduino - Parte 3: Obtenção do ângulo de oscilação da carga

Para obter o valor do ângulo de oscilação da carga, o Arduino comunica-se via I^2C com o CI MPU-6050 para obter de seus registradores os valores medidos pelos sensores acelerômetro e giroscópio. Estes valores são então tratados e filtrados para ser obtido o ângulo final de oscilação.

3.6.1 A comunicação entre o Arduino e o módulo GY-521

A comunicação I^2C *Inter-Integrated Circuit* foi criada com o intuito de facilitar a integração de circuitos de caráter final de aplicação, como por exemplo sensores e conversores, com um sistema de controle, como o Arduino, de modo que eles possam trabalhar com seus

sinais de maneira direta (BOXALL, 2010). A partir desta comunicação, o Arduino acessa os registradores do CI MPU-6050 para realizar a configuração dos sensores e obter os valores lidos pelos mesmos.

Dito isto, o primeiro passo realizado para a obtenção do ângulo de oscilação, após serem realizadas as configurações iniciais do sensor, é obter dos registradores os valores de aceleração e velocidade angular de cada um dos eixos do acelerômetro e do giroscópio. Esta leitura é realizada utilizando a biblioteca I^2C criada por *Kristian Lauszus* (LAUSZUS, 2012). Obtidos os valores dos registradores, calcula-se o valor da variação angular no eixo de oscilação da carga. No presente projeto, a oscilação da carga gera uma variação no eixo Z de medição do acelerômetro e do giroscópio. Deste modo o cálculo da rotação no eixo Z a partir das acelerações obtidas é realizado através da seguinte fórmula, com posterior conversão de radianos para graus:

$$Rotação_z = \arctan(\frac{A_x}{\sqrt{A_y^2 + A_z^2}}) \tag{3.1}$$

O terceiro passo é a obtenção do valor real da velocidade angular, já que quando são obtidas dos registradores elas estão em um valor não tratado, e precisam ser convertidas para graus/segundo através de uma divisão pelo fator apropriado fornecido pelo *datasheet* (INVENSENSE, 2013). Com o valor da rotação medida a partir da aceleração e das velocidades angulares nos eixos mencionados, pode-se então aplicar o filtro de Kalman para combinar as leituras dos sensores e assim obter a inclinação instantânea da carga durante a movimentação. A função que realiza este procedimento de filtragem foi derivada da opção proposta por (LAUSZUS, 2011), e implementada em C++ pela maior facilidade desta linguagem em trabalhar com *structs*.

3.6.2 O filtro de Kalman e sua implementação

O filtro de Kalman é um algoritmo de filtragem que utiliza uma série de medições observadas no tempo, onde neste caso são observadas as medições realizadas pelo acelerômetro e pelo giroscópio. Estas medições fatalmente apresentam ruídos, que contribuirão com o aparecimento de erros na leitura da medição. O filtro de Kalman irá então tentar estimar o estado do sistema baseando-se nos estados presente e anterior, buscando assim obter um resultado mais preciso do que unicamente os resultados obtidos através de leituras puras das variáveis desejadas. A teoria do filtro desenvolvido nesta seção foi baseada no

artigo publicado por *Welch* e *Bishop* (WELCH; BISHOP, 2006), com sua implementação em C++ realizada com o auxílio do artigo publicado por *Lauszus* (LAUSZUS, 2011).

Neste contexto, um problema muito comum apresentado no processo de medição de ângulo a partir do acelerômetro é que o valor da aceleração gravitacional lido pode ser muito ruidoso quando há variações rápidas de movimentação ou de sentido da oscilação durante esta medição (LAUSZUS, 2011). Da mesma maneira, o giroscópio apresenta o problema de deriva (*drift*) de medição, onde mesmo quando o sensor está praticamente parado, a leitura proveniente do giroscópio continua sendo incrementada ou decrementada gradativamente com o tempo, gerando um erro crescente de leitura de ângulo quando o giroscópio é utilizado isoladamente (LAUSZUS, 2011). Em suma, pode-se dizer que o giroscópio só é confiável a curto prazo, enquanto o acelerômetro só é confiável a longo prazo (LAUSZUS, 2011).

Uma das maneiras de tratar os problemas apresentados é utilizando o chamado Filtro Complementar, que consiste basicamente de um filtro digital passa-baixas aplicado às leituras do acelerômetro e um filtro digital passa-altas aplicado às leituras do giroscópio (LAUSZUS, 2011). Este filtro é mais simples de ser implementado, ao custo de uma menor precisão de leitura quando comparado ao filtro de Kalman. Devido à pequena amplitude de oscilação apresentada pela carga no presente trabalho, optou-se pela implementação do filtro de Kalman para obter uma maior precisão de leitura, possibilitando assim uma análise mais clara dos dados gerados.

O filtro de Kalman opera produzindo uma estimativa estatística ótima do estado do sistema, baseando-se nas sucessivas medições deste estado (AGUIRRE, 2007). Para fazer isto, é necessário conhecer o ruído presente na entrada do filtro, chamado de ruído de medição, além também do ruído do sistema propriamente dito, chamado de ruído de processo. Outro ponto que precisa ser atendido para uma aplicação eficiente deste filtro é que estes ruídos precisam apresentar-se com uma distribuição Gaussiana com média igual a zero, o que felizmente é a forma predominantemente apresentada pela grande maioria dos ruídos presentes nas medições provenientes do acelerômetro e do giroscópio (LAUSZUS, 2011).

3.6.2.1 O estado do sistema x_k

Antes de entrar em detalhes sobre a predição do estado do sistema, vale a pena detalhar algumas notações a serem utilizadas no decorrer da explicação para um melhor entendimento. Primeiramente, seja definido o chamado *estado anterior*, como segue:

$$\hat{\boldsymbol{x}}_{k-1|k-1}$$

O *estado anterior* é definido como sendo o estado anterior que foi estimado baseando-se no próprio estado anterior e nas estimativas dos estados precedentes a ele.

Em seguida, seja definido o chamado *estado a priori*:

$$\hat{\boldsymbol{x}}_{k|k-1}$$

O *estado a priori* é a estimativa da matriz de estados no tempo atual k, baseada no estado anterior do sistema e nas estimativas dos estados precedentes a ele.

Por ultimo, tem-se o chamado *estado a posteriori*:

$$\hat{\boldsymbol{x}}_{k|k}$$

O *estado a posteriori* é a estimativa do estado no tempo k, dadas as observações anteriores ao tempo k e, inclusive, no tempo k propriamente dito.

O problema é que o estado do sistema propriamente dito é "oculto" e pode ser observado somente através da observação z_k. Esta modelagem é conhecida como Modelo Oculto de Markov (ESPINDOLA, 2009). Isto significa que o estado será determinado a partir do estado no tempo k e todos os outros estados anteriores. O que também implica em dizer que a estimativa do estado não é confiável enquanto o filtro Kalman não tenha se estabilizado adequadamente.

O circunflexo acima de $\hat{x}$ indica que esta variável é uma estimativa do estado. Diferentemente de um x puro, que significa o estado real, ou seja, é a variável final a ser obtida pelo processo. Assim, a notação para o estado no tempo k é:

$$\boldsymbol{x}_k$$

Desse modo, o estado do sistema no tempo k é dado por (WELCH; BISHOP, 2006):

$$\boldsymbol{x}_k = \boldsymbol{F}x_{k-1} + \boldsymbol{B}u_k + w_k \tag{3.2}$$

Onde $\boldsymbol{x}_k$ é a matriz de estados, definida como:

$$\boldsymbol{x}_k = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_k \tag{3.3}$$

Nota-se que a saída do filtro será o ângulo θ, além também do desvio (*bias*) $\dot{\theta}_b$, determinados com base nas medições do acelerômetro e do giroscópio. O *bias* representa o quanto o giroscópio desviou da referência devido ao seu *drift* (ou deriva) inerente. Isto significa que para obter a variação angular correta, é necessário subtrair o *bias* do valor indicado pela medição do giroscópio.

Em seguida, tem-se a matriz $\boldsymbol{F}$, que é a matriz de transição de estados a ser aplicada ao estado anterior $\boldsymbol{x}_{k-1}$. Neste caso, $\boldsymbol{F}$ é definida como:

$$\boldsymbol{F} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \tag{3.4}$$

Onde Δt é o intervalo de tempo entre as leituras. A próxima grandeza a ser definida é a variável de entrada de controle u_k, que neste caso será a medição do giroscópio em graus por segundo (°/s) no tempo k, também chamado de taxa de variação angular (*rate*) $\dot{\theta}$. Pode-se então reescrever a equação de estado 3.2 como segue:

$$\boldsymbol{x}_k = \boldsymbol{F} x_{k-1} + \boldsymbol{B}\dot{\theta}_k + w_k \tag{3.5}$$

Onde a matriz $\boldsymbol{B}$ é denominada modelo de controle de entrada, sendo definida como:

$$\boldsymbol{B} = \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \tag{3.6}$$

Nota-se a partir das definições acima que pode-se obter o ângulo de variação medido pelo giroscópio ao multiplicar-se o *rate* $\dot{\theta}$ pelo intervalo de tempo Δt, e como não é possível calcular diretamente o *bias* a partir do *rate*, define-se 0 na segunda linha de $\boldsymbol{B}$.

$\boldsymbol{w}_k$ é o ruído de processo, que possui como característica ser uma distribuição Gaussiana com média zero, e com covariância Q no tempo k, como segue:

$$\boldsymbol{w}_k \sim N\left(0, \boldsymbol{Q}_k\right) \tag{3.7}$$

$\boldsymbol{Q}_k$ é a matriz de covariância do ruído de processo, que neste caso é a matriz de covariância das estimativas dos estados do acelerômetro e do *bias*. Desse modo, como as estimativas do *bias* e do acelerômetro são independentes entre si, $\boldsymbol{Q}_k$ contará somente com a variância das estimativas dos estados do acelerômetro e do *bias*, resultando na seguinte forma:

$$\boldsymbol{Q}_k = \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t \tag{3.8}$$

Nota-se que a matriz de covariância $\boldsymbol{Q}_k$ depende do tempo atual k. Logo, a variância do acelerômetro Q_θ e a variância do *bias* $Q_{\dot{\theta}_b}$ são multiplicadas pelo intervalo de tempo entre amostras Δt. Esta multiplicação mostra-se pertinente ao passo de que quanto maior o intervalo de tempo entre amostras, mais ruído de processo terá sido introduzido na leitura, como por exemplo o giroscópio introduzirá mais *drift* quanto maior for o tempo decorrente entre leituras. É necessário conhecer as variâncias mencionadas para o correto funcionamento do filtro, e tais valores podem ser encontrados no *datasheet* do sensor MPU-6050 (INVENSENSE, 2013).

Nota-se também que quanto maior for o valor da variância, mais ruído será introduzido na estimativa do estado. Então se por exemplo a estimativa do ângulo começar a apresentar o efeito de *drift*, é necessário aumentar o valor de $Q_{\dot{\theta}_b}$. Por outro lado, se a estimativa do estado estiver muito lenta, significa que o sistema está confiando excessivamente na estimativa do ângulo, sendo necessário diminuir o valor de Q_θ para tornar o sistema mais sensível às variações.

3.6.2.2 A medição z_k

Agora será analisada a medição ou observação z_k do estado real x_k. A medição z_k é dada por:

$$\boldsymbol{z}_k = \boldsymbol{H} x_k + v_k \tag{3.9}$$

Ou seja, a medição $\boldsymbol{z}_k$ resulta do estado atual x_k multiplicado pela matriz $\boldsymbol{H}$, somado com o ruído de medição v_k. $\boldsymbol{H}$ é chamada de modelo de observação e é usada para mapear o espaço de estados real no espaço observado. O estado real não pode ser observado. Como a medição provém apenas da medição do acelerômetro, $\boldsymbol{H}$ é dada por:

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{3.10}$$

O ruído de medição necessita apresentar distribuição Gaussiana com média zero, com covariância R:

$$\boldsymbol{v}_k \sim N\left(0, \boldsymbol{R}\right) \tag{3.11}$$

Porém, como neste caso R não é uma matriz, o ruído de medição será simplesmente igual à variância da medição, já que a covariância de uma mesma variável com ela própria é igual à própria variância. Desse modo, pode-se definir R como sendo:

$$\boldsymbol{R} = E\begin{bmatrix} v_k & {v_k}^T \end{bmatrix} = var(v_k) \tag{3.12}$$

Assumindo que o ruído de medição permanece o mesmo e não depende do tempo k:

$$var(v_k) = var(v) \tag{3.13}$$

Nota-se que caso a variância do ruído de medição $var(v)$ seja definido como um valor muito alto, o filtro responderá muito lentamente, já que estará confiando menos nas novas medições. Já se for definido como um valor muito baixo, poderão ocorrer variações muito bruscas já que desse modo o filtro estará confiando muito mais nas medições provenientes do acelerômetro.

Assim, para que o filtro seja eficiente deve-se, além de utilizar valores adequados para as variâncias de ruído de processo Q_θ e $Q_{\dot{\theta}_b}$, encontrar também um valor ótimo para a variância de ruído de medição $var(v)$.

Serão apresentadas a seguir as equações a serem utilizadas para estimar o estado real do sistema no tempo k ($\hat{x}_k$). Esta estimativa pode ser dividida em duas fases: Predição e

Atualização.

3.6.2.3 Etapa de Predição

As duas próximas equações a serem apresentadas têm o propósito de tentar prever o estado atual ($\hat{\boldsymbol{x}}_{k|k-1}$) e a matriz de covariância do erro no tempo k ($\boldsymbol{P}_{k|k-1}$). Primeiramente o filtro tentará estimar o estado atual baseando-se em todos os estados anteriores e na medição do giroscópio, como segue:

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{F}\hat{x}_{k-1|k-1} + \boldsymbol{B}\dot{\theta}_k \tag{3.14}$$

Por este motivo a medição do giroscópio é tratada como uma entrada de controle, já que é utilizada como uma entrada extra para a estimativa do estado no tempo atual k, também chamado de *estado a priori* $\hat{\boldsymbol{x}}_{k|k-1}$, como definido anteriormente.

O próximo passo é tentar estimar a matriz de covariância do erro *a priori* $\boldsymbol{P}_{k|k-1}$, definida a seguir, baseando-se na matriz de covariância do erro anterior $\boldsymbol{P}_{k-1|k-1}$:

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}^T + \boldsymbol{Q}_k \tag{3.15}$$

Esta matriz é utilizada para estimar o quanto pode-se confiar nos valores atuais do estado estimado. Quão menores forem os valores desta matriz, mais pode-se confiar no estado estimado atual. Uma análise da equação acima ajuda a compreender a afirmação anterior, já que a covariância do erro deverá crescer desde a última atualização das estimativas do estado. Logo, para obter a matriz *a priori* multiplica-se a matriz de covariância do erro anterior pela matriz de transição de estados $\boldsymbol{F}$ e sua transposta $\boldsymbol{F}^T$, somando também o ruído de processo atual Q_k no tempo k. No caso abordado neste trabalho, a matriz de covariância do erro $\boldsymbol{P}$ é uma matriz 2x2:

$$\boldsymbol{P} = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} \tag{3.16}$$

3.6.2.4 Etapa de Atualização

O primeiro passo a ser realizado na fase de atualização é calcular o vetor de inovação $\tilde{\boldsymbol{y}}_k$, que consiste na diferença entre a medição z_k e o *estado a priori* $x_{k|k-1}$:

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - \boldsymbol{H}\hat{x}_{k|k-1} \tag{3.17}$$

O modelo de observação $\boldsymbol{H}$ é utilizado para mapear o *estado a priori* $x_{k|k-1}$ dentro do estado observado, que é a medição proveniente do acelerômetro. Deste modo, a inovação não é uma matriz:

$$\tilde{\boldsymbol{y}}_k = \left[\tilde{\boldsymbol{y}}\right]_k \tag{3.18}$$

O próximo passo é calcular a denominada covariância da inovação:

$$\boldsymbol{S}_k = \boldsymbol{H}\boldsymbol{P}_{k|k-1}\boldsymbol{H}^T + \boldsymbol{R} \tag{3.19}$$

A covariância da inovação tenta prever o quanto deve-se confiar na medição baseada na matriz de covariância do erro *a priori* $\boldsymbol{P}_{k|k-1}$ e na variância de medição R. O modelo de observação $\boldsymbol{H}$ é utilizado para mapear a matriz de covariância do erro *a priori* $\boldsymbol{P}_{k|k-1}$ no espaço observado. Quão maior for o valor do erro de medição, maior será o valor de S, significando que o filtro não irá confiar tanto nas medições subsequentes. Neste caso S não é uma matriz, e pode ser reescrita como:

$$\boldsymbol{S}_k = \left[\boldsymbol{S}\right]_k \tag{3.20}$$

O próximo passo é calcular o ganho de Kalman. O ganho de Kalman é utilizado para indicar o quanto deve-se confiar na inovação, sendo definido como segue:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{H}^T\boldsymbol{S}_k^{-1} \tag{3.21}$$

Analisando a equação anterior nota-se que caso não se deva confiar muito na inovação, a covariância da inovação S será alta, e caso se deva confiar na estimativa do estado atual, a

matriz de covariância do erro $\boldsymbol{P}$ será pequena, resultando então em um valor pequeno para o ganho de Kalman. Da mesma maneira, tem-se um valor alto para o ganho de Kalman na situação onde se confia na inovação (S com valor baixo) porém não se confia na estimativa do estado atual ($\boldsymbol{P}$ com valores altos). Analisando mais a fundo, nota-se que é utilizada a matriz transposta do modelo de observação $\boldsymbol{H}$ para mapear a matriz de covariância do erro de estado $\boldsymbol{P}$ no espaço observado, comparando posteriormente $\boldsymbol{P}$ ao mutiplicá-la pela inversa da covariância da inovação S.

Conhecendo o estado inicial e obtendo o *bias* do giroscópio por calibração antes da aplicação do filtro, a inicialização da matriz de covariância do erro pode ser feita como segue:

$$\boldsymbol{P} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{3.22}$$

Caso as variáveis mencionadas não fossem conhecidas ao início do processo, a matriz $\boldsymbol{P}$ deveria conter números altos em sua diagonal principal. O ganho de Kalman será então uma matriz 2x1:

$$\boldsymbol{K} = \begin{bmatrix} K_0 \\ K_1 \end{bmatrix} \tag{3.23}$$

Pode-se então atualizar a estimativa *a posteriori* do estado atual. Isto é feito somando o *estado a priori* $\hat{\boldsymbol{x}}_{k|k}$ com o ganho de Kalman multiplicado pela inovação $\tilde{\boldsymbol{y}}_k$, como a seguir:

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \, \tilde{\boldsymbol{y}}_k \tag{3.24}$$

Vale ressaltar que a inovação $\tilde{\boldsymbol{y}}_k$ é a diferença entre a medição z_k e a estimativa do *estado a priori* $\hat{\boldsymbol{x}}_{k|k-1}$, podendo adquirir então tanto valores positivos quanto negativos. Esta equação define a estimativa final do estado atual, onde analisando-a pode-se observar que o estado final é simplesmente o *estado a priori* $\hat{\boldsymbol{x}}_{k|k-1}$, que foi calculado utilizando o estado anterior e a medição do giroscópio, corrigido pelo fator da combinação entre o ganho de Kalman e a inovação, que é calculado a partir das medições do acelerômetro.

O último passo do processo é atualizar a matriz de covariância do erro *a posteriori*:

$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})\boldsymbol{P}_{k|k-1} \tag{3.25}$$

Onde $\boldsymbol{I}$ é chamada matriz identidade, definida como:

$$\boldsymbol{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.26}$$

O que o filtro realiza neste passo é basicamente auto-corrigir a matriz de covariância do erro $\boldsymbol{P}_{k|k}$ baseando-se no quanto foi corrigida a estimativa. Isto faz sentido ao passo que o estado foi corrigido baseando-se na matriz de covariância do erro *a priori* $\boldsymbol{P}_{k|k-1}$, em conjunto com a covariância da inovação $\boldsymbol{S}_k$.

3.6.2.5 Implementando o filtro

Nesta seção as equações descritas anteriormente serão utilizadas para implementar o filtro Kalman em linguagem C++, possibilitando sua utilização para filtragem do sinal proveniente do sensor MPU-6050. As equações serão desenvolvidas e seus resultados finais simplificados serão implementados em C++, originando a base para o código de filtragem desenvolvido que está descrito no Apêndice D deste trabalho.

Passo 1: Estimativas dos estados *a priori* $\hat{\boldsymbol{x}}_{k|k-1}$:

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{F}\hat{x}_{k-1|k-1} + \boldsymbol{B}\dot{\theta}_k$$

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_k$$

$$= \begin{bmatrix} \theta - \dot{\theta}_b\Delta t \\ \dot{\theta}_b \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_k$$

$$= \begin{bmatrix} \theta - \dot{\theta}_b\Delta t + \dot{\theta}\Delta t \\ \dot{\theta}_b \end{bmatrix}$$

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} = \begin{bmatrix} \theta + \Delta t(\dot{\theta} - \dot{\theta}_b) \\ \dot{\theta}_b \end{bmatrix} \tag{3.27}$$

Nota-se que a estimativa *a priori* do angulo $\hat{\theta}_{k|k-1}$ é igual à estimativa do estado anterior $\hat{\theta}_{k-1|k-1}$ somado com a multiplicação de Δt com o *rate* sem *bias*. Como não é possível medir diretamente o *bias*, a estimativa do *bias a priori* é simplesmente igual ao anterior, como pode ser visto na segunda linha da matriz resultante.

O resultado da primeira linha da matriz pode ser escrito em C++ como segue:

```
1 rate = rate_novo - bias;
2 angulo = angulo + (dt*rate);
```

Passo 2: Estimativa da matriz de covariância do erro *a priori* $\boldsymbol{P}_{k|k-1}$:

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}^T + \boldsymbol{Q}_k$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

$$= \begin{bmatrix} P_{00} - \Delta t P_{10} & P_{01} - \Delta t P_{11} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

$$= \begin{bmatrix} P_{00} - \Delta t P_{10} - \Delta t(P_{01} - \Delta t P_{11}) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

$$= \begin{bmatrix} P_{00} - \Delta t P_{10} - \Delta t(P_{01} - \Delta t P_{11}) + Q_\theta \Delta t & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q_{\dot{\theta}_b} \Delta t \end{bmatrix}$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} = \begin{bmatrix} P_{00} + \Delta t(\Delta t P_{11} - P_{01} - P_{10} + Q_\theta) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q_{\dot{\theta}_b} \Delta t \end{bmatrix} \tag{3.28}$$

O resultado obtido acima pode ser escrito em C++ como:

```
1 P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angulo);
2 P[0][1] -= dt * P[1][1];
3 P[1][0] -= dt * P[1][1];
4 P[1][1] += Q_bias * dt;
```

Passo 3: Cálculo da inovação $\tilde{\boldsymbol{y}}_k$:

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - \boldsymbol{H}\hat{x}_{k|k-1}$$

$$= \boldsymbol{z}_k - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1}$$

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - \theta_{k|k-1} \tag{3.29}$$

Desse modo, o cálculo da inovação pode ser implementada em C++ como segue:

```
1 y = novo_angulo - angulo;
```

Passo 4: Cálculo da covariância da inovação S:

$$\boldsymbol{S}_k = \boldsymbol{H}\boldsymbol{P}_{k|k-1}\boldsymbol{H}^T + \boldsymbol{R}$$

$$= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \boldsymbol{R}$$

$$= P_{00_{k|k-1}} + \boldsymbol{R}$$

$$\boldsymbol{S}_k = P_{00_{k|k-1}} + var(v) \tag{3.30}$$

Da mesma maneira, a covariância da inovação S é implementada em C++ como segue:

```
1 S = P[0][0] + R_medicao;
```

Passo 5: Cálculo do ganho de Kalman $\boldsymbol{K}_k$:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{H}^T\boldsymbol{S}_k^{-1}$$

$$\begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \boldsymbol{S}_k^{-1}$$

$$= \begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1} \boldsymbol{S}_k^{-1}$$

$$\begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k = \frac{\begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1}}{\boldsymbol{S}_k} \tag{3.31}$$

Nota-se que em outros casos S pode ser uma matriz, inviabilizando a simples divisão de $\boldsymbol{P}$ por S, sendo necessário calcular a inversa de S. Porém como neste caso S não é uma matriz, a implementação em C++ do resultado obtido acima é simples, como segue:

```
1 K[0] = P[0][0] / S;
2 K[1] = P[1][0] / S;
```

Passo 6: Estimativa dos estados *a posteriori* $\hat{\boldsymbol{x}}_{k|k}$:

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \, \tilde{\boldsymbol{y}}_k$$

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k} = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} + \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k \tilde{\boldsymbol{y}}_k$$

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k} = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} + \begin{bmatrix} K_0 \, \tilde{\boldsymbol{y}} \\ K_1 \, \tilde{\boldsymbol{y}} \end{bmatrix}_k \tag{3.32}$$

Mais uma vez nota-se que o resultado obtido é bem direto, sendo sua implementação em C++ simples de ser realizada:

```
1 angulo = angulo + K[0] * y;
2 bias = bias + K[1] * y;
```

Passo 7: Atualização da matriz de covariância do erro *a posteriori* $\boldsymbol{P}_{k|k}$:

$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}) \boldsymbol{P}_{k|k-1}$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k} = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k \begin{bmatrix} 1 & 0 \end{bmatrix} \right) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1}$$

$$= \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_0 & 0 \\ K_1 & 0 \end{bmatrix}_k \right) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1}$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k} = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} - \begin{bmatrix} K_0\,P_{00} & K_0\,P_{01} \\ K_1\,P_{00} & K_1\,P_{01} \end{bmatrix} \tag{3.33}$$

A implementação em C++ da equação simplificada obtida resulta no seguinte trecho:

```
1 float P00_temp = P[0][0];
2 float P01_temp = P[0][1];
3
4 P[0][0] -= K[0] * P00_temp;
5 P[0][1] -= K[0] * P01_temp;
6 P[1][0] -= K[1] * P00_temp;
7 P[1][1] -= K[1] * P01_temp;
```

Finalmente, pesquisando no *datasheet* do sensor MPU-6050 e ajustando heurísticamente os valores das variâncias de ruído de processo Q_θ e $Q_{\dot{\theta}_b}$, assim como para a variância de ruído de medição $var(v)$, obteve-se os seguintes valores a serem utilizados pelo filtro de Kalman, respectivamente:

```
1 float Q_angulo = 0.001;     //variancia de ruido de processo para o angulo
2 float Q_bias = 0.003; //variancia de ruido de processo para o giroscopio
3 float R_medicao = 0.03;    //variancia de ruido de medicao
```

Assim, com o filtro implementado foi possível obter valores mais confiáveis para a medição da oscilação da carga durante o movimento na planta. O código completo da implementação do filtro desenvolvido nesta seção pode ser visto no Apêndice D.

4 ANÁLISE DOS RESULTADOS OBTIDOS

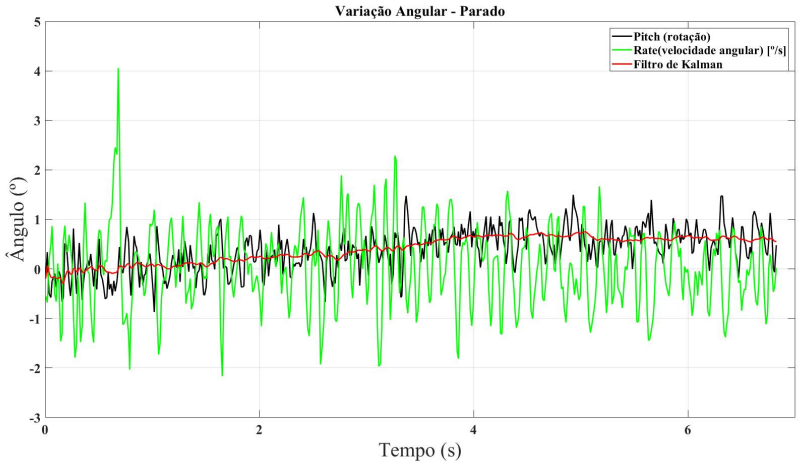
Após o desenvolvimento dos algoritmos de controle e monitoramento, foram realizados testes para validar o sistema implementado. Neste capítulo serão apresentados os resultados obtidos a partir dos dados provenientes dos *encoders* e do módulo acelerômetro/giroscópio, que representam a movimentação horizontal e vertical da carga ao longo da planta, assim como sua oscilação em relação ao eixo vertical ao longo deste processo. Porém, antes de realizar os testes com a planta propriamente dita, foi necessário validar o sistema de filtragem de ângulo, implementado pelos motivos citados na Seção 3.6.2.

4.1 Validação do filtro de Kalman

Visando validar o sistema de filtragem de ângulo para que a leitura da oscilação da carga fosse feito com eficiência, foram realizados testes variando a inclinação do sensor e aplicando diferentes tipos de perturbação para analisar seu comportamento. Como dito anteriormente, as leituras provenientes do acelerômetro apresentam ruídos e podem variar de maneira excessiva na presença de certos tipos de perturbação, como por exemplo uma variação brusca de inclinação, já que o sensor faz a leitura da aceleração que é aplicada sobre o mesmo. Da mesma maneira, rápidas variações implicam em uma velocidade angular alta, não podendo então ser tomada como base somente a leitura proveniente do giroscópio, além do mesmo apresentar uma deriva inerente com o passar do tempo.

Desta forma, o primeiro teste feito foi com o sensor imóvel, visando analisar somente a variação inerente dos sensores. A figura a seguir ilustra o resultado obtido.

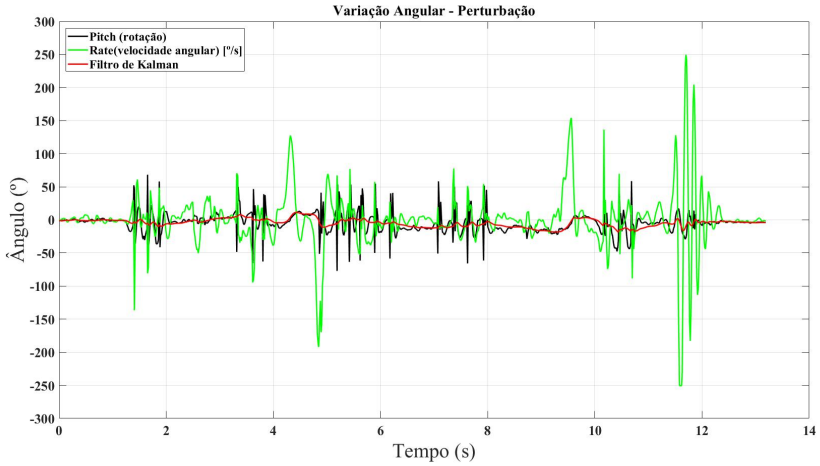
Figura 28 – Variação angular com o módulo MPU-6050 imóvel.



Fonte: Elaborado pelo Autor.

Como pode-se observar na Figura 28, mesmo sem haver uma mudança na inclinação do sensor, nota-se a existência de rápidas variações nos sinais apresentados pelo acelerômetro (*pitch*), em preto, e pelo giroscópio (*rate*), em verde. Enquanto isso, o sinal obtido pelo filtro de Kalman mostra-se consideravelmente mais estável, representando mais fielmente a inclinação apresentada pelo módulo GY-521 ao longo do tempo. Desse modo, foram aplicados dois tipos de perturbação no sistema para analisar sua reação. O primeiro deles foram variações bruscas de aceleração, que podem ocorrer através de colisões com o objeto onde o sensor está aplicado. Os resultados obtidos estão presentes na figura a seguir.

Figura 29 – Aplicação de perturbação durante a leitura do ângulo filtrado.

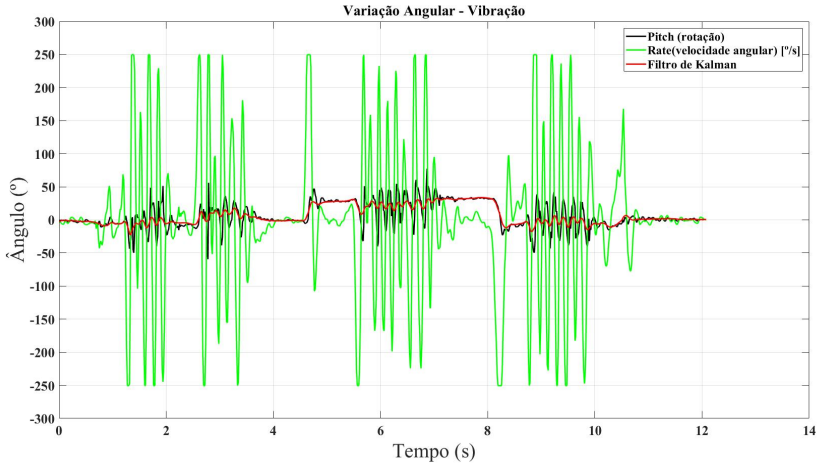


Fonte: Elaborado pelo Autor.

Nota-se que durante as colisões aplicadas, as leituras provenientes do acelerômetro e do giroscópio sofreram rápidas variações, formando diversos picos em suas leituras, o que acarretaria em uma informação errônea de ângulo obtida naquele instante. Apesar disso, o sinal apresentado pelo filtro de Kalman mostrou-se bastante estável. Mesmo durante situações extremas, como na variação apresentada entre 11 e 12 segundos onde a velocidade angular chega ao seu pico de 250 graus/s, que é o valor limite configurado para o giroscópio no código implementado no Arduino, o filtro de Kalman foi capaz de manter uma leitura fiel da real inclinação do sensor.

A segunda perturbação analisada foi a presença de vibrações durante o procedimento de medição. A figura a seguir ilustra os resultados obtidos neste processo.

Figura 30 – Aplicação de vibração durante a leitura do ângulo filtrado.

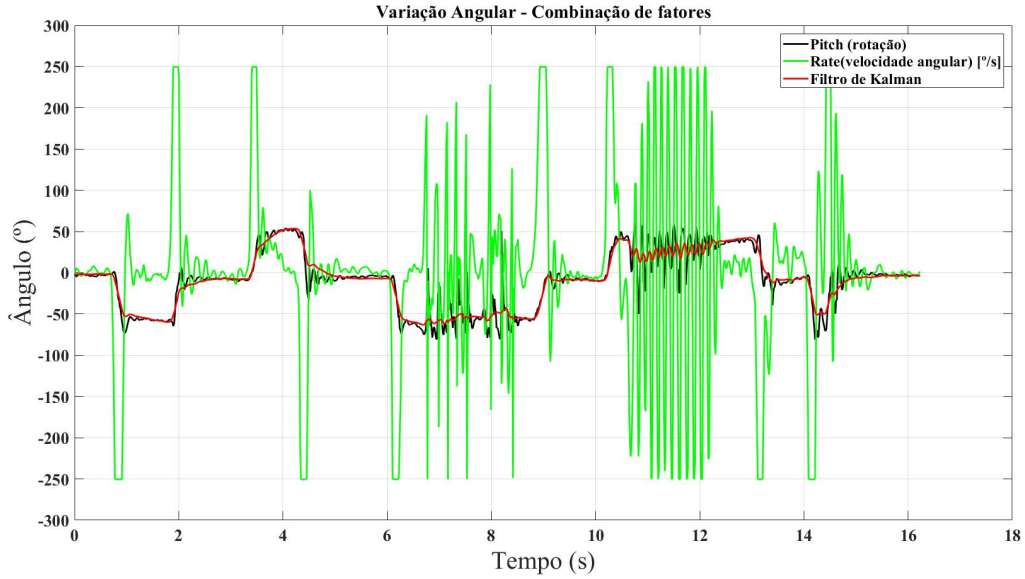


Fonte: Elaborado pelo Autor.

Analisando a figura nota-se que com a presença de fortes vibrações o sinal proveniente do giroscópio fica ainda mais comprometido, gerando picos em diversos momentos de sua leitura. Da mesma forma, a rotação medida pelo acelerômetro apresenta variações bruscas, chegando a 100 graus pico a pico, quando na verdade as vibrações estavam provocando variações muito menores na inclinação do objeto. Mais uma vez nota-se a robustez do filtro aplicado, capaz de manter a oscilação medida consideravelmente mais próxima da realidade.

Por fim, foi realizada uma medição em um ambiente onde, além da mudança de inclinação do sensor, há também a presença dos diferentes tipos de perturbação apresentados anteriormente. O resultado obtido está disponível na figura a seguir.

Figura 31 – Leitura da inclinação do sensor com presença de perturbações.



Fonte: Elaborado pelo Autor.

Com este último teste verifica-se claramente a confiabilidade do sinal filtrado, já que o mesmo é capaz de acompanhar a rotação medida pelo acelerômetro com rapidez, além de não ser influenciado pelas bruscas variações apresentadas pelo mesmo. Desta forma, confirma-se a eficiência do filtro implementado, onde a curto prazo confia-se na medição de ângulo proveniente do giroscópio, enquanto a longo prazo confia-se na rotação calculada a partir do acelerômetro. Com a implementação do filtro apresentando um resultado satisfatório, pôde-se então realizar os testes com a planta, sabendo-se que os resultados da medição da oscilação da carga possuirão alto grau de confiabilidade.

4.2 Análise das respostas ao comando de movimentação

Visando analisar o comportamento da planta mediante os comandos dados pelo usuário para a movimentação do conjunto *trolley*/carga, foram coletados dados das trajetórias verticais e horizontais do conjunto, além da oscilação da carga em relação ao eixo vertical durante o processo. Deste modo, foram realizados testes para os diferentes tipos de comandos possíveis de serem realizados através da interface gráfica.

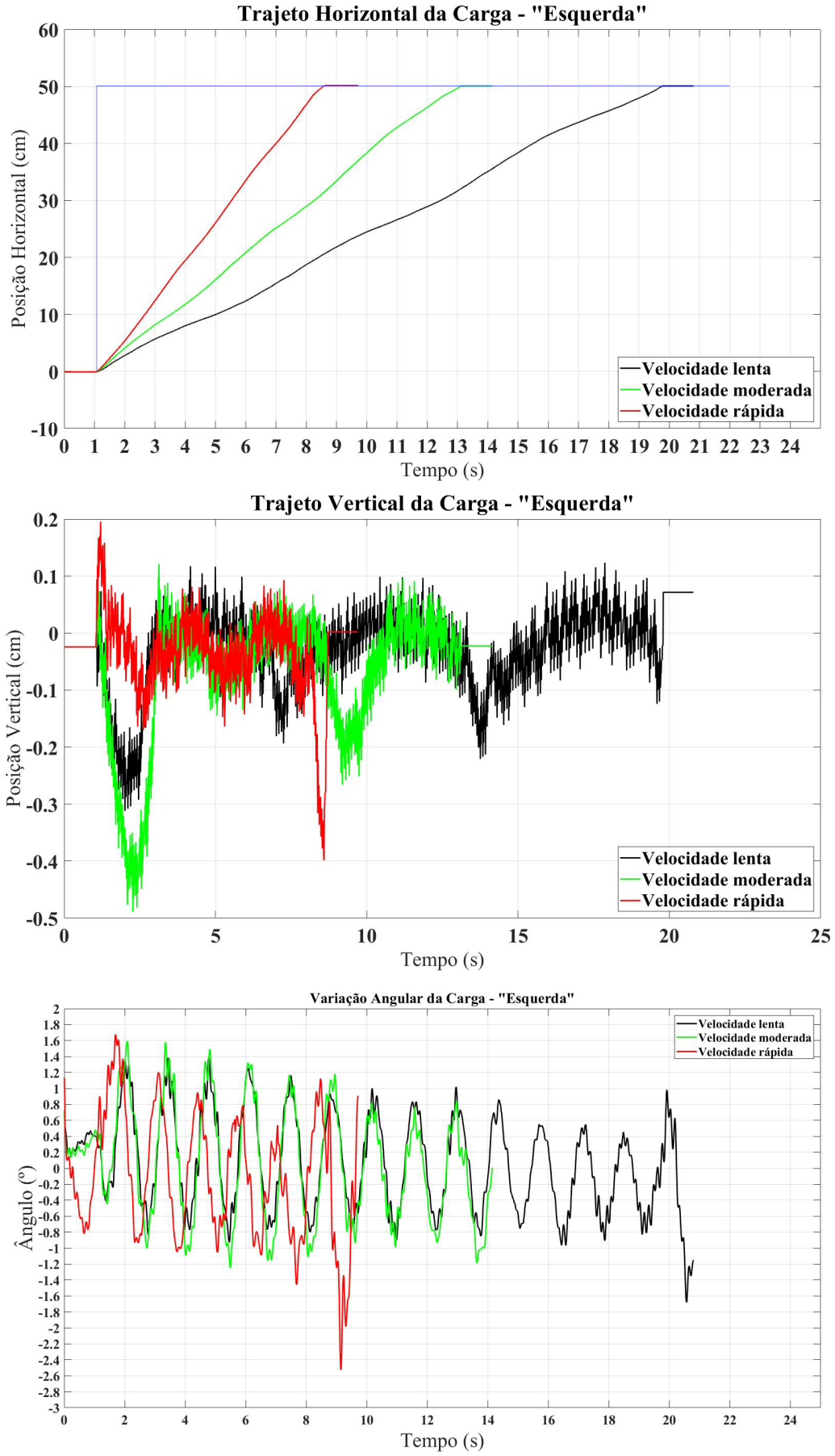
4.2.1 Resposta ao comando de movimentação para a esquerda

O primeiro teste feito foi para um deslocamento horizontal do conjunto, desde a origem do sistema de coordenadas até a posição 50 cm. Como mencionado anteriormente, o plano de coordenadas da planta é tal que os movimentos para a esquerda e para baixo representam aumentos nos valores das posições horizontal e vertical, respectivamente. O degrau foi aplicado no tempo igual a 1.1 segundos e foram coletados os dados referentes aos sensores. A resposta da planta para os três modos de velocidade pré-definidas pode ser visualizado a seguir.

Tabela 1 – Características das respostas ao comando de movimentação para a esquerda

Velocidade	**Tempo de subida (s)**	**Sobressinal (%)**	**Erro estacionário (cm)**	**Deslocamento vertical máximo (cm)**	**Oscilação máxima (º)**
Lenta	18.71	0.04	0.02	-0.31	1.7
Moderada	12.05	0.04	0.02	-0.49	1.6
Rápida	7.52	0.2	0.1	-0.40	2.5

Figura 32 – Resposta ao degrau para movimentação para a esquerda.



Fonte: Elaborado pelo Autor.

Analisando a Figura 32 no gráfico intitulado "Trajeto Horizontal da Carga - Esquerda", nota-se uma maior oscilação apresentada no trajeto horizontal percorrido desde a origem até 50 cm quando o comando é dado para baixas velocidades, apesar da tensão aplicada nos motores ser constante. Esta oscilação se dá pela maior influência do atrito presente no sistema para baixas velocidades, tanto no contato entre o *trolley* e os trilhos, quanto entre as engrenagens e a correia que puxa o *trolley*.

Já para a velocidade rápida, analisando a figura em conjunto com a Tabela 1, nota-se que apesar da movimentação não sofrer tanta influência do atrito e conseguir atingir o *setpoint* em menor tempo, há uma maior probabilidade de erro estacionário devido à inércia do conjunto em movimento. Para amenizar este erro, foi adotada uma estratégia de controle em que quando o erro (diferença entre o *setpoint* e a posição atual) for menor do que dois centímetros, os motores são automaticamente configurados para diminuir a velocidade e assim chegar ao ponto desejado de maneira mais suave. Nota-se que mesmo assim há um pequeno erro estacionário ao fim do percurso devido ao modo que o sistema de controle foi implementado, onde não há uma mudança na rotação do motor para ajustar estes pequenos erros, já que devido às grandes zonas mortas e inércias presentes neste sistema, uma mudança de rotação para tentar corrigir este erro geraria um erro equivalente na direção oposta, fazendo com que o sistema entre em uma oscilação constante.

Analisando o gráfico intitulado "Trajeto Vertical da Carga - Esquerda"na Figura 32, percebe-se uma oscilação em torno do ponto zero, resultado da estratégia de controle adotada para tentar manter a carga nivelada verticalmente enquanto o conjunto se move horizontalmente, já que como mencionado anteriormente, o motor responsável pelo movimento vertical não está no *trolley*, e sim preso na estrutura fixa da planta, sendo necessário então um acionamento deste motor mesmo quando se deseja somente um movimento horizontal, como no presente caso. Nota-se que apesar da rápida oscilação, o sistema de controle nivela a carga de maneira eficiente, já que os picos de diferença de nível apresentados na Tabela 1 se dão nos momentos de partida e de chegada ao ponto desejado, que são os momentos de máximas aceleração e desaceleração do conjunto, respectivamente.

Através do gráfico de variação angular da carga, nota-se novamente que os picos de oscilação se dão nos momentos de maior aceleração e desaceleração do sistema, onde quanto maior a velocidade de cruzeiro, e por consequência maior aceleração e desaceleração, maiores são as amplitudes de oscilação da carga nestes momentos. Apesar disso, as oscilações nestes períodos não são tão superiores às oscilações presentes durante o movimento do conjunto com velocidade aproximadamente constante. Estas oscilações ocorrentes mesmo durante o trajeto com velocidade constante se dão principalmente pelos atritos presentes durante o percurso em conjunto fato do sistema de cabeamento para a sustentação da carga consistir

de um material fino e elástico, como o *nylon*, que acaba deixando o conjunto susceptível à oscilações, por menores que sejam as forças atuantes. Da mesma forma, outro fator que pode influenciar na oscilação da carga é o fato de não haver um ponto fixo de suporte entre o *trolley* e a carga, já que os cabos de sustentação estão afixados na estrutura da planta.

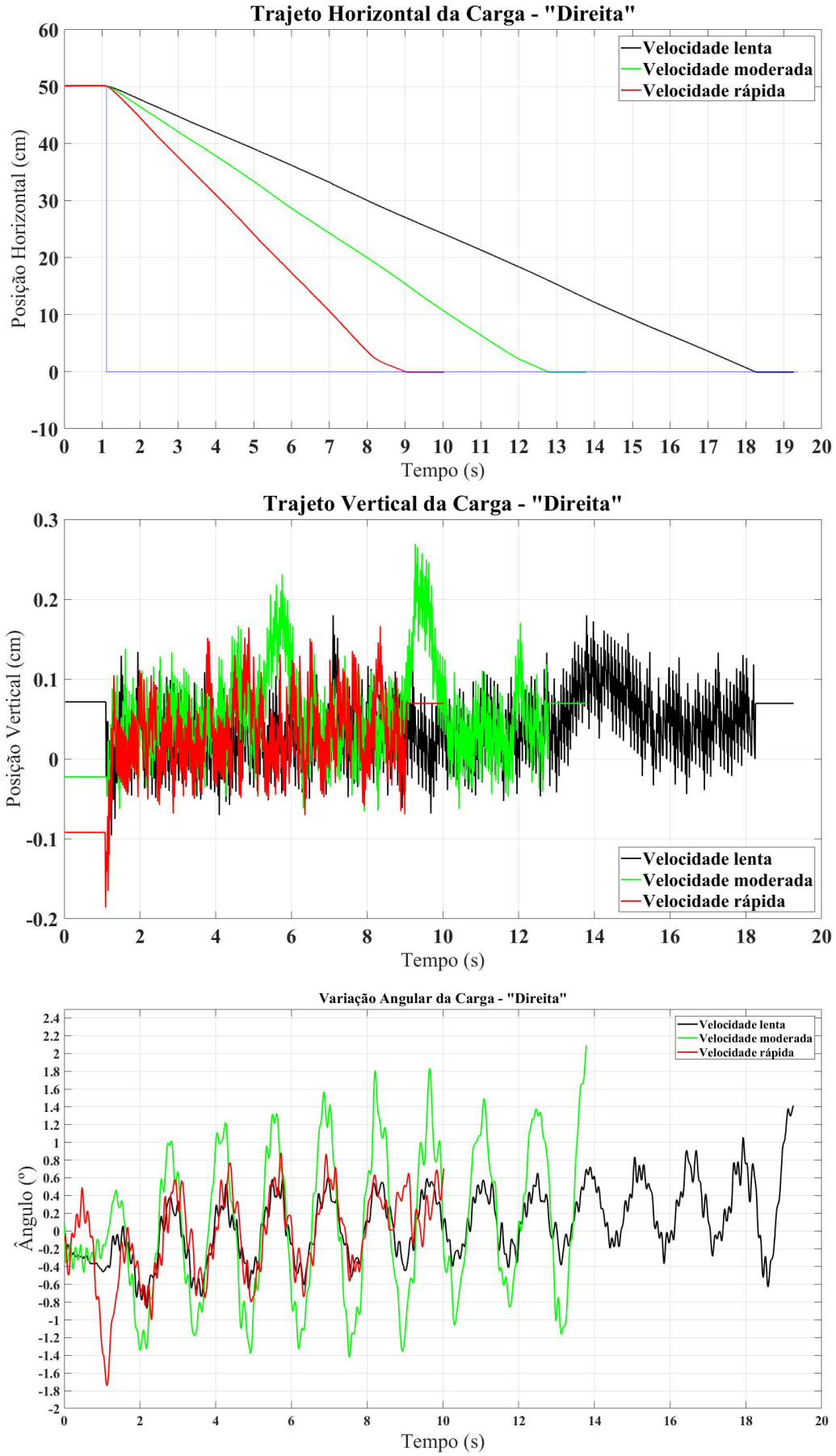
4.2.2 Resposta ao comando de movimentação para a direita

Da mesma forma, foram realizados testes para um comando de movimento horizontal para a direita, visando analisar as possíveis diferenças entre este e o processo de movimentação para a esquerda. Resultados diferentes eram esperados devido à já notada diferença no acionamento dos motores entre uma direção e outra durante a implementação do código, já que tensões PWM menores eram necessárias para que os motores girassem para a direita com velocidades parecidas às que apresentavam no comando para a esquerda, com pode ser visualizado no código do Apêndice B. Deste modo, já suspeitava-se de diferenças no atrito entre um sentido de rotação e o outro. Tais suspeitas foram comprovadas com os resultados obtidos na figura a seguir.

Tabela 2 – Características das respostas ao comando de movimentação para a direita

Velocidade	**Tempo de subida (s)**	**Sobressinal (%)**	**Erro estacionário (cm)**	**Deslocamento vertical máximo (cm)**	**Oscilação máxima (º)**
Lenta	17.13	0.12	0.06	0.18	1.4
Moderada	11.71	0.12	0.06	0.26	2.1
Rápida	7.89	0.2	0.1	0.16	1.7

Figura 33 – Resposta ao degrau para movimentação para a direita.



Fonte: Elaborado pelo Autor.

Como pode-se visualizar no trajeto horizontal percorrido para a direita, mesmo para as velocidades lenta e moderada, as respostas ao degrau mantiveram-se com inclinação aproximadamente constante durante todo o processo, não apresentando as ondulações vistas na movimentação para a esquerda. Isto comprova que os atritos presentes no sistema para este sentido não são tão intensos como os observados para o sentido oposto, resultando então em um percurso mais suavizado, com menores oscilações verticais e menores picos de oscilação da carga. Devido a esta menor amplitude de oscilação, nota-se oscilações de maior frequência inseridas na oscilação predominante, sendo resultantes da vibração do conjunto *trolley*/carga durante seu deslocamento.

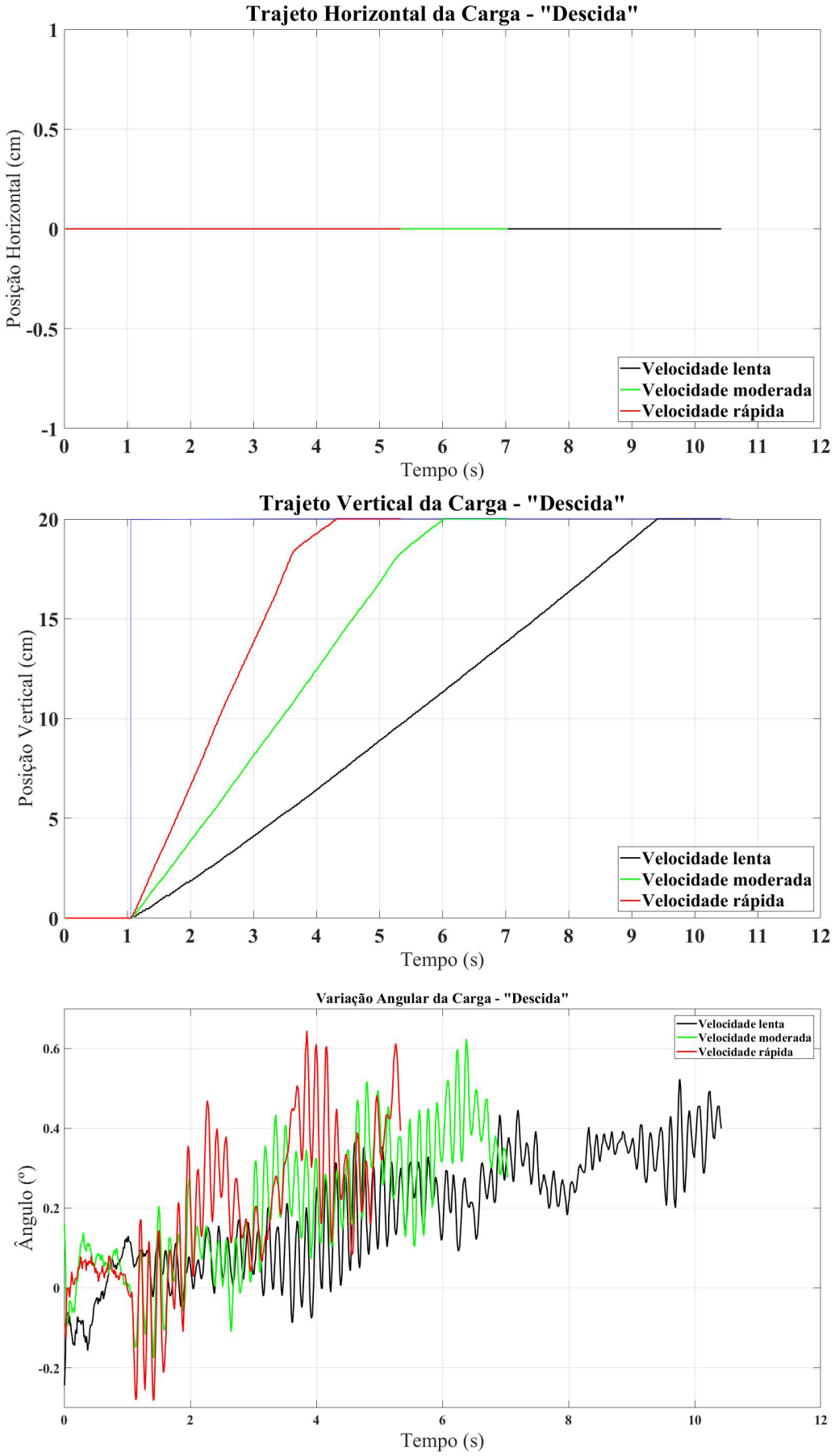
Outro fator que contribui para o resultado mais favorável apresentado neste tipo de movimento é o fato do sistema de roldanas que sustenta a cara estar acoplado ao *trolley* em seu lado esquerdo, o que implica dizer que quando o conjunto move-se para a esquerda, o acionamento horizontal do *trolley* precisa também arcar com o peso da carga, e por consequência com uma força contrária maior em relação a quando o movimento é para a direita. Já quando se movimenta para a direita, o motor vertical é quem sofre uma maior força de oposição ao movimento e acaba contribuindo também para a movimentação horizontal do *trolley* para a direita.

A tabela 2 ilustra o que foi dito, mostrando que os tempos de trajeto foram similares aos da movimentação vertical, porém ocorrendo menores erros estacionários e picos de oscilação, tanto vertical quanto de ângulo da carga. Nota-se uma oscilação maior para o movimento com velocidade moderada, o que não era esperado, já que quanto maior for a velocidade, maior é a aceleração necessária inicialmente, o que implicaria em uma maior oscilação da carga ao longo do trajeto. Esta diferença pode ter sido causada pela maior dificuldade apresentada pelo sistema de controle para manter a carga verticalmente nivelada nesta velocidade durante o percurso, como pode ser visualizado no gráfico intitulado "Trajeto Vertical da Carga - Direita"na Figura 33.

4.2.3 Resposta ao comando de movimentação para baixo

Visando analisar o comportamento da planta também para as movimentações de subida e descida da carga, foram feitos testes de posicionamento vertical para um percurso de 20 cm, nas três velocidades pré-definidas. A seguir tem-se a resposta obtida para um comando de descida de carga.

Figura 34 – Resposta ao degrau para movimentação para a baixo.



Fonte: Elaborado pelo Autor.

Tabela 3 – Características das respostas ao comando de movimentação para baixo

Velocidade	Tempo de subida (s)	Sobressinal (%)	Erro estacionário (cm)	Deslocamento horizontal máximo (cm)	Oscilação máxima (º)
Lenta	8.33	0.05	0.01	0	0.5
Moderada	4.88	0.05	0.01	0	0.61
Rápida	3.24	0.1	0.02	0	0.65

A partir dos gráficos e da Tabela 3, nota-se que o movimento vertical possui muito menos perturbações em relação ao horizontal das seções anteriores, possibilitando assim resultados mais precisos de posicionamento e menor oscilação da carga durante o processo. Verifica-se também com mais clareza a diminuição de velocidade de movimentação da carga nos movimentos com velocidade rápida e moderada ao chegar perto do *setpoint*, para garantir um posicionamento mais preciso da carga. Desta forma nota-se que mesmo utilizando a velocidade rápida, o erro final de posicionamento se assemelha aos erros utilizando menores velocidades.

Além disso, observa-se que os picos de oscilação da carga se dão justamente neste momento de mudança de velocidade de descida. Porém, mesmo ocasionando este aumento na oscilação, a amplitude da mesma é consideravelmente menor do que a oscilação durante os movimentos horizontais, e se dão majoritariamente pelas vibrações presentes no processo de descida, que fazem a carga oscilar devido à alta sensibilidade dos cabos de sustentação, que como já mencionado, são constituídos de material flexível.

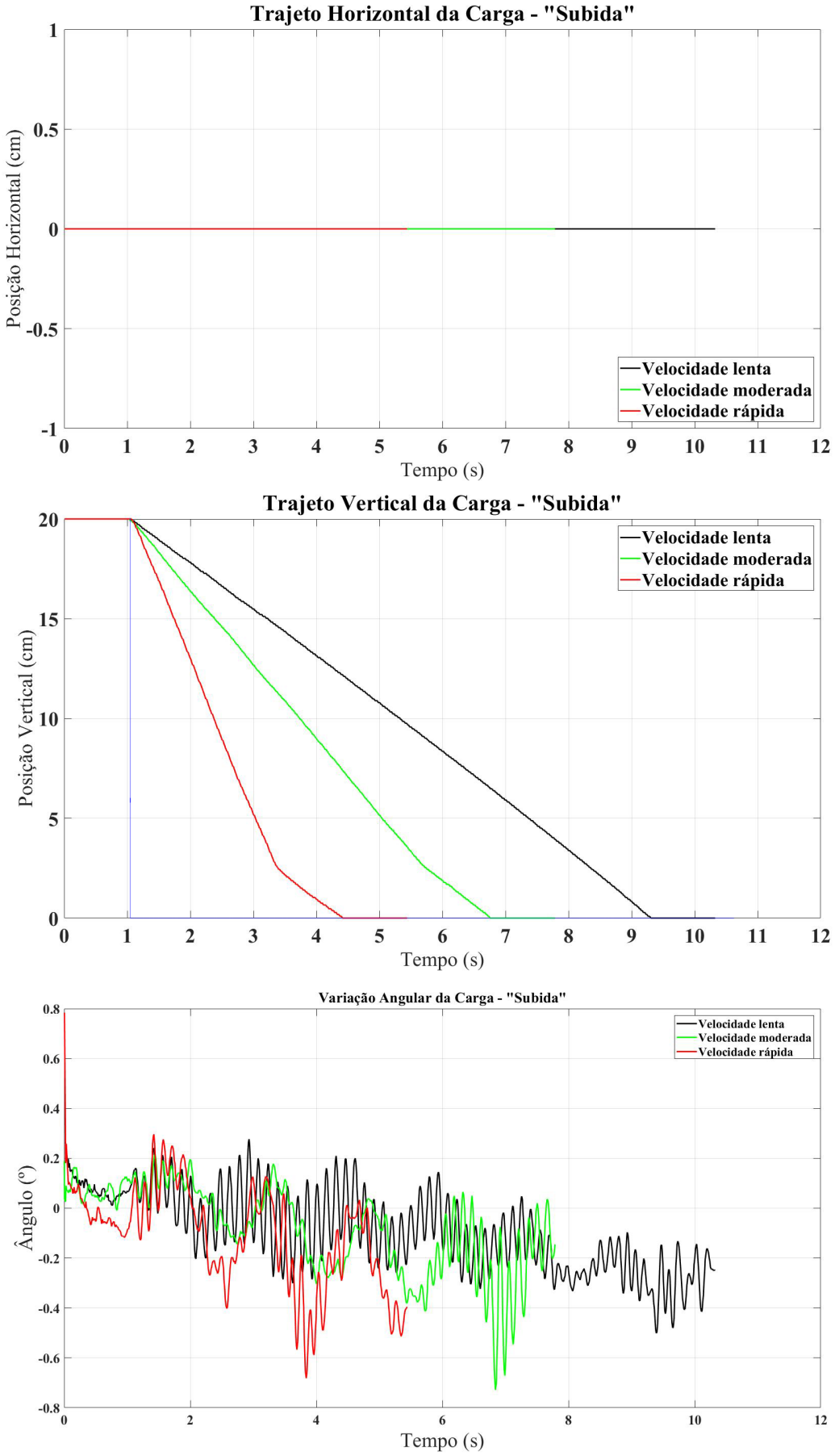
4.2.4 Resposta ao comando de movimentação para cima

Da mesma maneira, foram feitos testes de movimentação vertical da carga, desta vez com um movimento de subida para as três velocidades pré-definidas. Os resultados obtidos podem ser visualizados a seguir.

Tabela 4 – Características das respostas ao comando de movimentação para cima

Velocidade	Tempo de subida (s)	Sobressinal (%)	Erro estacionário (cm)	Deslocamento horizontal máximo (cm)	Oscilação máxima (º)
Lenta	8.1	0.15	0.03	0	0.5
Moderada	5.72	0.2	0.04	0	0.7
Rápida	3.32	0.45	0.09	0	0.65

Figura 35 – Resposta ao degrau para movimentação para a cima.



Fonte: Elaborado pelo Autor.

Analisando a Figura 35 e a Tabela 4 nota-se um resultado similar à da movimentação de descida, como esperado, já que as únicas perturbações presentes neste tipo de movimentação são as vibrações causadas pelo acionamento do motor e pela rotação das roldanas que auxiliam na movimentação vertical. Todas as respostas apresentaram um pequeno sobressinal que resultou em um erro estacionário remanescente, sendo estes cada vez maiores de acordo com o aumento da velocidade utilizada no trajeto. Apesar disso, as respostas apresentaram ângulos de oscilação baixos, causados pelas vibrações mencionadas.

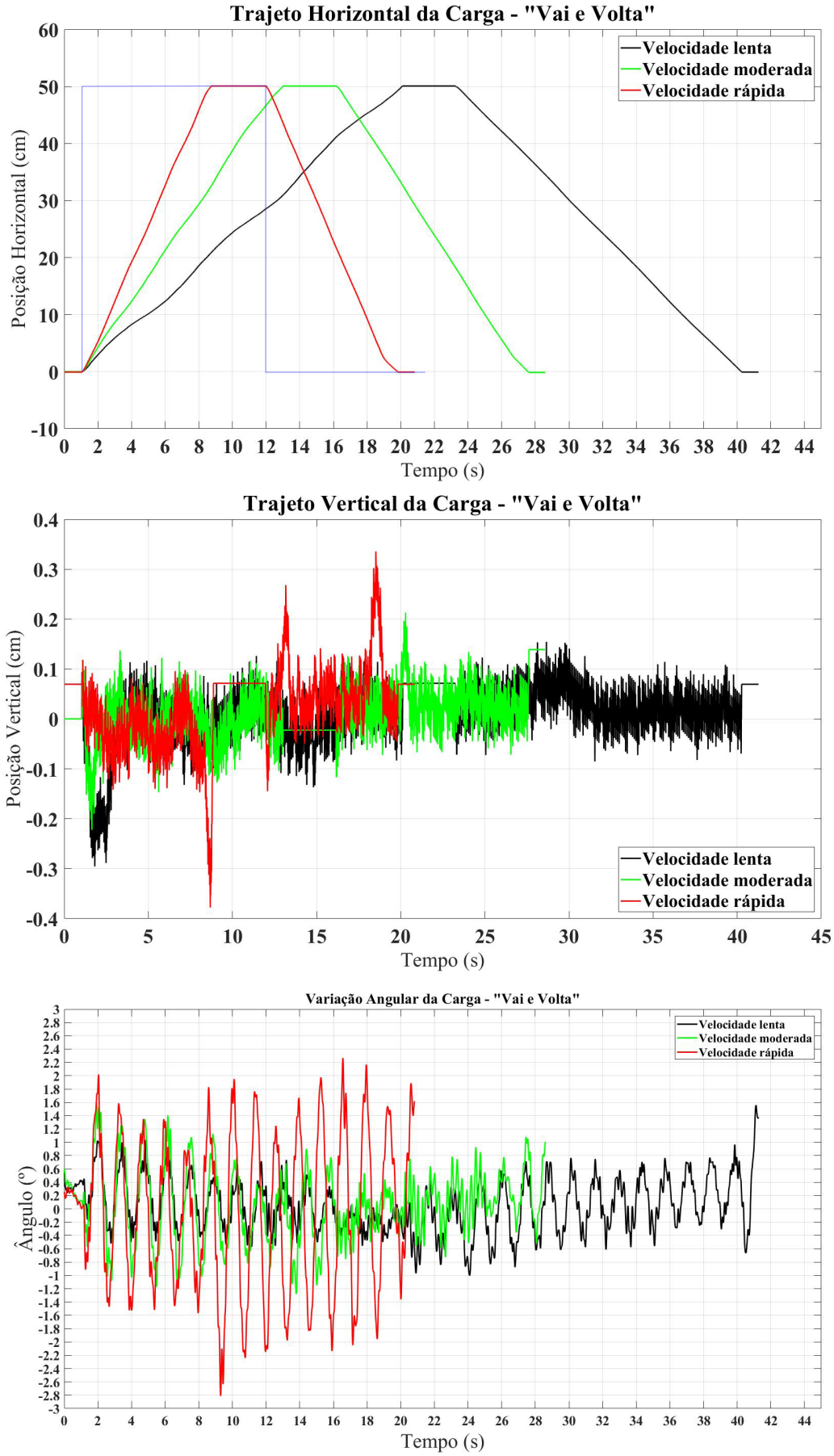
4.2.5 Resposta ao comando de movimentação Vai e Volta

Visando analisar desta vez o comportamento do sistema em uma movimentação em que o conjunto se move na direção horizontal para ambos os sentidos, foram coletados os dados referentes ao movimento pré-definido nomeado "Vai e Volta". Neste movimento, puramente horizontal, o conjunto *trolley*/carga move-se até a posição de 50 cm, faz uma breve pausa, retornando à origem em seguida. Foram realizados testes para as três velocidades pré-definidas, com os resultados apresentados nos gráficos e na tabela a seguir.

Tabela 5 – Características das respostas ao comando de movimentação "Vai e Volta"

Velocidade	**Tempo de ida (s)**	**Tempo de volta (s)**	**Sobressinal (%)**	**Erro estacionário (cm)**	**Deslocamento horizontal máximo (cm)**	**Oscilação máxima (º)**
Lenta	18.9	17.35	0.16	-0.08 (volta)	0.29	1.58
Moderada	11.78	11.4	0.14	-0.07 (volta)	0.22	1.4
Rápida	7.83	7.74	0.1	-0.05 (volta)	0.38	2.8

Figura 36 – Resposta para a movimentação pré-definida "Vai e Volta".



Fonte: Elaborado pelo Autor.

Neste tipo de movimento fica evidente a diferença entre os processos de movimentação para a esquerda e para a direita, sendo notável a presença de ondulações no trajeto horizontal durante a movimentação para a esquerda utilizando baixas velocidades. Da mesma maneira, devido à essas ondulações, desnivelamentos verticais ocorrem neste trecho do movimento, já que o sistema de controle vertical encontra maiores dificuldades para manter a carga em uma posição vertical fixa. Nota-se também que os picos de desnivelamento ocorrem novamente nos trechos iniciais e finais do movimento, onde há maiores aceleração e desaceleração horizontais, respectivamente. Apesar disso, os maiores valores de desnivelamento ocorrem durante um período muito curto de tempo, sendo rapidamente compensados pelo sistema de controle vertical.

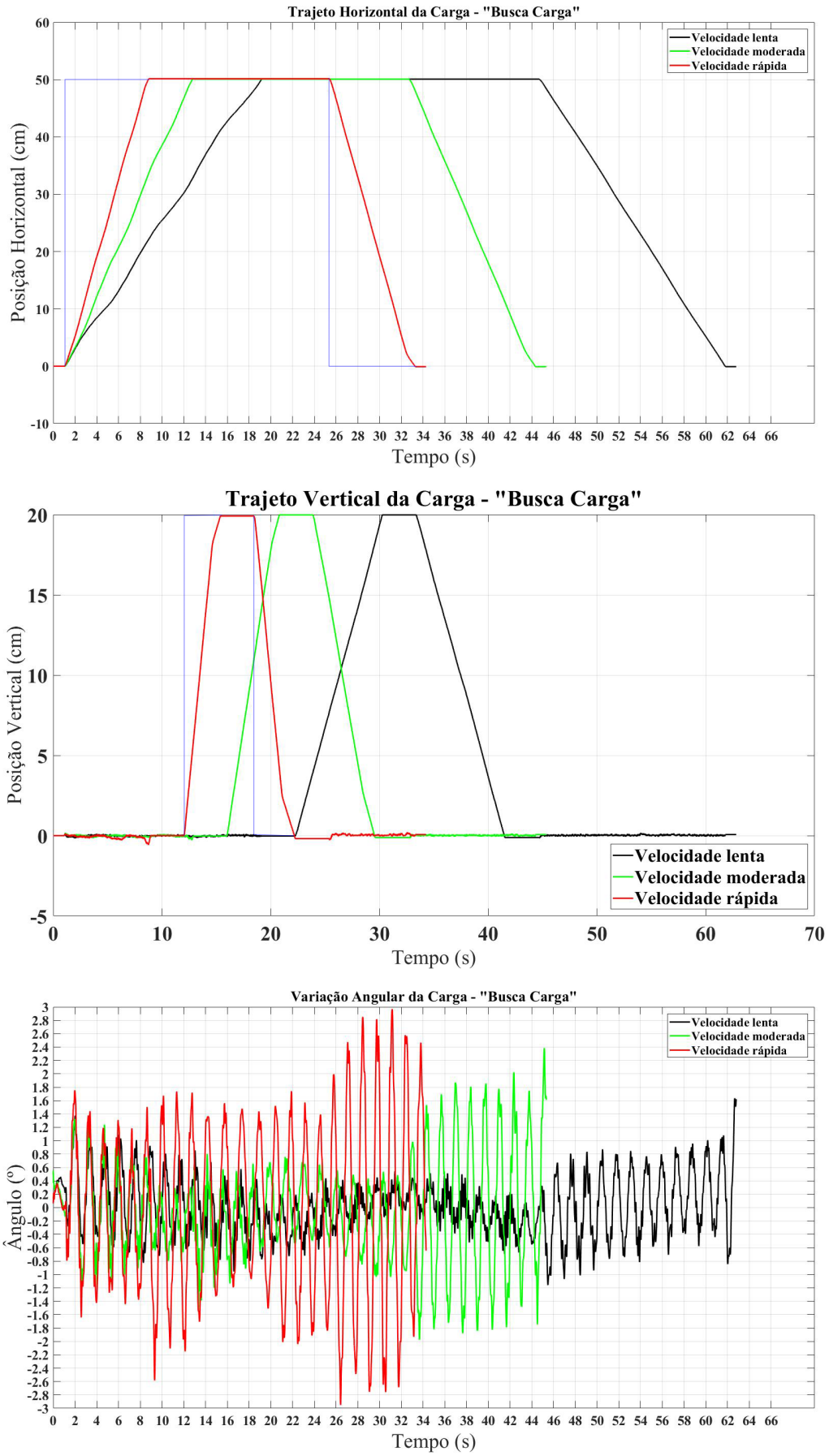
Analisando as oscilações da carga durante o trajeto, nota-se um aumento na inclinação da carga para a velocidade rápida exatamente no ponto de desnivelamento citado anteriormente, já que há uma mudança de aceleração horizontal que projeta a carga devido à sua inércia. Deste modo, mesmo ao modificar o sentido de movimento esta oscilação se manteve até o fim do percurso. Para baixas velocidades este efeito não fica tão evidente, já que a oscilação foi majoritariamente causada pela aceleração inicial, ocorrendo uma oscilação remanescente devido à alta sensibilidade dos cabos de sustentação da carga. Nota-se também um leve aumento na inclinação da carga ao fim do percurso realizado com velocidade lenta, também devido à inércia da carga que estava em movimento.

4.2.6 Resposta ao comando de movimentação Busca Carga

O último teste realizado foi para o movimento pré-definido nomeado "Busca Carga", onde o acionamento simula o processo de deslocamento do *trolley* até a posição horizontal onde a carga se encontra, realiza a descida para simular a coleta da carga, retorna à posição vertical inicial e por fim se desloca para a direita até a posição de origem do sistema de coordenadas. Neste último teste espera-se uma oscilação maior da carga, já que serão combinados todos os movimentos básicos descritos anteriormente, com diversos momentos de aceleração e desaceleração durante o processo.

A Tabela 6 a seguir expõe as características da resposta do sistema durante os movimentos horizontais de ida e de volta, enquanto a Tabela 7 resume as características durante a movimentação vertical de coleta da carga. A Figura 37 ilustra as respostas de todos os movimentos realizados pelos componentes monitorados.

Figura 37 – Resposta para a movimentação pré-definida Busca Carga.



Fonte: Elaborado pelo Autor.

Tabela 6 – Características das respostas horizontais ao comando de movimentação "Busca Carga"

Velocidade	**Tempo de ida (s)**	**Tempo de volta (s)**	**Sobressinal (%)**	**Erro estacionário (cm)**	**Deslocamento vertical máximo (cm)**	**Oscilação máxima (°)**
Lenta	17.05	17.3	0.12	0.06	0.16	1.62
Moderada	11.9	11.47	0.12	0.06	0.28	2.4
Rápida	7.82	7.74	0.26	0.13	0.58	2.95

Tabela 7 – Características das respostas verticais ao comando de movimentação "Busca Carga"

Velocidade	**Tempo de descida (s)**	**Tempo de subida (s)**	**Sobressinal (%)**	**Erro estacionário (cm)**	**Deslocamento horizontal máximo (cm)**	**Oscilação máxima (°)**
Lenta	7.96	8.11	0.55	-0.11	0	0.61
Moderada	4.9	5.68	0.55	-0.11	0	1.02
Rápida	3.3	3.75	0.9	-0.18	0	2.14

Analisando os resultados obtidos confirma-se o esperado aumento na oscilação da carga em relação aos outros tipos de percurso testados, chegando a uma oscilação máxima de 2.95 graus ao final do trajeto percorrido com velocidade rápida. Ainda no gráfico de oscilação da carga, desta vez o aumento da oscilação com o aumento da velocidade se tornou ainda mais evidente, já que o trajeto percorrido foi mais longo e com mais momentos de aceleração e desaceleração do conjunto. Analisando os trajetos horizontais nota-se que os percursos de ida e volta foram realizados em tempos semelhantes para cada velocidade, e que os desnivelamentos ocorridos foram poucos, quase não havendo para as velocidades lenta e moderada, e com dois momentos de pico na velocidade rápida, que ocorrem exatamente nos momentos de diminuição de velocidade para o ajuste fino de posição final.

Durante os movimentos de descida e subida, as oscilações de carga diminuíram, corroborando com os resultados verificados para os tipos de movimentação vertical testados anteriormente. Os sobressinais verificados verticalmente ocorreram somente no momento de reposicionamento vertical da carga, sendo sempre inferiores a 1%. Os erros estacionários presentes ocorreram no mesmo momento ao final do movimento de subida, sendo corrigidos logo ao início do movimento horizontal de retorno à origem do sistema de coordenadas.

5 CONCLUSÃO

Os resultados obtidos para a Planta operando nas três diferentes velocidades mostram que os valores para a oscilação angular da carga e o erro estacionário de posicionamento permaneceram sempre abaixo de 3 graus e 0,2 centímetros , respectivamente. Deste modo, considerando-se as não-linearidades e o atrito presentes na planta na qual o presente trabalho foi realizado, conclui-se que o sistema de controle implementado foi eficiente ao posicionar o conjunto *trolley*/carga nas posições definidas pelo usuário através da interface gráfica desenvolvida, tentando minimizar ao máximo as oscilações de carga que pudessem ocorrer durante este trajeto.

O controle e sincronia durante o percurso a ser realizado desde a posição inicial até o *setpoint* nesta planta não é tão simples de ser implementado como é em uma ponte rolante convencional, já que para realizar um movimento horizontal simples com uma velocidade constante para a esquerda, por exemplo, é necessário também acionar o motor vertical para que ele mova a carga para baixo com velocidade tangencial idêntica à do motor horizontal, visando mantê-la no mesmo nível durante sua movimentação. Isto causa um problema complexo ao tentar realizar um controle otimizado do deslocamento da carga, já que qualquer que seja o padrão de movimentação horizontal a ser realizado, é necessário haver um movimento compensatório igualmente eficiente por parte do motor vertical para tentar manter a carga nivelada verticalmente durante todo o percurso.

Esta questão da necessidade de sincronia entre os motores nesta planta torna-se um empecilho devido ao fato de haver consideráveis não-linearidades e diferentes regiões de atrito tanto para o movimento horizontal, quanto para o vertical, que quando estão acoplados tornam complexa uma implementação eficiente de um controlador mais sofisticado. A possível solução de transportar o motor vertical para junto do *trolley* mostrou-se inviável de ser realizada dentro do tempo hábil para a realização deste projeto devido às características construtivas da maquete e das dimensões, tanto do motor, quanto do *trolley*.

Outros fatores envolvendo os motores e a estrutura em geral prejudicaram uma implementação eficiente de técnicas de controle mais sofisticadas para a planta deste projeto. Uma delas é o fato dos motores possuírem consideráveis zonas mortas em seus acionamentos, onde o acionamento por PWM, que deveria idealmente ser efetivo desde 0 (0% da velocidade máxima) até 255 (100% da velocidade máxima), só causa movimentação da carga desde 140 a 255, no caso de movimentação para a esquerda, e desde 150 a 255, no caso de movimentação para cima, por exemplo, dificultando um ajuste fino de posicionamento

final e até mesmo uma implementação de mudança rápida de sentido de rotação para tentar diminuir o balanço de carga e posteriormente anular o erro estacionário verificado durante a análise das respostas ao degrau.

Como mencionado, outro fator que contribuiu para aumentar a complexidade de controle da planta foi a grande quantidade de atrito presente durante a movimentação do *trolley*, proveniente tanto do contato entre o *trolley* e a estrutura fixa, quanto internamente por parte dos motores. Este atrito varia consideravelmente ao longo do percurso devido às imperfeições das engrenagens que comandam a movimentação do *trolley*, dificultando a sintonia entre as velocidades dos motores horizontal e vertical para que a carga não perca o nivelamento vertical durante o percurso, especialmente durante velocidades baixas, como pôde ser visto durante a análise das respostas.

Para futuras melhorias sugere-se reconstruir toda a estrutura de movimentação vertical da carga, fixando o motor responsável pela movimentação vertical juntamente ao *trolley*, fazendo com que o sistema de movimentação horizontal e vertical passem a estar desacoplados. Desta maneira, a implementação de controladores mais sofisticados torna-se plausível, já que os sistemas horizontal e vertical podem ser tratados de maneira independente. Ainda tratando do sistema de movimentação vertical, outro ponto de melhoria na estrutura seria substituir o *nylon* que realiza a sustentação da carga por outro material mais robusto, para que assim as oscilações geradas pela movimentação do conjunto sejam mais rapidamente extinguidas, já que como foi analisado no Capítulo 4.2.6, as oscilações geradas ao início do movimento tendiam a se perpetuar devido à alta elasticidade do material utilizado para sustentar a carga.

Para garantir uma maior confiabilidade durante o acionamento dos motores, recomenda-se também substituir o motor de acionamento vertical por outro que drene uma corrente menor, para que assim seja viável a utilização de uma ponte H comercial para acioná-lo com maior estabilidade, já que esta é fabricada de acordo com normas e padronizações. Ao contrário da ponte H utilizada neste projeto, que teve de ser confeccionada manualmente devido à indisponibilidade de modelos comerciais capazes de suprir a alta corrente necessária para acionar o motor atualmente utilizado na planta, deixando o sistema de acionamento vertical sujeito à imperfeições não desejáveis.

REFERÊNCIAS BIBLIOGRÁFICAS

AGILENT, H.-P. *HEDS-5700 Datasheet*. 2016. Disponível em: <http://www.alldatasheet.com/datasheet-pdf/pdf/254226/HP/HEDS-5700.html>. Acesso em: 09 de novembro de 2016. Citado na página 31.

AGUIRRE, L. A. Introdução à identificação de sistemas - técnicas lineares e não-lineares aplicadas a sistemas reais. UFMG, 2007. Citado 2 vezes nas páginas 37 e 55.

ARDUINO, G. *Referência da Linguagem*. 2016. Disponível em: <https://playground.arduino.cc/Portugues/Referencia>. Acesso em: 07 de julho de 2017. Citado na página 38.

ARDUINO, G. *Arduino Mega 2560 Rev3*. 2017. Disponível em: <https://store.arduino.cc/usa/arduino-mega-2560-rev3>. Acesso em: 07 de julho de 2017. Citado 4 vezes nas páginas 18, 20, 21 e 22.

BOXALL, J. *A tutorial on the Arduino universe*. 2010. Disponível em: <http://tronixstuff.com/2010/10/20/tutorial-arduino-and-the-i2c-bus/>. Acesso em: 09 de novembro de 2016. Citado na página 54.

BRASIL, H. V. Máquinas de levantamento. 1 ed. Rio de Janeiro, Editora Guanabara, 1987. Citado na página 14.

CHWA, D. Nonlinear tracking control of 3-d overhead cranes against the initial swing angle and the variation of payload weight. IEEE Transactions on Control Systems Technology, vol. 17, no 4, July, 2009. Citado 2 vezes nas páginas 14 e 15.

COSTA, H. C. d. Aplicação de técnicas de modelagem e controle em sistemas tipo ponte rolante. Dissertação (Mestrado em Engenharia Elétrica) — Instituto Militar de Engenharia, Rio de Janeiro, 2010. Citado na página 15.

ESPINDOLA, L. d. S. Um estudo sobre modelos ocultos de markov hmm - hidden markov model. Pos-Graduacão em Ciência da Computacão - Faculdade de Informática, PUC-RS, 2009. Citado na página 56.

FILHO, D. O. B. *Ponte-H 12V 40A*. 2012. Disponível em: <http://www.robotizando.com.br/artigo_ponte_h_pg4.php>. Acesso em: 09 de novembro de 2016. Citado 2 vezes nas páginas 27 e 28.

FURTADO, D. M. d. C. Controle Ótimo de um protótipo de uma ponte rolante. Monografia (Trabalho de Graduação) - Curso de Engenharia de Controle e Automação, UFMG, 2006. Citado na página 15.

INVENSENSE. *MPU-6000 and MPU-6050 Product Specification Revision 3.4*. 2013. Disponível em: <https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>. Acesso em: 09 de novembro de 2016. Citado 7 vezes nas páginas 30, 32, 33, 34, 36, 54 e 58.

KUO, B. C.; GOLNARAGHI, F. *Sistemas de Controle Automático*. 9.ed. ed. 111 River Street, Hoboken, New Jersey: [s.n.], 2012. Nenhuma citação no texto.

LAUSZUS, K. Kalman filteret. Holstebro Tekniske Gymnasium, 2011. Citado 2 vezes nas páginas 54 e 55.

LAUSZUS, K. *Guide to gyro and accelerometer with Arduino including Kalman filtering*. 2012. Disponível em: <http://forum.arduino.cc/index.php/topic,58048.0.html>. Acesso em: 09 de novembro de 2016. Citado na página 54.

MARCOS, E. C. P. Modelagem e controle de uma ponte rolante de três graus de liberdade utilizando controle por planejamento e rastreamento de trajetória. Dissertação (Mestrado em Sistemas Mecatrônicos) — Universidade de Brasília, 2014. Citado 2 vezes nas páginas 14 e 15.

MATHWORKS, T. *MATLAB Documentation*. 2016. Disponível em: <https://www.mathworks.com/help/matlab/>. Acesso em: 10 de novembro de 2016. Citado na página 39.

MULTILóGICA, S. *Guia do Arduino*. 2016. Disponível em: <https://multilogica-shop.com/download_guia_arduino>. Acesso em: 09 de novembro de 2016. Citado na página 19.

OGATA, K. *Engenharia de Controle Moderno*. 5.ed. ed. Upper Saddle River, New Jersey: [s.n.], 2011. Nenhuma citação no texto.

OLIVEIRA, V. M. R. d. Desenvolvimento dos sensores e atuadores de uma maquete de ponte rolante. Iniciação Cientifica do Curso de Engenharia Elétrica, UFES, Vitória, 2015. Citado 2 vezes nas páginas 18 e 22.

SALDANHA, T. *Encoder - Tipos e funcionamento*. 2016. Disponível em: <http://sistemaembutido.com.br/article.php?id=34>. Acesso em: 09 de novembro de 2016. Citado na página 30.

TEIXEIRA, H. T. Desenvolvimento de um ambiente de controle e monitoramento em tempo real usando o malab e a placa de aquisição nudaq pci-9112. Monografia (Trabalho de Graduação) - Curso de Engenharia Elétrica, UFES, 2008. Citado na página 40.

THOMSEN, A. *Motor DC com driver ponte H L298N*. 2013. Disponível em: <http://blog.filipeflop.com/motores-e-servos/motor-dc-arduino-ponte-h-l298n.html>. Acesso em: 09 de novembro de 2016. Citado 3 vezes nas páginas 24, 25 e 26.

WELCH, G.; BISHOP, G. An introduction to the kalman filter. Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175, 2006. Citado 2 vezes nas páginas 55 e 56.

YANG, J. H. On the adaptative tracking control of 3-d overhead crane systems. Adaptative control, Kwanho You (Ed.), ISBN: 978-953-7619-47-3, 2009. Citado na página 14.

Apêndices

APÊNDICE A – ALGORITMO DE INTERFACE GRÁFICA E COMANDOS DO MATLAB

```
function varargout = p_rolante(varargin)
% P_ROLANTE MATLAB code for P_ROLANTE.fig
%      P_ROLANTE, by itself, creates a new P_ROLANTE or raises the existing
%      singleton*.
%
%      H = P_ROLANTE returns the handle to a new P_ROLANTE or the handle to
%      the existing singleton*.
%
%      P_ROLANTE('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in P_ROLANTE.M with the given input arguments.
%
%      P_ROLANTE('Property','Value',...) creates a new P_ROLANTE or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before p_rolante_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to p_rolante_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help p_rolante

% Last Modified by GUIDE v2.5 27-Jul-2017 13:18:50

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @p_rolante_OpeningFcn, ...
                   'gui_OutputFcn',  @p_rolante_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before p_rolante is made visible.
function p_rolante_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to p_rolante (see VARARGIN)
```

```
54
55 % Centraliza janela
56 movegui('center');
57
58 % Choose default command line output for p_rolante
59 handles.output = hObject;
60
61 % Update handles structure
62 guidata(hObject, handles);
63
64 % UIWAIT makes p_rolante wait for user response (see UIRESUME)
65 % uiwait(handles.figure1);
66
67 %Configuracao inicial dos botoes da interface
68 set(handles.radio_velo_lenta,'Value',0);
69 set(handles.radio_velo_lenta,'Enable','off');
70 set(handles.radio_velo_moderada,'Value',0);
71 set(handles.radio_velo_moderada,'Enable','off');
72 set(handles.radio_velo_rapida,'Value',0);
73 set(handles.radio_velo_rapida,'Enable','off');
74 set(handles.radio_vai_e_volta,'Value',0);
75 set(handles.radio_vai_e_volta,'Enable','off');
76 set(handles.edit_vai_e_volta,'Enable','off');
77 set(handles.radio_busca_carga,'Value',0);
78 set(handles.radio_busca_carga,'Enable','off');
79 set(handles.edit_busca_carga,'Enable','off');
80 set(handles.radio_automatico,'Value',0);
81 set(handles.radio_automatico,'Enable','off');
82 set(handles.radio_manual,'Value',0);
83 set(handles.radio_manual,'Enable','off');
84 set(handles.radio_horizontal,'Value',0);
85 set(handles.radio_horizontal,'Enable','off');
86 set(handles.radio_vertical,'Value',0);
87 set(handles.radio_vertical,'Enable','off');
88 set(handles.edit_horizontal,'Enable','off');
89 set(handles.edit_vertical,'Enable','off');
90 set(handles.edit_nome_arquivo,'Enable','off');
91 set(handles.push_remove_arduino,'Enable','off');
92 set(handles.push_zera,'Enable','off');
93 set(handles.toggle_start,'Enable','off');
94 set(handles.toggle_stop,'Enable','off');
95 set(handles.toggle_sobe,'Enable','off');
96 set(handles.toggle_desce,'Enable','off');
97 set(handles.toggle_direita,'Enable','off');
98 set(handles.toggle_esquerda,'Enable','off');
99
100
101 subplot(2,2,4)
102 axis([0 20 -90 90])
103 title( 'Variacao Angular da Carga' );
104 grid on
105 xlabel('Tempo(s)');
106 ylabel('Angulo (graus)');
107 subplot(2,2,2)
108 axis([0 20 0 100])
109 title( 'Trajeto Horizontal da Carga' );
110 grid on
111 xlabel('Tempo(s)');
112 ylabel('Posicao Horizontal (cm)');
113 subplot(2,2,3)
```

```
114 axis([0 20 0 100])
115 title( 'Trajeto Vertical da Carga' );
116 grid on
117 xlabel('Tempo(s)');
118 ylabel('Posicao Vertical(cm)');
119
120
121 % --- Outputs from this function are returned to the command line.
122 function varargout = p_rolante_OutputFcn(hObject, eventdata, handles)
123 % varargout  cell array for returning output args (see VARARGOUT);
124
125 % Get default command line output from handles structure
126 varargout{1} = handles.output;
127
128
129 % --- Executa ao pressionar o botao 'Inicializar Arduino'
130 function push_setuparduino_Callback(hObject, eventdata, handles)
131
132 %Declaracao das variaveis globais a serem utilizadas
133 global a setpoint_h setpoint_v setpoint_vai_e_volta setpoint_busca_carga offset_angulo
        flag_inicio flag_fim falha_i2cAddress falha_i2cRead falha_i2cWrite falha_i2cTimeout
        erro_medicao;
134
135 delete(instrfindall); %Encerra todas as conexoes 'serial' atuais
136 pause(0.1);
137
138 offset_angulo=0; %Variavel para ajustar o offset de leitura do angulo
139
140 baud = 115200;%Define a velocidade em bit/s (baud rate) de comunicacao entre arduino e
        matlab
141 setpoint_h=0; %Coordenada desejada de posicao horizontal em [cm] a partir do referencial
142 setpoint_v=0; %Coordenada desejada de posicao vertical em [cm] a partir do referencial
143 setpoint_vai_e_volta=0; %Coordenada desejada de posicao horizontal em [cm] a partir do
        referencial
144 setpoint_busca_carga=0; %Coordenada desejada de posicao horizontal em [cm] a partir do
        referencial
145 flag_inicio=9999;       %Flag de identificacao de inicio de comunicacao entre arduino e
        matlab
146 flag_fim=10000;         %Flag de identificacao de fim de comuicacao entre arduino e
        matlab
147 falha_i2cAddress=11000; %Flag de identificacao de falha de comunicacao i2c entre arduino
        e MPU-6050
148 falha_i2cRead=12000;    %Flag de identificacao de falha de comunicacao i2c entre arduino
        e MPU-6050
149 falha_i2cWrite=13000;   %Flag de identificacao de falha de comunicacao i2c entre arduino
        e MPU-6050
150 falha_i2cTimeout=14000; %Flag de identificacao de falha de comunicacao i2c entre arduino
        e MPU-6050
151 erro_medicao=15000;     %Flag de identificacao de erro de medicao
152
153
154 a=serial('COM4','BaudRate',baud); %Cria a comunicacao serial com o arduino usando o baud
        rate pre-definido
155 flushinput(a);                    %Limpa o buffer da porta de entrada serial
156 flushoutput(a);                   %Limpa o buffer da porta de saida serial
157 fopen(a);                         %Abre a comunicacao com o arduino
158 flushinput(a);                    %Limpa o buffer da porta de entrada serial
159 flushoutput(a);                   %Limpa o buffer da porta de saida serial
160 pause(3);                         %Faz pausa para calibracao da medicao do angulo pelo
        arduino
```

```
161
162 %Configura as condicoes iniciais dos botoes da interface apos estabelecer a comunicacao
        com o arduino
163 set(handles.radio_automatico,'Value',1);
164 set(handles.radio_automatico,'Enable','on');
165 set(handles.radio_manual,'Enable','on');
166 set(handles.radio_horizontal,'Value',1);
167 set(handles.radio_horizontal,'Enable','on');
168 set(handles.radio_vertical,'Value',0);
169 set(handles.radio_vertical,'Enable','on');
170 set(handles.radio_vai_e_volta,'Value',0);
171 set(handles.radio_vai_e_volta,'Enable','on');
172 set(handles.radio_busca_carga,'Value',0);
173 set(handles.radio_busca_carga,'Enable','on');
174 set(handles.radio_velo_lenta,'Value',0);
175 set(handles.radio_velo_lenta,'Enable','on');
176 set(handles.radio_velo_moderada,'Value',1);
177 set(handles.radio_velo_moderada,'Enable','on');
178 set(handles.radio_velo_rapida,'Value',0);
179 set(handles.radio_velo_rapida,'Enable','on');
180 set(handles.edit_horizontal,'Enable','on');
181 set(handles.edit_nome_arquivo,'Enable','on');
182 set(handles.edit_vertical,'Enable','off');
183 set(handles.edit_vai_e_volta,'Enable','off');
184 set(handles.edit_busca_carga,'Enable','off');
185 set(handles.push_zera,'Enable','on');
186 set(handles.toggle_start,'Enable','on');
187 set(handles.push_setuparduino,'Enable','off');
188 set(handles.push_remove_arduino,'Enable','on');
189
190
191 % --- Executa ao pressionar o botao 'Zera Coordenadas'
192 function push_zera_Callback(hObject, eventdata, handles)
193
194 %Declaracao das variaveis globais a serem utilizadas
195 global a flag_inicio flag_fim;
196
197 primeiro_contato = 0;        %Variavel de indicacao de primeiro contato entre matlab e
        arduino
198 byte_lido = 0;               %Variavel que guarda o byte lido pela porta serial
199 flushinput(a);              %Limpa o buffer da porta de entrada serial
200 flushoutput(a);              %Limpa o buffer da porta de saida serial
201
202 while (byte_lido ~= flag_fim) %Executa enquanto nao receber a sinalizacao de fim por
        parte do arduino
203
204     byte_lido = fread(a,1,'float');     %Le byte da porta serial
205
206     if (byte_lido ~= flag_fim)          %Se o byte lido nao for de fim,
207       if (primeiro_contato == 0)        %Se for o primeiro contato entre o matlab e o
        arduino,
208           if (byte_lido == flag_inicio) %Se o byte recebido for para inicializar a
        transmissao de comandos,
209               flushinput(a);            %Limpa o buffer da porta de entrada serial
210               flushoutput(a);           %Limpa o buffer da porta de saida serial
211               primeiro_contato = 1;     %Muda a flag indicando que nao e mais o primeiro
        contato
212               posicao = 0;              %Variavel necessaria para o complemento da
        mensagem a ser enviada
```

```
213                 fprintf(a, '%s', sprintf('<zera,null,null,null,%f>', posicao)); %Envia o
        comando de 'Zera Coordenadas' via serial
214             end
215         end
216         else                                    %Se recebeu o byte de fim de comunicacao,
217             zeracoordenadas;
218             disp('Zerou Coordenadas');          %Mostra no workspace que zerou as coordenadas
219         end
220     end
221     pause(0.1);
222     flushinput(a);                              %Limpa o buffer da porta de entrada serial
223     flushoutput(a);                             %Limpa o buffer da porta de saida serial
224
225
226     % --- Executa ao pressionar o botao 'Remove Arduino'
227     function push_remove_arduino_Callback(hObject, eventdata, handles)
228
229     removearduino;                          %Chama funcao para mostrar caixa de mensagem
230     enable_push_setuparduino = 0;           %Variavel auxiliar
231     while enable_push_setuparduino == 0 %Enquanto nao tiver pressionado 'Sim'
232         drawnow
233         enable_push_setuparduino = getappdata(0,'evalue');%Verifica se pressionou 'Sim'
234         if enable_push_setuparduino == 1 %Se pressionou, desabilita os botoes da interface
235             set(handles.push_setuparduino,'Enable','on');
236             set(handles.radio_automatico,'Value',0);
237             set(handles.radio_automatico,'Enable','off');
238             set(handles.radio_manual,'Value',0);
239             set(handles.radio_manual,'Enable','off');
240             set(handles.radio_horizontal,'Value',0);
241             set(handles.radio_horizontal,'Enable','off');
242             set(handles.radio_vertical,'Value',0);
243             set(handles.radio_vertical,'Enable','off');
244             set(handles.radio_vai_e_volta,'Value',0);
245             set(handles.radio_vai_e_volta,'Enable','off');
246             set(handles.radio_busca_carga,'Value',0);
247             set(handles.radio_busca_carga,'Enable','off');
248             set(handles.radio_velo_lenta,'Value',0);
249             set(handles.radio_velo_lenta,'Enable','off');
250             set(handles.radio_velo_moderada,'Value',0);
251             set(handles.radio_velo_moderada,'Enable','off');
252             set(handles.radio_velo_rapida,'Value',0);
253             set(handles.radio_velo_rapida,'Enable','off');
254             set(handles.edit_horizontal,'Enable','off');
255             set(handles.edit_vertical,'Enable','off');
256             set(handles.edit_vai_e_volta,'Enable','off');
257             set(handles.edit_busca_carga,'Enable','off');
258             set(handles.edit_nome_arquivo,'Enable','off');
259             set(handles.push_remove_arduino,'Enable','off');
260             set(handles.push_zera,'Enable','off');
261             set(handles.toggle_start,'Enable','off');
262             set(handles.toggle_stop,'Enable','off');
263             set(handles.toggle_sobe,'Enable','off');
264             set(handles.toggle_desce,'Enable','off');
265             set(handles.toggle_direita,'Enable','off');
266             set(handles.toggle_esquerda,'Enable','off');
267         end
268     end
269
270
271     % --- Executa ao pressionar o botao 'Start'
```

```
272 function toggle_start_Callback(hObject, eventdata, handles)
273
274 %Desabilita os botoes da interface por seguranca
275 set(handles.radio_horizontal,'Enable','off');
276 set(handles.edit_horizontal,'Enable','off');
277 set(handles.radio_vertical,'Enable','off');
278 set(handles.edit_vertical,'Enable','off');
279 set(handles.radio_vai_e_volta,'Enable','off');
280 set(handles.edit_vai_e_volta,'Enable','off');
281 set(handles.radio_busca_carga,'Enable','off');
282 set(handles.edit_busca_carga,'Enable','off');
283 set(handles.edit_nome_arquivo,'Enable','off');
284 set(handles.toggle_start,'Enable','off');
285
286
287 rotina_principal(handles);
288
289
290 set(handles.radio_velo_lenta,'Enable','on');                    %Reabilita os botoes da
        interface ao final da execucao do comando
291 set(handles.radio_velo_moderada,'Enable','on');
292 set(handles.radio_velo_rapida,'Enable','on');
293 set(handles.radio_automatico,'Enable','on');
294 set(handles.radio_manual,'Enable','on');
295 set(handles.radio_horizontal,'Enable','on');
296 set(handles.radio_vertical,'Enable','on');
297 set(handles.radio_vai_e_volta,'Enable','on');
298 set(handles.radio_busca_carga,'Enable','on');
299 set(handles.toggle_stop,'Value',0);
300 set(handles.toggle_stop,'Enable','off');
301 set(handles.toggle_start,'Value',0);
302 set(handles.toggle_start,'Enable','on');
303 set(handles.push_zera,'Enable','on');
304 set(handles.push_remove_arduino,'Enable','on');
305 set(handles.edit_nome_arquivo,'Enable','on');
306 if get(handles.radio_horizontal,'Value')==1
307     set(handles.edit_horizontal,'Enable','on');
308 elseif get(handles.radio_vertical,'Value')==1
309     set(handles.edit_vertical,'Enable','on');
310 elseif get(handles.radio_vai_e_volta,'Value')==1
311     set(handles.edit_vai_e_volta,'Enable','on');
312 elseif get(handles.radio_busca_carga,'Value')==1
313     set(handles.edit_busca_carga,'Enable','on');
314 end
315
316
317 % --- Executa ao pressionar o botao 'Desce', realizando o movimento
318 % vertical para baixo da carga ate que seja pressionado o botao 'Stop'
319 function toggle_desce_Callback(hObject, eventdata, handles)
320
321 set(handles.toggle_desce,'Enable','off');
322 set(handles.toggle_desce,'Value',1);
323 set(handles.toggle_sobe,'Enable','off');
324 set(handles.toggle_sobe,'Value',0);
325 set(handles.toggle_direita,'Enable','off');
326 set(handles.toggle_direita,'Value',0);
327 set(handles.toggle_esquerda,'Enable','off');
328 set(handles.toggle_esquerda,'Value',0);
329
330 rotina_principal(handles);
```

```
331
332
333 % --- Executes on button press in toggle_sobe.
334 function toggle_sobe_Callback(hObject, eventdata, handles)
335
336 set(handles.toggle_desce,'Enable','off');
337 set(handles.toggle_desce,'Value',0);
338 set(handles.toggle_sobe,'Enable','off');
339 set(handles.toggle_sobe,'Value',1);
340 set(handles.toggle_direita,'Enable','off');
341 set(handles.toggle_direita,'Value',0);
342 set(handles.toggle_esquerda,'Enable','off');
343 set(handles.toggle_esquerda,'Value',0);
344
345 rotina_principal(handles);
346
347
348 % --- Executes on button press in toggle_direita.
349 function toggle_direita_Callback(hObject, eventdata, handles)
350
351 set(handles.toggle_desce,'Enable','off');
352 set(handles.toggle_desce,'Value',0);
353 set(handles.toggle_sobe,'Enable','off');
354 set(handles.toggle_sobe,'Value',0);
355 set(handles.toggle_direita,'Enable','off');
356 set(handles.toggle_direita,'Value',1);
357 set(handles.toggle_esquerda,'Enable','off');
358 set(handles.toggle_esquerda,'Value',0);
359
360 rotina_principal(handles);
361
362
363 % --- Executes on button press in toggle_esquerda.
364 function toggle_esquerda_Callback(hObject, eventdata, handles)
365
366 set(handles.toggle_desce,'Enable','off');
367 set(handles.toggle_desce,'Value',0);
368 set(handles.toggle_sobe,'Enable','off');
369 set(handles.toggle_sobe,'Value',0);
370 set(handles.toggle_direita,'Enable','off');
371 set(handles.toggle_direita,'Value',0);
372 set(handles.toggle_esquerda,'Enable','off');
373 set(handles.toggle_esquerda,'Value',1);
374
375 rotina_principal(handles);
376
377
378 % --- Executa ao pressionar o botao 'Stop'
379 function toggle_stop_Callback(hObject, eventdata, handles)
380
381 if get(handles.radio_manual,'Value')==1            %Se o botao foi pressionado enquanto
      fazia movimentacao manual,
382     set(handles.radio_velo_lenta,'Enable','on');                    %Reabilita os botoes
      da interface ao final da execucao do comando
383     set(handles.radio_velo_moderada,'Enable','on');
384     set(handles.radio_velo_rapida,'Enable','on');
385     set(handles.radio_automatico,'Enable','on');    %Reabilita os botoes que haviam sido
      desabilitados ao inicio da execucao
386     set(handles.radio_manual,'Enable','on');
387     set(handles.toggle_stop,'Enable','off');
```

```
388     set(handles.toggle_desce,'Enable','on');
389     set(handles.toggle_desce,'Value',0);
390     set(handles.toggle_sobe,'Enable','on');
391     set(handles.toggle_sobe,'Value',0);
392     set(handles.toggle_direita,'Enable','on');
393     set(handles.toggle_direita,'Value',0);
394     set(handles.toggle_esquerda,'Enable','on');
395     set(handles.toggle_esquerda,'Value',0);
396     set(handles.push_zera,'Enable','on');
397     set(handles.push_remove_arduino,'Enable','on');
398 end
399
400
401 function rotina_principal(handles)
402
403 set(handles.radio_automatico,'Enable','off'); %Desabilita botoes por seguranca ao inicio
        do movimento
404 set(handles.radio_manual,'Enable','off');
405 set(handles.radio_velo_lenta,'Enable','off');
406 set(handles.radio_velo_moderada,'Enable','off');
407 set(handles.radio_velo_rapida,'Enable','off');
408 set(handles.toggle_stop,'Enable','on');
409 set(handles.toggle_stop,'Value',0);
410 set(handles.push_zera,'Enable','off');
411 set(handles.push_remove_arduino,'Enable','off');
412
413 %Declara variaveis globais a serem utilizadas
414 global a setpoint_h setpoint_v setpoint_vai_e_volta setpoint_busca_carga offset_angulo
        flag_inicio flag_fim;
415
416 vetor_bytes_lidos = zeros(1,3);      %Vetor para guardar cada trio de leituras recebidas
417 count_bytes_lidos = 1;               %Contador para auxiliar quantos bytes foram recebidos
418 primeiro_contato = 0;                %Variavel de indicacao de primeiro contato entre
        matlab e arduino
419 indice = 1;                          %Indice do vetor de dados recebidos
420 inicia_tic = 1;                      %Variavel auxiliar para inicio de contagem de tempo
421 posicao_h = [];                      %Vetor que guardara as leituras de posicao horizontal
         recebidas via serial
422 posicao_v = [];                      %Vetor que guardara as leituras de posicao vertical
        recebidas via serial
423 angulo = [];                         %Vetor que guardara as leituras de angulo recebidas
        via serial
424 tempo = [];                          %Vetor que guardara o tempo em que cada leitura
        ocorreu
425 byte_lido = 0;                       %Variavel que guarda o byte lido pela porta serial
426 flushinput(a);                       %Limpa o buffer da porta de entrada serial
427 flushoutput(a);                      %Limpa o buffer da porta de saida serial
428 offset_angulo = 3.9;
429
430 while (byte_lido < flag_fim)         %Executa enquanto nao receber a sinalizacao de fim
        por parte do arduino
431     drawnow                          %Necessario para habilitar o pressionamento de botoes
         enquanto a aplicacao roda
432
433     byte_lido = fread(a,1,'float'); %Le byte da porta serial
434
435     if (byte_lido < flag_fim)            %Se o byte lido nao for de fim,
436       if (primeiro_contato == 0)         %Se for o primeiro contato entre o matlab e o
        arduino,
```

```
437         if (byte_lido == flag_inicio) %Se o byte recebido for para inicializar a
    transmissao de comandos,
438             flushinput(a);            %Limpa o buffer da porta de entrada serial
439             flushoutput(a);           %Limpa o buffer da porta de saida serial
440             primeiro_contato = 1;     %Muda a flag indicando que ja ocorreu o primeiro
    contato
441             if get(handles.radio_automatico,'Value')==1      %Verifica se o comando
    dado foi para movimentacao automatica
442                 if get(handles.radio_horizontal,'Value')==1  %Se foi, verifica se o
    comando dado foi para movimentacao horizontal,
443                     setpoint_h = str2double(get(handles.edit_horizontal,'String')); %Se
     sim, pega o valor de posicao horizontal inserido na caixa de texto
444
445                      if setpoint_h < 0                                    %Se o valor
    inserido for menor que zero,
446                          setpoint_h = 0;                                  %Limita o
    valor em zero
447                          set(handles.edit_horizontal,'String','0');       %Modifica o
    valor da caixa de texto para zero
448                      end
449
450                     if setpoint_h > 100                                    %Se o valor
    inserido for maior que 100cm,
451                         setpoint_h = 100;                                  %Limita o
    valor em 100cm
452                         set(handles.edit_horizontal,'String','100');       %Modifica o
    valor da caixa de texto para 100
453                     end
454
455                     if get(handles.radio_velo_lenta,'Value')==1            %Se o
    comando foi dado com o botao de velocidade lenta ativo,
456                         fprintf(a, '%s', sprintf('<start,automatico,horizontal,lenta,%f
    >', setpoint_h)); %Manda via serial para o arduino o comando de iniciar movimento
    automatico horizontal lento ate setpoint_h
457                         pause(0.1);
458                     elseif get(handles.radio_velo_moderada,'Value')==1     %Se o
    comando foi dado com o botao de velocidade moderada ativo,
459                         fprintf(a, '%s', sprintf('<start,automatico,horizontal,moderada
    ,%f>', setpoint_h)); %Manda via serial para o arduino o comando de iniciar movimento
    automatico horizontal moderado ate setpoint_h
460                         pause(0.1);
461                     elseif get(handles.radio_velo_rapida,'Value')==1       %Se o
    comando foi dado com o botao de velocidade rapida ativo,
462                         fprintf(a, '%s', sprintf('<start,automatico,horizontal,rapida,%
    f>', setpoint_h)); %Manda via serial para o arduino o comando de iniciar movimento
    automatico horizontal rapido ate setpoint_h
463                         pause(0.1);
464                     end
465                 elseif get(handles.radio_vertical,'Value')==1 %Se nao foi para
    movimentacao horizontal, verifica se foi para movimento vertical
466                     setpoint_v = str2double(get(handles.edit_vertical,'String')); %Se
    foi, pega o valor da posicao vertical inserido na caixa de texto
467                     if setpoint_v < 0                                      %Se o
     valor inserido for menor que zero
468                         setpoint_v = 0;                                    %
    Limita o valor em zero
469                         set(handles.edit_vertical,'String','0');           %
    Modifica o valor da caixa de texto para zero
470                     end
471
```

```
472                     if setpoint_v > 65                                          %Se o valor inserido for maior que 65cm,
473                         setpoint_v = 65;                                        %Limita o valor em 65cm
474                         set(handles.edit_vertical,'String','65');               %Modifica o valor da caixa de texto para 65
475                     end
476                     if get(handles.radio_velo_lenta,'Value')==1             %Se o comando foi dado com o botao de velocidade lenta ativo,
477                         fprintf(a, '%s', sprintf('<start,automatico,vertical,lenta,%f>', setpoint_v)); %Manda via serial para o arduino o comando de iniciar movimento automatico vertical lento ate setpoint_v
478                         pause(0.1);
479                     elseif get(handles.radio_velo_moderada,'Value')==1      %Se o comando foi dado com o botao de velocidade moderada ativo,
480                         fprintf(a, '%s', sprintf('<start,automatico,vertical,moderada,%f>', setpoint_v)); %Manda via serial para o arduino o comando de iniciar movimento automatico vertical moderado ate setpoint_v
481                         pause(0.1);
482                     elseif get(handles.radio_velo_rapida,'Value')==1        %Se o comando foi dado com o botao de velocidade rapida ativo,
483                         fprintf(a, '%s', sprintf('<start,automatico,vertical,rapida,%f>', setpoint_v)); %Manda via serial para o arduino o comando de iniciar movimento automatico vertical rapido ate setpoint_v
484                         pause(0.1);
485                     end
486                 elseif get(handles.radio_vai_e_volta,'Value')==1 %Se nao foi para movimentacao horizontal nem vertical, verifica se foi para movimento vai e volta
487                     setpoint_vai_e_volta = str2double(get(handles.edit_vai_e_volta,'String')); %Se foi, pega o valor da posicao horizontal inserido na caixa de texto
488                     if setpoint_vai_e_volta < 0
        %Se o valor inserido for menor que zero
489                         setpoint_vai_e_volta = 0;
        %Limita o valor em zero
490                         set(handles.edit_vai_e_volta,'String','0');
        %Modifica o valor da caixa de texto para zero
491                     end
492
493                     if setpoint_vai_e_volta > 100
        %Se o valor inserido for maior que 100cm,
494                         setpoint_vai_e_volta = 100;
        %Limita o valor em 100cm
495                         set(handles.edit_vai_e_volta,'String','100');
        %Modifica o valor da caixa de texto para 100
496                     end
497                     if get(handles.radio_velo_lenta,'Value')==1             %Se o comando foi dado com o botao de velocidade lenta ativo,
498                         fprintf(a, '%s', sprintf('<start,automatico,vaievolta,lenta,%f>', setpoint_vai_e_volta)); %Manda via serial para o arduino o comando de iniciar movimento automatico vai e volta lento
499                         pause(0.1);
500                     elseif get(handles.radio_velo_moderada,'Value')==1      %Se o comando foi dado com o botao de velocidade moderada ativo,
501                         fprintf(a, '%s', sprintf('<start,automatico,vaievolta,moderada,%f>', setpoint_vai_e_volta)); %Manda via serial para o arduino o comando de iniciar movimento automatico vai e volta moderado
502                         pause(0.1);
503                     elseif get(handles.radio_velo_rapida,'Value')==1        %Se o comando foi dado com o botao de velocidade rapida ativo,
```

```
504                         fprintf(a, '%s', sprintf('<start,automatico,vaievolta,rapida,%f
    >', setpoint_vai_e_volta)); %Manda via serial para o arduino o comando de iniciar
    movimento automatico vai e volta rapido
505                         pause(0.1);
506                     end
507                 elseif get(handles.radio_busca_carga,'Value')==1 %Se nao foi para
    movimentacao horizontal nem vertical nem vai e volta, verifica se foi para movimento
    busca carga
508                     setpoint_busca_carga = str2double(get(handles.edit_busca_carga,'
    String')); %Se foi, pega o valor da posicao horizontal inserido na caixa de texto
509                     if setpoint_busca_carga < 0
        %Se o valor inserido for menor que zero
510                         setpoint_busca_carga = 0;
        %Limita o valor em zero
511                         set(handles.edit_busca_carga,'String','0');
        %Modifica o valor da caixa de texto para zero
512                     end
513
514                     if setpoint_busca_carga > 100
        %Se o valor inserido for maior que 100cm,
515                         setpoint_busca_carga = 100;
        %Limita o valor em 100cm
516                         set(handles.edit_busca_carga,'String','100');
        %Modifica o valor da caixa de texto para 100
517                     end
518                     if get(handles.radio_velo_lenta,'Value')==1             %Se o
    comando foi dado com o botao de velocidade lenta ativo,
519                         fprintf(a, '%s', sprintf('<start,automatico,buscacarga,lenta,%f
    >', setpoint_busca_carga)); %Manda via serial para o arduino o comando de iniciar
    movimento automatico busca carga lento
520                         pause(0.1);
521                     elseif get(handles.radio_velo_moderada,'Value')==1      %Se o
    comando foi dado com o botao de velocidade moderada ativo,
522                         fprintf(a, '%s', sprintf('<start,automatico,buscacarga,moderada
    ,%f>', setpoint_busca_carga)); %Manda via serial para o arduino o comando de iniciar
    movimento automatico busca carga moderado
523                         pause(0.1);
524                     elseif get(handles.radio_velo_rapida,'Value')==1        %Se o
    comando foi dado com o botao de velocidade rapida ativo,
525                         fprintf(a, '%s', sprintf('<start,automatico,buscacarga,rapida,%
    f>', setpoint_busca_carga)); %Manda via serial para o arduino o comando de iniciar
    movimento automatico busca carga rapido
526                         pause(0.1);
527                     end
528                 end
529             elseif get(handles.radio_manual,'Value')==1 %Se o comando dado foi para
    movimentacao manual, realiza os mesmos procedimentos acima porem enviando o comando
    correspondente ao movimento manual
530                 if get(handles.toggle_desce,'Value')==1 %Se o botao apertado foi o '
    Desce', envia o comando com a velocidade selecionada correspondente
531                     if get(handles.radio_velo_lenta,'Value')==1
532                         fprintf(a, '%s', sprintf('<start,manual,desce,lenta,%f>',
    setpoint_v));
533                         pause(0.1);
534                     elseif get(handles.radio_velo_moderada,'Value')==1
535                         fprintf(a, '%s', sprintf('<start,manual,desce,moderada,%f>',
    setpoint_v));
536                         pause(0.1);
537                     elseif get(handles.radio_velo_rapida,'Value')==1
```

```
538                         fprintf(a, '%s', sprintf('<start,manual,desce,rapida,%f>',
    setpoint_v));
539                         pause(0.1);
540                     end
541                 elseif get(handles.toggle_sobe,'Value')==1 %Se o botao apertado foi o '
    Sobe', envia o comando com a velocidade selecionada correspondente
542                     if get(handles.radio_velo_lenta,'Value')==1
543                         fprintf(a, '%s', sprintf('<start,manual,sobe,lenta,%f>',
    setpoint_v));
544                         pause(0.1);
545                     elseif get(handles.radio_velo_moderada,'Value')==1
546                         fprintf(a, '%s', sprintf('<start,manual,sobe,moderada,%f>',
    setpoint_v));
547                         pause(0.1);
548                     elseif get(handles.radio_velo_rapida,'Value')==1
549                         fprintf(a, '%s', sprintf('<start,manual,sobe,rapida,%f>',
    setpoint_v));
550                         pause(0.1);
551                     end
552                 elseif get(handles.toggle_direita,'Value')==1 %Se o botao apertado foi
    o 'Direita', envia o comando com a velocidade selecionada correspondente
553                     if get(handles.radio_velo_lenta,'Value')==1
554                         fprintf(a, '%s', sprintf('<start,manual,direita,lenta,%f>',
    setpoint_h));
555                         pause(0.1);
556                     elseif get(handles.radio_velo_moderada,'Value')==1
557                         fprintf(a, '%s', sprintf('<start,manual,direita,moderada,%f>',
    setpoint_h));
558                         pause(0.1);
559                     elseif get(handles.radio_velo_rapida,'Value')==1
560                         fprintf(a, '%s', sprintf('<start,manual,direita,rapida,%f>',
    setpoint_h));
561                         pause(0.1);
562                     end
563                 elseif get(handles.toggle_esquerda,'Value')==1 %Se o botao apertado foi
     o 'Esquerda', envia o comando com a velocidade selecionada correspondente
564                     if get(handles.radio_velo_lenta,'Value')==1
565                         fprintf(a, '%s', sprintf('<start,manual,esquerda,lenta,%f>',
    setpoint_h));
566                         pause(0.1);
567                     elseif get(handles.radio_velo_moderada,'Value')==1
568                         fprintf(a, '%s', sprintf('<start,manual,esquerda,moderada,%f>',
     setpoint_h));
569                         pause(0.1);
570                     elseif get(handles.radio_velo_rapida,'Value')==1
571                         fprintf(a, '%s', sprintf('<start,manual,esquerda,rapida,%f>',
    setpoint_h));
572                         pause(0.1);
573                     end
574                 end
575             end
576         end
577       else
578         x=10;
579         vetor_bytes_lidos(count_bytes_lidos) = byte_lido; %Guarda o ultimo byte lido no
     vetor de bytes lidos
580         count_bytes_lidos = count_bytes_lidos + 1;        %Incrementa o indice do vetor
     de bytes lidos
581         if inicia_tic == 1                                %Se ainda nao iniciou o
    contador de tempo
```

```
582                 tic                                                 %Inicia o contador de tempo
583                 inicia_tic = 0;                                     %Desabilita flag de inicio de
        contagem de tempo
584             end
585
586             if (count_bytes_lidos > 3 )                                 %Se ja leu o trio de
        bytes (posicao_h, posicao_v e angulo),
587                 posicao_h(indice) = vetor_bytes_lidos(1);               %Guarda a
        medicao de posicao horizontal em seu vetor correspondente
588                 posicao_v(indice) = vetor_bytes_lidos(2);               %Guarda a
        medicao de posicao vertical em seu vetor correspondente
589                 angulo(indice) = vetor_bytes_lidos(3) + offset_angulo; %Guarda a
        medicao de angulo + offset em seu vetor correspondente
590                 tempo(indice) = toc;                                    %Guarda o tempo
        de leitura em seu vetor correspondente
591                 count_bytes_lidos = 1;                                  %Reinicia o
        contador de bytes lidos
592                 indice = indice + 1;                                    %Incrementa o
        indice dos vetores de leituras
593             end
594         end
595     end
596     if get(handles.toggle_stop,'Value')==1                          %Se foi pressionado o
        botao 'Stop' da interface grafica,
597         flushinput(a);                                              %Limpa o buffer da porta
        de entrada serial
598         flushoutput(a);                                             %Limpa o buffer da porta
        de saida serial
599         fprintf(a, '%s', sprintf('<stop,null,null,null,0>'));%Manda via serial para o
        arduino o comando de Stop para parar os motores
600         pause(0.1);
601         set(handles.toggle_stop,'Value',0);
602         break;
603     end
604 end
605 tempo_total = toc
606 pause(0.1);
607 flushinput(a);                                                      %Limpa o buffer da porta
        de entrada serial
608 flushoutput(a);                                                     %Limpa o buffer da porta
        de saida serial
609
610 checa_erro(byte_lido);                                              %Verifica se houve erro
        na execucao da aplicacao do arduino
611
612 atualiza_grafico(tempo,angulo,posicao_h,posicao_v, handles);            %Plota os
        graficos na interface e guarda os dados no arquivo de backup
613
614
615 function atualiza_grafico(tempo,angulo,posicao_h,posicao_v, handles) %Funcao que atualiza
        os graficos ao final do processo e escreve os dados no arquivo txt
616
617 x=subplot(2,2,4);
618
619 plot(tempo,angulo,'LineWidth',1,'Color',[0 0 0]);
620 title( 'Variacao Angular da Carga' );
621 grid on
622 xlabel('Tempo(s)');
623 ylabel('Angulo (graus)');
624
```

```
625 y=subplot(2,2,2);
626
627 plot(tempo,posicao_h,'LineWidth',1,'Color',[0 0 0]);
628 title( 'Trajeto Horizontal da Carga' );
629 grid on
630 xlabel('Tempo(s)');
631 ylabel('Posicao Horizontal (cm)');
632
633 z=subplot(2,2,3);
634
635 plot(tempo,posicao_v,'LineWidth',1,'Color',[0 0 0]);
636 title( 'Trajeto Vertical da Carga' );
637 grid on
638 xlabel('Tempo(s)');
639 ylabel('Posicao Vertical(cm)');
640
641 if get(handles.radio_automatico,'Value') == 1
642     nome_arquivo  = get(handles.edit_nome_arquivo,'String');
643     dados = [tempo' posicao_h' posicao_v' angulo'];
644     dlmwrite(nome_arquivo,dados,'delimiter','\t','precision','%.6f');
645 end
646
647
648 function checa_erro(byte_lido) %Funcao que identifica o erro gerado pelo Arduino
649
650 global a flag_fim setpoint_v falha_i2cAddress falha_i2cRead falha_i2cWrite
        falha_i2cTimeout erro_medicao;
651
652 if (byte_lido > flag_fim)
653     fprintf(a, '%s', sprintf('<stop,null,null,null,%f>', setpoint_v));
654     if (byte_lido == falha_i2cAddress)
655         disp('ERRO DE ENDERECO I2C');
656     elseif (byte_lido == falha_i2cRead)
657         disp('FALHA DE LEITURA I2C');
658     elseif (byte_lido == falha_i2cWrite)
659         disp('FALHA DE ESCRITA I2C');
660     elseif (byte_lido == falha_i2cTimeout)
661         disp('FALHA DE TIMEOUT I2C');
662     elseif (byte_lido == erro_medicao)
663         disp('ERRO DE MEDICAO');
664     else
665         disp('FALHA DESCONHECIDA');
666     end
667 end
668
669
670 function edit_horizontal_Callback(hObject, eventdata, handles)
671
672 global setpoint_h;
673
674 setpoint_h = str2double(get(handles.edit_horizontal,'String'));
675
676 if setpoint_h < 0
677     setpoint_h = 0;
678     set(handles.edit_horizontal,'String','0');
679 end
680
681 if setpoint_h > 100
682     setpoint_h = 100;
683     set(handles.edit_horizontal,'String','100');
```

```
684 end
685
686
687 % --- Executes during object creation, after setting all properties.
688 function edit_horizontal_CreateFcn(hObject, eventdata, handles)
689
690 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor
        '))
691     set(hObject,'BackgroundColor','white');
692 end
693
694
695 function edit_vertical_Callback(hObject, eventdata, handles)
696
697 global setpoint_v;
698
699 setpoint_v = str2double(get(handles.edit_vertical,'String'));
700
701 if setpoint_v < 0
702     setpoint_v = 0;
703     set(handles.edit_vertical,'String','0');
704 end
705
706 if setpoint_v > 65
707     setpoint_v = 65;
708     set(handles.edit_vertical,'String','65');
709 end
710
711
712
713 % --- Executes during object creation, after setting all properties.
714 function edit_vertical_CreateFcn(hObject, eventdata, handles)
715
716 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor
        '))
717     set(hObject,'BackgroundColor','white');
718 end
719
720
721 % --- Executes on button press in radio_horizontal.
722 function radio_horizontal_Callback(hObject, eventdata, handles)
723
724 set(handles.radio_horizontal,'Value',1);
725 set(handles.radio_vertical,'Value',0);
726 set(handles.edit_horizontal,'Enable','on');
727 set(handles.edit_vertical,'Enable','off');
728 set(handles.radio_vai_e_volta,'Value',0);
729 set(handles.edit_vai_e_volta,'Enable','off');
730 set(handles.radio_busca_carga,'Value',0);
731 set(handles.edit_busca_carga,'Enable','off');
732 set(handles.edit_horizontal,'String','');
733 set(handles.edit_vertical,'String','');
734 set(handles.edit_vai_e_volta,'String','');
735 set(handles.edit_busca_carga,'String','');
736
737
738 % --- Executes on button press in radio_vertical.
739 function radio_vertical_Callback(hObject, eventdata, handles)
740
741 set(handles.radio_vertical,'Value',1);
```

```
742 set(handles.radio_horizontal,'Value',0);
743 set(handles.edit_vertical,'Enable','on');
744 set(handles.edit_horizontal,'Enable','off');
745 set(handles.radio_vai_e_volta,'Value',0);
746 set(handles.edit_vai_e_volta,'Enable','off');
747 set(handles.radio_busca_carga,'Value',0);
748 set(handles.edit_busca_carga,'Enable','off');
749 set(handles.edit_horizontal,'String','');
750 set(handles.edit_vertical,'String','');
751 set(handles.edit_vai_e_volta,'String','');
752 set(handles.edit_busca_carga,'String','');
753
754
755 % --- Executa ao pressionar o botao de modo de controle 'Manual'.
756 function radio_manual_Callback(hObject, eventdata, handles)
757
758 %Desabilita todos os botoes de modo automatico e habilita os botoes manuais
759 set(handles.radio_manual,'Value',1);
760 set(handles.radio_automatico,'Value',0);
761 set(handles.radio_vertical,'Value',0);
762 set(handles.radio_horizontal,'Value',0);
763 set(handles.radio_vai_e_volta,'Value',0);
764 set(handles.radio_busca_carga,'Value',0);
765 set(handles.radio_horizontal,'Enable','off');
766 set(handles.radio_vertical,'Enable','off');
767 set(handles.radio_vai_e_volta,'Enable','off');
768 set(handles.radio_busca_carga,'Enable','off');
769 set(handles.edit_vertical,'Enable','off');
770 set(handles.edit_horizontal,'Enable','off');
771 set(handles.edit_vai_e_volta,'Enable','off');
772 set(handles.edit_busca_carga,'Enable','off');
773 set(handles.edit_horizontal,'String','');
774 set(handles.edit_vertical,'String','');
775 set(handles.edit_vai_e_volta,'String','');
776 set(handles.edit_busca_carga,'String','');
777 set(handles.edit_nome_arquivo,'String','');
778 set(handles.edit_nome_arquivo,'Enable','off');
779 set(handles.toggle_stop,'Enable','off');
780 set(handles.toggle_stop,'Value',0);
781 set(handles.toggle_start,'Enable','off');
782 set(handles.toggle_start,'Value',0);
783 set(handles.toggle_desce,'Enable','on');
784 set(handles.toggle_desce,'Value',0);
785 set(handles.toggle_sobe,'Enable','on');
786 set(handles.toggle_sobe,'Value',0);
787 set(handles.toggle_direita,'Enable','on');
788 set(handles.toggle_direita,'Value',0);
789 set(handles.toggle_esquerda,'Enable','on');
790 set(handles.toggle_esquerda,'Value',0);
791
792
793 % --- Executa ao pressionar o botao de modo de controle 'Automatico'.
794 function radio_automatico_Callback(hObject, eventdata, handles)
795
796 %Desabilita todos os botoes de modo manual e habilita os de modo automatico
797 set(handles.radio_manual,'Value',0);
798 set(handles.radio_automatico,'Value',1);
799 set(handles.radio_vertical,'Value',0);
800 set(handles.radio_horizontal,'Value',1);
801 set(handles.radio_vai_e_volta,'Value',0);
```

```
802 set(handles.radio_busca_carga,'Value',0);
803 set(handles.radio_horizontal,'Enable','on');
804 set(handles.radio_vertical,'Enable','on');
805 set(handles.radio_vai_e_volta,'Enable','on');
806 set(handles.radio_busca_carga,'Enable','on');
807 set(handles.edit_horizontal,'Enable','on');
808 set(handles.edit_nome_arquivo,'Enable','on');
809 set(handles.toggle_stop,'Enable','off');
810 set(handles.toggle_stop,'Value',0);
811 set(handles.toggle_start,'Enable','on');
812 set(handles.toggle_start,'Value',0);
813 set(handles.toggle_desce,'Enable','off');
814 set(handles.toggle_desce,'Value',0);
815 set(handles.toggle_sobe,'Enable','off');
816 set(handles.toggle_sobe,'Value',0);
817 set(handles.toggle_direita,'Enable','off');
818 set(handles.toggle_direita,'Value',0);
819 set(handles.toggle_esquerda,'Enable','off');
820 set(handles.toggle_esquerda,'Value',0);
821
822
823 % --- Executes on button press in radio_vai_e_volta.
824 function radio_vai_e_volta_Callback(hObject, eventdata, handles)
825
826 set(handles.radio_vai_e_volta,'Value',1);
827 set(handles.edit_vai_e_volta,'Enable','on');
828 set(handles.radio_vertical,'Value',0);
829 set(handles.radio_horizontal,'Value',0);
830 set(handles.edit_vertical,'Enable','off');
831 set(handles.edit_vertical,'String','');
832 set(handles.edit_horizontal,'Enable','off');
833 set(handles.edit_horizontal,'String','');
834 set(handles.radio_busca_carga,'Value',0);
835 set(handles.edit_busca_carga,'Enable','off');
836 set(handles.edit_busca_carga,'String','');
837
838
839 function edit_vai_e_volta_Callback(hObject, eventdata, handles)
840
841
842
843 % --- Executes during object creation, after setting all properties.
844 function edit_vai_e_volta_CreateFcn(hObject, eventdata, handles)
845
846 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor
        '))
847     set(hObject,'BackgroundColor','white');
848 end
849
850
851 % --- Executes on button press in radio_busca_carga.
852 function radio_busca_carga_Callback(hObject, eventdata, handles)
853
854 set(handles.radio_vai_e_volta,'Value',0);
855 set(handles.edit_vai_e_volta,'Enable','off');
856 set(handles.radio_vertical,'Value',0);
857 set(handles.radio_horizontal,'Value',0);
858 set(handles.edit_vertical,'Enable','off');
859 set(handles.edit_horizontal,'Enable','off');
860 set(handles.radio_busca_carga,'Value',1);
```

```
861 set(handles.edit_busca_carga,'Enable','on');
862 set(handles.edit_horizontal,'String','');
863 set(handles.edit_vertical,'String','');
864 set(handles.edit_vai_e_volta,'String','');
865 set(handles.edit_busca_carga,'String','');
866
867
868 function edit_busca_carga_Callback(hObject, eventdata, handles)
869
870
871
872 % --- Executes during object creation, after setting all properties.
873 function edit_busca_carga_CreateFcn(hObject, eventdata, handles)
874
875 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor
        '))
876     set(hObject,'BackgroundColor','white');
877 end
878
879
880 % --- Executes on button press in radio_velo_lenta.
881 function radio_velo_lenta_Callback(hObject, eventdata, handles)
882
883 set(handles.radio_velo_lenta,'Value',1);
884 set(handles.radio_velo_moderada,'Value',0);
885 set(handles.radio_velo_rapida,'Value',0);
886
887
888 % --- Executes on button press in radio_velo_moderada.
889 function radio_velo_moderada_Callback(hObject, eventdata, handles)
890
891 set(handles.radio_velo_lenta,'Value',0);
892 set(handles.radio_velo_moderada,'Value',1);
893 set(handles.radio_velo_rapida,'Value',0);
894
895
896 % --- Executes on button press in radio_velo_rapida.
897 function radio_velo_rapida_Callback(hObject, eventdata, handles)
898
899 set(handles.radio_velo_lenta,'Value',0);
900 set(handles.radio_velo_moderada,'Value',0);
901 set(handles.radio_velo_rapida,'Value',1);
902
903
904
905 function edit_nome_arquivo_Callback(hObject, eventdata, handles)
906 % hObject    handle to edit_nome_arquivo (see GCBO)
907 % eventdata  reserved - to be defined in a future version of MATLAB
908 % handles    structure with handles and user data (see GUIDATA)
909
910 % Hints: get(hObject,'String') returns contents of edit_nome_arquivo as text
911 %        str2double(get(hObject,'String')) returns contents of edit_nome_arquivo as a
        double
912
913
914 % --- Executes during object creation, after setting all properties.
915 function edit_nome_arquivo_CreateFcn(hObject, eventdata, handles)
916 % hObject    handle to edit_nome_arquivo (see GCBO)
917 % eventdata  reserved - to be defined in a future version of MATLAB
918 % handles    empty - handles not created until after all CreateFcns called
```

```
919
920 % Hint: edit controls usually have a white background on Windows.
921 %       See ISPC and COMPUTER.
922 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor
        '))
923     set(hObject,'BackgroundColor','white');
924 end
```

APÊNDICE B – ALGORITMO DE CONTROLE DO ARDUINO

```
/*********************     inicializacao     ********************/
#include <Wire.h>   // Inclui a biblioteca I2C
#include <Kalman.h> // Inclui a biblioteca para o Filtro Kalman

// Pinagem

#define encoderhPinA  2  //definicao dos pinos para o Encoder Horizontal
#define encoderhPinB  4

#define encodervPinA  3  //definicao dos pinos para o Encoder Vertical
#define encodervPinB  5

typedef union {          //definicao do tipo para facilitar a transmissao de variavel
    float via serial
 float floatingPoint;
 byte binary[4];
} binaryFloat;

//=================== variaveis ACIONAMENTO DOS MOTORES

int in_IN1 = 43;       // saida IN1 da ponte H L298N (controle horizontal) - FIO MARROM
int in_IN2 = 42;       // saida IN2 da ponte H L298 - FIO VERMELHO
int in_enA = 44;       // Pino do controle PWM da ponte H L298N

int in_A = 45;        // saida in_A PWM da ponte H do motor vertical
int in_B = 46;        // saida in_B PWM da ponte H do motor vertical


int CHAVE_H_ESQ = 38; //Chaves fim de curso
int CHAVE_H_DIR = 39;
int CHAVE_V_1 = 40;
int CHAVE_V_2 = 41;


int vhdireita = 0; //variaveis de duty cycle PWM (0 = 0%, 255 = 100%)
int vvdireita = 0;
int vhesquerda = 0;
int vvesquerda = 0;
int vvdescendo = 0;
int vvsubindo = 0;

int vhesquerda_lenta = 155;
int vvesquerda_lenta = 130;
int vhdireita_lenta = 95;
int vvdireita_lenta = 155;
int vvdescendo_lenta = 130;
int vvsubindo_lenta = 160;

int vhesquerda_moderada = 175;
int vvesquerda_moderada = 170;
int vhdireita_moderada = 110;
int vvdireita_moderada = 185;
int vvdescendo_moderada = 180;
int vvsubindo_moderada = 190;

```

```
55 int vhesquerda_rapida = 210;
56 int vvesquerda_rapida = 224;
57 int vhdireita_rapida = 135;
58 int vvdireita_rapida = 224;
59 int vvdescendo_rapida = 251;
60 int vvsubindo_rapida = 251;
61
62 int flag_horizontal = 1; // Flags utilizadas para garantir a frequencia do sinal pwm
       aplicado aos motores
63 int flag_v_mais = 1;
64 int flag_v_menos = 1;
65
66
67 //====================== variaveis LEITURA DE posicao
68 volatile float counth = 0;  //posicao atual horizontal (pulsos)
69 volatile float countv = 0;  //posicao atual vertical (pulsos)
70
71 float setpoint_h = 0.0;   // Coordenada desejada de posicao horizontal em [cm] a partir
       do referencial
72 float setpoint_v = 0.0;   // Coordenada desejada de posicao vertical em [cm] a partir do
       referencial
73
74 volatile float dh = 0.0;   // Coordenada atual de posicao em [cm] do trolley (horizontal)
        a partir do referencial
75 volatile float dv = 0.0;   // Coordenada atual de posicao em [cm] da carga (vertical) +
       deslocamento horizontal a partir do referencial
76
77 //definicao das variaveis float que serao enviadas para o matlab via serial
78 binaryFloat posicao_h;      //Coordenada de posicao horizontal
79 binaryFloat posicao_v;      //Coordenada de posicao vertical
80 binaryFloat acabou;         // sinalizacao de termino de movimentacao da carga
81 binaryFloat falha_i2cAddress;        // sinalizacao de erro de leitura de Angulo via I2C
82 binaryFloat falha_i2cRead;        // sinalizacao de erro de leitura de Angulo via I2C
83 binaryFloat falha_i2cWrite;        // sinalizacao de erro de leitura de Angulo via I2C
84 binaryFloat falha_i2cTimeout;        // sinalizacao de erro de leitura de Angulo via I2C
85 binaryFloat erro_medicao;
86 binaryFloat inicio;         // sinalizacao de inicio de comunicacao entre arduino e matlab
87
88 // Constantes
89 volatile float kh = 100.0/1440.0;  // Resolucao do encoder horizontal (cm/pulso)
90 volatile float kv = 100.0/1065.0;  // Resolucao do encoder vertical (cm/pulso)
91
92
93 //====================== variaveis LEITURA DE Angulo
94 Kalman kalmanZ; // criacao da variavel do tipo Kalman para aplicacao do filtro na leitura
        de Angulo
95
96 //====================== DADOS COLETADOS DO SENSOR acelerometro/giroscopio
97 double acel_X, acel_Y, acel_Z;       //variaveis de leitura dos dados gerados pelo
       acelerometro
98 double giro_Z;   //variavel de leitura dos dados gerados pelo giroscopio
99 double giro_Z_cal = 0;
100
101 binaryFloat angulo_kalman_Z;
102
103 uint32_t timer;       //Guarda o tempo para calculo de velocidade angular medida pelo
       giroscopio
104 uint8_t i2cData[14]; // Buffer para aquisicao dos dados via I2C
105
106 //====================== variaveis LEITURA DE ENTRADA SERIAL
```

```
107 const byte tamanho_string = 64;          // Tamanho do vetor de caracteres a serem lidos
108 char string_recebido[tamanho_string];     // Vetor que guarda os caracteres recebidos
109 char string_temp[tamanho_string];         // Vetor temporario para separacao dos
        caracteres
110 char comando[tamanho_string] = {0};      // Vetor que guarda os caracteres de comando (
        start ou stop)
111 char modo[tamanho_string] = {0};         // Vetor que guarda os caracteres de modo (
        automatico ou manual)
112 char direcao[tamanho_string] = {0};      // Vetor que guarda os caracteres de direcao (
        horizontal ou vertical)
113 char velocidade[tamanho_string] = {0};    // Vetor que guarda os caracteres de velocidade
        (lenta, moderada ou rapida)
114 float posicao_final = 0.0;         // Variavel que guarda a posicao final em [cm] a ser
        alcancada
115
116 boolean dado_novo = false;          // Variavel auxiliar para reconhecer se terminou de
        receber o novo comando
117
118 //==================== variaveis I2C
119 const uint8_t IMUAddress = 0x68; // endereco do MPU 6050
120 const uint16_t I2C_TIMEOUT = 1000; // Tempo de timeout para checar erros na comunicacao
        I2C
121
122 //==================== FIM DA inicializacao DE variaveis
123
124 //==================== SETUP
125 void setup() {
126   Serial.begin(115200);                          //Inicializa porta serial com 115.200
        bits/s (baud)
127   Wire.begin();                                  //Inicializa biblioteca de comunicacao
        I2C
128
129 #if ARDUINO >= 157                               //Rotina para definir frequencia a I2C
        para 400 kHz
130   Wire.setClock(400000UL);
131 #else
132   TWBR = ((F_CPU / 400000UL) - 16) / 2;
133 #endif
134
135   acabou.floatingPoint = 10000.0;                //Define o valor para indicar o termino
        da realizacao da tarefa
136   falha_i2cAddress.floatingPoint = 11000.0;              //Define o valor para indicar
        erro na comunicacao I2C
137   falha_i2cRead.floatingPoint = 12000.0;               //Define o valor para indicar erro
         na comunicacao I2C
138   falha_i2cWrite.floatingPoint = 13000.0;               //Define o valor para indicar
        erro na comunicacao I2C
139   falha_i2cTimeout.floatingPoint = 14000.0;              //Define o valor para indicar
        erro na comunicacao I2C
140   erro_medicao.floatingPoint = 15000.0;
141   inicio.floatingPoint = 9999.0;                 //Define o Valor para indicar o inicio da
         comunicacao entre Arduino e Matlab
142
143   //inicializacao de pinos do Arduino
144   pinMode(in_IN1, OUTPUT);                       //definicao dos pinos do Arduino como
        Entrada (INPUT) ou saida (OUTPUT)
145   pinMode(in_IN2, OUTPUT);
146   pinMode(encoderhPinA, INPUT);
147   pinMode(encoderhPinB, INPUT);
148   pinMode(encodervPinA, INPUT);
```

```
149   pinMode(encodervPinB, INPUT);
150   pinMode(CHAVE_H_ESQ, INPUT);
151   pinMode(CHAVE_H_DIR, INPUT);
152   pinMode(CHAVE_V_1, INPUT);
153   pinMode(CHAVE_V_2, INPUT);
154   attachInterrupt(0, ContaDistanciah, RISING);  //indicacao de geracao de interrupcao no
        pino 2 do Arduino para o encoder horizontal
155   attachInterrupt(1, ContaDistanciav, RISING);  //indicacao de geracao de interrupcao no
        pino 3 do Arduino para o encoder vertical
156
157   //inicializacao PWM
158   analogWrite(in_enA, 0);                       //Inicializa o sinal PWM em 0% para a
        ponte H de controle horizontal
159   digitalWrite(in_IN1, LOW);                    //Inicializa o pino IN1 da ponte H de
        controle horizontal (L298N) em LOW
160   digitalWrite(in_IN2, LOW);                    //Inicializa o pino IN2 da ponte H de
        controle horizontal em LOW
161   analogWrite(in_A, 0);                         //Inicializa o pino A da ponte H de
        controle vertical em 0%
162   analogWrite(in_B, 0);                         //Inicializa o pino B da ponte H de
        controle vertical em 0%
163
164   //inicializacao do Sensor acelerometro/giroscopio
165   i2cData[0] = 7; // Define a taxa de amostragem em 1000Hz. Via datasheet do MPU 6050 ->
        8kHz/(7+1) = 1000Hz
166   i2cData[1] = 0x00; // Define taxa de filtragem de dados do acelerometro em 260 Hz, do
        giroscopio em 256 Hz, com taxa de amostragem padrao de 8 KHz
167   i2cData[2] = 0x00; // Define a escala de leitura do giroscopio em +/- 250 graus/s
168   i2cData[3] = 0x00; // Deine a escala de leitura do acelerometro em +/-2g
169   while (i2cWrite(0x19, i2cData, 4, false)); // Escreve nos 4 devidos registradores (a
        partir do 0x19) os dados acima para aplicar as configuracoes descritas
170   while (i2cWrite(0x6B, 0x01, true)); // Inicializa o sensor, desabilitando o modo Sleep
        e utilizando o clock do giroscopio do eixo X (recomendado pelo datasheet)
171
172   while (i2cRead(0x75, i2cData, 1)); //Testa o registrador que guarda o endereco do MPU
        6050 para garantir uma correta comunicacao
173   if (i2cData[0] != 0x68) { // Le o registrador "WHO_AM_I", caso o endereco do MPU 6050
        seja diferente de 0x68, envia sinal de erro para o Matlab e trava o programa
174     para_motores();
175     Serial.write(falha_i2cAddress.binary,4);
176     while (1);
177   }
178   delay(100); // Delay de 100ms para estabilizacao dos sensores
179   //Leitura inicial para definicao dos angulos iniciais lidos pelo acelerometro e
        giroscopio
180   while (i2cRead(0x3B, i2cData, 6)); //Recolhe as leituras armazenadas nos 6
        registradores do acelerometro, onde cada par guarda a leitura de cada um de seus 3
        eixos
181   acel_X = (int16_t)((i2cData[0] << 8) | i2cData[1]); //Combina o primeiro par de
        registradores para formar a leitura 'crua' do valor de aceleracao no eixo X
182   acel_Y = (int16_t)((i2cData[2] << 8) | i2cData[3]); //Combina o segundo par de
        registradores para formar a leitura 'crua' do valor de aceleracao no eixo Y
183   acel_Z = (int16_t)((i2cData[4] << 8) | i2cData[5]); //Combina o terceiro par de
        registradores para formar a leitura 'crua' do valor de aceleracao no eixo Z
184
185   double rotacao = atan(acel_X / sqrt(acel_Y * acel_Y + acel_Z * acel_Z)) * RAD_TO_DEG;
        //Calcula a rotacao em graus no eixo Z (rotacao) a partir dos valores lidos
        anteriormente
186   delay(3);
187
```

```
188   for (int cal_int = 0; cal_int < 1000 ; cal_int ++){                  //Le os valores do
        giroscopio do MPU-6050 1000 vezes para calibracao
189     while (i2cRead(0x47, i2cData, 2));    //Le os 2 registradores que guardam o valor da
      velocidade angular do eixo Z
190     giro_Z = (int16_t)((i2cData[0] << 8) | i2cData[1]); //Trata e armazena o valor 'cru'
      da velocidade angular no eixo Z lido pelo giroscopio
191     giro_Z_cal += giro_Z;                                                //Soma o offset do
        eixo Z do giroscopio a variavel de calibracao
192     delay(3);
193   }
194   giro_Z_cal /= 1000; //Faz a media para calibracao
195   kalmanZ.define_angulo(rotacao);// Define o angulo inicial no eixo Z para a aplicacao do
        filtro Kalman
196   estabelece_contato();  //Envia um byte para estabelecer a comunicacao com o Matlab e
      aguarda resposta
197   timer = micros();     //Inicializa a contagem de tempo para o calculo de Variacao
      angular
198 }
199
200 //====================== LOOP
201 void loop() {
202
203   le_serial();  //Le a porta serial para verificar se houve algum comando dado pelo
      Matlab, armazenando os comandos em suas respectivas variaveis
204
205   if(strcmp(comando,"start") == 0){ //Verifica se o comando dado pelo Matlab foi de '
      start'
206     if(strcmp(modo,"automatico") == 0){ //Se sim, verifica se o modo dado pelo Matlab foi
        'automatico'
207       if (strcmp(direcao,"horizontal") == 0)   //Se sim, verifica se a direcao dada pelo
      Matlab foi 'horizontal'
208       {
209         setpoint_h = posicao_final; //Guarda o valor da posicao final dada pelo matlab em
        'setpoint_h'
210
211         if(setpoint_h > 100){ setpoint_h = 100; } //Limita posicao final maxima permitida
        horizontal em 100 cm
212
213         if (dv > dh){ sobe_ref(); }        //Move a carga para o nivel de referencia antes
        de se mover horizontalmente
214         else if(dv < dh){ desce_ref(); }  //Move a carga para o nivel de referencia antes
        de se mover horizontalmente
215
216         if(setpoint_h >= dh){ move_esquerda(); } //Se a posicao final for maior que a
      posicao atual, define sentido para se mover para a esquerda
217         else { move_direita(); }                 //Se a posicao final for menor que a
      posicao atual, define sentido para se mover para a direita
218       }
219       else if(strcmp(direcao,"vertical") == 0){ //Se a direcao nao foi 'horizontal',
      verifica se a direcao dada pelo Matlab foi 'vertical'
220
221         if(posicao_final > 65){ posicao_final = 65; } //Limita posicao final maxima
      permitida vertical em 65 cm
222
223         setpoint_v = posicao_final + dh; //Guarda o valor da posicao final dada pelo
      matlab em 'setpoint_v'
224
225         if(setpoint_v >= dv){ desce(); } //Se a posicao final for maior que a posicao
      atual, define sentido para se mover para baixo
```

```
226           else { sobe(); }                //Se a posicao final for menor que a posicao
      atual, define sentido para se mover para cima
227         }
228         else if(strcmp(direcao,"vaievolta") == 0){//Se a direcao nao foi nenhuma das acima,
       verifica se a direcao dada pelo Matlab foi 'vai e volta'
229           setpoint_h = posicao_final; //Guarda o valor da posicao final dada pelo matlab em
       'setpoint_h'
230           if(setpoint_h > 100){ setpoint_h = 100; } //Limita posicao final maxima permitida
       horizontal em 100 cm
231           move_vaievolta(); //Chama a funcao que realiza o movimento 'vai e volta'
232         }
233         else if(strcmp(direcao,"buscacarga") == 0){//Se a direcao nao foi nenhuma das acima
      , verifica se a direcao dada pelo Matlab foi 'busca carga'
234           setpoint_h = posicao_final; //Guarda o valor da posicao final dada pelo matlab em
       'setpoint_h'
235           if(setpoint_h > 100){ setpoint_h = 100; } //Limita posicao final maxima permitida
       horizontal em 100 cm
236           move_buscacarga(); //Chama a funcao que realiza o movimento 'busca carga'
237         }
238         else{                                 //Se a direcao nao foi nenhuma das acima,
      para todos os motores
239           para_motores();
240         }
241       }
242       else if(strcmp(modo,"manual") == 0){  //Se o modo nao foi 'automatico', verifica se o
       modo dado pelo Matlab foi 'manual'
243         if (strcmp(direcao,"esquerda") == 0){ move_esquerda(); }       //Se a direcao dada
      for 'esquerda', move a carga para esquerda no modo manual
244         else if (strcmp(direcao,"direita") == 0) { move_direita(); }  //Se a direcao dada
      for 'direita', move a carga para direita no modo manual
245         else if (strcmp(direcao,"desce") == 0){ desce(); }             //Se a direcao dada
      for 'desce', move a carga para baixo no modo manual
246         else if (strcmp(direcao,"sobe") == 0){ sobe(); }               //Se a direcao dada
      for 'sobe', move a carga para cima no modo manual
247         else{                                                          //Se nao for nenhuma
      das direcoes acima, para todos os motores
248           para_motores();
249         }
250       }
251     }
252     else if(strcmp(comando,"zera") == 0){ //Se o comando dado pelo Matlab for 'zera', zera
       as coordenadas horizontal e vertical
253         counth = 0;
254         countv = 0;
255         dh = 0.0;
256         dv = 0.0;
257         para_motores();
258         Serial.write(acabou.binary,4); //Envia flag de indicacao de fim de realizacao de
      tarefa
259       }
260     else{                                 //Se o comando dado pelo Matlab nao for 'start'
       nem 'zera', para todos os motores
261       para_motores();
262     }
263
264     delay(100);
265     estabelece_contato();  //Aguarda novo comando do Matlab
266   }
267   //=====================
268   void para_motores(){ //Funcao que zera o pwm desabilitando todos os motores
```

```
269     analogWrite(in_enA, 0);
270     digitalWrite(in_IN2, LOW);
271     digitalWrite(in_IN1, LOW);
272     analogWrite(in_A, 0);
273     analogWrite(in_B, 0);
274     flag_horizontal = 1;
275     flag_v_mais = 1;
276     flag_v_menos = 1;
277   }
278   //=======================
279   void estabelece_contato() {      //funcao de estabelecimento de comunicacao com o Matlab
280     while (Serial.available() <= 0) {  //Enquanto o Matlab nao envia comando, permanece
        neste loop
281       Serial.write(inicio.binary,4);    //Envia o byte de inicio para o Matlab e aguarda
        resposta com o comando
282       delay(300);                       //Delay de 300ms entre envios
283     }
284   }
285   //=======================
286   void le_serial(){             //funcao de leitura de comandos enviados pelo Matlab
287       recebe_string();  //Chama funcao que recolhe os caracteres recebidos
288       if (dado_novo == true)         //Se recebeu um comando novo
289       {
290         strcpy(string_temp, string_recebido);    //Faz copia de seguranca dos caracteres
        recebidos, ja que a funcao strtok usada em separa_palavras() substitui as virgulas
        por \0
291         separa_palavras();                         //Guarda cada fracao dos caracteres
        recebidos em suas respectivas variaveis pertinentes para a realizacao dos comandos
292         dado_novo = false;                      //Reinicia flag de indicacao de recebimento
        de comando novo
293       }
294   }
295   //=======================
296   void recebe_string() { //funcao de recebimento e separacao do string de caracteres
        recebidos do Matlab, na forma <comando,modo,direcao,velocidade,posicao_final>
297       static boolean esta_recebendo = false; //Variavel de indicacao de recebimento
298       static byte indice = 0;                    //Variavel de indice de vetor
299       char marcador_inicio = '<';                 //Variavel de marcacao de inicio de
        caracteres a serem guardados
300       char marcador_fim = '>';                    //Variavel de marcacao de fim de caracteres
         a serem guardados
301       char char_recebido;                                 //Variavel que guarda cada caracter
         recebido via serial
302
303       while (Serial.available() > 0 && dado_novo == false) //mantem o loop enquanto dados
        estiverem sendo recebidos e o string recebido nao acabou
304       {
305         char_recebido = Serial.read();                    //Le o caracter recebido
306
307         if (esta_recebendo == true)          //Se ja reconheceu o caracter de marcacao de
        inicio de caracteres a serem recebidos ('<'), comeca o armazenamento
308         {
309           if (char_recebido != marcador_fim)               //Se o caracter recebido nao for
         o de marcacao de fim de caracteres a serem recebidos ('>'),
310           {
311             string_recebido[indice] = char_recebido;         //Guarda o caracter no vetor
312             indice++;                                 //Incrementa o indice do vetor de
        caracteres recebidos
313             if (indice >= tamanho_string)             //Se o indice se tornar maior que o
        tamanho do vetor,
```

```
314           {
315             indice = tamanho_string - 1;            //Decrementa o indice
316           }
317         }
318         else                                  //Se o caracter recebido for o de marcacao de
    fim de caracteres a serem recebidos ('>'),
319         {
320           string_recebido[indice] = '\0';       //Finaliza o vetor de caracteres
    recebidos
321           esta_recebendo = false;           //Define a flag indicando que o processo
    recebimento de caracteres foi finalizado
322           indice = 0;                           //Zera o indice
323           dado_novo = true;                    //Define a flag indicando que foi recebido
    um novo comando
324         }
325       }
326       else if (char_recebido == marcador_inicio) {        //Se o caracter recebido for o
    de marcacao de inicio de caracteres a serem recebidos ('<'),
327         esta_recebendo = true;              //Define a flag indicando que o recebimento de
     caracteres esta em andamento
328       }
329     }
330 }
331 //======================
332 void separa_palavras() {                        //funcao que separa os dados recebidos do
    Matlab via serial para armazenamento em suas respectivas variaveis
333
334     char * indice_strtok;                     //Ponteiro para indicacao de posicao a ser
    utilizada pela funcao strtok()
335
336     indice_strtok = strtok(string_temp,",");  // Pega a primeira parte do string de
    caracteres recebidos (ate a virgula), correspondente ao comando dado pelo Matlab
337     strcpy(comando, indice_strtok);           // Copia o string de comando para a variavel
    comando
338
339     indice_strtok = strtok(NULL, ",");        // Pega a partir de onde o anterior parou, e
    guarda a segunda parte do string, correspondente ao modo dado pelo Matlab
340     strcpy(modo, indice_strtok);              // Copia o string de modo para a variavel
    modo
341
342     indice_strtok = strtok(NULL, ",");        // Pega a partir de onde o anterior parou, e
    guarda a terceira parte do string, correspondente a direcao dada pelo Matlab
343     strcpy(direcao, indice_strtok);           // Copia o string de direcao para a variavel
    direcao
344
345     indice_strtok = strtok(NULL, ",");        // Pega a partir de onde o anterior parou, e
    guarda a quarta parte do string, correspondente a velocidade dada pelo Matlab
346     strcpy(velocidade, indice_strtok);          // Copia o string de velocidade para a
    variavel velocidade
347
348     indice_strtok = strtok(NULL, ",");        // Pega a partir de onde o anterior parou, e
    guarda a quinta parte do string, correspondente a posicao final dada pelo Matlab
349     posicao_final = atof(indice_strtok);      // Converte o string recebido para float e
    guarda na variavel posicao_final
350 }
351 //======================
352 void calcula_angulo(){                    //funcao que calcula o Angulo medido pelo sensor
    MPU 6050
353
```

```
354   while (i2cRead(0x3B, i2cData, 14));   //Le os 14 registradores que guardam os valores das aceleracoes e velocidades angulares em cada um dos 3 eixos, alem da temperatura
355   acel_X = (int16_t)((i2cData[0] << 8) | i2cData[1]); //Trata e armazena o valor 'cru' da aceleracao no eixo X lido pelo acelerometro
356   acel_Y = (int16_t)((i2cData[2] << 8) | i2cData[3]); //Trata e armazena o valor 'cru' da aceleracao no eixo Y lido pelo acelerometro
357   acel_Z = (int16_t)((i2cData[4] << 8) | i2cData[5]); //Trata e armazena o valor 'cru' da aceleracao no eixo Z lido pelo acelerometro
358   giro_Z = (int16_t)((i2cData[12] << 8) | i2cData[13]); //Trata e armazena o valor 'cru' da velocidade angular no eixo Z lido pelo giroscopio
359
360   double dt = (double)(micros() - timer) / 1000000; // Calcula a variacao de tempo entre leituras para o calculo da variacao angular no tempo entre leituras
361   timer = micros(); //Guarda o tempo atual para posterior calculo de delta t
362
363   double rotacao = atan(-acel_X / sqrt(acel_Y * acel_Y + acel_Z * acel_Z)) * RAD_TO_DEG; //Calcula a rotacao em graus no eixo Z (rotacao) a partir dos valores lidos anteriormente
364
365   double rate_giro_Z = (giro_Z - giro_Z_cal) / 131.0; // Converte o valor 'cru' da velocidade angular no eixo Z lido pelo giroscopio para graus/s
366
367   angulo_kalman_Z.floatingPoint = kalmanZ.filtra_angulo(rotacao, rate_giro_Z, dt); // Calcula o Angulo no eixo Z filtrado utilizando o filtro Kalman
368 }
369 //=======================
370 void ContaDistanciah()    //funcao de contagem de distancia percorrida, chamada na interrupcao gerada pelo encoder horizontal
371 {
372 if (digitalRead(encoderhPinA) == digitalRead(encoderhPinB)) //Se os niveis lidos pelos canais A e B do encoder horizontal forem HIGH, incrementa contagem de distancia (esquerda)
373 {counth++;}
374 else { counth--; }                                          //Se nao forem iguais, significa que o sentido de rotacao e outro (direita), entao decrementa contagem de distancia
375 dh = (counth*kh);                                          //Calcula a posicao atual em centimetros, a partir da resolucao horizontal em cm/pulsos
376 }
377 //=======================
378 void ContaDistanciav()    //funcao de contagem de distancia percorrida, chamada na interrupcao gerada pelo encoder vertical
379 {
380 if (digitalRead(encodervPinA) == digitalRead(encodervPinB)) //Se os niveis lidos pelos canais A e B do encoder vertical forem HIGH, incrementa contagem de distancia (desce)
381 {countv++;}
382 else { countv--; }                                          //Se nao forem iguais, significa que o sentido de rotacao e outro (subindo), entao decrementa contagem de distancia
383 dv = (countv*kv);                                          //Calcula a posicao atual em centimetros, a partir da resolucao vertical em cm/pulsos
384 }
385 //=======================
386 void atualiza_saida(){
387   calcula_angulo();                          //Chama funcao para atualizar a leitura do Angulo de carga
388   posicao_h.floatingPoint = dh;              //Coleta a posicao horizontal atual para ser enviada em bytes
389   posicao_v.floatingPoint = dv - dh;         //Coleta a posicao vertical atual para ser enviada em bytes
```

```
390   if(posicao_h.floatingPoint > acabou.floatingPoint || posicao_v.floatingPoint > acabou.
        floatingPoint || angulo_kalman_Z.floatingPoint > acabou.floatingPoint){ //Verifica se
         houve erro
391     para_motores();
392     Serial.write(erro_medicao.binary,4);            //Envia via serial para o Matlab o
        erro ocorrido
393   }
394   Serial.write(posicao_h.binary,4);           //Envia via serial para o Matlab a posicao
        horizontal atual
395   delay(3);
396   Serial.write(posicao_v.binary,4);           //Envia via serial para o Matlab a posicao
        vertical atual
397   delay(3);
398   Serial.write(angulo_kalman_Z.binary,4);           //Envia via serial para o Matlab o
        Angulo de carga atual
399   delay(3);
400 }
401 //======================
402 void pausa_movimento(){ //Funcao de pausa de movimento para melhor analise dos dados
      antes e depois do inicio da movimentacao
403   for (int pausa = 0; pausa<100; pausa++){
404     atualiza_saida();
405   }
406 }
407 //======================
408 void sobe_ref(){          //Funcao de subida da carga ate o nivel de referencia, antes de
       iniciar movimento horizontal
409
410   if(strcmp(velocidade,"lenta") == 0){ //Determina a velocidade a partir do comando dado
        pelo usuario
411     vvsubindo = vvsubindo_lenta;
412   }
413   else if(strcmp(velocidade,"moderada") == 0){
414     vvsubindo = vvsubindo_moderada;
415   }
416   else if(strcmp(velocidade,"rapida") == 0){
417     vvsubindo = vvsubindo_rapida;
418   }
419   while (dv > dh){                      //Enquanto a carga nao estiver no nivel vertical
        de referencia,
420     if(digitalRead(CHAVE_V_1) == LOW && digitalRead(CHAVE_V_2) == LOW){ //Verifica se as
        chaves fim de curso nao estao acionadas
421       if((dv - dh) < 2){                //Se a diferenca entre a posicao atual e a
        posicao final for menor que 2 cm,
422         vvsubindo = vvsubindo_lenta;                          //Diminui a rotacao do motor
        vertical, para um posicionamento mais preciso
423         flag_v_menos = 1;
424       }
425       if(flag_v_menos == 1){
426         analogWrite(in_A, vvsubindo);       //Aciona PWM no motor vertical para subir
427         analogWrite(in_B, 0);
428       }
429       flag_v_menos = 0;
430       le_serial();                         //Faz leitura da porta serial para verificar se
        foi dado algum comando de parada emergencial
431     }
432     else {strcpy(comando,"stop");}       //Se a chave estiver acionada, para os motores
433     if (strcmp(comando,"stop") == 0){  //Se foi dado o comando de parada emergencial '
        stop', para todos os motores e sai do loop
434       para_motores();
```

```
435         break;
436         }
437     }
438     para_motores();               //Se chegou no nivel de referencia vertical, para todos os
          motores para posteriormente iniciar o movimento horizontal
439 }
440 //=======================
441 void desce_ref(){           //funcao de descida da carga ate o nivel de referencia, antes
      de iniciar movimento horizontal
442
443     if(strcmp(velocidade,"lenta") == 0){ //Determina a velocidade a partir do comando dado
        pelo usuario
444       vvdescendo = vvdescendo_lenta;
445     }
446     else if(strcmp(velocidade,"moderada") == 0){
447       vvdescendo = vvdescendo_moderada;
448     }
449     else if(strcmp(velocidade,"rapida") == 0){
450       vvdescendo = vvdescendo_rapida;
451     }
452     while (dv < dh){                      //Enquanto a carga nao estiver no nivel vertical
        de referencia,
453       if((dh - dv) < 2){                 //Se a diferenca entre a posicao atual e a posicao
        final for menor que 2 cm,
454         vvdescendo = vvdescendo_lenta;                        //Diminui a rotacao do motor
        vertical, para um posicionamento mais preciso
455         flag_v_mais = 1;
456       }
457       if(flag_v_mais == 1){
458       analogWrite(in_A, 0);               //Aciona PWM no motor vertical para descer
459       analogWrite(in_B, vvdescendo);
460       }
461       flag_v_mais = 0;
462       le_serial();                        //Faz leitura da porta serial para verificar se
        foi dado algum comando de parada emergencial
463       if (strcmp(comando,"stop") == 0){  //Se foi dado o comando de parada emergencial '
        stop', para todos os motores e sai do loop
464         para_motores();
465         break;
466         }
467     }
468     para_motores();               //Se chegou no nivel de referencia vertical, para todos os
          motores para posteriormente iniciar o movimento horizontal
469 }
470 //=======================
471 void move_esquerda(){      //funcao de movimentacao horizontal da carga para a esquerda
472
473     if(strcmp(velocidade,"lenta") == 0){ //Determina a velocidade a partir do comando dado
        pelo usuario
474       vhesquerda = vhesquerda_lenta;
475       vvesquerda = vvesquerda_lenta;
476     }
477     else if(strcmp(velocidade,"moderada") == 0){
478       vhesquerda = vhesquerda_moderada;
479       vvesquerda = vvesquerda_moderada;
480     }
481     else if(strcmp(velocidade,"rapida") == 0){
482       vhesquerda = vhesquerda_rapida;
483       vvesquerda = vvesquerda_rapida;
484     }
```

```
485   if(strcmp(modo,"automatico") == 0){              //Se o modo recebido do Matlab via
        serial for modo automatico, realiza movimentacao automaticamente para a posicao final
         definida
486     pausa_movimento();
487     while((dh < setpoint_h) || (dv < setpoint_h)) //Enquanto o motor horizontal ou o
        motor vertical nao tiverem chegado na posicao final definida, continua no loop
488     {
489       if(digitalRead(CHAVE_H_ESQ)==LOW && digitalRead(CHAVE_V_1)==LOW){ //Verifica se as
        chaves fim de curso nao estao acionadas
490         if(dh < setpoint_h)                          //Se a posicao horizontal final ainda
         nao tiver sido atingida, aciona o motor horizontal para rodar para a esquerda
491         {
492           if((setpoint_h - dh) < 2){                 //Se a diferenca entre a posicao
        atual e a posicao final for menor que 2 cm,
493             vhesquerda = vhesquerda_lenta;                        //Diminui a rotacao do
        motor horizontal, para um posicionamento mais preciso
494             flag_horizontal = 1;
495           }
496           if(flag_horizontal == 1){
497           digitalWrite(in_IN2, LOW);                 //Configura para que o motor gire
        para a esquerda
498           digitalWrite(in_IN1, HIGH);
499           analogWrite(in_enA, vhesquerda);           //Aciona o motor horizontal atraves
        de sinal PWM
500           }
501           flag_horizontal = 0;
502         }
503         else                                         //Se a posicao horizontal final for
        atingida,
504         {
505           flag_horizontal = 1;
506           analogWrite(in_enA, 0);                    //Para o motor horizontal
507           digitalWrite(in_IN1, LOW);
508           digitalWrite(in_IN2, LOW);
509         }
510       }
511       else if(digitalRead(CHAVE_H_ESQ)==HIGH){       //Se a chave esquerda estiver
        acionada, para os motores
512         para_motores();
513         break;
514       }
515       else{
516         flag_horizontal = 1;
517         analogWrite(in_enA, 0);                    //Se somente a chave vertical estiver
        acionada, para o motor horizontal
518         digitalWrite(in_IN1, LOW);
519         digitalWrite(in_IN2, LOW);
520       }
521       if(dv < setpoint_h)                          //Se a posicao vertical final ainda nao
         tiver sido atingida, aciona o motor vertical para rodar para baixo para compensacao
        de nivel
522       {
523         if((setpoint_h - dv) < 2){                 //Se a diferenca entre a posicao atual
        e a posicao final for menor que 2 cm,
524           vvesquerda = vvesquerda_lenta;                        //Diminui a rotacao do
        motor vertical, para um posicionamento mais preciso
525           flag_v_mais = 1;
526           flag_v_menos = 1;
527         }
```

```
528         if(dv > dh)                                  //Se a posicao atual vertical for maior
      que a horizontal,
529         {
530           flag_v_mais = 1;
531           if(flag_v_menos == 1){
532           analogWrite(in_A, 0);                      //Diminui a rotacao do motor vertical,
    para tentar manter o nivel vertical da carga constante
533           analogWrite(in_B, (vvesquerda-30));
534           }
535           flag_v_menos = 0;
536         }
537         else                                         //Se a posicao atual vertical for menor
      que a horizontal,
538         {
539           flag_v_menos = 1;
540           if(flag_v_mais == 1){
541           analogWrite(in_A, 0);                      //Aumenta a rotacao do motor vertical,
    para tentar manter o nivel vertical da carga constante
542           analogWrite(in_B, (vvesquerda+30));
543           }
544           flag_v_mais = 0;
545         }
546       }
547       else                                           //Se o nivel vertical estiver na
    referencia,
548       {
549         flag_v_mais = 1;
550         flag_v_menos = 1;
551         analogWrite(in_A, 0);                        //Para o motor vertical
552         analogWrite(in_B, 0);
553       }
554
555       le_serial();                                   //Faz leitura da porta serial para
    verificar se foi dado algum comando de parada emergencial
556       if (strcmp(comando,"stop") == 0){              //Se foi dado o comando de parada
    emergencial 'stop', para todos os motores e sai do loop
557         para_motores();
558         break;
559       }
560       atualiza_saida();                              //Chama funcao para enviar os dados dos
      sensores para o MATLAB
561     }
562   }
563   else if(strcmp(modo,"manual") == 0){               //Se o modo recebido nao foi '
    automatico', verifica se o modo recebido foi para operacao no modo Manual
564     while(strcmp(comando,"stop") != 0)               //Se sim, entra no loop de movimentacao
      para a esquerda ate que receba o comando de parada 'stop'
565     {
566       if(digitalRead(CHAVE_H_ESQ)==LOW && digitalRead(CHAVE_V_1)==LOW){ //Verifica se as
    chaves fim de curso nao estao acionadas
567         if(flag_horizontal == 1){
568         digitalWrite(in_IN2, LOW);                   //Configura o sentido de rotacao do
    motor horizontal para girar para esquerda
569         digitalWrite(in_IN1, HIGH);
570         analogWrite(in_enA, vhesquerda);             //Aciona o motor horizontal com o
    sinal PWM
571         }
572         flag_horizontal = 0;
573       }
```

```
574         else if(digitalRead(CHAVE_H_ESQ)==HIGH){        //Se a chave esquerda estiver
        acionada, para os motores
575           para_motores();
576           break;
577         }
578         else{
579           flag_horizontal = 1;
580           analogWrite(in_enA, 0);                       //Se somente a chave vertical estiver
        acionada, para o motor horizontal
581           digitalWrite(in_IN1, LOW);
582           digitalWrite(in_IN2, LOW);
583         }
584
585         if(dv > dh)                                     //Verifica se a distancia percorrida
        verticalmente esta igual a horizontal, para manter a carga nivelada durante o
        percurso
586         {
587           flag_v_mais = 1;
588           if(flag_v_menos == 1){
589           analogWrite(in_A, 0);                         //Se esta maior, diminui a velocidade
        de rotacao vertical para manter as distancias percorridas iguais nos dois sentidos
590           analogWrite(in_B, (vvesquerda-30));
591           }
592           flag_v_menos = 0;
593         }
594         else                                            //Se esta menor, aumenta a velocidade
        de rotacao vertical para manter as distancias percorridas iguais nos dois sentidos
595         {
596           flag_v_menos = 1;
597           if(flag_v_mais == 1){
598           analogWrite(in_A, 0);
599           analogWrite(in_B, (vvesquerda+30));
600           }
601           flag_v_mais = 0;
602         }
603
604         le_serial();                                    //Faz leitura da porta serial para
        verificar se foi dado algum comando de parada emergencial
605
606         if (strcmp(comando,"stop") == 0){               //Se foi dado o comando de parada
        emergencial 'stop', para todos os motores e sai do loop
607           para_motores();
608           break;
609         }
610         atualiza_saida();                               //Chama funcao para enviar os dados dos
         sensores para o MATLAB
611       }
612     }
613     para_motores();
614     pausa_movimento();
615     if(strcmp(direcao,"horizontal") == 0 || strcmp(direcao,"esquerda") == 0){
616       Serial.write(acabou.binary,4);                    //Envia via serial ao Matlab a
        indicacao de conclusao da movimentacao da carga
617     }
618   }
619   //=======================
620   void move_direita(){      //funcao de movimentacao horizontal da carga para a direita
621
622     if(strcmp(velocidade,"lenta") == 0){ //Determina a velocidade a partir do comando dado
        pelo usuario
```

```
623       vhdireita = vhdireita_lenta;
624       vvdireita = vvdireita_lenta;
625     }
626     else if(strcmp(velocidade,"moderada") == 0){
627       vhdireita = vhdireita_moderada;
628       vvdireita = vvdireita_moderada;
629     }
630     else if(strcmp(velocidade,"rapida") == 0){
631       vhdireita = vhdireita_rapida;
632       vvdireita = vvdireita_rapida;
633     }
634     if(strcmp(modo,"automatico") == 0){              //Se o modo recebido do Matlab via
        serial for modo automatico, realiza movimentacao automaticamente para a posicao final
         definida
635       if(strcmp(direcao,"vaievolta") == 0 || strcmp(direcao,"buscacarga") == 0){
636         setpoint_h = 0;
637       }
638       pausa_movimento();
639       while((dh > setpoint_h) || (dv > setpoint_h)) //Enquanto o motor horizontal ou o
        motor vertical nao tiverem chegado na posicao final definida, continua no loop
640       {
641         if(digitalRead(CHAVE_H_DIR) == LOW){          //Verifica se a chave fim de curso nao
         esta acionada
642           if(dh > setpoint_h)                          //Se a posicao horizontal final ainda
         nao tiver sido atingida, aciona o motor horizontal para rodar para a direita
643           {
644             if((dh - setpoint_h) < 3){                 //Se a diferenca entre a posicao
        atual e a posicao final for menor que 3 cm,
645               vhdireita = vhdireita_lenta;                          //Diminui a rotacao do
        motor horizontal, para um posicionamento mais preciso
646               flag_horizontal = 1;
647             }
648             if(flag_horizontal == 1){
649             digitalWrite(in_IN2, HIGH);                //Configura o sentido de rotacao do
        motor horizontal para girar para direita
650             digitalWrite(in_IN1, LOW);
651             analogWrite(in_enA, vhdireita);            //Aciona o motor horizontal com o
        sinal PWM
652             }
653             flag_horizontal = 0;
654           }
655           else                                         //Se a posicao horizontal final for
        atingida,
656           {
657             flag_horizontal = 1;
658             analogWrite(in_enA, 0);                    //Para o motor horizontal
659             digitalWrite(in_IN1, LOW);
660             digitalWrite(in_IN2, LOW);
661           }
662         }
663         else{                                          //Se a chave estiver acionada, para
        os motores
664           para_motores();
665           break;
666         }
667         if(digitalRead(CHAVE_V_1) == LOW && digitalRead(CHAVE_V_2) == LOW){  //Verifica se
        as chaves fim de curso nao estao acionadas
668           if(dv > setpoint_h)                          //Se a posicao vertical final ainda
        nao tiver sido atingida, aciona o motor vertical para rodar para cima para
        compensacao de nivel
```

```
669         {
670           if((dv - setpoint_h) < 3){                //Se a diferenca entre a posicao
      atual e a posicao final for menor que 3 cm,
671             vvdireita = vvdireita_lenta;                        //Diminui a rotacao do
      motor vertical, para um posicionamento mais preciso
672             flag_v_mais = 1;
673             flag_v_menos = 1;
674           }
675           if(dv > dh)                                 //Se a posicao atual vertical for
      maior que a horizontal,
676           {
677             flag_v_menos = 1;
678             if(flag_v_mais == 1){
679             analogWrite(in_A, (vvdireita+30));        //Aumenta a rotacao do motor
      vertical, para tentar manter o nivel vertical da carga constante
680             analogWrite(in_B, 0);
681             }
682             flag_v_mais = 0;
683           }
684           else                                        //Se a posicao atual vertical for
      menor que a horizontal,
685           {
686             flag_v_mais = 1;
687             if(flag_v_menos == 1){
688             analogWrite(in_A, (vvdireita-30));        //Diminui a rotacao do motor
      vertical, para tentar manter o nivel vertical da carga constante
689             analogWrite(in_B, 0);
690             }
691             flag_v_menos = 0;
692           }
693         }
694         else                                          //Se o nivel vertical estiver na
      referencia,
695         {
696           flag_v_mais = 1;
697           flag_v_menos = 1;
698           analogWrite(in_A, 0);                       //Para o motor vertical
699           analogWrite(in_B, 0);
700         }
701       }
702       else{
703         flag_v_mais = 1;
704         flag_v_menos = 1;
705         analogWrite(in_A, 0);                       //Se as chaves estiverem acionadas,
      para o motor vertical
706         analogWrite(in_B, 0);
707       }
708       le_serial();                                  //Faz leitura da porta serial para
      verificar se foi dado algum comando de parada emergencial
709       if (strcmp(comando,"stop") == 0){             //Se foi dado o comando de parada
      emergencial 'stop', para todos os motores e sai do loop
710         para_motores();
711         break;
712       }
713       atualiza_saida();                             //Chama funcao para enviar os dados dos
       sensores para o MATLAB
714     }
715   }
716   else if(strcmp(modo,"manual") == 0){              //Se o modo recebido nao foi '
      automatico', verifica se o modo recebido foi para operacao no modo Manual
```

```
717       while(strcmp(comando,"stop") != 0)              //Se sim, entra no loop de movimentacao
           para a direita ate que receba o comando de parada 'stop'
718       {
719         if(digitalRead(CHAVE_H_DIR) == LOW){          //Verifica se a chave fim de curso nao
          esta acionada
720           if(flag_horizontal == 1){
721           digitalWrite(in_IN2, HIGH);                   //Configura o sentido de rotacao do
          motor horizontal para girar para direita
722           digitalWrite(in_IN1, LOW);
723           analogWrite(in_enA, vhdireita);               //Aciona o motor horizontal com o
          sinal PWM
724           }
725           flag_horizontal = 0;
726         }
727         else{                                         //Se a chave estiver acionada, para os
          motores
728           para_motores();
729           break;
730         }
731         if(digitalRead(CHAVE_V_1) == LOW && digitalRead(CHAVE_V_2) == LOW){ //Verifica se
          as chaves fim de curso nao estao acionadas
732           if(dv > dh)                                 //Verifica se a distancia percorrida
          verticalmente esta igual a horizontal, para manter a carga nivelada durante o
          percurso
733           {
734             flag_v_menos = 1;
735             if(flag_v_mais == 1){
736             analogWrite(in_A, (vvdireita+30));          //Se esta maior, aumenta a
          velocidade de rotacao vertical para manter as distancias percorridas iguais nos dois
          sentidos
737             analogWrite(in_B, 0);
738             }
739             flag_v_mais = 0;
740           }
741           else                                        //Se esta menor, diminui a velocidade
           de rotacao vertical para manter as distancias percorridas iguais nos dois sentidos
742           {
743             flag_v_mais = 1;
744             if(flag_v_menos == 1){
745             analogWrite(in_A, (vvdireita-30));
746             analogWrite(in_B, 0);
747             }
748             flag_v_menos = 0;
749           }
750         }
751         else{
752           flag_v_mais = 1;
753           flag_v_menos = 1;
754           analogWrite(in_A, 0);                       //Se a chave estiver acionada, para o
          motor vertical
755           analogWrite(in_B, 0);
756         }
757         le_serial();                                  //Faz leitura da porta serial para
          verificar se foi dado algum comando de parada emergencial
758         if (strcmp(comando,"stop") == 0){             //Se foi dado o comando de parada
          emergencial 'stop', para todos os motores e sai do loop
759           para_motores();
760           break;
761         }
```

```
762         atualiza_saida();                          //Chama funcao para enviar os dados dos
          sensores para o MATLAB
763       }
764     }
765     para_motores();                          //Ao chegar na posicao desejada, para todos os
        motores
766     pausa_movimento();
767     Serial.write(acabou.binary,4);                  //Envia via serial ao Matlab a
        indicacao de conclusao da movimentacao da carga
768   }
769   //=======================
770   void desce(){      //funcao de movimentacao vertical da carga para baixo
771
772     if(strcmp(velocidade,"lenta") == 0){ //Determina a velocidade a partir do comando dado
        pelo usuario
773       vvdescendo = vvdescendo_lenta;
774     }
775     else if(strcmp(velocidade,"moderada") == 0){
776       vvdescendo = vvdescendo_moderada;
777     }
778     else if(strcmp(velocidade,"rapida") == 0){
779       vvdescendo = vvdescendo_rapida;
780     }
781     if (strcmp(modo,"automatico") == 0){              //Se o modo recebido do Matlab via
        serial for modo automatico, realiza movimentacao automaticamente para a posicao final
         definida
782       if(strcmp(direcao,"buscacarga") == 0){
783         setpoint_v = setpoint_h + 20;
784       }
785       pausa_movimento();
786       while (dv < setpoint_v){                        //Enquanto o motor vertical nao tiver
        chegado na posicao final definida, continua no loop
787         if((setpoint_v - dv) < 2){                //Se a diferenca entre a posicao atual e
        a posicao final for menor que 2 cm,
788           vvdescendo = vvdescendo_lenta;                        //Diminui a rotacao do motor
         vertical, para um posicionamento mais preciso
789           flag_v_menos = 1;
790         }
791         if(flag_v_menos == 1){
792         analogWrite(in_A, 0);                         //Aciona o motor vertical com o sinal
       PWM e sentido de descida
793         analogWrite(in_B, vvdescendo);
794         }
795         flag_v_menos = 0;
796
797         le_serial();                                  //Faz leitura da porta serial para
        verificar se foi dado algum comando de parada emergencial
798         if (strcmp(comando,"stop") == 0){             //Se foi dado o comando de parada
        emergencial 'stop', para todos os motores e sai do loop
799           para_motores();
800           break;
801         }
802         atualiza_saida();                          //Chama funcao para enviar os dados dos
          sensores para o MATLAB
803       }
804     }
805     else if(strcmp(modo,"manual") == 0){              //Se o modo recebido nao foi '
        automatico', verifica se o modo recebido foi para operacao no modo Manual
806       while (strcmp(comando,"stop") != 0){            //Se sim, entra no loop de movimentacao
         para baixo ate que receba o comando de parada 'stop'
```

```
807         if(flag_v_menos == 1){
808         analogWrite(in_A, 0);                          //Aciona o motor vertical com o sinal
        PWM e sentido de descida
809         analogWrite(in_B, vvdescendo);
810         }
811         flag_v_menos = 0;
812         le_serial();                                   //Faz leitura da porta serial para
        verificar se foi dado algum comando de parada emergencial
813         if (strcmp(comando,"stop") == 0){              //Se foi dado o comando de parada
        emergencial 'stop', para todos os motores e sai do loop
814           para_motores();
815           break;
816         }
817         atualiza_saida();                              //Chama funcao para enviar os dados dos
         sensores para o MATLAB
818       }
819     }
820     para_motores();                          //Ao chegar na posicao desejada, para todos os
        motores
821     pausa_movimento();
822     if(strcmp(direcao,"vertical") == 0 || strcmp(direcao,"desce") == 0){
823       Serial.write(acabou.binary,4);                   //Envia via serial ao Matlab a
        indicacao de conclusao da movimentacao da carga
824     }
825   }
826   //=======================
827   void sobe(){      //funcao de movimentacao vertical da carga para cima
828
829     if(strcmp(velocidade,"lenta") == 0){ //Determina a velocidade a partir do comando dado
        pelo usuario
830       vvsubindo = vvsubindo_lenta;
831     }
832     else if(strcmp(velocidade,"moderada") == 0){
833       vvsubindo = vvsubindo_moderada;
834     }
835     else if(strcmp(velocidade,"rapida") == 0){
836       vvsubindo = vvsubindo_rapida;
837     }
838     if (strcmp(modo,"automatico") == 0){             //Se o modo recebido do Matlab via
        serial for modo automatico, realiza movimentacao automaticamente para a posicao final
         definida
839       if(strcmp(direcao,"buscacarga") == 0){
840         setpoint_v = setpoint_h;
841       }
842       pausa_movimento();
843       while (dv > setpoint_v){                       //Enquanto o motor vertical nao tiver
        chegado na posicao final definida, continua no loop
844         if(digitalRead(CHAVE_V_1) == LOW && digitalRead(CHAVE_V_2) == LOW){ //Verifica se
        as chaves fim de curso nao estao acionadas
845           if((dv - setpoint_v) < 3){                 //Se a diferenca entre a posicao atual
        e a posicao final for menor que 3 cm,
846             vvsubindo = vvsubindo_lenta;                          //Diminui a rotacao do
        motor vertical, para um posicionamento mais preciso
847             flag_v_mais = 1;
848           }
849           if(flag_v_mais == 1){
850           analogWrite(in_A, vvsubindo);                //Aciona o motor vertical com o sinal
         PWM e sentido de subida
851           analogWrite(in_B, 0);
852           }
```

```
853           flag_v_mais = 0;
854           le_serial();                                    //Faz leitura da porta serial para
        verificar se foi dado algum comando de parada emergencial
855         }
856         else {strcpy(comando,"stop");}                    //Se as chaves estiverem acionadas,
        para os motores
857         if (strcmp(comando,"stop") == 0){                 //Se foi dado o comando de parada
        emergencial 'stop', para todos os motores e sai do loop
858           para_motores();
859           break;
860         }
861         atualiza_saida();                                 //Chama funcao para enviar os dados dos
         sensores para o MATLAB
862       }
863     }
864     else if(strcmp(modo,"manual") == 0){                  //Se o modo recebido nao foi '
        automatico', verifica se o modo recebido foi para operacao no modo Manual
865       while (strcmp(comando,"stop") != 0){                //Se sim, entra no loop de movimentacao
         para baixo ate que receba o comando de parada 'stop'
866         if(digitalRead(CHAVE_V_1) == LOW && digitalRead(CHAVE_V_2) == LOW){ //Verifica se
        as chaves fim de curso nao estao acionadas
867           if(flag_v_mais == 1){
868           analogWrite(in_A, vvsubindo);                   //Aciona o motor vertical com o sinal
         PWM e sentido de subida
869           analogWrite(in_B, 0);
870           }
871           flag_v_mais = 0;
872           le_serial();                                    //Faz leitura da porta serial para
        verificar se foi dado algum comando de parada emergencial
873         }
874         else {strcpy(comando,"stop");}                    //Se as chaves estiverem acionadas,
        para os motores
875         if (strcmp(comando,"stop") == 0){                 //Se foi dado o comando de parada
        emergencial 'stop', para todos os motores e sai do loop
876           para_motores();
877           break;
878         }
879         atualiza_saida();                                 //Chama funcao para enviar os dados dos
         sensores para o MATLAB
880       }
881     }
882     para_motores();                                     //Ao chegar na posicao desejada, para
        todos os motores
883     pausa_movimento();
884     if(strcmp(direcao,"vertical") == 0 || strcmp(direcao,"sobe") == 0){
885       Serial.write(acabou.binary,4);                    //Envia via serial ao Matlab a
        indicacao de conclusao da movimentacao da carga
886     }
887   }
888   //=======================
889   void move_vaievolta(){      //funcao de movimentacao horizontal da carga para o movimento
        'vai e volta'
890     move_esquerda();
891     pausa_movimento();
892     move_direita();
893   }
894   //=======================
895   void move_buscacarga(){      //funcao de movimentacao da carga para o movimento 'busca
        carga'
896     move_esquerda();
```

```
897     pausa_movimento();
898     desce();
899     pausa_movimento();
900     sobe();
901     pausa_movimento();
902     move_direita();
903   }
904   //=======================
905   uint8_t i2cWrite(uint8_t registerAddress, uint8_t data, bool sendStop) { //funcao de
        escrita I2C
906     return i2cWrite(registerAddress, &data, 1, sendStop); // Retorna 0 se escreveu com
        sucesso
907   }
908   //=======================
909   uint8_t i2cWrite(uint8_t registerAddress, uint8_t *data, uint8_t length, bool sendStop) {
         //funcao de escrita I2C
910     Wire.beginTransmission(IMUAddress);
         //Inicia transmissao com o sensor
911     Wire.write(registerAddress);
         //Acessa o registrador a ser escrito
912     Wire.write(data, length);
         //Escreve o dado no registrador
913     uint8_t rcode = Wire.endTransmission(sendStop);
         //Retorna 0 se escreveu com sucesso
914     if (rcode) {
         //Se rcode = 1,
915       para_motores();
916       Serial.write(falha_i2cWrite.binary,4);
           //Gera aviso dizendo que houve erro de escrita I2C
917     }
918     return rcode;
919   }
920   //=======================
921   uint8_t i2cRead(uint8_t registerAddress, uint8_t *data, uint8_t nbytes) {      //funcao
        de leitura I2C
922     uint32_t timeOutTimer;                                                      //Contador
         de timeout
923     Wire.beginTransmission(IMUAddress);                                         //Inicia
        transmissao com o sensor
924     Wire.write(registerAddress);                                                //Acessa o
         registrador a ser lido
925     uint8_t rcode = Wire.endTransmission(false);                                //Mantem o
         canal de comunicacao ocupado
926     if (rcode) {                                                                //Se nao
        retornou 0,
927       para_motores();
928       Serial.write(falha_i2cRead.binary,4);                                      //Gera
        aviso diendo que houve erro de leitura I2C
929       return rcode;
930     }
931     Wire.requestFrom(IMUAddress, nbytes, (uint8_t)true);                        //Envia
        requisicao leitura de nbytes para o endereco, liberando o canal de comunicacao em
        seguida
932     for (uint8_t i = 0; i < nbytes; i++) {                                      //Entra no
         loop para armazenamento de dados obtidos do registrador
933       if (Wire.available())                                                     //Se ha
        dados a serem lidos
934         data[i] = Wire.read();                                                  //Faz a
        leitura do dado e coloca no vetor
935       else {
```

```
936         timeOutTimer = micros();                                                    //Se nao
      ha mais dados, entra em rotina para verificacao de timeout
937         while (((micros() - timeOutTimer) < I2C_TIMEOUT) && !Wire.available());  //Enquanto
       nao ha dados e nao atingiu o tempo de timeout,
938         if (Wire.available())                                                       //Verifica
       se ha dados a serem lidos
939           data[i] = Wire.read();
940         else {                                                                      //Se
      atingiu o tempo de timout,
941           para_motores();
942           Serial.write(falha_i2cTimeout.binary,4);                                    //
      Indica que houve erro de leitura I2C
943           return 5;
944         }
945       }
946     }
947     return 0;                                                                       //Retorna
       0 se fez a leitura com sucesso
948 }
```

APÊNDICE C – KALMAN.H

```
1  #ifndef _Kalman_h_
2  #define _Kalman_h_
3
4  class Kalman {                                          //Definicao da classe Kalman
5  public:
6      Kalman();                                           //Declara variaveis e
       matrizes a serem usadas nos calculos do angulo filtrado
7
8      float filtra_angulo(float angulo_novo, float rate_novo, float dt); //Funcao que
       calcula angulo filtrado, com angulo_novo em graus, rate_novo em graus/s e dt em
       segundos
9
10     void define_angulo(float angulo);                   //Funcao que define o
       angulo inicial a ser usado nos calculos.
11
12 private:
13     //Variaveis do filtro Kalman
14     float Q_angulo;   // Variancia do ruido do processo para o acelerometro
15     float Q_bias;     // Variancia do ruido do processo para o desvio do giroscopio
16     float R_medicao; // Variancia do ruido da medicao
17
18     float angulo;     // Angulo calculado pelo filtro Kalman (parte do vetor de estado 2
       x1)
19     float bias;       // Desvio do giroscopio calculado pelo filtro Kalman (parte do vetor
        de estado 2x1)
20     float rate;       // Variacao angular sem contar o desvio do giroscopio
21
22     float P[2][2];    // Matriz de covariancia do erro (matriz 2x2)
23 };
24
25 #endif
```

APÊNDICE D – KALMAN.CPP

```
#include "Kalman.h"

Kalman::Kalman() {
    //Variancias de ruido de processo e de medicao
    Q_angulo = 0.001f;
    Q_bias = 0.003f;
    R_medicao = 0.03f;

    angulo = 0.0f; // Inicializa o angulo
    bias = 0.0f; // Inicializa o bias

    P[0][0] = 0.0f; // Como o bias e 0 e sabe-se o angulo inicial a partir da funcao define_angulo,,
    P[0][1] = 0.0f; // A matriz de covariancia do erro e inicializada com zeros
    P[1][0] = 0.0f;
    P[1][1] = 0.0f;
};
// Funcao que filtra o angulo, tendo como entrada o angulo em graus, o rate em graus/s e delta t em segundos
float Kalman::filtra_angulo(float angulo_novo, float rate_novo, float dt) {

    // Passo 1: Estimativa dos estados a priori
    rate = rate_novo - bias;
    angulo += dt * rate;
    //Passo 2: Estimativa da matriz de covariancia do erro a priori
    P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angulo);
    P[0][1] -= dt * P[1][1];
    P[1][0] -= dt * P[1][1];
    P[1][1] += Q_bias * dt;
    // Passo 3: Calculo da inovacao y[k]
    float y = angulo_novo - angulo; // diferenca no angulo
    // Passo 4: Calculo da covariancia da inovacao
    float S = P[0][0] + R_medicao; // Erro estimado
    // Passo 5: Calculo do ganho de Kalman K
    float K[2];
    K[0] = P[0][0] / S;
    K[1] = P[1][0] / S;
    //Passo 6: Estimativa dos estados a posteriori
    angulo += K[0] * y;
    bias += K[1] * y;
    //Passo 7: Atualizacao da matriz de covariancia do erro a posteriori
    float P00_temp = P[0][0];
    float P01_temp = P[0][1];

    P[0][0] -= K[0] * P00_temp;
    P[0][1] -= K[0] * P01_temp;
    P[1][0] -= K[1] * P00_temp;
    P[1][1] -= K[1] * P01_temp;

    return angulo;
};

void Kalman::define_angulo(float angulo) { this->angulo = angulo; }; //Funcao para definir o angulo inicial
```