GUILHERME VINICIUS DE SOUSA SOARES D'OSWALDO
VINICIUS MELO DE SOUZA

ESPECIFICAÇÃO E SIMULAÇÃO DE HARDWARE DE CONTROLE DE UM MANIPULADOR ROBÓTICO DE 5 GRAUS DE LIBERDADE

São Paulo
2022

GUILHERME VINICIUS DE SOUSA SOARES D'OSWALDO
VINICIUS MELO DE SOUZA

ESPECIFICAÇÃO E SIMULAÇÃO DE HARDWARE DE CONTROLE DE UM MANIPULADOR ROBÓTICO DE 5 GRAUS DE LIBERDADE

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Elétrico com ênfase em Automação e Controle.

Área de Concentração:

Especificação e Simulação de Sistemas Embarcados

Orientador:

Fábio de Oliveira Fialho

São Paulo
2022

Catalogação-na-publicação

Souza, Vinícius
Especificação e simulação de hardware de controle de um manipulador robótico de 5 graus de liberdade / V. Souza, G. D'Oswaldo -- São Paulo, 2022.
98 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Telecomunicações e Controle.

1.Manipulador Robótico 2.Especificação de Hardware 3.Simulação de Hardware 4.Sistemas Embarcados I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Telecomunicações e Controle II.t. III.D'Oswaldo, Guilherme

RESUMO

Manipuladores robóticos são muito utilizados na indústria em diversas aplicações, como na indústria automotiva, até mesmo em trabalhos mais precisos como na medicina. Com isso em mente, é necessário que alunos que pretendem seguir carreira profissional nesta área apresentem uma formação teórica robusta, que apenas é possível a partir do controle prático desses manipuladores, assim como a experiência com a simulações.

Dessa forma, está sendo projetado um manipulador robótico com 5 graus de liberdade e o objetivo deste projeto é especificar a parte de eletrônica e hardware de controle do braço, além de simular esse hardware de forma a imitar o seu funcionamento prático para o controle do manipulador real. Atualmente o projeto encontra-se com uma modelagem robusta de controle porém o controle prático conta com uma eletrônica não especificada, e a eletrônica utilizada atualmente impede o seu uso prático. Além disso, será refinada a Interface Humano-Máquina para a utilização didática, preparando seu funcionamento para o uso simplificado pelo aluno, criando novas funcionalidades como rotas pré-determinadas e também permitindo formas mais complexas de controle com configurações mais específicas com scripts. Para a realização das atividades foi proposto um estudo teórico sobre o projeto de sistemas embarcados, utilização de ferramentas de programação e simulação (MATLAB, Simulink), além da simulação do microcontroladores (Proteus) para a realização de testes com software embarcado e também análises de carga. Afinal, terá sido realizado uma especificação do hardware, sendo possível delimitar informações do microcontrolador a partir das simulações e além disso será feito uma análise do método utilizado para a especificação, compreendendo os benefícios de utilizar uma metodologia híbrida de especificação top-down e bottom-up, na qual enquanto é realizado a especificação top-down é realizados simulações com hardwares específicos de uma forma bottom-up.

Espera-se desse projeto a possibilidade de, a partir de outras atividades relacionadas com o controle do manipulador, ser possível implementar um sistema embarcado que irá funcionar sem grandes alterações. Além também para aqueles alunos que pretendem seguir carreira na área possam utilizar uma ferramenta amigável e robusta que possa aprimorar seus conhecimentos na área de manipuladores robóticos.

Palavras-Chave – Manipuladores Robóticos, Sistemas Embarcados, Especificação de Hardware, Simulação de Hardware, Interface Humano-Máquina.

SUMÁRIO

1 INTRODUÇÃO

O primeiro robô industrial foi criado em 1954 e sua função era pegar pedaços quentes de metal e colar essas peças nos chassis dos carros. Dessa forma em plena revolução industrial começaram a surgir os primeiros manipuladores robóticos a partir do alinhamento de diversos conhecimentos como Eletrônica, Mecânica, Computação e Teorias de Controle. Esse robôs eram utilizadas em tarefas repetitivas e muitas vezes perigosas como soldagem, movimentação de cargas pesadas e realização de testes, por exemplo.

1.1 Antecedentes

Para a realização de todos os trabalhos está sendo utilizado um manipulador robótico de 5 juntas construído em colaboração com os professores Arthur Vieira Netto da Fatec Sorocaba, Fábio de Oliveira Fialho e Fuad Kassab Junior da Escola Politécnica da USP, com adaptações em suas instalações elétricas pelo doutorando Lucas Franco da Silva também da Escola Politécnica da USP. A figura 1.1 é uma foto do braço robótico que será utilizado ao longo do desenvolvimento.

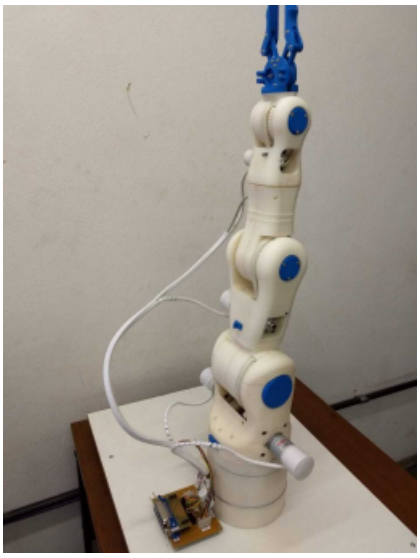


Figura 1.1: Braço Robótico Disponibilizado

Nesse manipulador são utilizados motores do tipo CC da Premium Planetary Gear Motor com *encoder* de quadratura magnética. Abaixo é apresentado uma tabela com as características desses motores.

Junta	Motor	Relação de Engrenagem	Voltagem
1, 2 e 4	Premium Planetary Gear Motor com 60 RPM	231.22:2	12V
3 e 5	Premium Planetary Gear Motor com 60 RPM	139.138:1	12V

Tabela 1.1: Especificação dos motores das juntas

Anteriormente a esse projeto foi realizado um projeto na área de controle, analisando os métodos de controle de manipuladores robóticos, que depende de conceitos avançados de cinemática, dinâmica e teoria de controle. Foram implementados sistemas de controle como um controle independente por juntas, controlador PD e controle *feedfoward*, na qual considerava a modelagem de diversas não linearidade do sistema em um braço simulado, como o atrito de coulomb e efeito *backlash*.

Além disso, foi implementando uma Interface Humano-Máquina básica com algumas funcionalidades iniciais, utilizando a ferramentas App Designer: plataforma de desenvolvimento de aplicativos integrada ao MATLAB. A IHM apresenta diversas funcionalidades que se assemelham às de interfaces utilizadas atualmente na indústria, podendo o aluno ter um contato com um protótipo e com uma IHM que se assemelha a uma encontrada no mercado, porém diversas funcionalidades não estavam completamente implementadas. A figura X mostra uma foto do estado inicial da IHM.

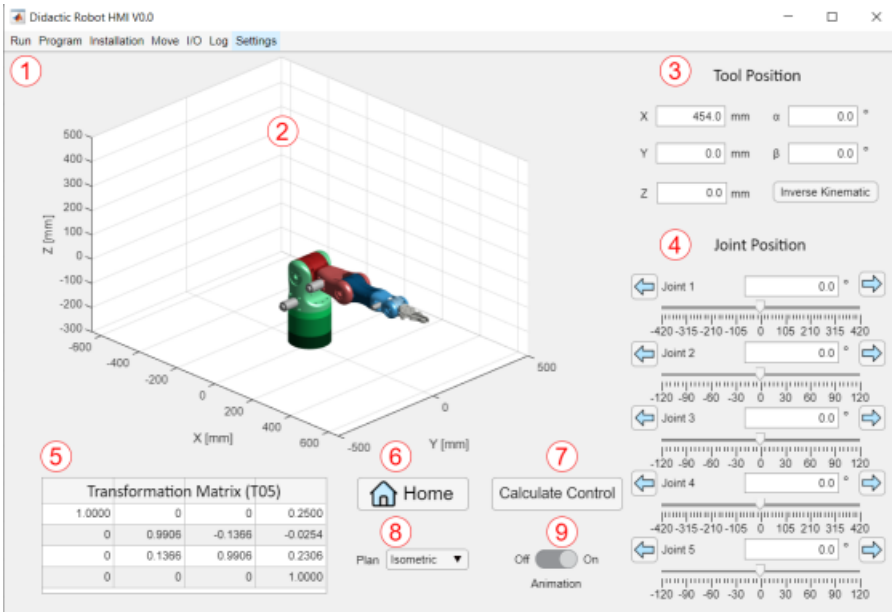


Figura 1.2: Interface principal da IHM desenvolvida

As peças do manipulador robótico didático já foram modeladas no software Solidworks pelo professor Arthur Vieira Netto da Fatec Sorocaba. No trabalho anterior foram realizadas a integração dessas peças com o MATLAB na qual é possível visualizar o conjunto completo por meio de uma função *patch*, pré-definida pelo MATLAB.

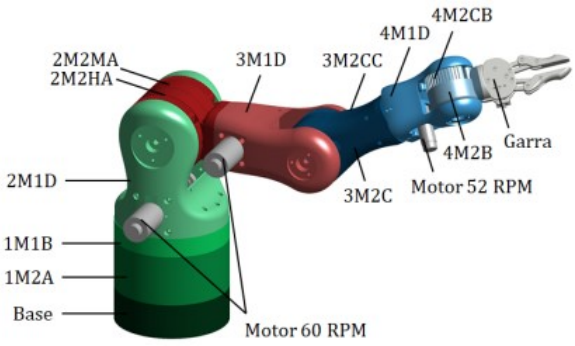


Figura 1.3: Braço Robótico modelado no Solidworks

Por fim existe uma eletrônica de potência projetada na qual é utilizado um *driver* que tem como finalidade receber sinais de baixa corrente provenientes do controlador e amplificá-los para que o motor possa ser acionado corretamente.

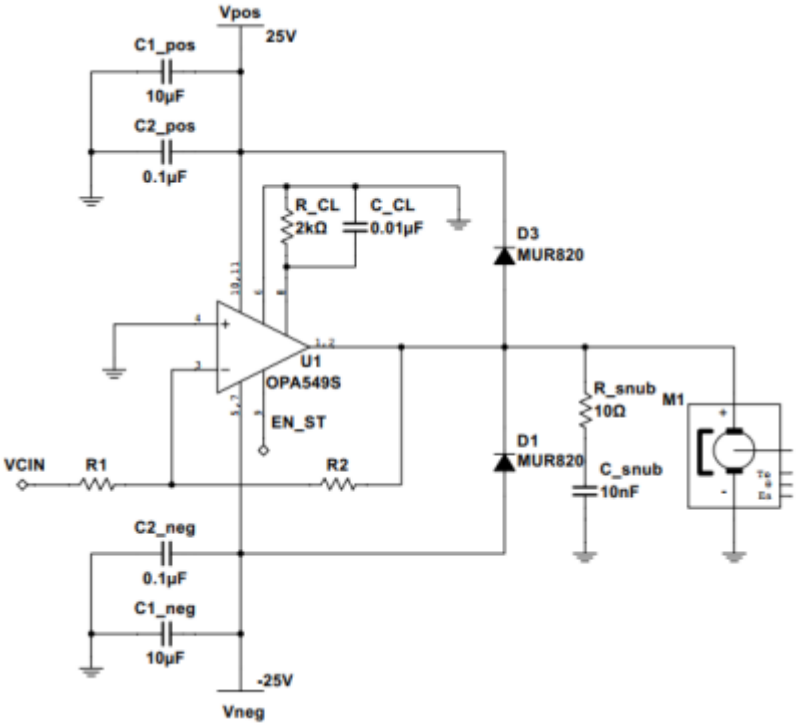


Figura 1.4: Circuito eletrônico - *driver*

Para analisar o correto funcionamento dos motores e possibilitar a criação de novas implementações utilizando a corrente como informação para o controle do manipulador foi implementado um sensor de corrente utilizando um resistor *shunt*, porém o seu funcionamento não estava ideal.

1.2 Motivação

Os manipuladores robóticos são muito úteis e têm se tornado cada vez mais comuns. Inicialmente auxiliando o trabalho repetitivo, atualmente eles têm ganhado tanta autonomia, flexibilidade e complexidade que grande parte de trabalhos manuais e repetitivos são realizados unicamente por manipuladores em cadeias industriais. Eles estão presentes na indústria com o intuito de diminuir custos, atuar em tarefas que poderiam ser prejudiciais à saúde humana, além de realizarem suas atividades incessantemente permitindo que indústrias nunca parem suas atividades. Porém para o manuseio correto de objetos, sem quebrá-los, derrubá-los e posicionando de forma exata na sua necessidade, esses manipuladores dependem de controles muito precisos que consideram erros na movimentação, mudanças de direção bruscas que dependem da zona de contato das engrenagens e relações de atrito que alteram com questões microscópicas de sua constituição das peças.

Atualmente esses robôs têm se tornado mais conectados com a utilização de dados em massa presentes nas indústrias 4.0 a na cooperação de suas atividades para a realização de forma mais eficiente, o que exige de um profissional uma melhor compreensão e adequação às necessidades de conhecimentos para a atuação com esses equipamentos na atualidade. Dessa forma, além de sua teoria não ser trivial, a sua utilização prática tem se tornado um pré-requisito para alunos que pretendem ingressar nesse mercado de trabalho, compreendendo também questões de conectividade com esses equipamentos, e utilização de interfaces complexas presentes na indústria.

Uma eletrônica dedicada que leve em consideração os diversos ambientes de uso, formas de trabalho, e realizações de atividades é algo cada vez mais utilizado, além de que em um contexto educacional a possibilidade de alterar questões de controle com um simples clique de forma que um microcontrolador passe a trabalhar com métodos distintos de controle é de grande utilidade para um aluno que deseje compreender esses diversos métodos, seus benefícios e contrapontos.

Para tal o intuito desse projeto de especificar essa eletrônica e simulá-la é de fundamental importância para a continuidade do projeto do manipulador robótico didático, com o intuito de evitar retrabalhos futuros na qual seja necessário alterar aspectos do projetos para se adequar a novos métodos de controle propostos no meio educacional. E não apenas isso, mas uma interface amigável e de fácil uso, que representam interfaces utilizadas no mercado, é de grande importância para preparar o aluno a utilizar esses sistemas no meio industrial sem frustrações.

1.3 Relevância

Qualquer manipulador robótico depende de uma eletrônica dedicada com por exemplo: eletrônica de potência, sistema de processamentos podendo ser microcontroladores com conjuntos de pinos dedicados ou de propósito geral. E para o seu correto funcionamento é necessário um estudo e especificação das necessidades daquele sistema em específico o que constitui um projeto de Sistema Embarcado.

Para um manipulador utilizado em meio educacional a sua complexidade acaba sendo ainda maior devido a necessidade de flexibilidade do sistema que se beneficia com diversos métodos de conectividade, diversos métodos de controle implementados e uma necessidade de estar preparado e seguro para a incorreta utilização de seus componentes devido estarem sendo utilizados por usuários que estarão experimentando com esse sistema.

Dessa forma um hardware robusto que possa ser incrementado futuramente, que seja utilizados de diversas maneiras é que apresentam uma flexibilidade sem prejudicar sua complexidade é de grande necessidade, o que torna o trabalho de especificação desse sistema mais complexo e árduo. Porém o resultado é muito proveitoso dado a possibilidade de ser utilizado sem grandes dificuldades e preocupações, já que diversos aspectos de uso foram planejados para serem utilizados da forma com que os componentes foram requisitados. Se espera de um aluno que estude nessa área a ocorrência de falhas e descuidos, por conta disso se espera desse sistema uma robustez e adaptabilidade muito maior.

Além disso, o uso de simulações para a criação do hardware, não só tem sido uma tendência no mercado com o uso de FPGA's e programas de simulação de microcontroladores, como é um meio de documentação deste sistema que permite que alunos interessados em questões mais intrínsecas ao sistema tenham alí um fruto de aprendizado e experimentação. Logo é de suma importância no projeto de criação de um manipulador robótico utilizado no meio educacional, um sistema muito bem especificado, organizado, documentado e testado para a utilização.

1.4 Declaração do Problema

Assim como descrito anteriormente a teoria presente nos sistemas de controle não é nada trivial dependendo de diversos cálculos e etapas na sua realização. Dessa forma a sua computação também se torna complexa dependendo da computação da cinemática direta e inversa, cinemática de velocidade, dinâmica de cada junta, geração de trajetórias, determinação de controle por juntas, além de coerência dos cálculos e precisão. Atividades essas que devem ser realizadas em tempo real ou em tempo hábil suficiente para simular uma dinâmica em tempo real de funcionamento do manipulador robótico.

Fora isso, os atuais sistemas utilizados devem ser adaptáveis para serem utilizados em um microcontrolador de forma que seja criado um código compilável para esse novo sistema, que tenha integração com os diversos programas utilizados como MATLAB e Simulink. Dessa forma é necessário determinar necessidade intrínsecas a esse sistema como a possibilidade de rodar esse programas ou no caso gerar códigos compilados a partir desse programas para esse hardware embarcado.

Outra questão é a dependência com os diversos sinais de entrada e saída com os sistemas externos ao hardware. A consideração da forma de realizar a comunicação com a

Interface Humano-Máquina na qual se deve ser estudado a possibilidade de utilizar diversos tipos de comunicação como serial, bluetooth ou Wifi e os diversos tipos de informações e sinais que precisam ser enviados, com seus respectivos protocolos e taxas de envio. Além também das conexões e dependências com os motores e *drivers* de potência, identificando todos os sinais de entrada e saída do sistema e às necessárias taxas de amostragem e frequência de processamento para ser possível realizar os mais variados tipos de controle sem um grande comprometimento do processamento, permitindo maior maleabilidade futuramente.

Por fim é formulada um IHM robusta que esteja adaptada a toda essa comunicação e seja simples, flexível e robusta permitindo que seja utilizado em diversos ambientes como sistemas operacionais distintos, apresentando funcionalidades pré-definidas e também possibilidade de implementação própria por parte do aluno a partir de novos métodos de controle e geração de trajetórias por meio de *scripts*.

1.5 Declaração da Necessidade

A geração de um produto tão robusto como delimitado nas seções anteriores não é uma tarefa simples. Por conta disso são necessários a realização de projetos menores, como o projeto e modelagem do manipulador, a criação do manipulador físico e realização da sua mecânica, a simulação e implementação dos diversos tipos de controles, a análise das diversas não linearidades, a configuração e determinação dos diversos coeficientes de controle, a especificação e simulação do hardware embarcado, a prototipação do hardware, assim como outros passos a serem definidos futuramente, procurando sempre melhorar o produto inicial. Dessa forma o nosso projeto é uma dessas etapas que visa permitir o uso prático desse sistema.

O hardware embarcado é de suma importância pois é ele que realiza todo o controle, fazendo os cálculos, conectando os sinais dos sistema físico com as informações de manipulação definidas pelo usuário, configurando proteções para o sistema físico impedindo comandos do usuário advindo da IHM que possam prejudicar o sistema, e verificando o correto funcionamento do sistema identificando possíveis falhas na partes mecânicas e possibilitando tanto a segurança do usuário como a segurança dos componentes do manipulador robótico

Por fim, assim como identificado anteriormente a IHM não deve apenas representar um sistema utilizado no mercado, mas também deve possibilitar uma gama de ações

e funcionalidades que não estão presentes em sistemas do mercado dado que constituem funcionalidade necessárias para o estudo dos sistemas de controle de manipuladores. Dessa forma essa Interface Humano-Máquina deve ser adaptável aos diversos sistemas de controle e deve permitir configuração de trajetórias que muitas vezes não existem em sistemas do mercado.

1.6 Requisitos de Marketing

O projeto atualmente constitui um sistema a ser desenvolvido apenas internamente para a utilização pelos alunos das disciplinas de controle. Sendo então a sua divulgação inicial apenas realizada para aqueles que pretendem cursar a ênfase de controle e automação e tem interesse em dar continuidade no projeto.

Além disso os estudos em si por terem teor acadêmico tem como requisito a divulgação em meios científicos permitindo a ampla divulgação dos métodos de simulação de hardware que muitas vezes são poucos utilizados e conhecidos nesse meios, o que por ventura deverá levar a melhores implementações desse tipo de projeto.

Por fim um projeto acadêmico desse porte pode ser após a sua realização divulgado amplamente no meio universitário permitindo que outras instituições consigam implementar esses equipamentos dado a sua fácil criação utilizando peças impressas em 3D, com componentes de fácil troca.

1.7 Requisitos de Engenharia

Para realizar a implementação do estado atual do projeto, com a especificação e simulação de hardware, é necessário o conhecimento da arquitetura de sistemas embarcados, conhecendo os diversos tipos de microcontroladores, as diversas arquitetura de processadores e conjunto de instruções. Também a utilização de programas que possibilitem a implementação de softwares embarcados, os benefícios de cada sistema assim como possíveis contrapontos que os mesmos possam apresentar.

É necessário também conhecer as diversas metodologias de projeto de sistemas embarcados, sendo possível utilizar a criação desses sistemas a partir de métodos *top-down* ou *bottom-up*, a etapas de modelagem, design e prototipação desses sistemas, e a forma com que os diferentes temas de estudo se integram em um único projeto, unindo questão físicas e mecânicas dado por sensores, atuadores, dinâmica e modelagem dos sistemas, questões

de armazenamento, capacidade de processamento e comunicação entre usuário e sistema físico que constituem o hardware do sistema, e também questões de arquitetura de software, programação nas mais diversas linguagens como C, C++ e Matlab, e organização e boas práticas de programação, que constituem questões de software do projeto.

Para o projeto em questão é necessário conhecer os diversos tipos de controle como controle PID, controle por junta e controle *feedforward*, compreendendo como implementá-los tanto em linguagens de mais baixo nível como C e C++, como em programação de mais alto nível utilizando programas como MATLAB e Simulink.

Por fim, também é necessário saber utilizar programas da indústria que permitem a simulação de microcontroladores, como no software Proteus. É necessário compreender como é possível executar um software nesse hardware simulado e como é possível verificar a utilização da carga desse sistema. A conexão entre os diversos programas como MATLAB e Proteus também é de suma importância o que permite simular a dinâmica do manipulador robótico e testar os softwares de controle sem a utilização prática do manipulador físico.

1.8 Árvore de Objetivos

Para a realização do objetivo final que consiste a simulação do hardware embarcado possibilitando controlar o manipulador a partir desse hardware simulado, foram definidos a seguinte árvore de objetivos:

- Identificar o problema presente no sistema de potência para o recebimento de sinais de corrente;
- Refatoração da IHM para a o correto funcionamento dela além de permitir a fácil configuração na sua utilização;
- Simulação do hardware inicial visualizando seu controle por meio de variáveis no Matlab;
- Implementação de um controle inicial no hardware simulado e visualização de seu funcionamento no Matlab;
- Implementação de um controle de 3 graus de liberdade em um hardware simulado com conexão com Matlab;

- Implementação de um controle de 5 graus de liberdade e um hardware simulado com conexão com o Matlab;
- Implementação de um controle simples com conexão com o Matlab para ser realizado o controle diretamente no braço robótico físico;
- Implementação de um controle simples com conexão com o Matlab para ser realizado o controle diretamente no braço robótico físico;
- Implementação de um controle mais complexo de 5 graus de liberdade em um hardware simulado com conexão com o Matlab para o controle do braço robótico o físico;
- Especificação do hardware em níveis 0, 1 e 2 (Verificando até onde é possível realizar essa especificação dado o atual estado do projeto).

2 ESTADO DA ARTE

Atualmente é comum vermos a realização de simulações de Hardware anteriores a sua prototipação. Isso é devido principalmente à possibilidade de projetar o software em união com o hardware permitindo criar hardwares mais específicos e com limitações mais bem definidas. Dessa forma a especificação de hardware utilizando esse métodos condiz melhor com as expectativas para o funcionamento do hardware no momento da realização de testes práticos do Sistema Embarcado.

2.1 Conceitos

- Sistema Embarcado: Sistema microprocessado no qual o computador é completamente dedicado ao dispositivo ou sistema que ele controla. Ele realiza um conjunto de tarefas predefinidas, com requisitos de especificação, sendo que sua especificidade permite a otimização desse sistema para a tarefa realizada. Normalmente não são reprogramáveis e utilizam de recursos computacionais limitados;

- Hardware: Parte física de um computador, isso é o conjunto de aparatos eletrônicos, peças e equipamentos que fazem o computador funcionar. Pode ser utilizada também para o conjunto de equipamentos acoplados que possuem algum tipo de processamento computacional;

- Software: Conjunto de instruções que devem ser executadas por um mecanismo seja ele um computador ou um aparelho eletromecânico. Muitas vezes é o termo utilizado para descrever programas, apps, scripts, macros e instruções de códigos embarcados diretamente (Firmware), de modo a ditar o que a máquina realiza;

- Especificação top-down: Método que visa a arquitetura de gestão, produtos e projetos começando com uma abordagem geral e descendo para níveis mais específicos.;

- Especificação bottom-up: Método que visa a arquitetura de gestão, produtos e projetos, começando com uma especificação completa nos subníveis mais detalhados

e compreendendo como esse subnível trabalha verificando o seu espaço dentro de um sistema maior;

- Graus de liberdade: Para a geometria são diferentes maneiras que um objeto pode se mover em um sistema, isto é no projeto são os diferentes eixos de movimentação que o manipular apresenta;
- Cinemática: Parte da mecânica que estuda os movimentos de corpos. Descrição dos movimentos sem se preocupar com suas causas;
- Dinâmica: Parte da mecânica responsável por analisar as causa do movimentos e seus possíveis efeitos.

2.2 Definições

- Manipuladores robóticos: Dispositivo mecanicamente concebido para posicionar e orientar no espaço o seu órgão terminal: uma garra ou uma ferramenta. Executam a manipulação de objetos. Na indústria são muito para que sejam automatizados a passagem de produtos de uma linha de produção para outra;
- Microcontrolador: Conjunto de um núcleo de processador, memórias voláteis e não voláteis e diversos periféricos de entrada e saída de dados em um único circuito integrado;
- Firmware: Conjunto de instruções operacionais que são programadas diretamente no hardware de um equipamento eletrônico;
- Driver: Componente que torna possível que o sistema se comunique com o hardware. Para software é o programa que permite a comunicação do sistema operacional com o hardware. Para a eletrônica e o circuito que recebe uma entrada analógica e gera uma saída compreensível para um hardware;
- Interface Humano-Máquina: Painel de interação com o usuário que permite ele se comunicar com a máquina. Utilizado para descrever telas utilizadas em ambientes industriais que exibem dados em tempo real e permitem que o usuário controle o equipamento usando a interface gráfica;
- FPGA: Field-Programmable Gate Array é um dispositivo lógico programável que suporta a implementação e simulação de circuitos digitais. São diversas funções lógicas, memórias, DSP, blocos de entrada e saída que podem se conectar de diversas

formas diferentes para representar um hardware descrito por uma linguagem de descrição de hardware (HDL);

- Controle PID: Controle proporcional, integral e derivativo, que é uma técnica de controle mais empregada quando se deseja realizar o controle de variáveis contínuas mantendo elas em um valor a partir da compensação de seu erro de diferentes formas;
- Controle FeedForward: Controle que tem por objetivo antecipar as variações das variáveis e fazer correções antes da alteração dessas variáveis manipuladas;
- IDE: Ambiente de desenvolvimento integrado, isso é um software para criar aplicações que combinam ferramentas comuns de desenvolvimento em uma única interface gráfica de desenvolvimento;
- Protótipo: Produto de trabalho da fase de teste ou planejamento de um projeto. Servindo para fornecer especificações para um sistema funcional e real.

2.3 Abordagem Teórica

Para a realização do projeto será seguido a abordagem teórica top-down e bottom-up definidas na referência [1]. A abordagem top-down será utilizada para a especificação do microcontrolador a ser utilizado, abrindo uma gama de dispositivos que poderão ser utilizados dentro das limitações do projeto. Já a abordagem bottom-up será utilizada para melhorar definição top-down, verificando a funcionalidade de alguns microcontroladores na prática antes da especificação final. Isso é necessário, dado também à constituição do projeto na qual ainda não será possível definir um microcontrolador final de uso.

Porém, nesse caso, a partir da simulação desses microcontroladores utilizando uma abordagem bottom-up, verificando eles na prática e analisando as limitações de alguns dispositivos será possível já prosseguir com outras áreas do projeto de implementação do software e integração dos diversos sistemas como MATLAB e microcontrolador, sem necessariamente ter fechado a utilização de uma arquitetura, sistema ou microcontrolador específico.

Essa abordagem permitirá também a melhor adaptabilidade futuro do projeto, já que o dispositivo simulado poderá facilmente ser alterado no sistema de simulação contanto que ele esteja disponível e seja compatível com os atuais programas utilizados.

2.4 Soluções Existentes

Atualmente, a partir da utilização do programa MATLAB é possível facilmente converter programas do tipo .m para programas em C e C++ utilizando um toolbox de conversão, sendo esses programas em C e C++ executáveis em certos microcontroladores suportados pelo MATLAB. Nesse caso é possível gerar uma software de controle em matlab utilizando às atuais simulações do manipulador robótico para analisar seu funcionamento antes da utilização no manipulador real e convertê-los para diferentes arquiteturas de microcontroladores presentes na lista daqueles suportados para a conversão, realizando algumas alterações manuais devido algumas questões práticas desses microcontroladores. Isso é algo a ser testado e melhor verificado já que um dos pré-requisitos do projeto é que os alunos tenham maleabilidade de conseguirem alterar o software de controle criado sem a necessidade de conhecimento de programação diretamente em C ou C++.

Assim os códigos gerados em C e C++ podem ser compilados pelas respectivos compiladores de cada uma das empresas em suas respectivas IDE's. gerando códigos executáveis do tipo .hex. Esses códigos podem ser utilizados na simulação desse microcontrolador no software Proteus. Esse software simula o funcionamento do microcontrolador verificando a carga de processamento e a realização das instruções que geram as saídas do sistema. Dessa forma pode ser realizado uma simulação simples internamente no proteus verificando o controle de um motor genérico.

Além disso, é possível enviar essas saídas para um sistema externo por meio de uma saída serial virtual. Essa saída consegue se comunicar com o Matlab enviando essas informações para a simulação do manipulador já criada em Simulink e retornando os respectivos sinais necessários para o simulador. Nesse caso é possível fazer uma integração completa entre Proteus e MATLAB simulando o sistema por completo e verificando seu correto funcionamento. Por fim, assim como é possível enviar esses sinais por uma saída serial virtual é possível enviá-los para o manipulador físico e verificar o controle sendo realizado no microcontrolador simulado no próprio manipulador real.

2.5 Tendências

Cada vez mais tem sido uma tendência no mercado a simulação dos hardware do sistema antes da sua prototipação. Isso tem ocorrido com o intuito de diminuir custos, evitar retrabalho além de possibilitar a criação de softwares em paralelo com a especificação do hardware. Isso tem sido uma necessidade dado que o projeto de sistemas embarcados

tem se tornado cada vez mais caros e mais demorados devido à complexidade que esses sistemas têm apresentado.

Uma prática muito comum dessa questão de simulação de hardware antes da prototipação e obtenção do hardware final tem sido muito utilizado na criação de sistemas utilizando FPGA's. Nesse caso o hardware a ser utilizado ainda nem foi projetado, e a realização desse projeto em conjunto com o software tem facilitado a criação de hardwares mais especializados além de ter diminuído muito o tempo de produção já que o software é criado em paralelo com o projeto de hardware.

Isso é muito menos comum em projetos de sistemas embarcados como no caso desse projeto na qual será utilizado um microprocessador específico já constituído, porém pode ser implementado de forma muito similar a partir do atual poder de processamento presentes nas máquina modernas com programas específicos da indústria e facilita muito casos de projetos na qual nem todas os requisitos conseguem ser especificados devido a falta de etapas anteriores a especificação.

2.6 Conclusão

Espera-se desse projeto de TCC a geração de um grande avanço no projeto final. Podendo delimitar diversos dos requisitos do projeto, além de conseguir implementar muitos do sistema e analisar seu funcionamento a partir das simulações em programas da indústria. Será abordados temas de especificação de sistemas embarcados, junto com a implementação de software para esse sistema, e estudo de métodos de controle, além de realização de uma prática não muito comum na indústria para casos como esse que é de simular um microcontrolador já constituído visualizando seu funcionamento antes de ser realizado a prototipação e ter sido realizado o gasto financeiro para adquirir diversos dos produtos necessários para o projeto.

Dessa forma será possível verificar a possibilidade de novos métodos de especificação de hardware além de permitir a fácil troca de dispositivos a partir da manutenção do restante do sistema simulado. Será também possível verificar o atual estado de simulações desse tipo, compreendendo o atual estado do mercado e verificando possíveis melhorias que poderiam ser realizadas para que especificações utilizando esse método possam ser realizadas de forma mais prática.

3 MATERIAS E MÉTODOS

Para a realização do projeto é necessário diversos programas e uma grande dedicação dos colaboradores do projeto para ser possível realizar todas as etapa em tempo hábil. Dentre esses programas nos temos o Proteus que é um software que possibilita a simulação de microcontroladores, além da realização de circuitos eletrônicos. Também é muito comum o uso do MATLAB para o estudo dos diversos tipos de controle e para a continuidade desse projeto ele será de grande auxílio para a simulação da mecânica do manipulador robótico.

3.1 Prova de Conceito

Para provar a possibilidade de controlar o manipulador robótico utilizando um microcontrolador simulado será utilizado o software Proteus. Nesse software é possível simular um microcontrolador específico, sendo disponibilizado todos os pinos utilizados neste microcontrolador, e com isso é possível inserir um código no formato .hex para ser executado no microcontrolador. Também é possível debugar o software neste microcontrolador simulado além de que utilizando o ambiente de simulação do Proteus é possível simular o funcionamento de um motor genérico.

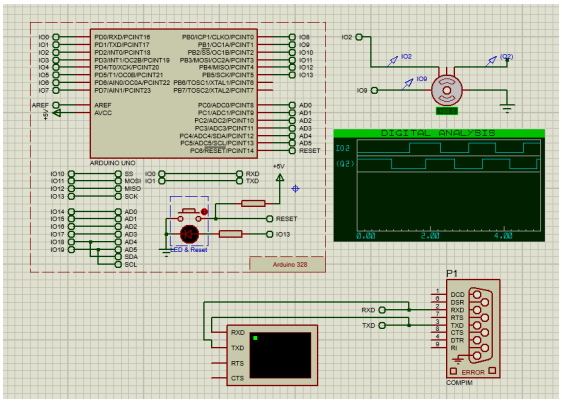


Figura 3.1: Imagem da tela do software Proteus com o esquemático de um Arduino Uno

Também é possível comunicar os dados do software Proteus com o MATLAB utilizando portas seriais virtuais. Essa porta permite enviar informações entre os dois softwares, o que permite simular um software de controle utilizando o Proteus e visualizar seus dados no MATLAB. Além disso, como já existe um modelo do braço robótico no MATLAB que simula o funcionamento do manipulador original, sendo a única questão que os coeficientes do fatores desse simulador ainda não estão definidos, o que é tema de estudo para um outro projeto de TCC, seria possível controlar o modelo simulado em Simulink. Por fim, é possível controlar o manipulador real a partir da comunicação do MATLAB com um placa de aquisição que envia dados analógicos para o driver de potência do manipulador e controla o braço físico. Dessa forma é possível delimitar que existindo a conexão entre MATLAB e Proteus também é possível controlar o manipulador real passando às informações do Proteus para a placa de aquisição por meio do MATLAB.

Como foi definido que a intenção final do projeto é de simular um software de controle em um microcontrolador simulado por meio do Proteus e controlar o manipulador físico devemos constar que os pontos principais são possíveis, sendo esses: simular um software de controle básico em um microcontrolador utilizando o Proteus; realizar a comunicação entre o Proteus e o MATLAB, enviando dados de atuação para o MATLAB; realizar o envio de dados de controle necessários de volta para o Proteus, sendo realizado em tempo real.

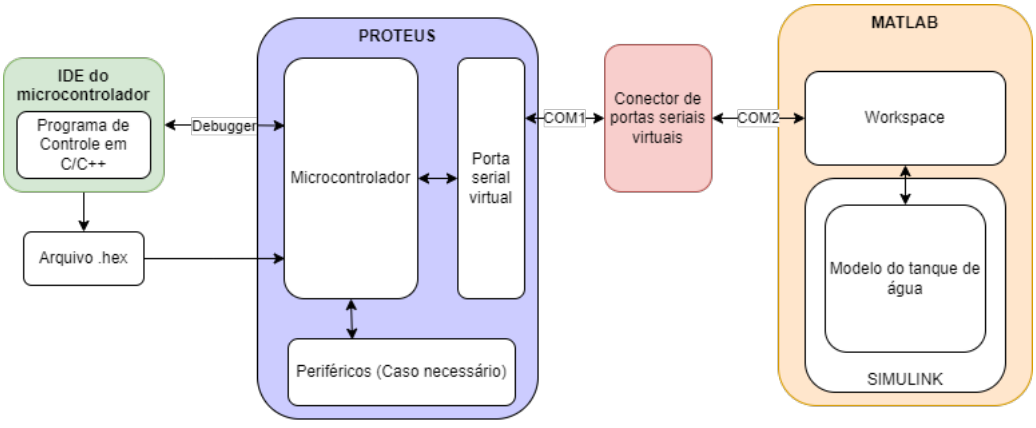


Figura 3.2: Esquema para a realização da Prova de Conceito

Dessa forma para a prova de conceito será criado um controle do tipo proporcional para controlar um tanque de água sendo a tensão de atuação sobre a abertura da válvula do tanque enviado ao MATLAB. Essa tensão de atuação será utilizada em um modelo projetado do tanque de água no MATLAB que terá seu controle em tempo real utilizando o Toolbox de tempo real do MATLAB. Esse tanque de água tem variáveis que modelam

seu sistemas estando elas relacionadas a sua vazão de entrada e de saída, e o valor esperado de altura do nível de água no tanque. Abaixo segue um esquema desse tanque:

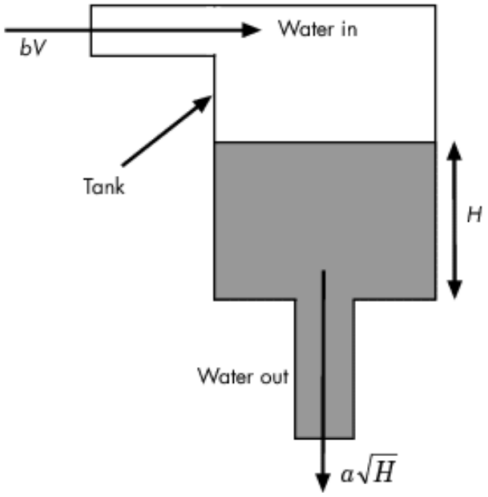


Figura 3.3: Esquema do tanque de água

O dado de altura atual do nível de água será retornado para o software Proteus utilizando a saída serial que irá fechar a malha de controle. Abaixo e demonstrado o modelo a ser utilizado do tanque de água em simulink:

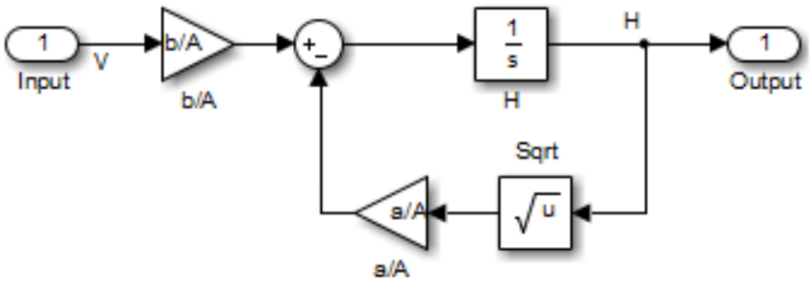


Figura 3.4: Modelo do tanque de água em Simulink

Caso seja possível realizar um controle simplificado dessa forma será também possível realizar o controle mais complicado do modelo de um manipulador. Como existe integração com o MATLAB e a placa de aquisição, então será possível enviar os dados de atuação para o manipulador real e recuperar dados de controle por meio de seus sensores.

3.2 Cronograma

As cores referenciam diferentes trabalhos a serem realizados: Amarelo - Estudos Iniciais; Vermelho - Engenharia de Sistemas; Roxo - Especificação de Hardware; Azul - Trabalhos com a IHM; Verde - Trabalhos com o driver de potência; Laranja - Trabalhos de Simulação de Hardware.

Lista de tarefas	Abril	Maio	Junho	Julho	Agosto
Estudo bibliográfico					
Trabalho de Engenharia de Sistemas					
Estudo da metodologia de especificação de projetos embarcados					
Adaptação da IHM a nova estrutura de diretórios					
Teste e simulação do comportamento esperado do driver de potência					
Especificação em Nível 0					
Modo de Operações (3R e 5R da IHM)					
Replicação e compreensão do problema do resistor Shunt					
Simulação inical do microntrolador					
Trajetória pré programadas de Reta					
Especificação em Nível 1					
Simulação de controle simples no Proteus					
Levantamento de Melhorias para o sensor de corrente					
Simulação do controle com 3R para o MATLAB					
Trajetória pré programadas de Arco					
Especificação em Nível 2					
Simulação do controle com 5R para o MATLAB					
Melhoria Visual da IHM					
Simulação para controle simples do braço real					
Simulação para controle 5R do braço real					

Lista de tarefas	Agosto	Setembro	Outubro	Novembro	Dezembro
Estudo bibliográfico					
Trabalho de Engenharia de Sistemas					
Estudo da metodologia de especificação de projetos embarcados					
Adaptação da IHM a nova estrutura de diretórios					
Teste e simulação do comportamento esperado do driver de potência					
Especificação em Nível 0					
Modo de Operações (3R e 5R da IHM)					
Replicação e compreensão do problema do resistor Shunt					
Simulação inical do microntrolador					
Trajetória pré programadas de Reta					
Especificação em Nível 1					
Simulação de controle simples no Proteus					
Levantamento de Melhorias para o sensor de corrente					
Simulação do controle com 3R para o MATLAB					
Trajetória pré programadas de Arco					
Especificação em Nível 2					
Simulação do controle com 5R para o MATLAB					
Melhoria Visual da IHM					
Simulação para controle simples do braço real					
Simulação para controle 5R do braço real					

Tabela 3.1: Cronograma Detalhado

4 PROPOSTA

Baseado na definição da Necessidade de Especificação e Simulação de Hardware de Controle de Um Manipulador Robótico de 5 Graus de Liberdade, tarefas foram identificadas com o intuito de alcançar o objetivo final. Essas tarefas foram separadas dos seguintes tópicos:

- Estudos - Na qual é definido os principais pontos de pesquisa para permitir a realização do projeto,
- Engenharia de Sistemas e Organização - Tópico para a organização do projeto consolidando a base de criação,
- Circuito dos Atuadores - Na qual foram realizados estudos para compreender o corretos funcionamento dos circuitos com identificação dos problemas e indicação de possíveis soluções,
- Interface Humano-Máquina - Tópico que centraliza todas as alterações realizadas na IHM,
- Especificação - Identificação de todos os níveis de especificação presentes na abordagem de especificação,
- Simulação de Hardware - Tópico com todas as atividades realizadas para a possibilitar realizar testes com o sistema especificados sem a necessidade prática de aquisição da placa.

4.1 Estudos

Inicialmente foram realizados estudos bibliográficos com intuito de encontrar os principais matérias que possibilitariam a realização do projeto, separando os matérias em dois grandes grupos: modelagem e controle, e especificação eletrônica e de sistemas embarcados. Dentre esses matérias as principais referências utilizadas foram *Introduction*

to Robotics: Mechanics and Control de *Patterson* e *Hennessy* [2] para a compreensão da modelagem do sistema e *Computer As Components* de *Marilyn Wolf* [1] para a definição da metodologia de especificação de sistemas embarcados. Dentre outras referencias secundárias utilizadas durante as atividades identificadas posteriormente nesse relatório foram utilizados slides da matéria de manipuladores robóticos dada pelo professor orientador *Fabio Fialho* slidesFabio, slides da matérias de Arquitetura de Sistemas Embarcados do professor *Gustavo Rehder* [3] e livro *Microeletrônica* de *Sedra* e *Smith* [4] para a realização do estudo do circuito analisado.

4.1.1 Metodologia para Especificação do Sistema Embarcado

Após o término do estudo bibliográfico foi feito um estudo sobre técnicas e metodologias de projetos de sistemas embarcados, mais especificamente a parte eletrônica desses sistemas no que se refere a microcontroladores e utilidades e funções que devem estar presentes no circuito.

A partir da referência [3] foram compiladas as informações do fluxo de desenvolvimento usual de projetos relacionados.

Primeiramente é feita a definição dos requisitos do projeto de modo funcional e não funcional, essa etapa define o "O que ?" do projeto. A descrição funcional exige as informações referentes a: entradas, saídas e comunicação que o sistema deve executar. Enquanto a descrição não funcional define requisitos como por exemplo: Peso, Potência, Velocidade ou Custo, todas as características não funcionais devem estar incluídas.

Em seguida é feita a definição das especificações do projeto de fato, é a etapa em que se define o "Como ?" do projeto levando em consideração todas as características funcionais e não funcionais descritas na etapa anterior. Geralmente neste momento são definidos os componentes necessários para atingir os objetivos do projeto.

Para a necessidade em questão, que é definir um circuito capaz de fazer a comunicação entre o controlador e o braço, podemos aplicar essa metodologia de especificação no que é descrito como diferentes "Níveis".

4.2 Engenharia de Sistemas e Organização

Como o projeto em questão tem como objetivo ser de fácil entendimento e alta utilidade para alunos, professores, técnicos e afins que futuramente devem utilizar o braço

robótico e sua documentação para uso didático ou até mesmo para melhorias e incrementos técnicos, entendeu-se que era necessário realizar a organização estrutural dos documentos e diretórios no ambiente Cloud do projeto.

O ambiente em cloud utilizado atualmente no projeto encontra-se no software OwnCloud do LAC. A atividade que foi guiada pelo professor Fábio Fialho durou duas semanas e durante este tempo foi replicado um projeto de Engenharia de Sistemas para organização dos diretórios e arquivos do projeto.

O objetivo de projetos relacionados a Engenharia de Sistemas é quebrar problemas em padrões. No caso o problema era a falta de organização estrutural e documentação para os arquivos e diretórios em cloud do projeto. A realização da atividade permite a coesão, padronização e organização da área de trabalho para qualquer pessoa que for utilizar o Braço Robótico, evitando confusão, perda de arquivos e re-trabalho.

Além das qualidades citadas acima, a realização de uma atividade de Engenharia de Sistemas também permite que a comunicação e integração entre diferentes áreas do projeto, como por exemplo a parte elétrica e a parte mecânica, seja simplificada e facilitada através da organização e padronização.

A estrutura atual é replicada a partir de projetos já realizados pelo professor Fábio Fialho na área de satélites, seguindo as guidelines *MIL-STD-881C* do *Departamento de Defesa dos Estados Unidos* [**?**], que se referem a "Work Breakdown Structure" (WBS), Estrutura Analítica de Projeto.

4.3 Circuito dos Atuadores

As atividades de laboratório, focadas em encontrar, documentar e propor soluções para os problemas existentes no circuito atual para os atuadores do braço são descritas a seguir.

Em termos gerais o plano de ação para atingir esse objetivo foram definidos em quatro etapas:

1. Simulações de comportamento esperado
2. Coleta de dados experimentais, replicação do erro da leitura da corrente no shunt
3. Levantamento de hipóteses da causa
4. Levantamento de soluções

4.3.1 Simulação do Comportamento Esperado

Utilizando o software NI Multisim 14.0 disponibilizado pelo Laboratório de Sistemas Integráveis da Universidade de São Paulo(LSI-USP), foram realizadas simulações sobre o circuito básico de testes, que não possui a resistência shunt e o C.I de medição desse shunt, além do circuito final, com a resistência shunt e o C.I de medição. O arquivo de simulação criado considera o Shunt e o C.I de medição de corrente LMP8601.

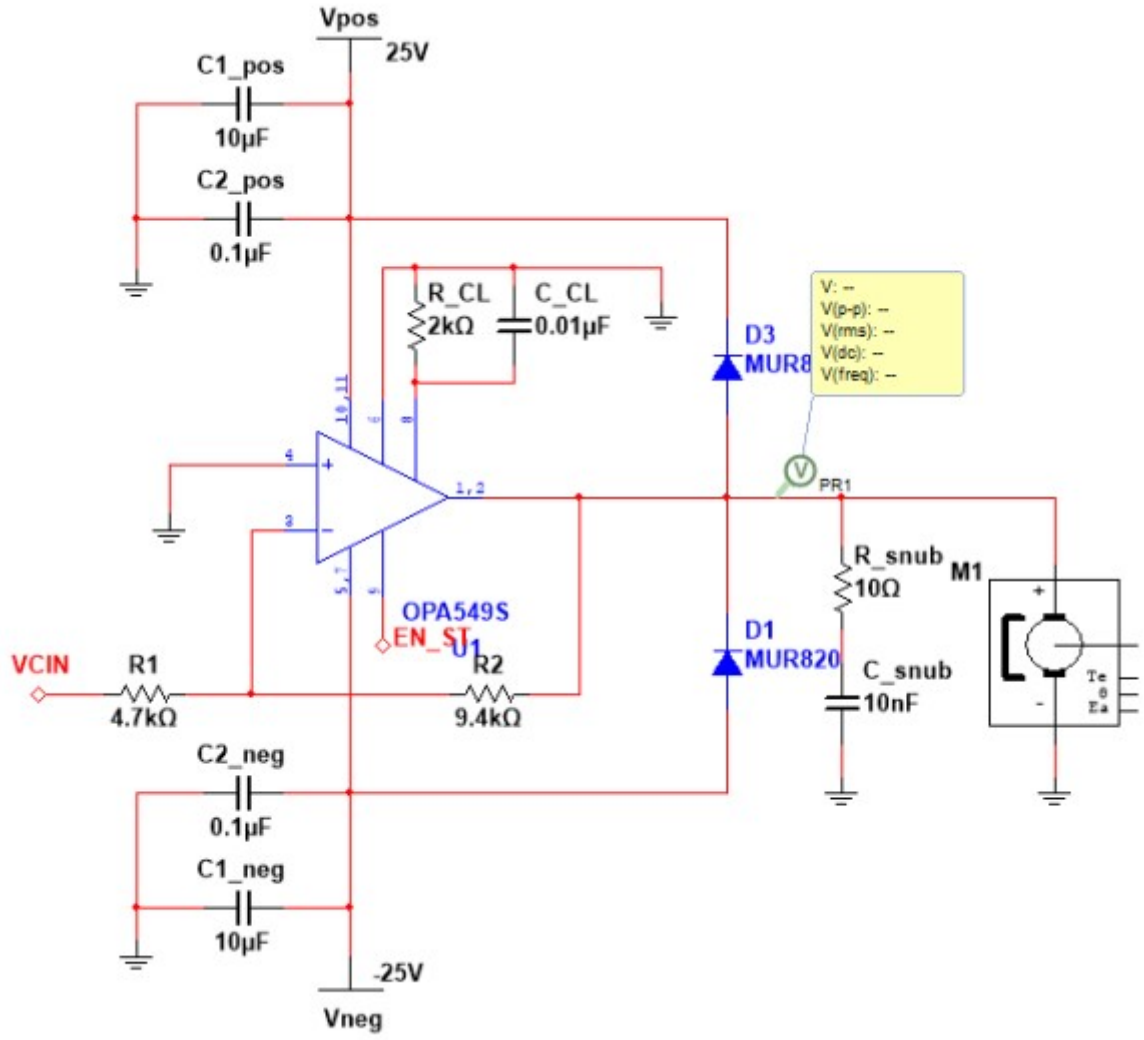


Figura 4.1: Diagrama do circuito de testes em ambiente multisim

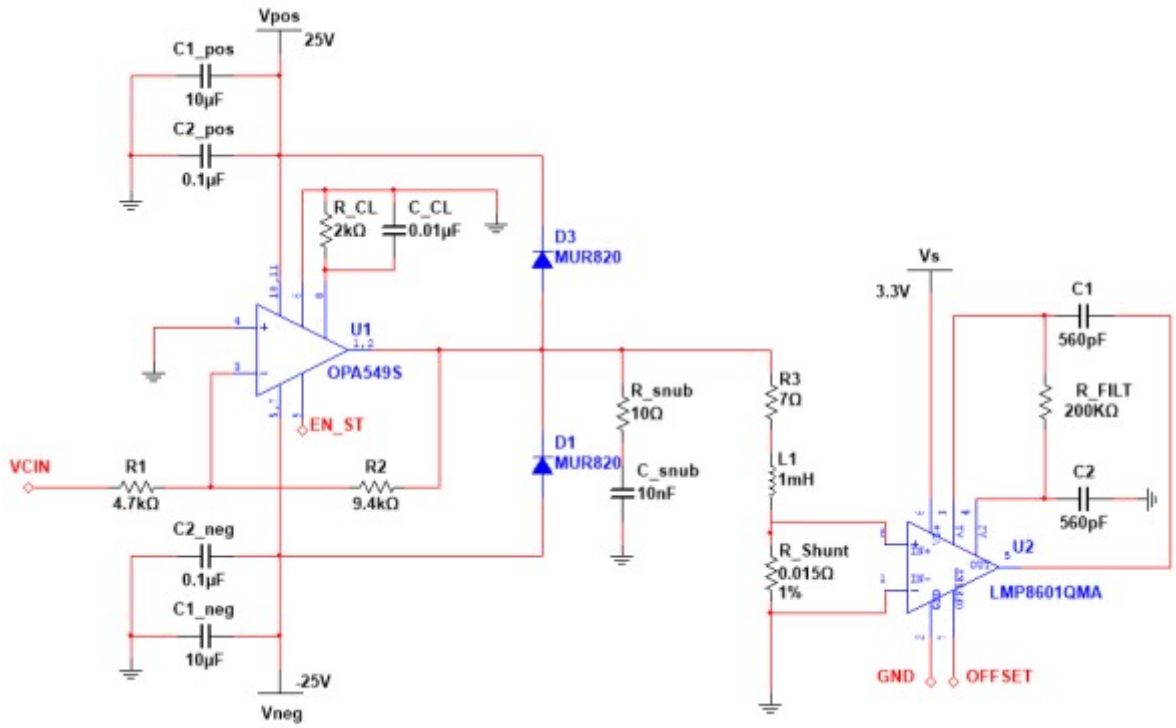


Figura 4.2: Diagrama do circuito real em ambiente multisim

De acordo com a montagem do amplificador operacional OPA549S em utilização no circuito, ele se encontra na configuração inversora e, portanto, o circuito possui um ganho que é representado pela seguinte fórmula:

$$G_o = -\frac{R_2}{R_1} \tag{4.1}$$

No circuito em bancada, no caso o circuito real, R_2 é um potenciômetro de valor máximo $20k\Omega$ e, portanto, considerando o valor fixo de R_1 no circuito, os valores máximos e mínimos de ganho a serem obtidos nesse circuito são aproximadamente:

$$-4,255 \leq G_o \leq 0 \tag{4.2}$$

Durante as simulações o ganho foi setado como fixo em $G_o = -2$, configurando portanto o valor de R_1 em $9.4k\Omega$ e R_2 em $4.7k\Omega$. Foram feitas variações na tensão de entrada V_{cin}, a tensão de saída V_{out} e a corrente de saída I_{out} foram medidas antes da resistência que simula o motor presente nas juntas do braço. Os dados de simulação do circuito podem ser encontrados na tabela abaixo:

Através dos dados obtidos que até a voltagem de entrada V_{cin} de 2V os circuitos

V_{cin} **[V]**	V_{out} **Teórico [V]**	V_{out} **sem Shunt [V]**	I_{out} **sem Shunt [A]**	V_{out} **com Shunt [V]**	I_{out} **com Shunt [A]**
1	-2	-2	-0.277	-2	-0.285
1,5	-3	-3	-0.390	-3	-0.427
2	-4	-4	-0.551	-4	-0.570
5	-10	-10	-0.771	-10	-1.43

Tabela 4.1: Dados de simulação dos circuitos em ambiente multisim

operam de forma semelhante, com desvios minimos na corrente de saída I_{out} entre eles, contudo, com 5V de entrada no circuito a corrente, quando consideramos o shunt no modelo, torna-se o dobro da corrente do modelo sem considerarmos o funcionamento do C.I de shunt.

Após uma investigação, percebe-se que essa diferença é causada pelo componente a ser utilizado em ambiente multisim para representar o funcionamento do motor. Utilizando o componente de motor *DC MACHINE PERMANENT MAGNET* disponível no software é possível configurar um ambiente de simulação mais verossímil a realidade em laboratório. Contudo, esse componente é apenas funcional no modelo de circuito sem a consideração do C.I de medição de corrente através do shunt.

Para contornar essa limitação é possível considerar o motor por sua representação mais simples como uma resistência ($R = 7\Omega$) e uma indutância ($L = 1mH$) em série, como dizem as referências em eletrônica [4], todavia, o software não parece ser capaz de realizar a simulação compreendendo o funcionamento real da resistência shunt, fornecendo uma resposta idêntica para a simulação de ambos os modelos:

V_{cin} **[V]**	V_{out} **Teórico [V]**	V_{out} **sem Shunt [V]**	I_{out} **sem Shunt [A]**	V_{out} **com Shunt [V]**	I_{out} **com Shunt [A]**
1	-2	-2	-0.277	-2	-0.277
1,5	-3	-3	-0.390	-3	-0.390
2	-4	-4	-0.551	-4	-0.551
5	-10	-10	-0.771	-10	-0.771

Tabela 4.2: Dados de simulação dos circuitos em ambiente multisim utilizando a representação do motor como um modelo R-L simples

4.3.2 Identificação de Problemas

Em prosseguimento das ações referentes ao circuito existente dos atuadores, após a simulação foram obtidos os dados experimentais e a formalização dos erros que haviam acontecendo com o circuito presente em laboratório.

- **Relato do Problema:** Através do uso do circuito dos atuadores, ao realizar a medição do valor de corrente sendo consumida pelo Motor através do CI LMP 8601-Q1, o valor obtido era: Não condizente com o valor de multímetro de bancada e, diferente entre os 6 distintos circuitos impressos em laboratório.

No laboratório, foram impressos e numerados 6 circuitos, em teoria idênticos, como o relato do problema não indicava a quantidade de circuitos com problema ou quais apresentavam o problema, logo, todos foram testados da mesma forma para a coleta de dados experimentais em bancada.

Através do relato do problema foram feitos diferentes testes em cima do circuito em bancada. Devido a estrutura de solda e montagem do circuito, é possível testar as duas funções, amplificação e leitura de corrente, de forma independente. A funcionalidade de amplificação foi a primeira a ser testada, com a intenção de verificar se existia algum problema no funcionamento da peça eletrônica mais cara que no caso é o C.I do amplificador operacional OPA549.

Para a realização dos testes, foram usados os valores de ganho $G_o = -2$ com $R_1 = 4.7k\Omega$ e $R_2 = 9.4k\Omega$ a fim de se replicar os testes realizados durante a simulação em multisim. Os resultados podem ser encontrados abaixo:

V_{cin} **[V]**	V_{out} **Teórico [V]**	V_{out} **Simulado [V]**
1	-2	-2
1,5	-3	-3
2	-4	-4
5	-10	-10

Tabela 4.3: Resultados teóricos e de simulação do amplificador operacional do circuito

V_{out} **Real - Circuito 1 [V]**	V_{out} **Real - Circuito 2 [V]**	V_{out} **Real - Circuito 3 [V]**	V_{out} **Real - Circuito 4 [V]**	V_{out} **Real - Circuito 5 [V]**	V_{out} **Real - Circuito 6 [V]**
-2,02	-1,99	-2,01	-2,00	-2,02	-1,98
-3,01	-3,04	-2,98	-2,99	-3,01	-3,01
-3,98	-4,01	-4,05	-2,02	-4	-3,96
-10,12	-9,98	-10,07	-9,91	-10,03	-10,09

Tabela 4.4: Resultados experimentais do amplificador operacional dos circuitos 1, 2, 3, 4, 5 e 6

Analisando os valores obtidos experimentalmente podemos perceber que, salvo pequenas variações, possivelmente provenientes de ruídos de montagem e conexão com aparelhos de medida como os multímetros, osciloscópios e gerador de tensão, todos os circuitos funcionam próximo ao modo esperado pela simulação e verificado pela teoria.

Através desses resultados podemos concluir que, em todos os circuitos em laboratório a funcionalidade de amplificação através do amplificador operacional OPA549 é ideal e, portanto, o problema relatado encontra-se em outra parte do circuito.

Em prosseguimento, verificada a funcionalidade de amplificação, deve ser verificada também a funcionalidade de medição de corrente através dos C.I's LMP-8601-Q1 individualmente. Estes componentes funcionam da seguinte maneira, através de uma corrente que passa pelo equipamento, gera-se uma tensão para ser lida, portanto associa-se a corrente de saída do circuito, com a tensão de saída do C.I, essa tensão de saída pode então ser enviada para leitura em um computador através de uma placa de aquisição.

Esta tensão é importante para o controle e proteção do braço robótico e seu operador em bancada, por isso ela é importante e o seu envio para o ambiente MATLAB-Simulink de controle e simulação do braço é necessária.

Para a aquisição desses dados é necessário variar a corrente e verificar os valores de tensão do C.I, portanto, pode ser montado um simples circuito com um resistor $R_{teste} = 10,3\Omega$ e então variar uma tensão de entrada $-3V \leq V_{CI} \leq 3V$ para causar uma variação na corrente do resistor $-0,25A \leq I_R \leq 0,25A$ e então verificar o funcionamento do C.I referente a essas correntes no resistor.

Além da montagem de um circuito é necessário passar uma tensão de alimentação V_s para a operação do C.I LMP-8601-Q1, existem duas possibilidades, $V_s = 3.3V$ ou $V_s = 5.0V$. Os testes foram realizados para $V_s = 3.3V$. Essa tensão de alimentação,

quando dividida pela metade devido a configuração escolhida na hora da impressão do circuito, é o valor de saída quando o C.I lê uma corrente de 0 Ampères. Portanto, caso forem necessários os valores para $V_s = 5.0V$ será necessário realizar novamente os testes já que os valores não serão aplicáveis nesta configuração.

Os resultados obtidos encontram-se na tabela abaixo:

I_R **[A]**	V_{CI} **Placa 1 [V]**	V_{CI} **Placa 2 [V]**	V_{CI} **Placa 3 [V]**	V_{CI} **Placa 4 [V]**	V_{CI} **Placa 5 [V]**	V_{CI} **Placa 6 [V]**	V_{CI} **Simulado [V]**
-0,24	1,506	1,524	0,031	-0,001	1,493	1,483	1,61
-0,2	1,529	1,545	0,032	-0,001	1,522	1,511	1,622
-0,16	1,552	1,566	0,033	0,000	1,550	1,539	1,634
-0,12	1,576	1,588	0,033	0,000	1,578	1,567	1,646
-0,08	1,599	1,609	0,034	0,001	1,606	1,595	1,658
-0,04	1,622	1,630	0,034	0,001	1,634	1,623	1,67
-0	1,646	1,651	0,035	0,002	1,662	1,651	1,682
0,04	1,669	1,672	0,036	0,002	1,691	1,678	1,694
0,08	1,693	1,693	0,036	0,003	1,719	1,706	1,706
0,12	1,716	1,714	0,036	0,003	1,747	1,734	1,718
0,16	1,739	1,735	0,037	0,003	1,775	1,762	1,73
0,20	1,763	1,756	0,037	0,004	1,803	1,790	1,742
0,24	1,786	1,777	0,039	0,004	1,831	1,818	1,754

Tabela 4.5: Resultados da leitura do CI LMP 8601-Q1 para uma corrente de um resistor de $R_{teste} = 10,3\Omega$

E na figura abaixo podemos ver os resultados que foram obtidos de forma gráfica:

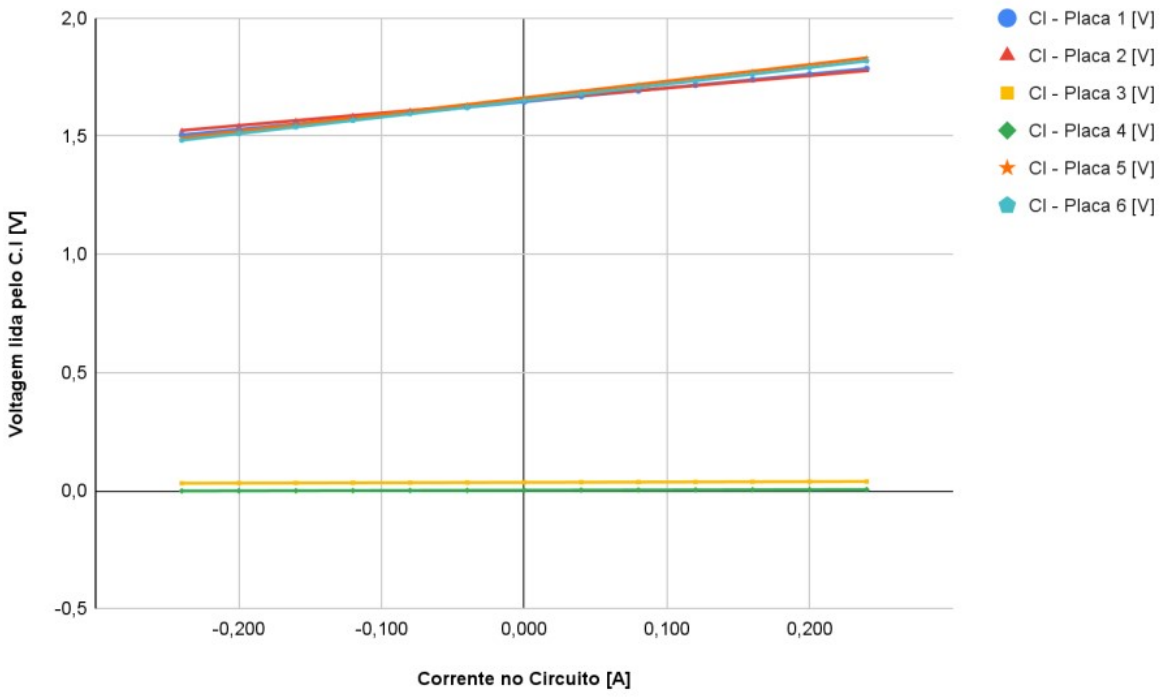


Figura 4.3: Resultados da leitura do CI LMP 8601-Q1 para uma corrente de um resistor de $R_{teste} = 10,3\Omega$

Pelos resultados obtidos, é perceptível, pela grande diferença dos valores na leitura de V_{CI}, que as placas 3 e 4 possuem algum problema. Excluindo os resultados dessas duas placas dos gráficos podemos obter uma comparação dos resultados reais com os resultados simulados:

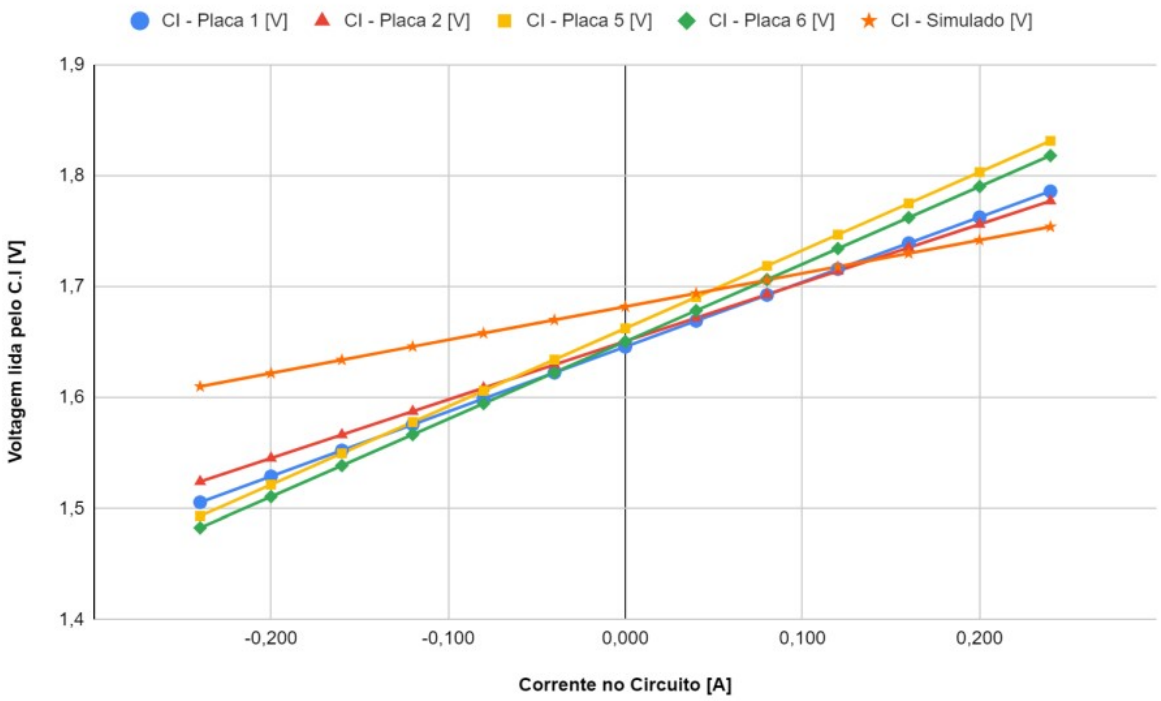


Figura 4.4: Comparação de resultados da leitura do CI LMP 8601-Q1 para uma corrente de um resistor $R_{teste} = 10,3\Omega$, simulado e experimental excluindo as placas 3 e 4

Podemos ver na figura acima o comportamento relatado do problema. Todos os C.I's funcionais estão promovendo uma leitura diferente de corrente no circuito. Excepcionalmente, descobrimos que as placas 3 e 4 estão com os LMP-8601-Q1 queimados e, através de testes de continuidade, percebemos que o problema dos C.I's não foi durante a montagem do circuito ou durante a solda, mas acreditamos que, durante testes no laboratório, devido a complexidade das conexões das placas para realizar testes, em conjunto com os diferentes equipamentos utilizados ao mesmo tempo, alguma conexão foi realizada de forma errada no próprio C.I ou algum curto foi realizado quando rodando o motor das juntas do robô e assim, os limites máximos de corrente suportados pelo equipamento foram ultrapassados e portanto eles abriram, ou "queimaram".

- **Formalização do Problema:** Devido às diferenças entre os LMP 8601-Q1 utilizados nos 6 circuitos, além das diferenças nas resistências shunt de precisão utilizadas para os circuitos, ao realizarmos a leitura da corrente do circuito pelo C.I obtivemos diferenças significativas entre eles.

- **Sugestão de Solução:** Considerando a impraticabilidade de substituição de todos os circuitos e a modelagem de outro circuito elétrico do zero, sugerimos a modelagem do comportamento de cada um dos C.Is para os circuitos sendo utilizados, assim,

sabendo o comportamento real dos dispositivos é possível fazer leituras de corrente e associá-las ao valor real esperado.

Seguindo a sugestão proposta, foram feitos os modelos lineares de cada um dos circuitos através do método da regressão linear em cima dos resultados experimentais obtidos para cada um dos C.I's:

$$\text{Placa 1: } I_s = 1,712 * V_{CI} - 2,818 \tag{4.3}$$

$$\text{Placa 2: } I_s = 1,897 * V_{CI} - 3,131 \tag{4.4}$$

$$\text{Placa 3: } I_s = 67,114 * V_{CI} - 2,349 \tag{4.5}$$

$$\text{Placa 4: } I_s = 85,47 * V_{CI} - 0,137 \tag{4.6}$$

$$\text{Placa 5: } I_s = 1,420 * V_{CI} - 2,360 \tag{4.7}$$

$$\text{Placa 6: } I_s = 1,431 * V_{CI} - 2,362 \tag{4.8}$$

Com os funcionamentos modelados foi possível então fazer a verificação final, a obtenção de dados reais de um dos motores sem carga e a obtenção de dados do C.I no ambiente MATLAB/Simulink, simulando uma utilização real do circuito:

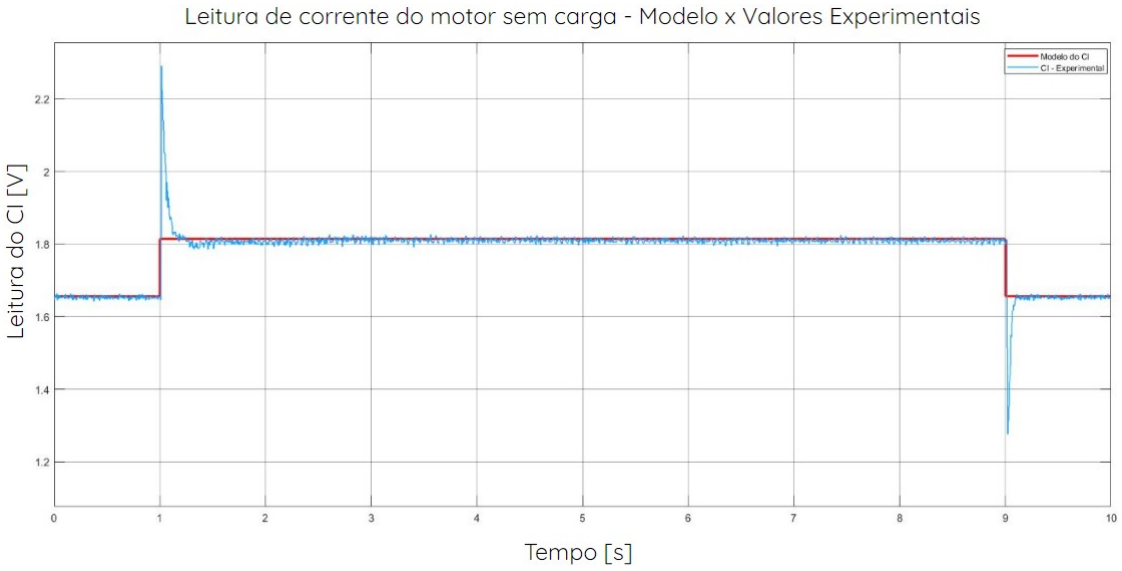


Figura 4.5: Leitura da corrente do motor sem carga pela Placa 1 - Modelo esperado x Valores Experimentais

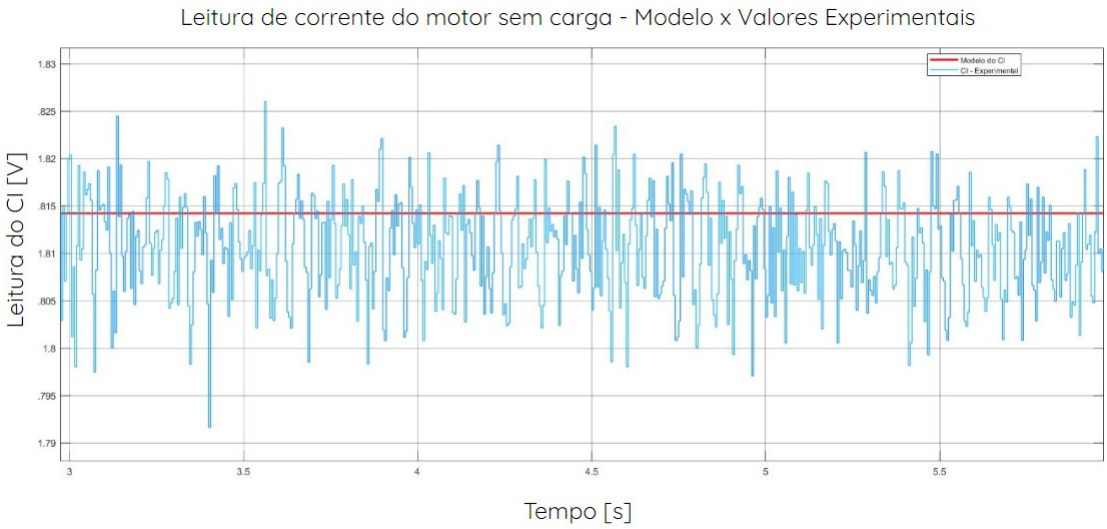


Figura 4.6: Leitura da corrente do motor sem carga pela Placa 1 - Modelo esperado x Valores Experimentais (Ampliado)

As respostas das figuras 4.6 e 4.7 acima foram obtidas utilizando a placa de aquisição NI-USB6251 integrada com o software MATLAB-Simulink. Em vermelho temos a resposta obtida do modelo linear da placa 1, em azul temos os dados reais obtidos pela leitura direta do C.I e aquisição pela placa.

Podemos observar, pela resposta ampliada que o modelo linear se aproxima bem da resposta real e, por ser um modelo linear, a principal diferença encontra-se nos momentos

de transição, quando o motor sai do estado estacionário ou tenta retornar ao estado estacionário há uma brusca mudança na tensão que o modelo não consegue replicar, contudo, em operação, ou um regime permanente, o modelo é adequado para representar o funcionamento do C.I.

Por fim, uma consideração importante sobre o circuito dos atuadores que foi percebida durante os testes em laboratório. Devido a corrente alta exigida para girar o motor, em conjunto com o amplificador operacional de alta potência OPA549, o circuito aumenta muito a sua temperatura durante a operação, caso o uso seja feito por períodos prolongados, a medida que a temperatura continuava elevando, percebeu-se, empiricamente, que a qualidade da resposta do modelo gradualmente diminuía a medida que a resposta real do C.I LMP-8601-Q1 aumentava o valor da tensão para um valor constante de corrente.

Isso leva a hipótese de que o modelo do C.I deve depender da temperatura. Como, em laboratório, não há presente um sensor de temperatura no momento atual, recomenda-se a modelagem do C.I em função da temperatura em um momento futuro do projeto.

4.4 Interface Humano-Maquina

4.4.1 Adequação da IHM para estrutura de diretórios

Devido ao trabalho de engenharia de sistemas realizado previamente, viu-se necessária a adequação do código estrutural da Interface Humano Máquina desenvolvida em MATLAB App Designer, para que ela contemple a nova estrutura de diretórios do ambiente OwnCloud.

Essa atividade se tornou uma necessidade a medida que, previamente, os "paths" no código da IHM, que indicam o caminho do diretório para os arquivos necessários para a utilização da interface, eram hardcoded para uma estrutura local diferente da estrutura utilizada em Cloud.

A solução contemplada foi a utilização de variáveis de ambiente para o sistema operacional utilizado no computador. No caso a variável criada pelo usuário na própria máquina deve possuir o nome `\DIDACTIC_ROBOT_PATH"`, caso a nomenclatura não seja utilizada corretamente a interface não irá funcionar.

Uma grande vantagem desta adaptação é que, qualquer usuário com acesso ao ambiente OwnCloud, uma vez configurada a variável de ambiente do projeto pode acessar e utilizar a Interface Humano Máquina, sem necessidade de alteração de código ou compre-

ensão técnica, aumentando assim o range de usuários que podem usar ela, indo de acordo com o objetivo de tornar o braço robótico em uma ferramenta didática.

4.4.2 Adição de Funcionalidade - Opção de graus de liberdade

No contexto atual do projeto, existem dois equipamentos diferentes, um braço robótico com 5 graus de liberdade, apelidado de 5R, e um braço robótico de 3 graus de liberdade, apelidado de RRR. Em laboratório possuímos o equipamento físico referente ao 5R porém, os arquivos de modelagem, controle e simulação para o braço RRR encontram-se parcialmente completos no ambiente cloud.

Com isso em vista, considerando que, futuramente, ambos os braços podem existir em um mesmo laboratório, viu-se a oportunidade de incrementar o design atual de Interface Humano Máquina para contemplar os arquivos já existentes para o braço com 3 graus de liberdade que ainda está em desenvolvimento.

Em termos gerais é uma adição em menor escala, considerando que os arquivos de controle e simulação estão parcialmente completos. Portanto, o que foi feito foi a preparação do ambiente para a futura simulação dos arquivos. Além da adaptação do código da interface para a compreensão do modo de operação desejado, foi feita também a importação do arquivo de modelagem 2D referente ao braço e a adaptação da interface para a utilização do braço nesta configuração RRR.

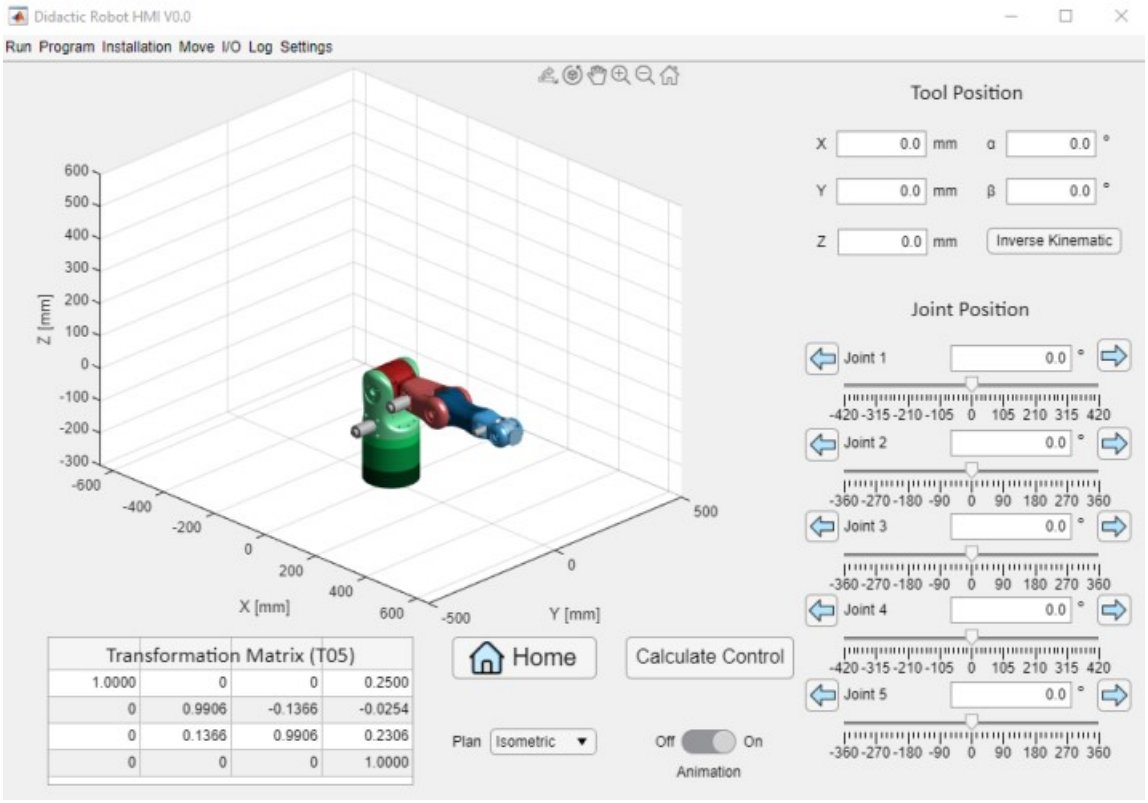


Figura 4.7: Configuração 5R inicial da IHM

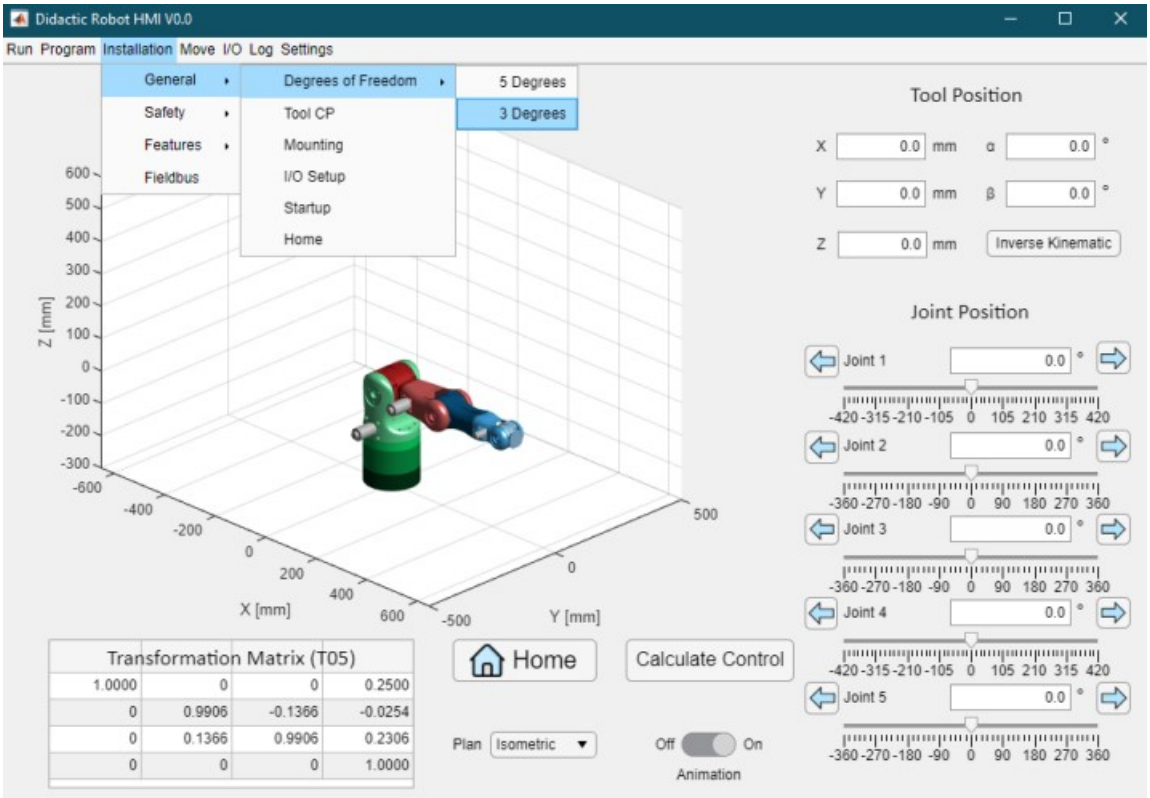


Figura 4.8: Caminho da funcionalidade adicionada

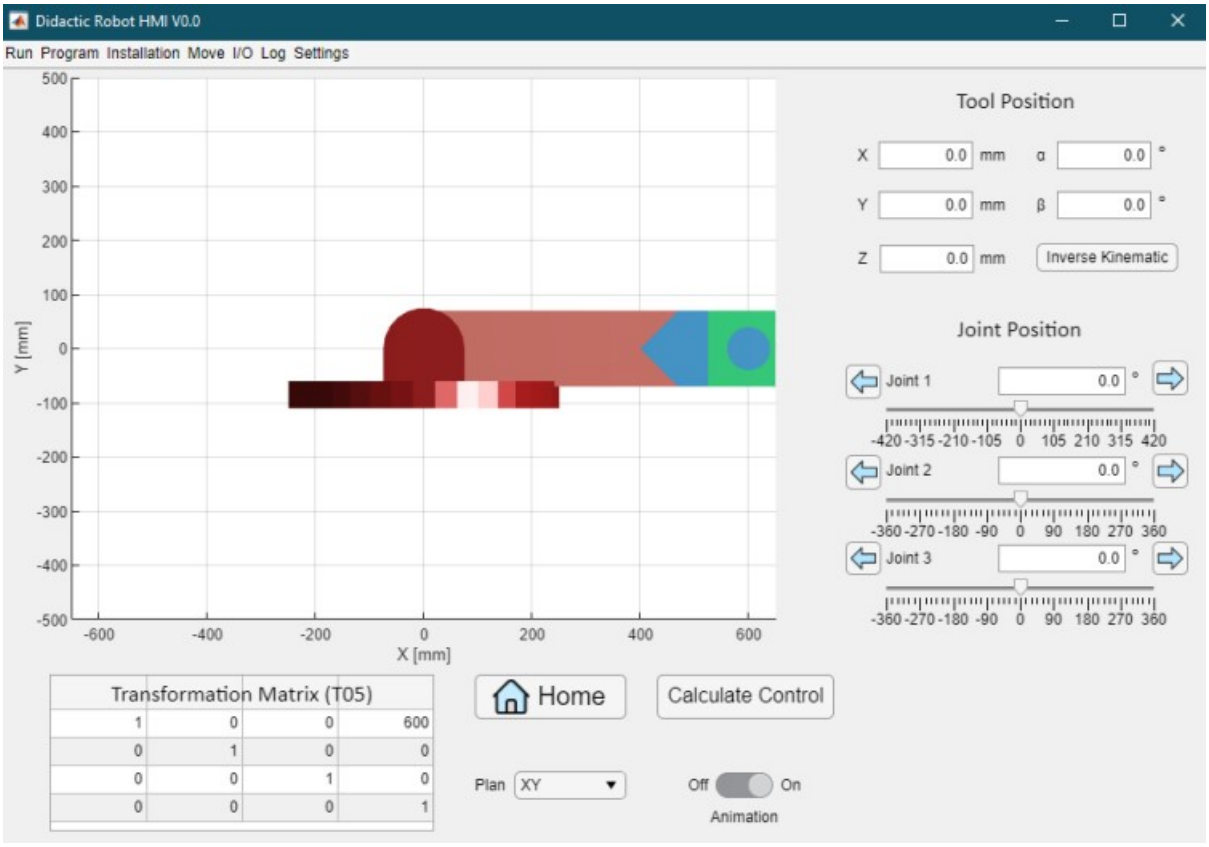


Figura 4.9: Configuração RRR da IHM

Como o modelo em 3 graus de liberdade, RRR, é 2D, a interface se adapta e força a visão do usuário para o plano XY, assim como ela se adapta a permitir apenas inputs coerentes a funcionalidade escolhida.

4.5 Especificação

4.5.1 Nível 0

Seguindo o estudo realizado previamente foi iniciada a especificação "Nível 0" ou geral. Durante a especificação geral é feita a descrição de quais entradas, saídas e funcionalidades o circuito deve possuir no geral. Além dessas descrições, também é recomendada a associação das funcionalidades das entradas e saídas definidas durante o trabalho. As informações foram compiladas em um documento descritivo em texto corrido e também por forma gráfica e visual para auxílio e compreensão do escopo do projeto.

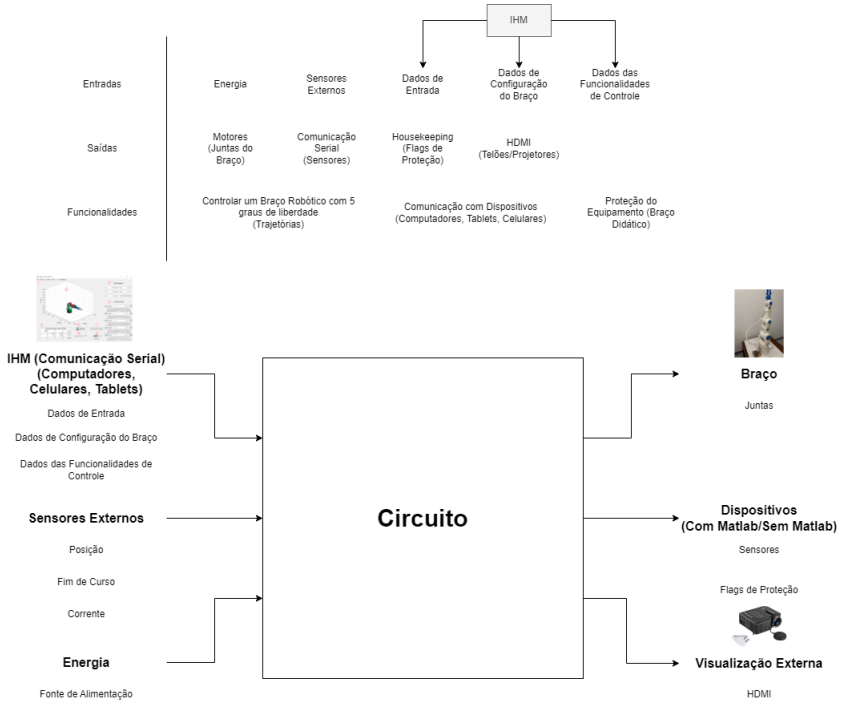


Figura 4.10: Documento gráfico de especificação em Nível 0 do circuito de controle

4.5.2 Nível 1

Em sequência as atividades realizadas previamente na especificação do circuito do equipamento foi concluída a segunda etapa na especificação da parte elétrica. Em contraste ao Nível 0, a especificação em Nível 1 é relativamente mais simples, principalmente pois é facilitada pelo trabalho já realizado e as informações desenvolvidas previamente.

De acordo com as funcionalidades que foram definidas, entendemos que a estrutura do circuito como um todo é relativamente simples, vimos que o microcontrolador a ser definido precisará ter as comunicações previstas com a interface humano máquina, seja através de módulos periféricos como Bluetooth e Wi-Fi ou internos caso já possuam.

Além disso os circuitos de energia e proteção do sistema como um todo foram decididos por serem externos aos circuito central do microcontrolador e o circuito dos atuadores, para garantir uma maior robustez ao equipamento elétrico, com uma modularização de suas funcionalidades, permitindo que eventuais problemas sejam identificados mais facilmente e o desenvolvimento futuro possa ser feito de forma paralela e independente.

De forma semelhante ao Nível 0, as informações técnicas compiladas durante esta atividade se encontram descritas em um documento de texto e também de forma gráfica para melhor compreensão sobre a situação e escopo do projeto.

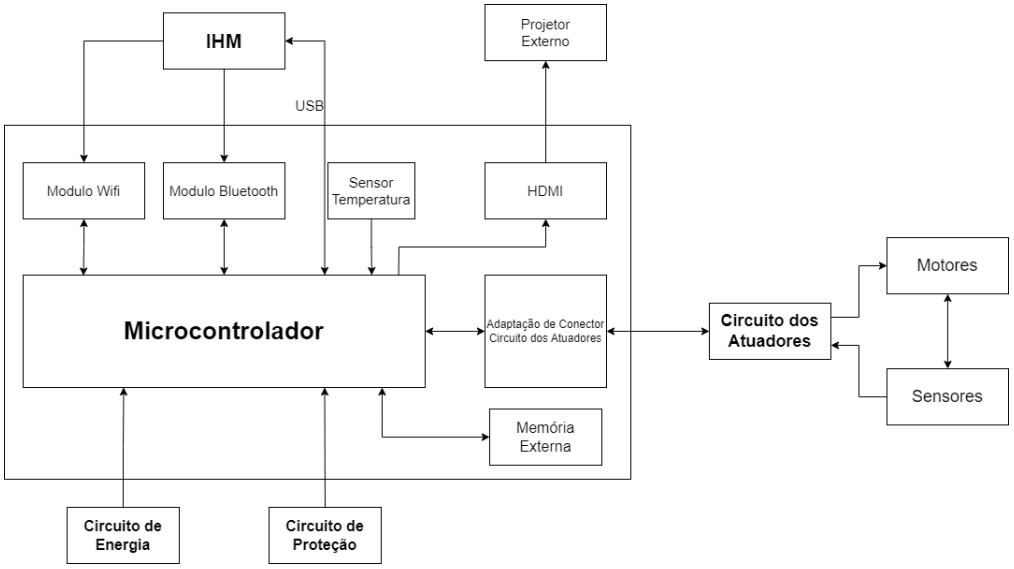


Figura 4.11: Documento gráfico de especificação em Nível 1 do circuito de controle

4.5.3 Nível 2

Definidas as necessidades gerais do circuito com a especificação em Nível 0 e Nível 1, pode-se dar início a especificação em Nível 2 do circuito, referente a escolha e definição dos componentes necessários para o circuito. Como indicam as referências, a escolha de componentes deve iniciar pelo principal, que no caso de sistemas embarcados, ainda mais no caso específico do projeto, é o microcontrolador que tem como função o controle em tempo real do equipamento, além da comunicação com a IHM em diferentes ambientes, computador, laptop, tablet e celular, com diferentes comunicações, por exemplo, bluetooth e wi-fi.

Contudo, na etapa atual do projeto, entendemos que, como existem incertezas e incógnitas referentes ao modelo do braço, como por exemplo a falta de definição dos parâmetros reais do equipamento físico, e, portanto, os requisitos para o controle e utilização do equipamento, a especificação não pode ser feita por completo. Logo, entende-se por especificação em Nível 2 no contexto deste projeto como o início desta etapa de planejamento do circuito.

4.5.4 Nível 2 - Definição da estratégia de especificação de microcontroladores

Como citado anteriormente, no contexto atual do projeto, não é prudente a escolha e definição por definitivo de componentes eletrônicos para o circuito de controle. Contudo,

é possível dar início da esta etapa, começando pela peça principal, o microcontrolador.

Como não será feita a especificação desse componente por definitivo, optou-se por listar as características desejadas no equipamento e preencher com as informações disponíveis no momento atual. Através das referências [3], foram obtidas os principais aspectos de um microcontrolador no contexto de sistemas embarcados. Em conjunto com as peculiaridades e necessidades do projeto, a atual lista de características para o microcontrolador segue na tabela abaixo: A frequência de conversão A/D e D/A com no

Característica	**Necessidade**
Comunicação	Wi-Fi, Bluetooth, USB
Sistema Operacional	Tempo Real (RTOS) - Preferência: Linux
Operação Matemática mais Complexa	Inversão Numérica de Matrizes
Velocidade de Processamento	Não Definido
Frequência de Conversão A/D	5 kHz
Frequência de Conversão D/A	500 Hz
Capacidade de Memória	Não Definido
Clock	Não Definido
Precisão da Arquitetura	Não Definido
Resistência Térmica	$\geq 60°$ C
Eficiência Energética	Não Definido
Disponibilidade	Não Definido
Custo	Não Definido

Tabela 4.6: Especificação do microcontrolador para o circuito de controle

mínimo 5 kHz e 500 Hz respectivamente, deve-se a dinâmica dos motores sendo utilizados. Como a constante de tempo de uma junta do motor é entendida por ser 10 milissegundos (10ms), a fim de realizar um controle com ao menos 10 pontos de informação, é necessária uma velocidade de controle de no mínimo 1 milissegundo (1ms). Entende-se então, que a frequência de controle ideal seria 1 kHz, para termos uma margem de segurança de 50% em caso de aumento de complexidade futura, teremos uma frequência de controle útil de 500 Hz e, portanto, uma frequência de conversão A/D de 5 kHz e uma frequência de conversão D/A de 500 Hz.

Pelos dados da tabela 4.1, fica evidente a falta de informações necessárias para se completar a especificação em nível 2 do circuito. As informações relacionadas a precisão, velocidade de processamento, capacidade de memória e clock devem ser definidas após a a evolução do projeto e identificadas através de simulações do hardware, identificação dos parâmetros reais do equipamento físico e simulações do modelo em software MATLAB-Simulink.

Através da lista de necessidades atuais, é possível traçar estratégias de especificação dos microcontroladores, no sentido de, realizar um estudo referente a tipos e modelos do equipamento que se adéquem as restrições existentes atualmente, sem necessariamente se comprometer a um específico, deixando a escolha definitiva em aberto para quando houverem mais informações sobre o projeto.

Através do estudo foi possível identificar duas estratégias diferentes, focando principalmente nas características de: Sistema Operacional de Tempo Real (RTOS) e a Conversão A/D e D/A que respeitem as frequências identificadas de 5 kHz e 500 Hz respectivamente, tendo em vista que, atualmente, são as informações mais importantes e, no contexto do projeto como um todo, são características essenciais para o microcontrolador.

4.5.4.1 Estratégia 1 - Baixo Custo

A primeira estratégia se caracteriza pela escolha de microcontroladores que atendem as necessidades desejadas de forma independente, isto é sem a necessidade da conexão com periféricos, que são acessíveis em mercado nacional e com uma faixa de preço entre R$ 10 e R$ 100, são eles: STM32F103C8T6 "BluePill", Raspberry Pi Pico e microcontroladores da família "ESP32".

Cada um dos microcontroladores possui um foco, ou uma especialidade, tendo uma performance melhor em uma área específica. A STM32 "BluePill"é um microcontrolador versátil que suporta diferentes sistemas operacionais em tempo real: Azure, FreeRTOS, Linux, permitindo maior escolha e sendo uma escolha menos restritiva no contexto do projeto.

Por outro lado, a Raspberry Pi Pico possui uma maior capacidade de processamento, através de um processador ARM Cortex M0+ com um clock mais rápido, comparado ao da STM32 de 72 MHz, com uma velocidade máxima de 133 MHz, mais capacidade de memória e com mais conversores A/D e D/A internos ao microcontrolador. Essas características se encontram em detrimento de uma menor escolha dentre sistemas operacionais e uma faixa de preço maior.

Por fim, a família de microcontroladores ESP32 se destacam pela compatibilidade com diferentes módulos de comunicação Wi-Fi, sejam externos como periféricos, ou internos ao próprio microcontrolador. Além de diferentes kits de desenvolvimento disponíveis tanto para comunicação Wi-Fi quanto Bluetooth, é uma escolha que permite a realização de testes de comunicação mais complexos com o equipamento. Em contrapartida, estes microcontroladores possuem uma capacidade de processamento menor, além do fato dos conversores A/D e D/A possuírem uma resolução menor comparados ao da STM32 e da Raspberry Pi, o que se traduz para ruídos maiores nos sinais enviados e no controle do braço.

Estratégia 1			
Característica	**STM32 "BluePill"**	**Raspberry Pi Pico**	**ESP32**
Processador	ARM Cortex M3 72 MHz 32 bits	Dual Core Cortex M0+ 133 MHz 32 bits	Xtensa Dual Core 32 bits 160-240 MHz
Sistemas Operacionais	FreeRTOS, Azure, Linux	FreeRTOS, Linux	FreeRTOS
Conversor A/D	2x - 12 bits Resolução Mínima $0{,}2\mu s/sample$	3x - 12 bits Resolução Mínima $2\mu s/sample$	3x - 8bits Resolução Mínima $5\mu s/sample$

Tabela 4.7: Comparação entre as características de microcontroladores da estratégia 1

4.5.4.2 Estratégia 2 - Alta Performance

Para a segunda estratégia definida, o foco foi estudar microcontroladores de alta performance, levando em consideração que o preço e a disponibilidade seriam menores, a complexidade de se trabalhar e configurar, em contrapartida, garantindo que os requisitos sejam atendidos com margens grandes de folga e altíssima confiabilidade no produto adquirido. No caso da escolha de um desses modelos listados nesta estratégia, espera-se que mesmo com o progresso e aumento da complexidade do projeto e de suas características, o equipamento escolhido não terá problemas de operação e performance.

Os microcontroladores estudados para a definição desta estratégia se encontram na família C2000 de microcontroladores da empresa Americana Texas Instruments, conhecida e confiável dentro da área de engenharia. O outro equipamento eletrônico encontrado foi o Zybo Z7, confeccionado pela Diligent, uma empresa associada a National Instruments, outra empresa confiável do ramo de engenharia.

Como comentado anteriormente, esses microcontroladores possuem uma melhor per-

formance, no caso da família C2000 por exemplo é oferecida uma grande variedade de produtos, alguns possuem maior capacidade de memória, outros possuem microcontroladores mais rápidos, até 900 MIPS (Millions of Instructions per Second). Contudo, o preço inicial desses equipamentos e suas placas de desenvolvimento começam em US$ 50 chegando a possíveis US$ 300 dependendo do modelo.

Similar aos microcontroladores da Texas Instruments, as ZYNQ Boards, como são chamados esses eletrônicos da Diligent, são opções semelhantes, na mesma faixa de preço e oferecem um range maior de periféricos e kits de desenvolvimento porém a custo de maior complexidade.

Estratégia 2		
Característica	**C2000 Texas Instruments**	**ZYBO Z7 Diligent**
Processador	2x C28x 2x CLA 200 MHz 32 Bits	Dual Core Cortex A9 667 MHz 32 bits
Sistemas Operacionais	TI-RTOS	FreeRTOS, Linux
Conversor A/D	4x - 12 ou 16 bits 3.5 MSPS (12 bits) 1.1 MSPS (16 bits)	4x - 12 bits 1 MSPS

Tabela 4.8: Comparação entre as características de microcontroladores da estratégia 2

4.6 Simulação de Hardware

Um dos principais objetivos com o trabalho proposto, além de realizar a especificação do do circuito de controle, em conjunto com a especificação inicial do microcontrolador, é estar realizando uma simulação inicial do hardware especificado. Através dessas simulações, no contexto do projeto, temos por objetivo obter mais informações sobre a operação do modelo do braço através do hardware e, principalmente, informações sobre a usabilidade do hardware, no sentido de: Utilização de memória para o código de controle e Utilização da capacidade de processamento (carga de processamento).

Estas informações obtidas através da simulação realizam um papel importante, além de auxiliar no avanço da especificação de hardware, também, em uma forma logística do projeto, auxiliam ao evitar retrabalho e reinvestimento em componentes eletrônicos, um problema que pode ser frequente ao se trabalhar com projetos em larga escala de sistemas embarcados e projetos de engenharia como um todo. Através das simulações futuras, espera-se que o circuito por completo seja simulado em ambiente digital, uma

vez funcional, o investimento para adquirir as peças e componentes eletrônicos, além da montagem do circuito pode ser realizado com maior certeza e menor retrabalho esperado. Todas as simulações do hardware do circuito de controle foram realizadas no software Proteus Design Suite v8.13.

A figura abaixo contempla os passos de simulações factíveis neste projeto antes da realização da montagem final, em destaque vermelho encontra-se a etapa de simulação descrita e iniciada neste projeto. Vale destacar, a primeira etapa, de configuração e montagem da IHM está parcialmente concluída, assim como a etapa de comunicação do simulink com o equipamento físico, ambas devem receber avanços futuramente.

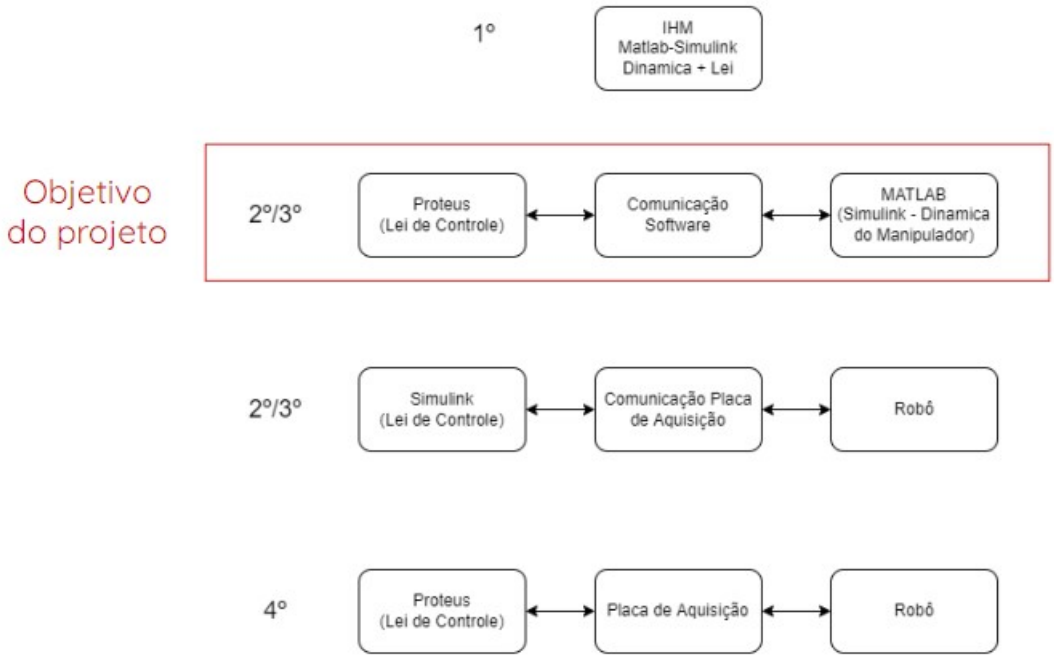


Figura 4.12: Etapas de simulação do projeto

4.6.1 Estratégia de simulação

O objetivo das simulações é obter informações de funcionamento do hardware em um contexto de operação de controle do braço robótico. O software de simulação de hardware escolhido foi o Proteus, por possuir a capacidade de simular os hardwares especificados anteriormente, além de permitir o design de placas de circuito impresso (PCB) e também conseguir simular circuitos e componentes eletrônicos de forma semelhante, porém menos intensa, ao software Multisim utilizado em outras etapas desse projeto.

Contudo, como os modelos desenvolvidos para o braço encontram-se no ambiente

MATLAB-Simulink como comentado anteriormente e como visto no precedente a este trabalho, é necessário encontrar uma forma de conexão entre ambos os softwares, a fim de evitar o retrabalho de modelar o braço em ambiente Proteus, uma função que não é de certeza se é possível neste software de simulação de hardware.

O protocolo definido para a comunicação entre ambos os softwares neste trabalho foi a comunicação Serial virtual UART. Ambos os softwares possuem outros métodos de comunicação disponíveis que devem ser possíveis de serem utilizados em conjunto, como por exemplo TCP/IP. Todavia, este protocolo foi escolhido para o projeto por ser um método de comunicação mais simples e comprovado, dentro da comunidade de usuários do Proteus, como um método válido para a comunicação com o software do MATLAB.

Um último software é necessário para realizar a integração entre o simulador de hardware e o simulador de controle. A comunicação serial virtual em uma mesma máquina é realizada através da conexão, no caso o pareamento, de portas COM físicas, do próprio equipamento ou máquina disponível que estará rodando os softwares, ou de portas COM virtuais, criadas por outros softwares que possuem essa capacidade. Existem diversos softwares, OpenSource, ou privados, que possuem a característica da criação e pareamento de portas seriais que são necessárias para a simulação do projeto.

No caso, o software "Null-modem emulator (com0com)"foi utilizado para este projeto, por ser um software OpenSource, que atende as características necessárias. Existem outros softwares que foram utilizados também, como o "Virtual COM Port Driver"em sua versão de licença grátis por 14 dias para a realização das simulações iniciais e prova de conceito com o Arduino.

Definidos então: o software de simulação de hardware (Proteus), o software de simulação de controle (MATLAB-Simulink), o protocolo de comunicação entre os softwares (Comunicação Serial Virtual UART) e o software de pareamento e criação de portas seriais virtuais (com0com), a última etapa foi a definição do hardware a ser simulado. No total foram simulados dois hardwares em ambiente Proteus, um Arduino UNO e a STM32F103C8T6 "BluePill"especificada para este projeto.

O ambiente proteus fornece módulos de simulação para o Arduino UNO, simplificando o seu uso e tornando-o compreensível para testes, contudo, foi visto durante a etapa de especificação que este hardware não atende as características necessárias do projeto e, portanto, foi apenas utilizado para uma prova de conceito e verificação da integração entre os softwares, se é possível e como que ela se comporta.

As simulações com objetivo de obter informações sobre a especificação e operação

do hardware foram feitas na STM32 especificada, já que ela atende as características do projeto. Contudo, a simulação deste software não é facilitada por módulos assim como o Arduino, e, portanto, é necessária a configuração do hardware em ambiente proteus sem configurações padrão. Além disso, a criação do código a ser rodado pelo hardware do Arduino pode ser feita diretamente em ambiente Proteus, com ferramentas de debugging, assim como uma interface de desenvolvimento (IDE) comum, porém, para a simulação da STM32, é necessário realizar a configuração do hardware e a compilação e geração do código na IDE própria "STM32CubeIDE"e então este codigo deve ser compactado e exportado em formato ".hex"para que o Proteus possa ser capaz de interpretar e compilar este código, no caso sem as ferramentas de debugging durante a simulação.

Portanto, a simulação da STM32 é mais complexa do que a simulação de prova de conceito feita com o Arduino porém ela é a que gera as informações pertinentes para o projeto, além de permitir a comunicação em tempo real entre MATLAB-Simulink, algo que não foi possível realizar com as simulações em Arduino UNO.

De forma gráfica, a estratégia de simulação encontra-se na figura abaixo para compreensão simplificada do texto:

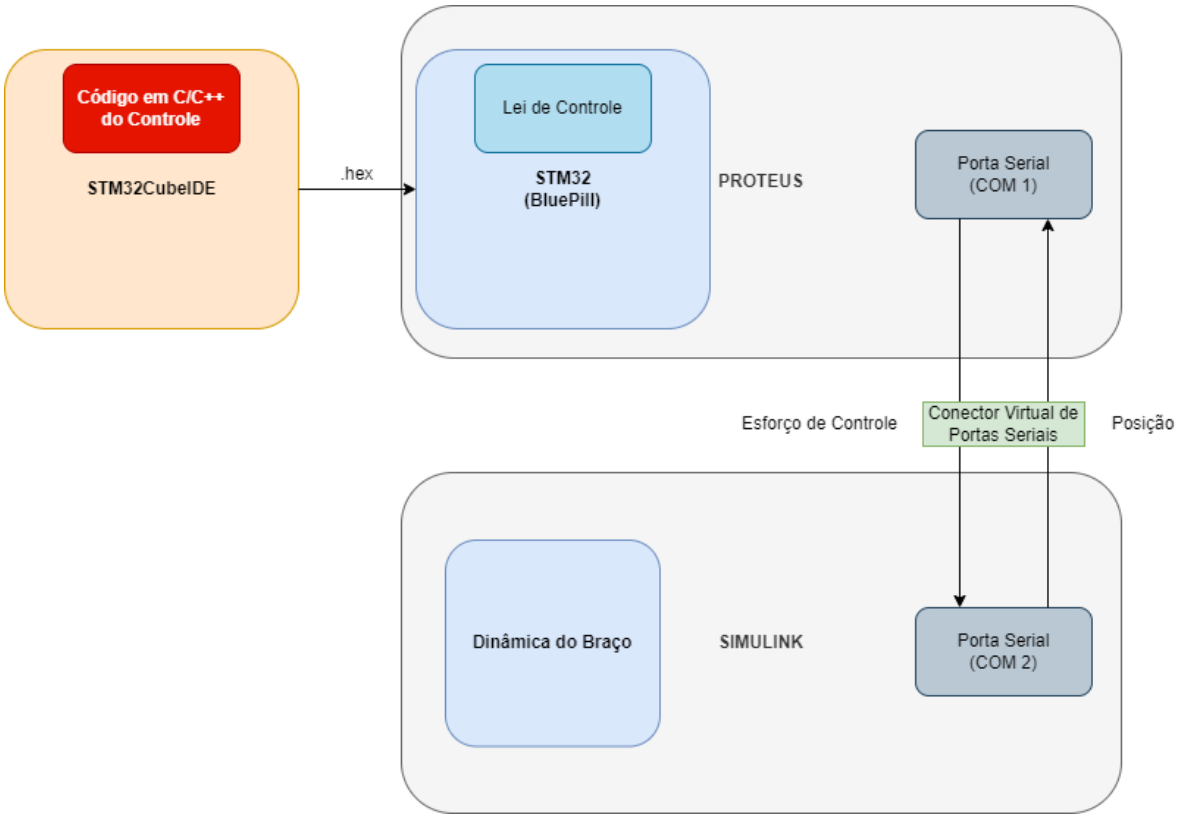


Figura 4.13: Estratégia gráfica de simulação entre os ambientes Proteus e MATLAB-Simulink para a simulação do hardware

A simulação da dinâmica do braço também foi simplificada para um modelo de primeira ordem com uma constante de tempo $\tau = 10$ $(10Hz)$ para simular o funcionamento do motor de modo mais simples. O modelo utilizado nas simulações representava apenas uma junta do robô porém é possível realizar a simulação de mais juntas concomitantemente.

Além de ser um modelo de primeira ordem, é importante que o modelo seja discreto considerando que o controle a ser realizado nesta etapa é um controle digital através da comunicação serial. Uma informação importante, como a dinâmica do motor é $10Hz$, é importante que o período de amostragem (T_s) do modelo seja no máximo $T_s = 0.1s$, caso seja maior a informação gerada não será pertinente ao projeto tendo em vista que não simula de modo apropriado a dinâmica do motor.

$$\text{Modelo de Primeira Ordem utilizado: } \frac{1}{s+10} \tag{4.9}$$

$$\text{Modelo de Primeira Ordem Discreto utilizado: } \frac{0.06321}{z-0.3679},\ T_s = 0.1 \tag{4.10}$$

As seções abaixo discutem sobre como foram realizados e como replicar as simulações, tanto do Arduino UNO quanto da STM32F103C8T6 em Ambiente proteus com integração ao MATLAB-Simulink.

4.6.2 Comunicação com MATLAB - Uso do Arduino

Para dar inicio das simulações, foi necessário primeiro a realização de uma prova de conceito, para verificar a funcionalidade da comunicação entre MATLAB-Simulink e o ambiente Proteus. Como citado anteriormente, estas simulações iniciais foram feitas com o Arduino UNO como o hardware de simulação, por ser mais simples de configurar e rodar no software.

Para realizar a comunicação de maneira útil as seguintes etapas devem ser realizadas, não necessariamente na ordem listada abaixo:

1. Configurar do hardware em ambiente Proteus;
2. Criar do código de controle a ser simulado no hardware;
3. Configurar do ambiente Proteus para o envio e recepção de dados seriais;

4. Configurar do ambiente MATLAB-Simulink para o envio e recepção de dados seriais;

5. Configurar das portais seriais virtuais;

Estes passos identificados devem ocorrer em todas as simulações de hardware no Proteus com integração ao MATLAB-Simulink, com variações dependendo do tipo de hardware ou tipo de simulação desejada. Para a simulação do Arduino UNO, por ser mais simples, segue os passos básicos sem variações.

4.6.2.1 Arduino - Configuração do hardware no Proteus

O primeiro passo de simulação reflete pela configuração do próprio hardware, ou microcontrolador no nosso caso, no software. Este passo é necessário pois, sem indicar ao software qual o tipo exato de hardware queremos simular não será possível obter informações pertinentes sobre os testes realizados e sobre a operação do microcontrolador.

Quando se fala sobre a configuração do hardware não é apenas a inserção do modelo desejado na área de trabalho do Proteus, é necessário indicar as características que desejamos replicar em hardware, como por exemplo: Velocidade/Frequência do Clock, Taxa de Transferência de Dados (Baud Rate), Flags de utilização para debugging, Pacote de simulação. Algumas dessas informações, dependendo do hardware escolhido e a compatibilidade com o Proteus podem ser feitas diretamente em código, em outros é necessária a configuração do componente.

No caso do Arduino UNO, devido ao módulo de compatibilidade presente no Proteus, existem configurações padrão para essas características que não foram alteradas, as duas mais importantes utilizadas para os testes foram:

- **Baud Rate (Taxa de Transferência de Dados)** 9600 bits/s;
- **Frequência do Clock** 16 MHz;

As outras configurações foram mantidas como padrão.

4.6.2.2 Arduino - Código de controle

O desenvolvimento do código de controle em ambiente Proteus pode ser feito de duas maneiras, caso o software possua módulos de compatibilidade com o microcontrolador escolhido, ao criar um projeto de simulação o próprio Proteus permite a edição e criação

do código através de uma interface de desenvolvimento interna, como é o caso do Arduino UNO. Caso não haja a compatibilidade padrão, é necessário criar o código em uma IDE própria que haja compatibilidade com o hardware escolhido e então exportar para um formato de arquivo ".hex", nesse caso, escolhido um modelo padrão de simulação no Proteus, é possível inserir o arquivo exportado para que o software consiga interpretar.

O código de controle no caso do Arduino UNO é escrito na Linguagem de Programação do Arduino, uma versão especializada de C++, com métodos e funções próprias, além de um método de escrita diferente. Devido ao volume extenso de utilização do Arduino nas comunidades de engenharia e eletrônica, não se torna um problema a criação ou adaptação de códigos para a realização de funções específicas. É recomendado a busca por códigos e métodos já existentes para simplificar o trabalho do usuário.

Para a simulação realizada em malha aberta, envia-se um degrau unitário como informação serial para o modelo do motor em Simulink através das funções internas a linguagem de programação: *Serial.write()* e *Serial.print*.

Para a simulação em malha fechada, utilizou-se um modelo de controle proporcional simples com ganho proporcional $K_p = 2$, para a recepção de dados do Simulink, a fim de fechar a malha de controle, foi utilizada outra função interna: *Serial.read()*.

Além da escrita e definição do código em sí é necessária a configuração da mensagem serial, tanto no envio quanto na recepção, além da definição do tipo de dado a ser enviado. Boas práticas sugerem o envio de mensagens com "cabeçalho"(header"e "terminador"(terminator), assim como o envio de bits de paridade e a informação de CheckSum para assegurar que a informação enviada não foi perdida no meio do caminho (packet loss). A comunicação entre MATLAB-Simulink e um Arduino de modo serial já foi feita em outros estudos tanto com o hardware físico quanto com o hardware simulado [5].

As características das mensagens enviadas e recebidas nos testes realizados com o Arduino UNO encontram-se abaixo. Elas foram desenvolvidas a fim de garantir a transferência de informação do modo mais rápido possível, evitando a perda de informações:

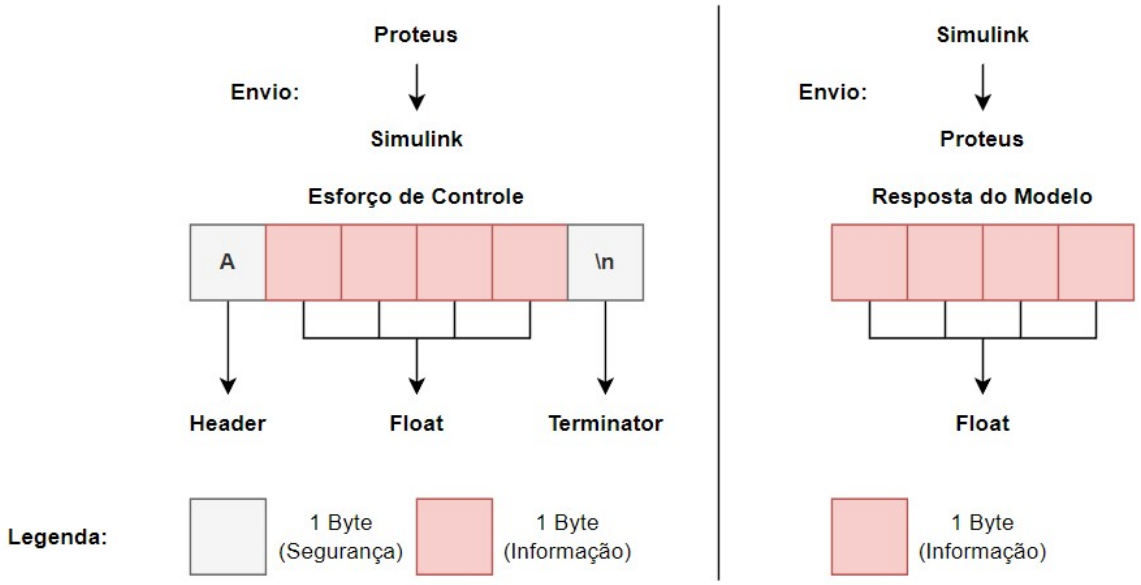


Figura 4.14: Mensagens enviadas pelo Arduino. Esquerda, envio do esforço de controle do Proteus para o MATLAB-Simulink. Direita, envio da resposta do modelo do MATLAB-Simulink para o Proteus

A principal diferença entre ambas as mensagens é que, para o envio de dados do Proteus em direção ao Simulink, é adicionado um header padrão com o caractere "A"e um terminator padrão com o caractere "\n"que se refere ao comando de "pular linha". No caso ele é apenas necessário para o envio de mensagens e não para a recepção de mensagens pois ela é a responsável por fazer o "handshaking"ou a sincronização entre os dois softwares, neste caso ela não está incluída para garantir a integridade da mensagem. Devido a isso, a simulação é feita de modo que o Proteus atua como "Leader"e o MATLAB-Simulink como "Follower".

Por fim, a informação enviada é um float, ou single pela nomenclatura utilizada pelo Simulink, um formato de dados que guarda informação de números flutuantes até 17 casas decimais de precisão em 4 bytes de dados, no caso 32 bits. O envio dessa informação é interessante pois permite um controle refinado do modelo, a informação não é tão precisa quanto um formato double, ou long double porém é mais do que suficiente para a obtenção de uma resposta útil.

4.6.2.3 Arduino - Configuração do Proteus para envio e recepção de dados seriais

Após feita a configuração do hardware e criado o código de controle, deve ser então criada a configuração, no ambiente de trabalho Proteus, do envio e recepção da comunicação serial. Essa configuração se dá através de um modelo de componente próprio chamado "COMPIM". Este é um modelo de interface física da porta serial, como o próprio software referencia *"Physical Interface Model - PIM"*.

Dados seriais de entrada são direcionados a um buffer a partir desse componente, e enviados para o circuito como um sinal digital. Além de permitir a comunicação, ele auxilia no método de handshaking para a comunicação entre diferentes máquinas, permitindo a configuração da Baud Rate e Paridade dos dados por exemplo, possibilitando também a comunicação com equipamentos físicos e virtuais através das portas seriais.

Para a configuração do componente, no caso de utilização do Arduino, é necessária a utilização de um componente COMPIM, através da porta serial "COM1", com 8 bits de dados, sem paridade atrelada, com Baud Rate virtual de 9600 bits/s. É importante que a informação da taxa de transferência de dados seja idêntica a que é utilizada no hardware, caso contrário existirão erros de simulação.

Além disso, é importante realizar as conexões corretas no componente. As portas "RXD"e "TXD"são referentes a "Receive Data"(Recepção de dados) e "Transmit Data"(Transmissão de dados). Através da comunicação Serial UART sendo utilizada, o envio de dados de uma máquina se conecta a recepção de dados na máquina conectada e vice versa, por isso é importante manter a conexão correta, caso contrário as simulações não serão funcionais.

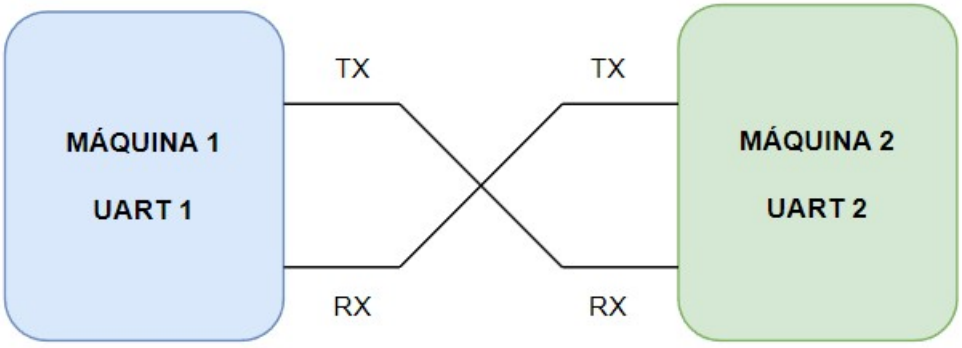


Figura 4.15: Diagrama de funcionamento da comunicação Serial UART

Dependendo do microcontrolador e o hardware escolhido, existirão opções de debug-

ging internas ao próprio software, como é o caso do Arduino UNO, caso contrário é necessário desenvolver outras estratégias para o debugging. Uma ferramenta útil a ser considerada para essa tarefa é o monitor serial, um componente próprio do Proteus que simula um monitor das mensagens sendo enviadas, mostrando ao usuário as informações transmitidas e recebidas.

4.6.2.4 Arduino - Configuração do MATLAB-Simulink para envio e recepção de dados seriais

A configuração do ambiente MATLAB-Simulink pode ser feita antes ou depois da configuração do Proteus. Em comparação com a configuração do hardware ela é mais simples pois depende da criação do diagrama de blocos que consiga fazer: recepção dos dados seriais virtuais, transmissão de dados seriais virtuais e a simulação do modelo.

Para conseguir fazer a integração, é necessária a utilização dos seguintes toolboxes (módulos) do software: Instrument Control Toolbox, Simulink Desktop Real-Time, Embedded Coder, pois blocos desses módulos são utilizados.

A configuração e utilização do modelo discreto é feita como sugerem as referências de controle digital [6], e a configuração das portas seriais, o envio e recepção de dados foi feita de acordo com as referências de integração entre Arduino e MATLAB-Simulink [5].

4.6.2.5 Arduino - Configuração das portas seriais virtuais

Este passo diferencia-se dependendo do software utilizado para a criação e pareamento das portas seriais virtuais. Como via de regra, porém, é necessário que tanto em ambiente Proteus quanto em ambiente MATLAB-Simulink, as portas criadas e pareadas estejam configuradas corretamente. Por exemplo, caso as portas criadas e pareadas sejam as portas "COM1"e "COM2", uma delas deve ser utilizada apenas no Proteus enquanto a outra deve ser utilizada apenas em MATLAB, caso contrário a comunicação não ocorrerá.

4.6.2.6 Arduino - Resultados da simulação em Malha Aberta

Feitas as configurações, a simulação pode ser realizada. Nos apêndices [A], [B] e [C] encontram-se, respectivamente, o diagrama do circuito em ambiente proteus, o código utilizado para a simulação em malha aberta do arduino e o diagrama de blocos em ambiente simulink. As figuras abaixo mostram as respostas obtidas da simulação em malha aberta.

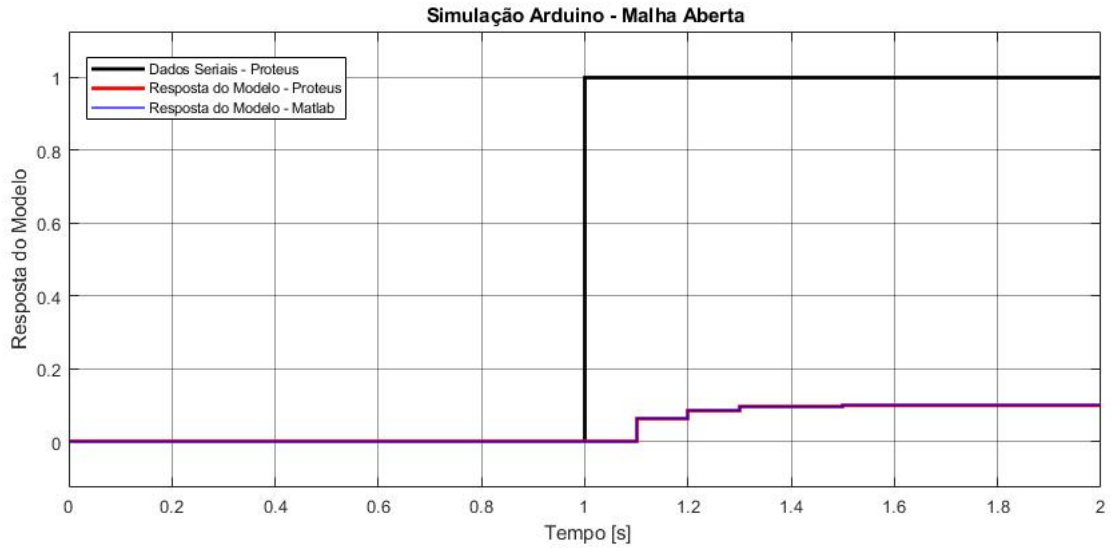


Figura 4.16: Simulação de hardware. Arduino UNO, Malha aberta.

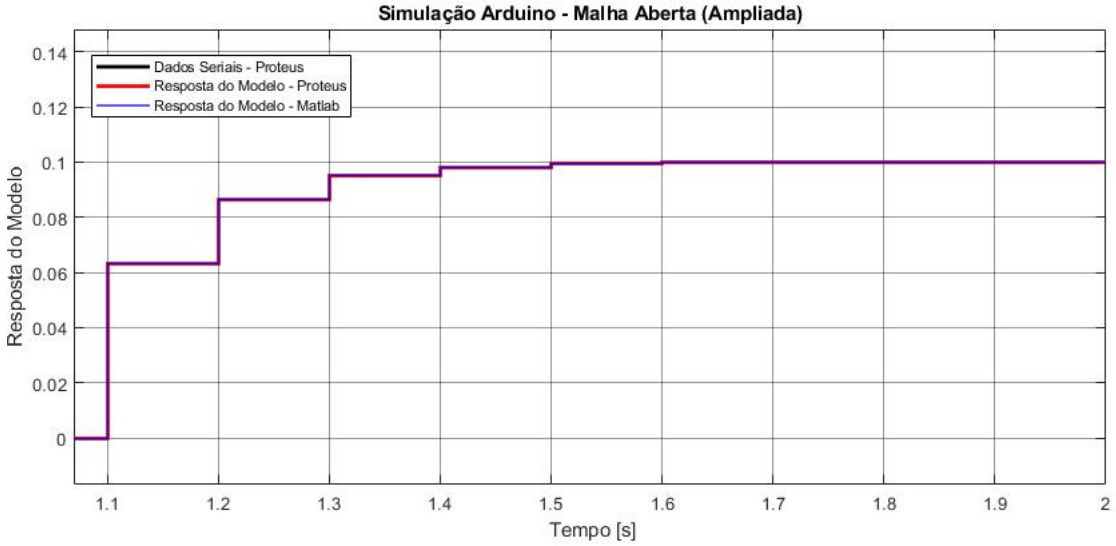


Figura 4.17: Simulação de hardware. Arduino UNO, Malha aberta, ampliada.

Através da comparação da resposta ideal, em azul, simulada completamente em MATLAB-Simulink, e da resposta obtida através da comunicação Proteus-Simulink, podemos perceber que a transmissão de dados foi feita de maneira correta, sem que ocorresse a perda de dados ou a interpretação incorreta de informações.

4.6.2.7 Arduino - Resultados da simulação em Malha Fechada

Nas figuras abaixo seguem os resultados da simulação feita em malha fechada com o Arduino UNO. Nos apêndices [A], [D] e [E] encontram-se, respectivamente, o diagrama do circuito em ambiente proteus, o código utilizado para a simulação em malha aberta do arduino e o diagrama de blocos em ambiente simulink:

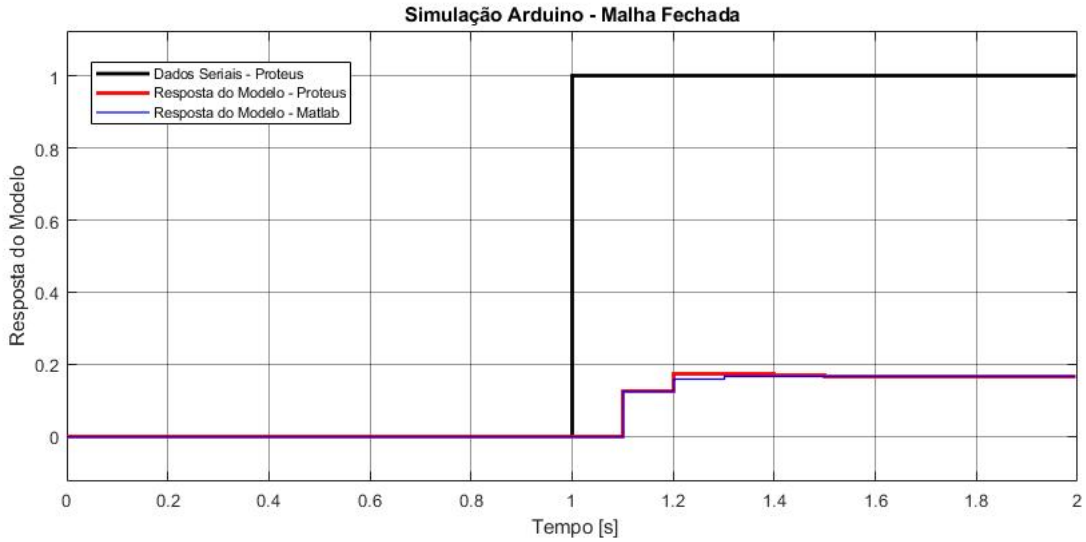


Figura 4.18: Simulação de hardware. Arduino UNO, Malha fechada.

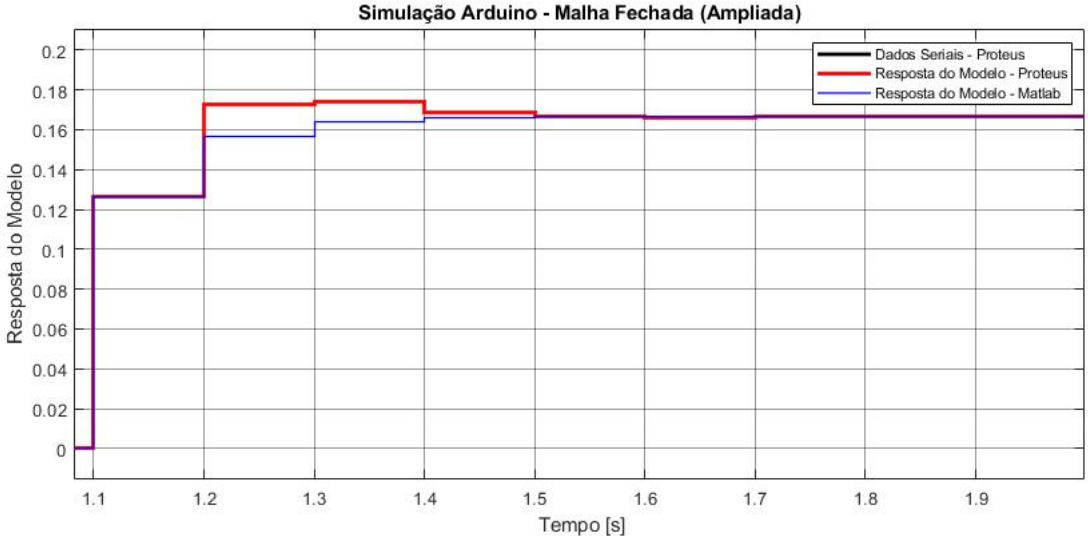


Figura 4.19: Simulação de hardware. Arduino UNO, Malha fechada, ampliada.

Através da comparação da resposta ideal, em azul, simulada completamente em

MATLAB-Simulink, e da resposta obtida através da comunicação Proteus-Simulink, podemos perceber que a transmissão de dados foi feita de maneira correta, sem que ocorresse a perda de dados ou uma interpretação incorreta.

Contudo, também é perceptivel através das imagens que há um erro entre ambas as respostas mesmo utilizando o mesmo controle. Esse erro não se deve a falha na transmissão mas sim na precisão de envio dos dados, como citado anteriormente, os dados enviados do simulink para o proteus tem o formato "float", ou "single"em nomenclatura Mathworks. Enquanto a simulação feita com o modelo puramente em simulink é feita com dados em formato "double", a conversão entre formatos de dados quando é feita a comunicação interefere na precisão do controle e dos dados, porém, não mostrou ser um problema, tendo em vista que em estado estacionário as respostas são praticamente idênticas, com erro máximo de $E_{max} = 0.0008$.

4.6.3 Comunicação com MATLAB - Uso da STM32

Com os resultados obtidos pela prova de conceito através do arduino UNO, foi possível progredir para as simulações mais complicadas com a STM32F103C8T6 especificada para o projeto. Como o hardware em questão não possui um módulo de compatibilidade, a estratégia alternativa citada previamente, de exportar um arquivo .hex após a criação de um código de controle na IDE própria do microcontrolador.

4.6.3.1 STM32 - Configuração da STM32CubeIDE e Proteus

A IDE compatível com o hardware utilizada é a STM32CubeIDE, durante as simulações a versão 1.10.1 disponível gratuitamente foi utilizada. A IDE permite a configuração das características do microcontrolador de modo mais complexo, desde a escolha de quais pinos serão utilizados a quais funcionalidades serão ativas, permitindo ao usuário a adaptação do microcontrolador ao sua necessidade.

Para o projeto em questão, é necessário configurar manualmente: a velocidade do Clock HCLK, ativar e configurar a comunicação UART/USART e por fim configurar a exportação do código compilado para os formatos .bin e .hex.

As configurações utilizadas, tanto em ambiente proteus, quanto na STM32CubeIDE, para as simulações foram:

- **Baud Rate (Taxa de Transferência de Dados)** 115200 bits/s;

- **Frequência do Clock** 8 MHz (HSE - HCLK);
- **Canais de comunicação serial** USART1, USART2, USART3 em modo assíncrono, interrupt de DMA ativo;

A taxa de transferência de dados maior para a STM32 foi escolhida devido ao tipo de transferência a ser utilizado, para garantir a funcionalidade da simulação em tempo real as mensagens enviadas e recebidas devem ser transferidas o mais rapidamente possível. Este valor em especifico é o limite recomendado para a comunicação serial DMA.

Por fim, é importante ressaltar a utilização das flags de interrupção pelo modo DMA. A comunicação serial na STM32 pode ser realizada de 3 métodos distintos: *Polling*, *Interrupt*, *DMA*. As simulações foram feitas utilizando o método DMA, ao final desta seção há uma comparação entre o funcionamento dos diferentes métodos.

4.6.3.2 STM32 - Código de controle

O desenvolvimento do código de controle a ser utilizada na STM32 foi desenvolvido na própria IDE e exportado como formato de arquivo ".hex". De forma semelhante, a simulação em malha fechada utilizou um controle proporcional simples, com ganho proporcional $K_p = 14$. O mesmo modelo de primeira ordem simplificado do motor foi utilizado para a simulação através da STM32.

Uma característica importante do microcontrolador é a compatibilidade com duas linguagens de programação, tanto C quanto C++, durante a criação de um projeto novo deve ser escolhida a linguagem preferida do usuário ou a mais útil. Os códigos do projeto foram realizados em C por familiaridade com a linguagem.

4.6.3.3 STM32 - Configuração do Envio de dados Proteus-Simulink

Para a configuração da transmissão de dados do Proteus com a STM32 é preciso criar uma mensagem, definindo suas características e o tipo de dado a ser enviado, assim como foi feito para o arduino. Contudo, existem certas peculiaridades que diferenciam o envio pelo Arduino do envio pela STM32.

A utilização de headers e terminators pode ser feita da mesma forma e é aconselhada para garantir estabilidade nos dados. Contudo, não é recomendada a utilização de configurações de paridade em nenhum dos modos: "EVEN", "ODD", "SPACE"ou "MARK"no proteus e também no simulink. Ao tentar simular em tempo real, a transferência de dados

mostra momentos de instabilidade dependendo da velocidade utilizada na transferência e no método escolhido para a transmissão serial e a utilização da paridade no envio desses dados agrava este problema.

Com a instabilidade, ocorre a perda de dados e, uma vez que um dado foi perdido ou corrompido nessa comunicação, o controle deve ser descartado, caso seja utilizado no equipamento físico pode se tornar um risco de segurança para o equipamento e o operador.

Portanto, para a configuração da transmissão de dados com a STM32, diferente do Arduino, o lado responsável por realizar a sincronização e o handshaking é o MATLAB, invertendo os papeis da ultima comunicação, o Proteus atuando como "Follower"e o MATLAB-Simulink como "Leader".

Como tipo de mensagem a ser enviado, tirando as peculiaridades citadas acima, pode ser realizada da mesma forma, a função continua sendo interna, similar ao arduino, sem a necessidade de criar uma função própria de envio, utilizando: *HAL_UART_Transmit*. E o tipo de dado a ser enviado, como feito anteriormente, é um float, dessa vez sem a utilização de header e terminator.

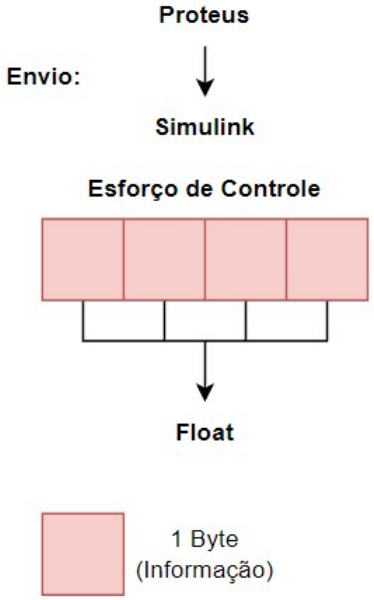


Figura 4.20: Mensagem de transmissão de dados Proteus-Simulink para as simulações com a STM32

4.6.3.4 STM32 - Configuração da Recepção de dados Simulink-Proteus

Como citado anteriormente, a configuração da recepção de dados, para as simulações feitas com a STM32 é a mais importante pois desta vez, o MATLAB-Simulink que realiza o papel de "Leader", garantindo que ambos os softwares estejam sincronizados nas informações.

De forma inversa a transmissão de dados, para a recepção é necessária a utilização de um bit de paridade, logo a configuração de paridade como: "EVEN"ou "ODD"é instrumental para que a comunicação funcione de forma correta. Além disso, é igualmente necessário o envio de headers e terminators para a recepção de dados, diferente da simulação pelo arduino, onde o Simulink recebia a mensagem com header e terminator e tratava deles pelo usuário, é necessário o tratamento manual em código para a simulação com a STM32.

Ambas as configurações são de extrema importância para o funcionamento da simulação. Caso elas não sejam respeitadas irá ocorrer a perda de dados, o microcontrolador não fará a interpretação correta das informações e o controle deverá ser descartado. Isso se deve ao modo com o qual o Simulink envia dados e o modo com o qual o Proteus recebe e interpreta eles. Pelos testes realizados, foi perceptível o fato de que o MATLAB-Simulink, ao enviar informações seriais, envia elas em formato de bloco, por exemplo, uma mensagem de 4 bytes e 32 bits será enviada de uma vez para o Proteus.

Contudo, é capaz de que, dependendo do momento em que se encontra o compilador do microcontrolador, por exemplo se estiver processando outra informação ou em um estado de *idle*, pode acontecer um atraso na tratativa dessas informações recebidas, e quando o microcontrolador começar a tratar desta mensagem, outra venha e sobrescreva a informação, misturando as mensagens e corrompendo os dados.

Ao se utilizar um header e terminator assim como a paridade, o próprio proteus pode verificar se a mensagem perdeu algum dado antes de chegar ao software e descartá-la, e dentro do processamento da mensagem, o microcontrolador, através do código, tem uma maior facilidade em interpretar o inicio, fim e a integridade de uma mensagem.

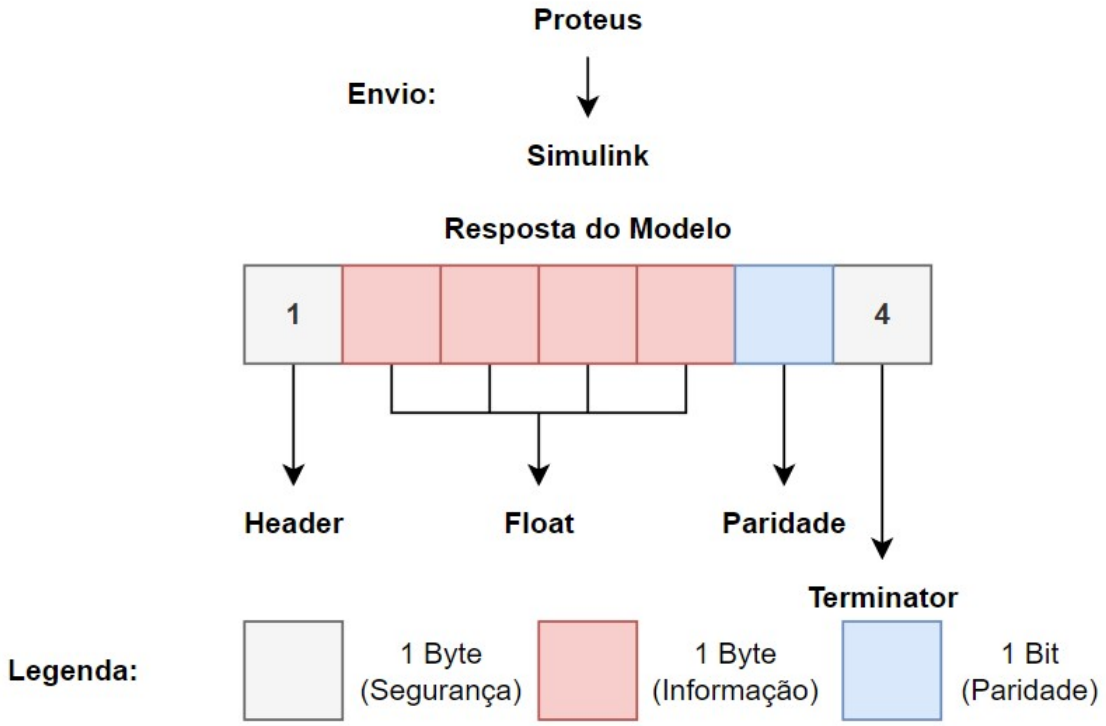


Figura 4.21: Mensagem de transmissão de dados Simulink-Proteus para as simulações com a STM32

O envio byte a byte, pelo MATLAB-Simulink é possível e auxilia na integridade de mensagens, porém atrasa a simulação e diminui a frequência máxima de simulação em tempo real. Portanto, é mais aconselhada a utilização da paridade, com headers e terminators padrão para garantir a integridade das mensagens.

Devido as peculiaridades existentes no envio e recepção de dados com a STM32, de forma diferente a simulação realizada com o arduino, é necessário o uso de 4 portas seriais virtuais, ou 2 pares de conexões entre portas seriais virtuais, todas respeitando a mesma taxa de transmissão de dados. Um par deve ser dedicado ao envio de dados Proteus-Simulink, sem paridade configurada e outro deve ser dedicado a recepção de dados Simulink-Proteus

4.6.3.5 STM32 - Resultados da simulação em Malha Fechada

Feitas as configurações, a simulação pode ser realizada. Nos apêndices [F], [G] e [H] encontram-se, respectivamente, o diagrama do circuito em ambiente proteus, o código utilizado para a simulação em malha fechada da STM32 e o diagrama de blocos em ambiente simulink. As figuras abaixo mostram as respostas obtidas da simulação em

malha fechada.

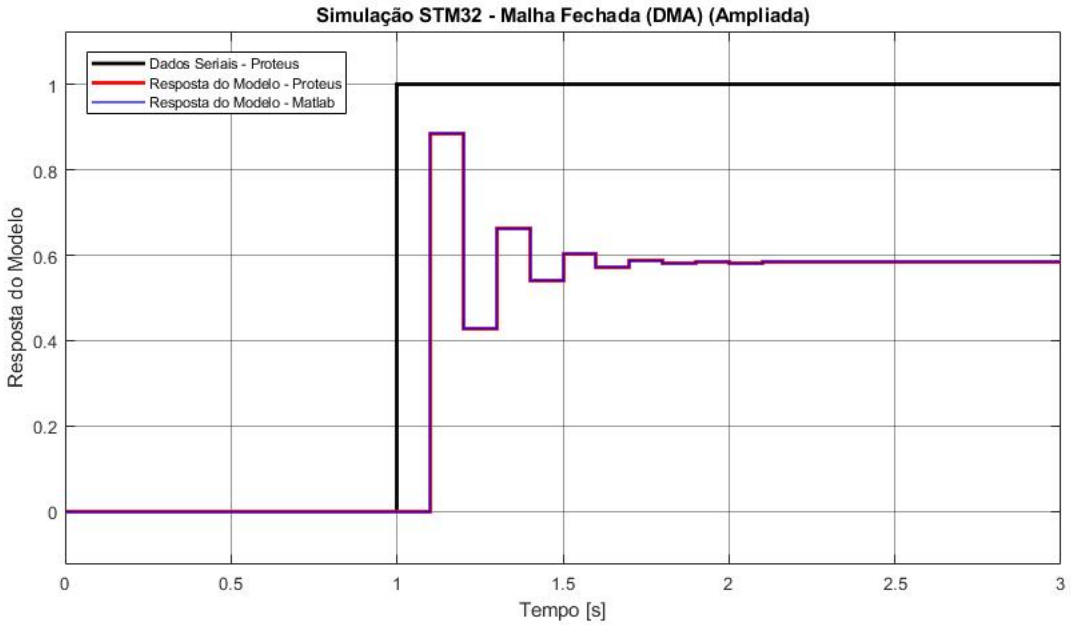


Figura 4.22: Simulação de hardware. STM32F103C8T6, Malha fechada.

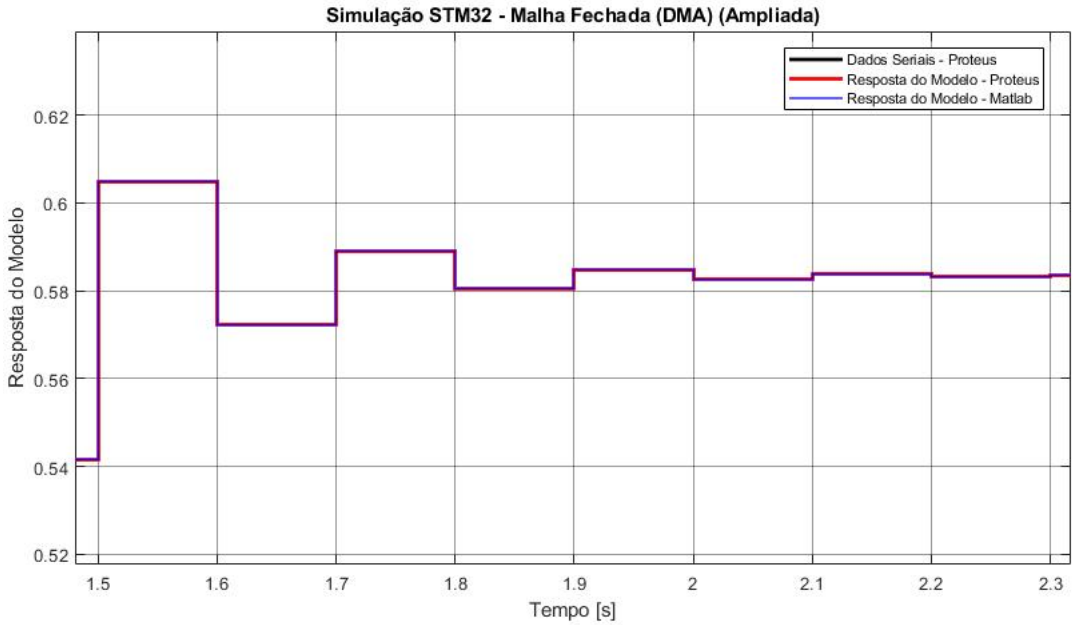


Figura 4.23: Simulação de hardware. STM32F103C8T6, Malha fechada, ampliada.

Através da comparação da resposta ideal, em azul, simulada completamente em MATLAB-Simulink, e da resposta obtida através da comunicação Proteus-Simulink, podemos perceber que a transmissão de dados foi feita de maneira correta, sem que ocorresse a perda de dados ou uma interpretação incorreta.

Uma outra informação importante gerada pela simulação através da STM32, como comentado anteriormente, se refere as informações e dados sobre a utilização do microcontrolador, no que diz a operação e funcionamento do hardware durante o controle. Além das simulações em malha fechada, foram realizadas simulações em malha aberta, com um código levemente alterado, no hardware especificado STM32F103C8T6 e um hardware similar STM32F103C6T6. Ambos da mesma família, porém com a grande diferença de que o modelo C6 dispõe de metade da memória flash e memória RAM disponível para o modelo C8.

Essas simulações foram realizadas com os mesmos códigos em ambos os microcontroladores, e a informação de uso de memória foi obtida através da própria STM32CubeIDE, os resultados podem ser vistos abaixo:

Consumo e utilização de memória		
Modelo	**% de utilização - Memória Flash**	**% de utilização - Memória RAM**
STM32F103C6T6 em Malha Aberta	21,73%	96,11%
STM32F103C6T6 em Malha Fechada	21,80%	96,26%
STM32F103C8T6 em Malha Aberta	10,86%	48,06%
STM32F103C8T6 em Malha Fechada	10,90%	48,15%

Tabela 4.9: Consumo e utilização de memória RAM e memória Flash entre dois modelos de hardware STM32

O modelo C6 dispõe de 10 KB de memória flash e 32 KB de memória RAM, enquanto o modelo C8, como comentado, possui o dobro, 20 KB de memória flash e 64 KB de memória RAM. Pelos resultados obtidos, fica evidente a necessidade de ao menos 64 KB de memória RAM, uma informação pertinente e importante para a especificação do projeto, anteriormente indisponível.

Sobre esses valores, outros testes de menor escala foram realizados, com a versão C6 e a versão C8, tentando simular a inclusão de mais juntas e a adição de portas de entrada e saída GPIO para a representação dos periféricos especificados previamente. A versão C6 passa de sua utilização máxima de 100% de memória RAM, não permitindo sequer a compilação de um código. Enquanto com as mesmas adições o modelo C8 chega a bater em 60% de utilização.

A ultima consideração a ser feita sobre esses números é que, esses resultados mostram que, pelo mais que o modelo especificado, a STM32F103C8T6, seja capaz de rodar

códigos mais complexos, uma vontade da equipe do projeto era manter uma margem de ao menos 50% de utilização em todas as características do hardware: utilização de CPU, utilização de Memória, etc. A fim de que, futuramente, caso necessário, com uma maior complexidade do projeto, não haja a necessidade de refazer a especificação. Através desses resultados, considerando as expectativas da equipe, é capaz de que nenhum dos microcontroladores especificados pela estratégia de baixo custo sejam viáveis no quesito de memória.

4.6.3.6 STM32 - Comparação entre funcionamento dos métodos de comunicação serial

Por fim, é importante a discussão sobre os diferentes métodos de comunicação serial disponíveis pela STM32 e seus métodos de funcionamento. Como citado anteriormente são 3 métodos [7], todos com diferentes propósitos, utilizações e configurações distintas, tanto em código quanto em configurações na IDE:

1. **Método Polling:** Através da função *HAL_UART _RECEIVE* ativa-se esse meio de comunicação serial, essencialmente se caracteriza por ser uma função que bloqueia a CPU do microcontrolador de avançar em instruções até que a quantidade de bytes, fornecida como parâmetro para a função, seja recebida. Após a recepção da quantidade esperada de dados o processador sai do modo de congelamento e continua a sua execução do código principal. Caso dados suficientes não tenham sido recebidos, o processador continua bloqueado até um período de timeout que força a função a terminar.

2. **Método Interrupt:** Através da função *HAL_UART _RECEIVE _IT*, em conjunto com a ativação da flag de interrupt para o canal USART dentro da STM32CubeIDE é possível ativar esse meio de comunicação serial. O modo de operação deste método é diferente ao anterior, ele é um método que não bloqueia ou congela o processador enquanto os dados estão sendo recebidos, ele cria um buffer de tamanho requisitado, apenas quando o buffer receber todas as informações e for completo, ativa-se uma flag de interrupção do processador, que então interrompe as atividades na função principal "main"e segue para a tratativa dos dados e da função descrita na flag de interrupção.

3. **Método DMA:** Através da função *HAL_UART _RECEIVE _DMA*, em conjunto da ativação da flag de interrupt DMA para o canal e método de recepção ou envio de dados dentro da STM32CubeIDE, ativa-se esse meio de comunicação serial. A

operação novamente se difere das outras duas a medida que, os dados recebidos pela comunicação UART são escritos diretamente na memória, não requisitando a intervenção da CPU para inseri-los em armazenamento. Quando o buffer configurado pelo DMA estiver completo, e também completo pela metade, ativa-se uma flag para caso o usuário deseje tratar desses dados ou processá-los de alguma forma. O buffer neste modo de operação pode funcionar de forma circular, semelhante a uma pilha FIFO (First-In-First-Out), onde cada dado novo recebido descarta o mais antigo no buffer que se encontra na ultima posição, escrevendo o novo dado na primeira posição do buffer de modo contínuo, ou no modo normal, onde a recepção é idêntica mas o funcionamento não é continuo, necessitando a reativação após o preenchimento do buffer.

Os 3 métodos podem ser utilizados, porém com diferentes níveis de sucesso e graus de qualidade de recepção de dados. Os 3 métodos foram listados apenas para a recepção de dados pois para o envio não é uma necessidade, o método de envio por polling normal atende as necessidades para a integração Proteus-Simulink com a STM32, o foco, e no caso onde a maioria dos problema de integração podem acabar ocorrendo, encontram-se na recepção de dados pelo lado do Proteus.

Iniciando a discussão pela utilização do método Polling. A utilização deste método não é ideal considerando as necessidades do projeto, devido a sua característica de block, ou congelamento, da CPU, e, considerando que o envio de dados entre os softwares deve ser rápido e ininterrupto para que o controle seja realizado em uma velocidade elevada, o método, ao forçar a CPU a ficar parada, pode comprometer a velocidade de controle.

Além deste ponto, o método não foi capaz de ser utilizado, nem sequer para a recepção de dados simples através do proteus em conjunto com o simulink. Por fim, este método não apenas compromete a velocidade de controle, mas como ele apenas recebe informações durante o momento de congelamento da CPU, ele corre um alto risco de perda de dados. Como o fluxo de comunicação não é apenas rápido, ele é simultâneo, o envio e a recepção em ambos os lados devem ocorrer ao mesmo tempo, ou de forma rápida o suficiente para simular essa comunicação simultânea, o microcontrolador não pode ficar em modo idle para recepção e por isso este método, mesmo se fosse funcional, foi compreendido como não ideal para a necessidade do projeto.

O método de comunicação por interrupt é mais viável, o fato da recepção de dados poder ocorrer paralelamente a tratativa de uma função principal é uma característica mais interessante e mais útil para o projeto. Contudo, através dos testes, viu-se que,

novamente, o fato da CPU ficar em idle quando ocorrer a ativação da flag de interrupt com o buffer completo, causa a perda de dados. Não distante, ao tratar a flag de interrupt, é necessária sua reativação, então por exemplo caso, durante a tratativa dos dados, com a CPU bloqueada, em um buffer de 8 bytes, 1 desses bytes seja ignorado a informação foi perdida e o controle deve ser descartado, mesmo utilizando a paridade, headers e terminators.

A utilização deste método trouxe resultados não satisfatórios, pelo mais que tenha sido possível realizar a comunicação, para garantir a recepção completa de dados, sem a perda de informações, a velocidade de transferência de dados foi de $t = 0.25s$, o que, quando transformado em frequência é $f = 4Hz$. Longe dos $10Hz$ representativos da dinâmica do motor.

Caso uma velocidade assim seja utilizada com o equipamento real, diversas informações dos motor serão perdidas e não é possível de realizar o controle. Por estes motivos, compreende-se que este método de recepção também não é ideal para as necessidades do projeto. Por fim, o método de comunicação por DMA é ideal caso a comunicação serial seja utilizada, devido ao fato da recepção de dados não bloquear a CPU, armazenando diretamente na memória, quando as flags são ativadas os dados continuam sendo recebidos de forma simultânea, logo, o envio e a recepção podem ser feitos ao mesmo tempo mesmo com um volume grande de informações sendo transferidas.

Abaixo, seguem informações pertinentes sobre cada método de utilização da comunicação serial:

Métodos de comunicação serial - STM32		
Método	**Velocidade Mínima de Recepção (Garantindo 100% de Recepção)**	**Frequência Máxima de Recepção (Garantindo 100% de Recepção)**
Polling	N/D	N/D
Interrupt	0.25s	4 Hz
DMA	0.06s	16,67 Hz

Tabela 4.10: Comparação entre métodos de comunicação serial para garantir 100% de recepção de dados - STM32

Por fim, é importante ressaltar, os resultados de simulação obtidos em proteus dependem do hardware sendo utilizado para rodar o software, no caso a máquina, sendo um computador ou laptop. Os resultados acima foram obtidos em um computador desktop com um processador Intel i7 de 6ª geração, 16 GB de memória RAM, contudo, esses resultados diferem levemente quando comparados a um mesmo setup sendo utilizado em

Métodos de comunicação serial - STM32	
Método	**Porcentagem de Recepção de dados**
Polling	N/D
Interrupt	70%-80%
DMA	100%

Tabela 4.11: Comparação entre métodos de comunicação serial com a transferência de dados à 10 Hz - STM32

um notebook. Portanto, é possível concluir que, dependendo do hardware utilizado essas velocidades e frequências podem ser maiores ou menores, sendo um fator importante na hora de se considerar a utilização desse método de comunicação em trabalhos futuros.

5 DISCUSSÃO

Nesta seção, pretende-se realizar uma avaliação geral de todas as etapas desenvolvidas neste trabalho de maneira completa.

Com relação ao circuito dos atuadores, é interessante fazer uma análise sobre a montagem de bancada do circuito e sobre a complexidade de operação. Os problemas detectados no final eram de menor importância, se resumindo a componentes eletrônicos em aberto. Contudo, não é uma descoberta necessariamente oportuna considerando o escopo do projeto no geral, a montagem do circuito completo, com utilização do C.I LMP-8601-Q1, pelas informações disponíveis nos diretórios Cloud projeto não havia sido estudada antes da implementação, havendo estudos extensos sobre a função de amplificação porém poucos relacionados a função de leitura de corrente do circuito. A geração desse conhecimento foi feita mas, é de nossa crença que esse vácuo de informações é o que causou os problemas em primeiro momento. A montagem do circuito em bancada é complicada, são diferentes equipamentos, cabos, conexões e até mesmo softwares que devem estar conectados e configurados corretamente, esperamos que com as informações providenciadas este problema seja uma ocorrência futura.

E com relação as simulações de hardware, através dos resultados obtidos com a STM32F103C8T6 utilizando a comunicação serial, vimos as limitações que o software, hardware e também protocolo de comunicação escolhido para a simulação. A comunicação serial virtual, pelo mais que capaz de enviar e receber dados e simular em frequências acima de 10 Hz, não possui uma margem satisfatória para que o controle do equipamento físico seja realizado, idealmente a comunicação deve ser feita na faixa de centenas de Hz, permitindo um controle refinado.

6 CONCLUSÃO

O trabalho desenvolvido permite tirar conclusões sobre simulação e especificação de hardware para sistemas embarcados, além de fornecer conhecimentos importantes sobre protocolos de comunicação, integração de software e projetos complexos de engenharia. Vimos que a metodologia de especificação de hardware é importante, não apenas para decidir os componentes que compõem um equipamento mas também para orientar e facilitar o trabalho dos integrantes de uma equipe. Devido a extensa quantidade de informações necessárias em um projeto de engenharia e sistemas embarcados, as metodologias existem para orientar e auxiliar engenheiros e técnicos a tomar as melhores decisões. É curioso comentar que existem diferentes metodologias, a metodologia utilizada foi de modelo Top-Down, onde se define detalhes de modo iterativo e de forma progressiva no projeto até a sua conclusão, por preferência da equipe e do professor Fábio, mas as metodologias Bottom-Up por exemplo também seriam mais do que capazes de atender as necessidades.

Com relação a simulação do hardware, como comentamos anteriormente, vimos os limites e as peculiaridades da comunicação serial, um protocolo útil e simples porém repleto de detalhes. As simulações iniciais feitas com o Arduino mostram também a complexidade entre diferentes hardwares, a existência de um modulo de configuração do Arduino no ambiente proteus e as ferramentas disponíveis através do próprio microcontrolador facilitaram as tarefas quando comparado as simulações da STM32. Desde a configuração do hardware para a simulação em proteus, ao envio e recepção de dados de forma serial, seus diferentes métodos de aplicação, limitações, utilidades e também o tipo de dado a ser enviado foram mais desafiadoras.

As simulações iniciais utilizando a STM32 foram feitas com o envio de informações por bytes singulares, enviando uma informação no formato uint8 para fechar a malha com o proteus e o simulink. Esse envio de dados simples nos mostrou técnicas de controle digital com dados menos complexos, como transformar os valores inteiros da comunicação e processá-los como sendo valores decimais, mantendo a comunicação simples porém aumentando a complexidade dos dados enviados foi interessante de se analisar.

Espera-se que esse trabalho de continuidade ao projeto do manipulador robótico didático dentro da Escola Politécnica da Universidade de São Paulo, e de início a uma série de trabalhos voltados a continuação da especificação do hardware e avanços na simulação do comportamento do circuito e do modelo.

Vale ressaltar que a computação, como ciência e tecnologia, avança rapidamente, com novas descobertas de softwares, hardwares e técnicas com aplicações e usos em sociedade, assim como faz a área da robótica, ambas requerem profissionais capacitados e treinados com sólidos conhecimentos para os avanços e retornos para a sociedade. Sendo assim, há a finalização do trabalho, com o desejo de que o leitor haja se interessado pela continuação de um projeto importante para a sociedade brasileira e a escola politécnica, e tenha aprendido e se interessado, assim como nós, nos temas de simulação, robótica e projetos em engenharia.

7 TRABALHOS FUTUROS

Futuramente, podem ser feitos estudos a fim de avançar os modelos do braço robótico, como não linearidades que ainda não foram consideradas como por exemplo a elasticidade dos ligamentos. Com o refinamento do modelo espera-se que as simulações do equipamento se aproximem cada vez mais da realidade. Além disso, é necessária a identificação dos parâmetros reais do equipamento, uma etapa ainda não realizada porém importante para o prosseguimento do projeto e da especificação do hardware.

Podem ser feitos avanços na interface humano máquina, adicionando funcionalidades como: instalação do robô(calibração, *hardware*, limites de juntas, posições iniciais), trajetórias pré programadas(linhas retas, circular), scripts próprios para geração de trajetórias.

Por fim, as simulações de hardware podem ser avançadas, testando diferentes métodos de comunicação e integração entre o Proteus e o ambiente MATLAB-Simulink, como o protocolo TCP/IP por exemplo. Além disso, é possível avançar em outros aspectos da simulação do hardware, como por exemplo a integração e funcionamento de periféricos. A criação de um aplicativo que realiza a comunicação Bluetooth com o ambiente Proteus, enviando informações para o microcontrolador e utilizando funcionalidades, simulando um operador utilizando o celular para controlar o braço.

REFERÊNCIAS

[1] WOLF, M. *Computers as Components: Principles of Embedded Computing System Design*. 3. ed. [S.l.]: Morgan Kaufmann Publishers, 2016.

[2] PATTERSON, D. A.; HENNESSY, J. L. *Computer Organization and Design Arm Edition: The Hardware Software Interface*. [S.l.]: Morgan Kaufmann Publishers, 2016.

[3] REHDER, G. *Slides da disciplina: Arquitetura de Sistemas Embarcados*. 2020. Departamento de Engenharia de Sistemas Eletrônicos da Escola Politecnica da Universidade de São Paulo.

[4] EMBARCADOS. *Microeletrônica*. https://embarcados.com.br/controle-pid-em-sistemas-embarcados/. Ultimo acesso: 05 de Agosto de 2022.

[5] MARIGA, L. *Simulink and Arduino connection*. https://github.com/leomariga/Simulink-Arduino-Serial. Ultimo acesso: 02 de Outubro de 2022.

[6] MARQUES, R. P. *Apostila e materiais da disciplina PTC3419-Controle Digital*. 2021. Laboratório de Automação e Controle do Departamento de Engenharia de Telecomunicações e Controle da Escola Politecnica da Universidade de São Paulo.

[7] MAGDY, K. *How To Receive UART Serial Data With STM32 – DMA / Interrupt / Polling*. Https://deepbluembedded.com/how-to-receive-uart-serial-data-with-stm32-dma-interrupt-polling/. Ultimo acesso: 06 de Dezembro de 2022.

[8] LEE, E. A.; SESHIA, S. A. *Introduction to Embedded Systemns - A Cyber-Physical Approach*. 2. ed. [S.l.]: MIT Press, 2017.

[9] NAKADAIRA, L. N.; SIMÕES, V. L.

Modelagem e Controle de um Manipulador Robótico Didático de 5 graus de Liberdade — Escola Politécnica da Universidade de São Paulo, São Paulo, 2020.

[10] FIALHO, F. *Slides da disciplina: Modelagem e Controle de Manipuladores Robóticos*. 2019. Laboratório de Automação e Controle do Departamento de Engenharia de Telecomunicações e Controle da Escola Politecnica da Universidade de São Paulo.

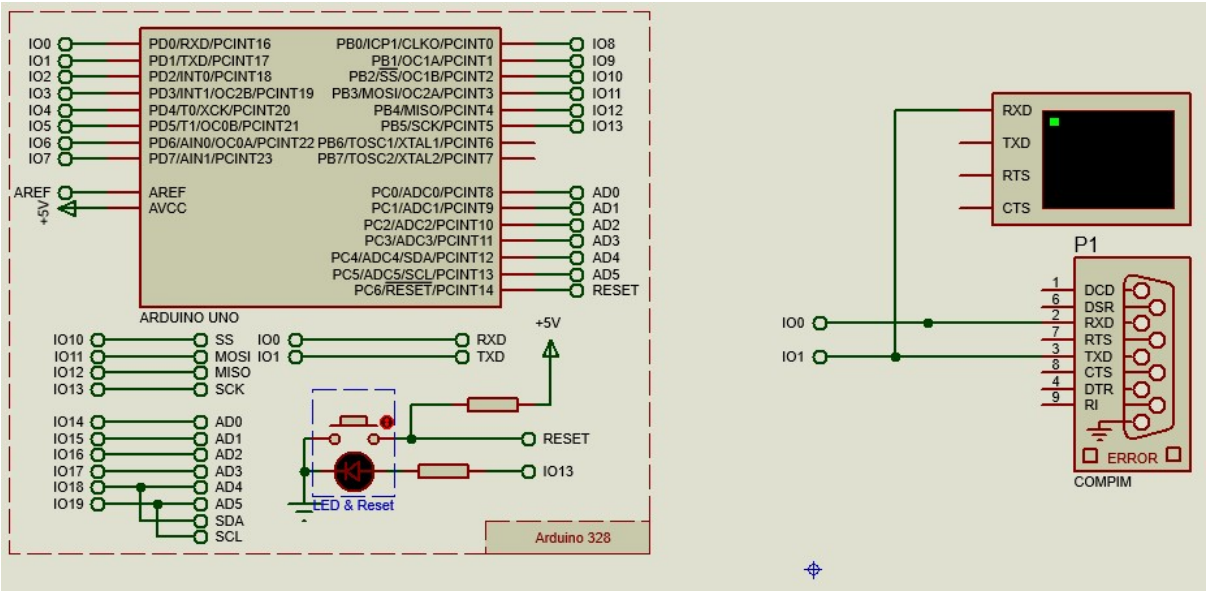
[11] CRAIG, J. J. *Introduction to Robotics: Mechanics and Control*. 4. ed. [S.l.]: Pearson, 2017.

[12] COPA-DATA. *O que é HMI? A interface homem-máquina*. https://www.copadata.com/pt/produtos/zenon-software-platform/visualizacao-controle/o-que-e-hmi-a-interface-homem-maquina-copa-data/. Ultimo acesso: 05 de Agosto de 2022.

[13] MATHWORKS. *watertank Simulink Model - MATLAB & Simulink*. https://www.mathworks.com/help/slcontrol/gs/watertank-simulink-model.html. Ultimo acesso: 05 de Agosto de 2022.

[14] EMBARCADOS. *Controle PID em Sistemas Embarcados*. https://embarcados.com.br/controle-pid-em-sistemas-embarcados/. Ultimo acesso: 05 de Agosto de 2022.

APÊNDICE A – SIMULAÇÃO ARDUINO UNO, ESQUEMÁTICO DO CIRCUITO EM AMBIENTE PROTEUS



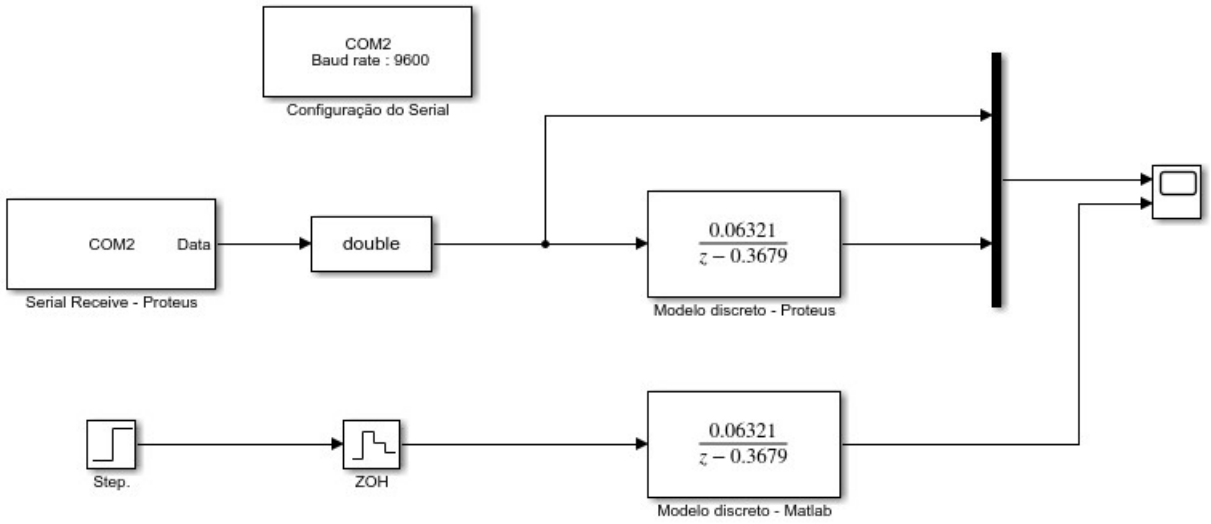
APÊNDICE B – SIMULAÇÃO ARDUINO UNO, CÓDIGO SIMULAÇÃO EM MALHA ABERTA

```
/* Main.ino file generated by New Project wizard
 *
 * Created:   ter jul 26 2022
 * Processor: Arduino Uno
 * Compiler:  Arduino AVR (Proteus)
 */

// Peripheral Configuration Code (do not edit)
//---CONFIG_BEGIN---
#pragma GCC push_options
#pragma GCC optimize ("Os")

#include <core.h> // Required by cpu
#include <cpu.h>

#pragma GCC pop_options

// Peripheral Constructors
CPU &cpu = Cpu;

void peripheral_setup () {
}
```

```
void peripheral_loop() {
}
//---CONFIG_END---

typedef union{
  float number;
  uint8_t bytes[4];
} FLOATUNION_t;

FLOATUNION_t myValue;
float step = 0;
float data = 0;

void setup() {

  Serial.begin(9600);

}

void loop() {

   //sendData(data);
   if(millis() >= 1000){
      step = 1;
   }
   data = step;
   sendData(data);
   if (Serial.available() > 0) {
    resposta = getFloat();
   }
   delay(100);

}

void sendData (float data){
```

```
    myValue.number = data;
    Serial.write('A');

    for (int i=0; i<4; i++){
            Serial.write(myValue.bytes[i]);
    }

    Serial.print('\n');

}
}
```

APÊNDICE C – SIMULAÇÃO ARDUINO UNO, DIAGRAMA DE BLOCOS MALHA ABERTA EM AMBIENTE SIMULINK



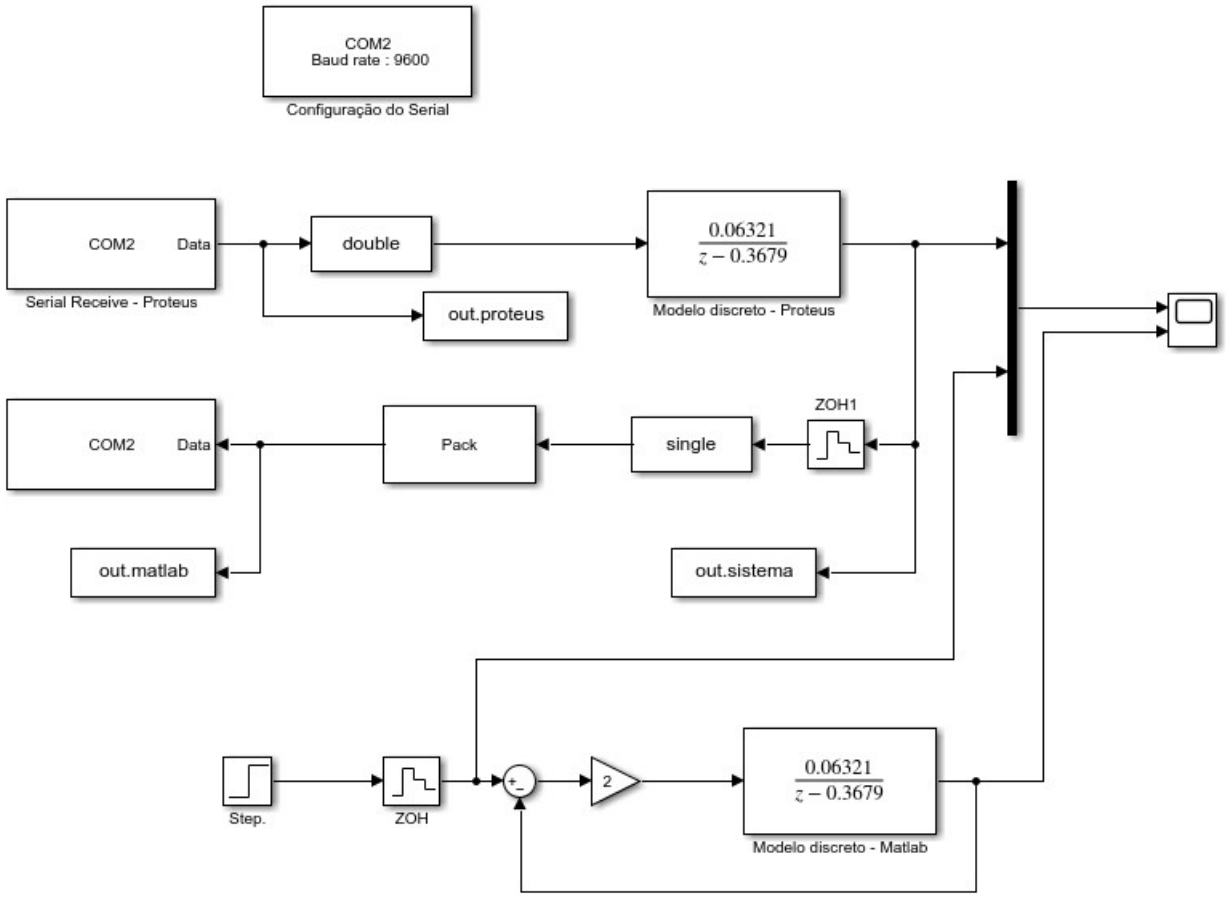
APÊNDICE D – SIMULAÇÃO ARDUINO UNO, CÓDIGO SIMULAÇÃO EM MALHA FECHADA

```
/* Main.ino file generated by New Project wizard
 *
 * Created:   ter jul 26 2022
 * Processor: Arduino Uno
 * Compiler:  Arduino AVR (Proteus)
 */

// Peripheral Configuration Code (do not edit)
//---CONFIG_BEGIN---
#pragma GCC push_options
#pragma GCC optimize ("Os")

#include <core.h> // Required by cpu
#include <cpu.h>

#pragma GCC pop_options

// Peripheral Constructors
CPU &cpu = Cpu;

void peripheral_setup () {
}

void peripheral_loop() {
}
//---CONFIG_END---

typedef union{
```

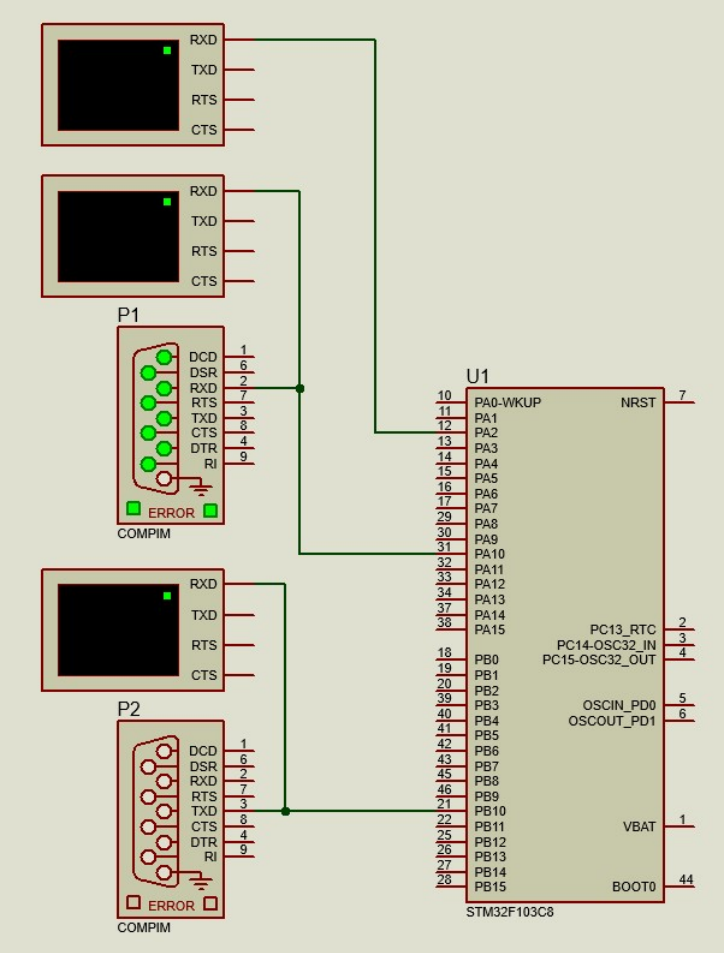
```
  float number;
  uint8_t bytes[4];
} FLOATUNION_t;

FLOATUNION_t myValue;
float step = 0;
float data = 0;
float resposta = 0;
float erro = 0;
int Kp = 2;

void setup() {

  Serial.begin(9600);

}

void loop() {

   //sendData(data);
   if(millis() >= 1000){
      step = 1;
   }
   double erro = step - resposta;
   data = Kp*erro;
   sendData(data);
   if (Serial.available() > 0) {
    resposta = getFloat();
   }
   delay(100);

}

void sendData (float data){

   myValue.number = data;
   Serial.write('A');

   for (int i=0; i<4; i++){
           Serial.write(myValue.bytes[i]);
   }

   Serial.print('\n');
```

```
}

float getFloat(){
    int cont = 0;
    FLOATUNION_t f;
    while (cont < 4 ){
        f.bytes[cont] = Serial.read() ;
        cont = cont +1;
    }
    return f.number;
}
```

APÊNDICE E – SIMULAÇÃO ARDUINO UNO, DIAGRAMA DE BLOCOS MALHA FECHADA EM AMBIENTE SIMULINK



APÊNDICE F – SIMULAÇÃO STM32, ESQUEMÁTICO DO CIRCUITO EM AMBIENTE PROTEUS



APÊNDICE G – SIMULAÇÃO STM32, CÓDIGO SIMULAÇÃO EM MALHA FECHADA

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************

  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************

  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the
     LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************

  */
/* USER CODE END Header */
/* Includes
   ------------------------------------------------------------------*/
#include "main.h"

/* Private includes
   ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
```

```
#include <stdio.h>
#include <string.h>

/* USER CODE END Includes */

/* Private typedef
   -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define
   -----------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro
   -----------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables
   -----------------------------------------------------------*/
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart3;
DMA_HandleTypeDef hdma_usart1_rx;
DMA_HandleTypeDef hdma_usart1_tx;

/* USER CODE BEGIN PV */
uint8_t dataBuffer[12] = {0};
union myNumber {
        uint8_t bytes[4];
        float myFloat;
} envia, recebe;
int count = 0;

float Kp = 14;
float setPoint = 0;
float error = 0;
```

```
uint8_t stepMoment = 0;

// Variaveis para o delay
uint32_t intervalo = 100;
uint32_t tempoAtual = 0;
uint32_t tempoAnterior = 0;

/* USER CODE END PV */

/* Private function prototypes
   -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART3_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code
   -------------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
  envia.myFloat = 0;
  recebe.myFloat = 0;
  /* USER CODE END 1 */

  /* MCU Configuration
     --------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
     Systick. */
```

```
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA_Init();
MX_USART1_UART_Init();
MX_USART2_UART_Init();
MX_USART3_UART_Init();
/* USER CODE BEGIN 2 */
HAL_UART_Receive_DMA(&huart1, dataBuffer, sizeof(dataBuffer));

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
      tempoAtual = HAL_GetTick();
      if(tempoAtual >= 1000 && stepMoment == 0) {
              setPoint = 1;
              stepMoment = 1;
      }

      tempoAtual = HAL_GetTick();
      if (tempoAtual - tempoAnterior > intervalo){
        tempoAnterior = tempoAtual;
        //char msg[64];
        //sprintf(msg, "%dM: %.6f \r\n", count, recebe1.myFloat);
        //HAL_UART_Transmit(&huart2, (uint8_t *) msg, strlen(msg),
           100);
        //sprintf(msg, "%dC: %.6f \r\n", count, recebe2.myFloat);
        //HAL_UART_Transmit(&huart2, (uint8_t *) msg, strlen(msg),
```

```
            100);
          error = setPoint - recebe.myFloat;
          envia.myFloat = Kp*error;
          HAL_UART_Transmit(&huart3, envia.bytes, sizeof(envia.bytes),
            10);
        }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified
     parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK
      )
  {
    Error_Handler();
  }
}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
```

```
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief USART3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART3_UART_Init(void)
{

  /* USER CODE BEGIN USART3_Init 0 */

  /* USER CODE END USART3_Init 0 */

  /* USER CODE BEGIN USART3_Init 1 */
```

```
  /* USER CODE END USART3_Init 1 */
  huart3.Instance = USART3;
  huart3.Init.BaudRate = 115200;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART3_Init 2 */

  /* USER CODE END USART3_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel4_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel4_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel4_IRQn);
  /* DMA1_Channel5_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel5_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
```

```
  */
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

}

/* USER CODE BEGIN 4 */
void HAL_UART_RxHalfCpltCallback(UART_HandleTypeDef *huart)
{
        //recebe1.myFloat = 0;
        //count ++;
        //uint8_t bytes[4];
        recebe.bytes[0] = dataBuffer[1];
        recebe.bytes[1] = dataBuffer[2];
        recebe.bytes[2] = dataBuffer[3];
        recebe.bytes[3] = dataBuffer[4];
        //float f;
        //memcpy (&f, bytes, 4);
        //char msg[64];
        //sprintf(msg, "%dM: %.6f \r\n", count, recebe1.myFloat);
        //sprintf(msg, "%dM: %u %u %u %u %u %u \r\n", count, dataBuffer
           [0], dataBuffer[1], dataBuffer[2], dataBuffer[3], dataBuffer
           [4], dataBuffer[5]);
        //HAL_UART_Transmit(&huart2, (uint8_t *) msg, strlen(msg), 100);
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
        //recebe2.myFloat = 0;
        //count ++;
        //uint8_t bytes[4];
        recebe.bytes[0] = dataBuffer[7];
        recebe.bytes[1] = dataBuffer[8];
        recebe.bytes[2] = dataBuffer[9];
        recebe.bytes[3] = dataBuffer[10];
        //float t;
        //memcpy (&t, bytes, 4);
        //char msg[64];
```

```
        //sprintf(msg, "%dC: %.6f \r\n", count, recebe2.myFloat);
        //sprintf(msg, "%dC: %u %u %u %u %u %u \r\n", count, dataBuffer
            [6], dataBuffer[7], dataBuffer[8], dataBuffer[9], dataBuffer
            [10], dataBuffer[11]);
        //HAL_UART_Transmit(&huart2, (uint8_t *) msg, strlen(msg), 100);
        HAL_UART_Receive_DMA(&huart1, dataBuffer, sizeof(dataBuffer));
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
     state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line
     number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
     line envia,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
        line) */
  /* USER CODE END 6 */
}
```

```
#endif /* USE_FULL_ASSERT */
```

APÊNDICE H – SIMULAÇÃO STM32, DIAGRAMA DE BLOCOS MALHA FECHADA EM AMBIENTE SIMULINK