Universidade de Brasília
Faculdade de Tecnologia

Implementação de Software para Teste de Prótese Transfemoral Semi-ativa

Gabriel de Szechy Vigna

PROJETO FINAL DE CURSO

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasília

2023

Universidade de Brasília
Faculdade de Tecnologia

Implementação de Software para Teste de Prótese Transfemoral Semi-ativa

Gabriel de Szechy Vigna

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Orientador: Prof. Dr. Geovany Araújo Borges

Brasília
2023

Vigna, Gabriel de Szechy.
V679i Implementação de Software para Teste de Prótese Transfemoral Semi-ativa / Gabriel de Szechy Vigna; orientador Geovany Araújo Borges. -- Brasília, 2023.
78 p.

Projeto Final de Curso (Engenharia de Controle e Automação) -- Universidade de Brasília, 2023.

1. Prótese transfemoral. 2. Joelho protético. 3. Sistema embarcado. 4. Sensores. I. Borges, Geovany Araújo, orient. II. Título

Universidade de Brasília
Faculdade de Tecnologia

Implementação de Software para Teste de Prótese Transfemoral Semi-ativa

Gabriel de Szechy Vigna

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Trabalho aprovado. Brasília, 9 de Fevereiro de 2023:

Prof. Dr. Geovany Araújo Borges,
UnB/FT/ENE
Orientador

Prof. Dr. Roberto de Souza Baptista,
UnB/FGA
Examinador interno

Prof. Dr. Jones Yudi Mori Alves da Silva,
UnB/FT/ENM
Examinador interno

Brasília
2023

Este trabalho é dedicado às vítimas de amputação que,
sonham em reaver seus movimentos.

Agradecimentos

Agradeço a todas as pessoas que me apoiaram na execução desse trabalho desde amigos e familiares que não me deixaram desanimar, colegas do laboratório sempre solícitos em ajudar e meu orientador pela elucidação de problemas quando tudo parecia perdido.

“‘Inventar é imaginar o que ninguém pensou;
é acreditar no que ninguém jurou;
é arriscar o que ninguém ousou;
é realizar o que ninguém tentou.
Inventar é transcender.””
(Santos Dumont)

Resumo

Visando contribuir para a melhoria do bem estar de amputados é retomado por meio desse trabalho o estudo e desenvolvimento do projeto RLEG. Tal projeto é uma iniciativa do LARA (Laboratório de Automação e Robótica da Universidade de Brasília) e atualmente propõe desenvolver uma prótese robótica semi-ativa para amputados transfemorais atuada por pistão magneto-relógico. O sistema está com o projeto mecânico e eletrônico desenvolvido e necessita de aplicações em software que permitam o avanço da prótese rumo à capacidade de mimetizar o movimento humano. Para alcançar esse objetivo o presente trabalho implementou um software capaz de testar dispositivos do sistema embarcado da prótese. O software em questão introduz uma interface de comunicação entre um computador e a eletrônica da prótese, permitindo o envio de comandos e dados por meio de um terminal. O programa é implementado em tempo real e aplica uma fila circular para transmissão de dados ao computador, o que permite que o processo de transmissão não atrase o processo de leitura dos sensores. Além disso, com o intuito de prevenir a perda de dados na fila, o processo de transmissão tem sua prioridade aumentada quando a fila está próxima de sua capacidade máxima de armazenamento. Os dispositivos presentes na prótese foram testados por meio da solução implementada e o correto funcionamento desses componentes e do programa foi constatado. Como atividades futuras para o projeto destaca-se a calibração dos sensores da prótese, o desenvolvimento de algoritmos de predição de movimento e a aplicação de estratégias de controle para o joelho mecânico.

Palavras-chave: Prótese transfemoral. Joelho protético. Sistema embarcado. Sensores.

Abstract

Aiming to contribute to the improvement of the well-being of amputees, this work resumes the study and development of the RLEG project. This project is an initiative of LARA (Automation and Robotics Laboratory of the University of Brasilia) and currently proposes to develop a semi-active robotic prosthesis for transfemoral amputees actuated by a magneto-relogical piston. The mechanical and electronic designs of the system are developed and it needs software applications that allow the advancement of the prosthesis towards the ability to mimicking human movement. To achieve this goal, the present work implemented a software capable of testing devices of the embedded system of the prosthesis. This software is a communication interface between a computer and the electronics of the prosthesis, allowing the sending of commands and data through a terminal. The program is implemented in real time and applies a circular queue for data transmission to the computer, which allows the transmission process not to delay the sensor reading process. Furthermore, in order to prevent data loss in the queue, the transmission process has its priority increased when the queue is close to its maximum storage capacity. The devices present in the prosthesis were tested through the implemented solution and the right functioning of the system was verified. As future activities for the project, the calibration of the prosthesis sensors, the development of movement prediction algorithms and the application of control strategies for the mechanical knee stand out.

Keywords: transfemoral prosthesis. prosthetic knee. embedded system. Sensors.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

1.1 Contextualização

A amputação pode ser descrita como a retirada parcial ou total de um membro do corpo, podendo ocorrer de forma traumática ou cirúrgica (JACQUELIN PERRY, 1992). Seu principal impacto é a deficiência infringida ao amputado, o que compromete funcionalidades e habilidades físicas do indivíduo, podendo acarretar em dificuldades para o trabalho e convívio social(SILVA GOMES et al., 2014). Amputações nos membros inferiores alteram ou impossibilitam o padrão de marcha natural humano, o que exige maior esforço e gasto energético por parte do amputado durante a caminhada. Esses fatores dificultam a mobilidade e reduzem a qualidade de vida de pessoas amputadas (MCGUIRE et al., 2007). Por isso, pesquisas e novas tecnologias são continuamente realizadas com o intuito de compensar as dificudades físicas desses indivíduos.

Um levantamento realizado pela Sociedade Brasileira de Angiologia e de Cirurgia Vascular (SBACV), constatou que no período de 2012 a 2021, 245.811 brasileiros sofreram amputação de membro inferior. Isso significa que em média 66 pacientes foram atendidos por dia e pelo menos três amputações foram feitas a cada hora. Das amputações inferiores as mais comuns são a transtibial e a transfemoral. A amputação transtibial é realizada entre a articulação tibiotársica e a articulação do joelho, enquanto que a amputação transfemoral é feita entre o joelho e o quadril (CARVALHO, J. A., 2003). É importante notar que um amputado transfemural vai necessitar de maior esforço energético que o amputado transtibial (CAMPOS; DRUMMOND; PAULA, 2022). Além disso, indivíduos que utilizam prótese em comparação com indivíduos saudáveis, durante a ambulação, vão precisar de cada vez mais esforço físico conforme a velocidade e a inclinação da superfície aumentam (CAMPOS; DRUMMOND; PAULA, 2022).

Considerando que uma prótese transfemoral passiva já é capaz de contribuir consideravelmente para a mobilidade do amputado, nota-se pelas informações acima, que o desenvolvimento de uma prótese semi-automatizada para amputados transfemorais é justificável e apresenta o potencial de melhorar e facilitar a vida de milhares de pessoas. Uma vez que, permite não apenas a recuperação de uma imagem corporal normal, como também compensação de parte do esforço desprendido pelo usuário. Isso ocorre, pois o objetivo do projeto é tornar a cinemática da prótese mais próxima da perna natural, o que mitiga a necessidade de compensações físicas por parte do membro saudável.

1.2 Objetivo

Com o intuito de ajudar no processo de recuperação de amputados e contribuir para o desenvolvimento tecnológico e científico brasileiro o Laboratório de Automação e Robótica da Universidade de Brasília (LARA) iniciou em 2005 o projeto RLEG para o desenvolvimento de uma prótese robótica transfemoral. Contudo, até o presente momento o projeto não foi finalizado, tendo passado por várias alterações físicas e conceituais. Atualmente o projeto tem as partes mecânica e eletrônica de uma prótese transfemoral semi-ativa já construídas.

Dada a situação descrita anteriormente, é foco desse trabalho estudar as pesquisas anteriores do projeto, a fim de retomar suas atividades. Assim como, realizar uma revisão e testagem dos componentes da prótese e implementar soluções em software que permitam o desenvolvimento de sistemas de controle mais robustos e adaptativos para cada usuário, permitindo a aproximação do movimento da prótese à cinemática da marcha humana. Tudo isso, com o fim de tornar a prótese RLEG um produto acessível e de grande impacto na melhoria do bem estar do usuário vítima de amputação.

1.3 Resultados e Apresentação

Através do presente trabalho, foi retomada a atividade do projeto RLEG, através de uma revisão eletrônica, um levantamento histórico e o desenvolvimento de uma aplicação para a realização de testes nos dispositivos da prótese. Foi possível observar a linha de evolução conceitual do projeto, começando como uma prótese ativa com controle mioelétrico até o formato atual que visa a criação de uma prótese semi-ativa com controle por identificação das fases do ciclo de marcha. Foi desenvolvido um programa em tempo real capaz de receber comandos e enviar dados através de comunicação USB com um computador. É possível por meio desse programa realizar testes eletrônicos nos dispositivos da prótese e receber relatórios contendo informações advindas dos sensores do sistema embarcado do projeto. Por fim, utilizando-se o programa implementado, foi feito o teste dos componentes eletrônicos do projeto e o bom funcionamento desses dispositivos foi verificado. Devido à solução criada neste trabalho, a testagem da prótese tornou-se muito mais simples, o que facilita as atividades futuras de desenvolvimento.

O presente manuscrito faz uma revisão de conceitos envolvendo marcha humana, amputação e prótese transfemoral, em seguida é apresentado um breve histórico do projeto RLEG que finaliza com uma revisão do sistema embarcado atual da prótese. O programa de testes implementado é então introduzido, seu funcionamento é mostrado, assim como as ferramentas necessárias para sua criação e aplicação. No final é revelado os resultados dos testes eletrônicos realizados com esse programa, o que demonstra seu funcionamento e dos dispositivos presentes na prótese.

2 Fundamentação

2.1 Ciclo de Marcha

2.1.1 Biomecânica da Marcha

A marcha humana pode ser definida como uma repetição de movimentos que visa a locomoção do corpo sem perda de estabilidade (JACQUELIN PERRY, 1992). Desse modo, o ciclo de marcha representa os eventos e movimentos que ciclicamente permitem a locomoção e a deambulação. A marcha humana possui duas fases principais: A fase de apoio e a fase de balanço (JACQUELIN PERRY, 1992). Essas fases são nomeadas de acordo com o membro de referência. Sendo a fase de apoio definida pelo contato do membro de referência com o solo, já na fase de balanço o membro em questão está no ar realizando um movimento pendular até o próximo contato com o chão. Cada um dos membros inferiores passa por essas fases de modo intermitente e com uma diferença de fase. Para um indivíduo saudável o ciclo de marcha é simétrico, pois, o movimento de um membro é posteriormente repetido pelo outro, o que facilita a estabilidade e reduz o gasto energético durante a caminhada.

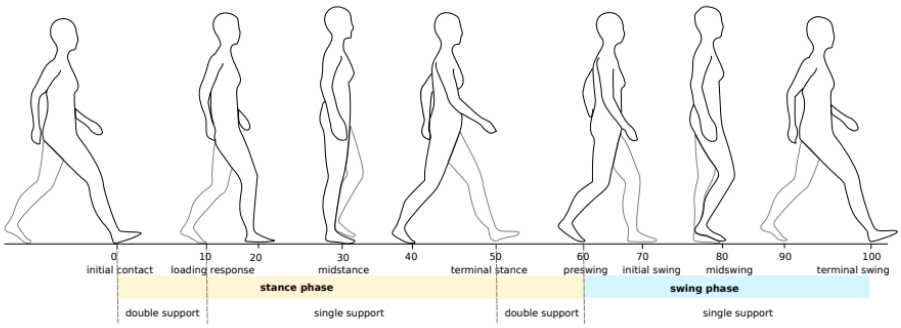

Figura 2.1 – Divisões Ciclo de Marcha

Adaptado de (NEUMANN, 2002)

Como explicitado pela Figura 2.1, o ciclo de marcha ainda pode ser dividido em subfases (JACQUELIN PERRY, 1992). A primeira subfase da marcha é chamada de contato inicial e corresponde ao momento em que o calcanhar da perna de referência toca o chão. A segunda subfase é denominada de resposta a carga e compreende o momento em que o pé se aplaina com o chão. Em seguida há a fase de apoio médio, onde o peso do corpo é totalmente distribuído sobre a sola do pé e o joelho apresenta maior extensão. A próxima subfase é a etapa final da fase de apoio, é referida como apoio terminal e representa o momento em o calcanhar perde o contato com o solo.

A fase de balanço inicia-se com a subfase pré-balanço, em que o apoio é transferido da perna de referência para a perna oposta. Seguem-se as subfases balanço inicial, quando os dedos deixam o solo, balanço médio, quando o membro de referência ultrapassa o membro oposto e, por fim, balanço terminal, quando o calcanhar retoma o contato com o chão. Na Tabela 2.1 os eventos que definem cada uma dessas subfases podem ser melhor observados.

Tabela 2.1 – Fases do ciclo de marcha

Fases do Ciclo de Marcha	Evento da Fase
Contato inicial	Contato do calcanhar com o chão
Resposta a carga	Pé se torna plano
Apoio médio	Carga total na perna de referência
Apoio terminal	Calcanhar deixa o solo
Pré-balanço	Início do apoio da perna oposta
Balanço inicial	Dedos deixam o solo
Balanço médio	Membro de referência ultrapassa o outro
Balanço terminal	Calcanhar retorna a posição de referência

A fase de apoio ainda pode ser dividida em apoio duplo e apoio simples. O apoio duplo ocorre no momento em que o peso do corpo é distribuído entre as duas pernas do indivíduo. O apoio simples, por outro lado, ocorre quando o peso atua apenas sobre o lado de referência. O apoio duplo ocorre desde o contato inicial até a resposta à carga, correspondendo a aproximadamente 10% do ciclo de marcha. Para todos os instantes remanescentes da fase de apoio, o suporte é do tipo simples.

2.1.2 Biomecânica do Joelho

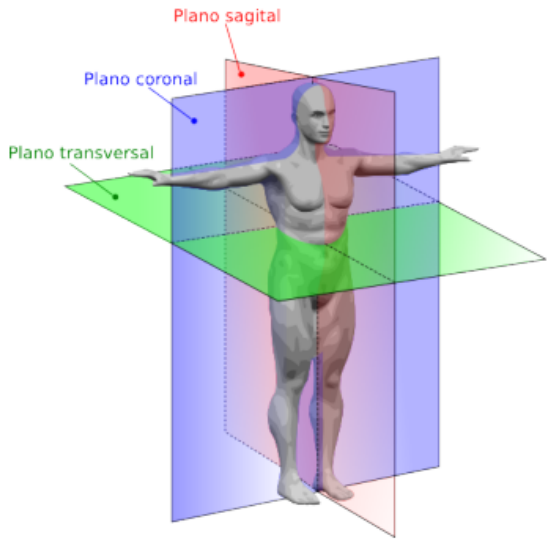

Figura 2.2 – Planos de referência do corpo humano

Fonte: Wikimedia Commons

O corpo é dividido em três planos de referência. A descrição desses planos pode ser vista na Figura 2.2. Com relação ao joelho o plano de maior relevância é o sagital, pois paralelamente a ele ocorrem as maiores amplitudes de movimento dos membros inferiores. Devido ao comportamento cíclico da marcha humana, as juntas do membro inferior possuem um padrão específico de repetição com relação à geração de torque e ao ângulo de extensão e flexão do joelho. Para este trabalho é de interesse a biomecânica do joelho, pois representa o padrão de comportamento biológico necessário para a identificação das fases do ciclo de marcha.

A Figura 2.3 demonstra a variação da angulação do joelho conforme o ciclo de marcha avança (NEUMANN, 2002). É possível observar a ocorrência de dois momentos em que a angulação do joelho é mais acentuada, o primeiro ocorre durante a fase de apoio em aproximadamente 10% do ciclo de marcha. Isso ocorre pois durante a fase de resposta a carga o joelho se flexiona para posteriormente estender-se de modo gradativo e abarcar o peso do corpo. Além disso, é importante perceber que durante a fase de balanço em aproximadamente 80% do ciclo de marcha há uma segunda flexão do joelho que ocorre devido ao movimento pendular da perna de referência.

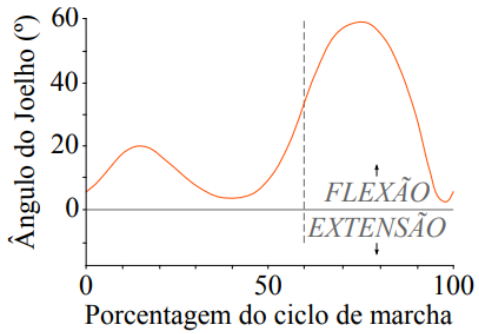

Figura 2.3 – Angulação do joelho conforme ciclo de marcha

Adaptado de (NEUMANN, 2002)

Com relação ao torque, na Figura 2.4 é possível observar o esforço desprendido pelo joelho durante o ciclo de marcha (NEUMANN, 2002). Durante o início da fase de apoio, quando há a tendência de extensão do joelho, pode-se observar um aumento do torque desprendido no membro, o que revela o esforço gradual para sustentar o peso do corpo. Em seguida há outra grande geração de torque, porém visando a flexão da articulação. Esse último movimento ajuda a gerar impulso para a fase de balanço. Já durante o balanço é possível perceber pouco esforço desprendido pelo joelho, o que é explicado pela ausência de contato com o chão. Nessa situação, a energia gasta no joelho visa manter um ângulo seguro e confortável para garantia de estabilidade durante o movimento e o retorno do pé ao solo.

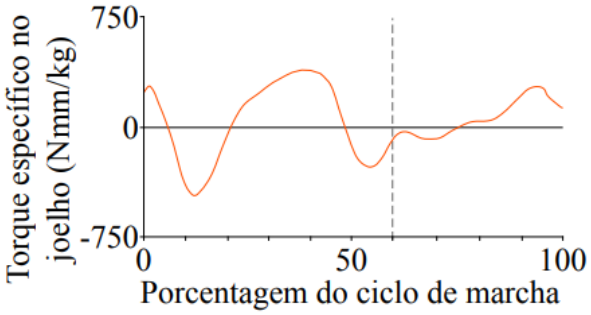

Figura 2.4 – Torque no joelho conforme ciclo de marcha

Adaptado de (NEUMANN, 2002)

2.1.3 Marcha do Amputado com Prótese

Um indivíduo que sofreu amputação apresenta perdas de funções anatômicas e físicas. Tal fator leva o paciente a desenvolver mecanismos de compensação física com o membro sadio ao utilizar uma prótese, o que implica em mais esforço e desconforto durante a locomoção. Alguns exemplos desses mecanismos de compensação são: Movimento lateral do tronco, inclinação para o lado não amputado e elevação do quadril (DIAZ, 2015). Mais exemplos desses tipos de compensação podem ser vistos na Tabela 2.2.

Um problema dessas estratégias de adaptação do corpo é o risco de desenvolvimento de novas patologias decorrentes do esforço aplicado. Exemplos de complicações advindas desse modo são a osteoartrite e a osteoporose (DIAZ, 2015). Com relação à dinâmica de movimento do paciente com prótese, é comum um prolongamento do tempo de apoio no membro saudável. Isso pode ser explicado devido ao pouco controle que o corpo possui sobre o membro protético, o que não gera confiança para o paciente manter mais tempo o apoio sobre a prótese, já que há a necessidade de manter a prótese estendida para garantir o apoio do corpo.

Tabela 2.2 – Problemas de marcha do amputado com prótese

Deficiência da Marcha	**Mecanismo de Compensação**
Flexão involuntária do joelho protético e má distribuição do peso corporal	Joelho protético completamente estendido, flexão anterior do tronco e maior tempo de apoio com o membro saudável
Falta de estabilidade durante deambulação	Movimento lateral do tronco
Encurtamento inadequado da prótese durante fase de balanço	Inclinação na direção do membro saudável
Interface inapropriada entre coto e soquete	Aumento da inclinação da pelve
Ausência de dorsiflexão do pé protético	Elevação do quadril

Fonte: (DIAZ, 2015)

2.2 Prótese Transfemoral

2.2.1 Componentes da Prótese

Uma prótese serve para substituir um membro amputado e auxiliar o paciente em algum trabalho de movimentação ao reaver parte da mobilidade perdida (MCGIMPSEY; BRADFORD, 2008). Ela precisa garantir o conforto e a saúde do paciente, além de apresentar uma aparência esteticamente agradável. Desse modo, buscando imitar uma perna humana uma prótese transfemoral é normalmente constituída de 4 componentes: Soquete, canela, joelho e conjunto pé-tornozelo .

Uma prótese genérica com todos esses componentes pode ser vista na Figura 2.5. O soquete é responsável pela conexão entre o amputado e a prótese, esse componente é normalmente fabricado de fibra de vidro, resina ou material acrílico e precisa ser ajustado com bastante precisão ao corpo do usuário. Um soquete mal colocado pode centralizar esforços e tensões cisalhantes em pontos indesejados do membro do usuário, trazendo lesões e desconforto durante a utilização da prótese (TROWER, 2006).

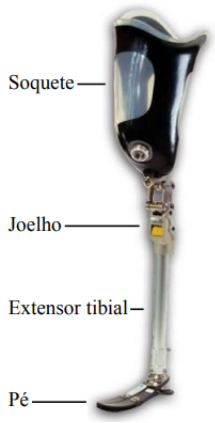

Figura 2.5 – Detalhamento de componentes de prótese transfemoral

Fonte: (ROCHA, T. S., 2015)

Os joelhos mecânicos apresentam uma grande variedade de tipos no mercado, desde sistemas simples apenas com um mancal rotativo, até complexos mecanismos com diversos elementos passivos e ativos para maior conforto e estabilidade. Contudo, os joelhos mecânicos podem ser divididos em dois grupos principais: Monocêntricos e Policêntricos. O primeiro, consiste de uma junta rotativa simples em que existe um centro de rotação fixo entre o soquete e a canela. O segundo, combina movimentos rotativos e translacionais de forma que o centro de rotação não é fixo e traça uma trajetória definida conforme o joelho é estendido e flexionado. Surge então um centro instantâneo de rotação que varia de acordo com o ângulo entre os eixos do soquete e da canela (ROCHA, T. S., 2015).

Um dos maiores fabricantes de joelhos protéticos na atualidade é a Ottobock, na Figura 2.6 tem-se um exemplo de joelho policêntrico desenvolvido por essa empresa. A canela, também chamada de extensor tibial, é um conjunto de hastes que faz a conexão entre o joelho mecânico e o conjunto tornozelo-pé. Geralmente feita de alumínio para maior resistência e menor peso é importante para ajustar a prótese à altura do usuário. Seus componentes podem ser estendidos ou retraídos para aproximar a prótese ao tamanho do membro sadio, o que ajuda a reaver parte da simetria corporal perdida com a amputação.

Figura 2.6 – Exemplo de joelho policêntrico Ottobock

Por fim, o conjunto tornozelo-pé é análogo ao próprio tornozelo humano, sendo fundamental para a estabilidade da caminhada e para absorção de impacto. O tornozelo também é a região do membro inferior responsável pela maior parte da potência empregada durante o movimento (HERR, H. M.; GRABOWSKI, 2012). No mercado o modelo de tornozelo-pé mais comum é o rígido, que não possui movimentação da articulação (TURCOT et al., 2013). Esse modelo garante mais estabilidade para a realização de movimentos mais simples, porém também reduz a qualidade e quantidade desses movimentos.

Um exemplo dessa redução é a incapacidade de realizar dorsiflexão do tornozelo. O que é suficiente para gerar anomalias na movimentação do corpo durante a caminhada, como visto na tabela 2.2. Uma alternativa para a ampliação da capacidade motora da prótese e a utilização de tornozelos articulados com relação ao plano sagital do corpo. Tal mecanismo permite a dorsiflexão e pode incluir dispositivos de armazenamento e geração de energia, tornando o movimento da junta mais similar com o de um tornozelo natural (TURCOT et al., 2013). Na Figura 2.7 é possível observar um exemplo de conjunto tornozelo-pé capaz de articulação no plano sagital.

Figura 2.7 – Exemplo de tornozelo-pé com articulação - Proprio Foot da Össur

2.2.2 Tipos de Prótese

Atualmente as próteses transfemorais são divididas em três grupos principais: As próteses passivas, ativas e semi-ativas (ASIF et al., 2021). Próteses passivas são caracterizadas pela ausência de dispositivos eletromecânicos que regulam a movimentação da prótese. Assim, tais próteses podem apresentar juntas simples ou policêntricas para substituir a função do joelho e componentes elásticos como molas e pistões para conservar parte da energia desprendida durante um período do ciclo de marcha, a fim de descarregá-la em outro momento. Em (UNAL et al., 2010) é idealizada uma prótese para esse propósito.

Esse tipo de prótese não possui recursos para ajudar o usuário no desprendimento de potência durante a locomoção, logo toda a energia e esforço necessários para movê-la são advindos do próprio usuário. Dados esses fatores, a prótese passiva possui a vantagem do baixo custo e da simplicidade, entretanto, sua capacidade de reconstituição da mobilidade do paciente é reduzida, pois gera ciclos de marcha assimétricos e reduz a velocidade de deambulação em comparação com indivíduos saudáveis. Além disso, exige maior gasto energético por parte do usuário (KAUFMAN; FRITTOLI; FRIGO, 2012). A imagem 2.8 representa o design conceitual da prótese transfemoral passiva desenvolvida por (UNAL et al., 2010).

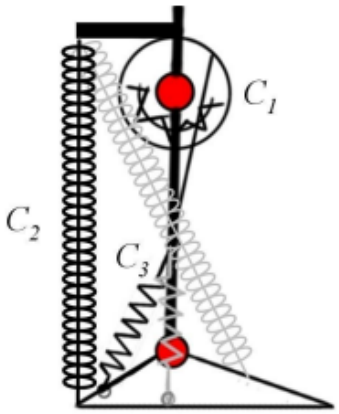

Figura 2.8 – Design conceitual de prótese passiva

Fonte: (UNAL et al., 2010)

Com relação à prótese ativa, seu diferencial é a presença de mecanismos ativos de aplicação de torque e potência ao movimento do amputado. Esse tipo de prótese necessita de dispositivos eletromecânicos para esse fim e por isso emprega também o uso de sistemas eletrônicos, sistemas embarcados e sensores. Os dispositivos mais utilizados nesse tipo de prótese são motores DC, motores de passo e pistões hidráulicos e pneumáticos. Para comandar a potência desprendida por tais elementos são necessários mecanismos de controle e identificação do ciclo de marcha, o que exige a utilização de microcontroladores e estratégias de controle.

Tais características tornam esse tipo de prótese mais cara e mais complexa de implementar. Por outro lado, sua capacidade de reposição da mobilidade do indivíduo é alta permitindo que o usuário volte a realizar atividades como subir escadas e levantar-se de um assento (ELERY et al., 2020). Na Figura 2.9 observa-se um exemplo de prótese ativa. Sua estratégia de atuação é realizada por meio de pistões pneumáticos.

Figura 2.9 – Exemplo de prótese ativa atuada por pistões pneumáticos

Fonte: (SUP; BOHARA; GOLDFARB, 2008)

Por fim, a prótese semi-ativa apresenta uma mesclagem entre os tipos anteriores de próteses. Esse modelo não possui ferramentas que empregam esforço diretamente ao movimento, mas sim formas de responder à potência empregada pelo próprio usuário (ADAMCZYK, 2020). Entre os dispositivos mais utilizados por esse tipo de prótese, se destaca o pistão magneto relógico. Tal equipamento é capaz de regular a própria resistência mecânica em função da corrente elétrica aplicada aos seus terminais.

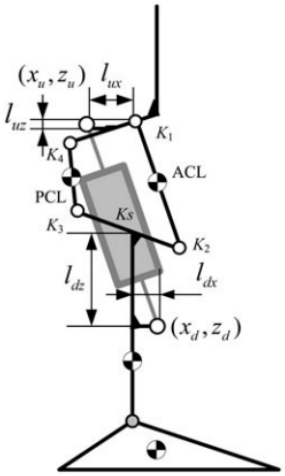

Figura 2.10 – Prótese semi-ativa com atuador magneto-relógico

Fonte: (XIE et al., 2010)

Também é necessária a aplicação de microcontroladores e sensores nesse modelo, visto o objetivo de amortecer a movimentação da prótese em momentos específicos do ciclo de marcha. Para isso é fundamental também a aplicação de estratégias de controle que regulam a resistência do pistão magneto-relógico e a implementação de algoritmos de identificação de fases do ciclo de marcha. Próteses semi-ativas apresentam um meio termo bastante vantajoso, pois são mais baratas que as ativas e apresentam melhor desempenho do que as passivas em estabilidade e conforto para o usuário. Na imagem 2.10 é possível observar o design conceitual de uma prótese com atuador magneto-relógico e junta policêntrica.

2.2.3 Estimação Ciclo de Marcha e Controle

Próteses de joelho tem o objetivo de mimetizar o comportamento de membros humanos saudáveis (ZHANG et al., 2021). Essa tarefa demonstra-se particularmente difícil no âmbito da intenção de movimento, pois, para que a prótese contribua para com a deambulação do usuário é necessário prever a movimentação desejada. Com esse intuito são desenvolvidos os algoritmos de predição de movimento, que no caso de uma prótese transfemoral irá descrever a angulação e as fases do ciclo de marcha durante a locomoção.

Existem várias estratégias para prever o comportamento esperado para o joelho durante a caminhada. Uma delas é descrever o padrão de movimentação humana através de captura por vídeo. Como realizado em (HUANG et al., 2007) é possível posicionar marcadores ao longo do corpo e ao gravar a deambulação descrever a trajetória realizada por cada marcador. Essa estratégia descreverá padrões relacionados à deambulação como: O comprimento da passada, o período da passada, o período em apoio duplo, os ângulos de saída e chegada do calcanhar no chão, a altura atingida pelo pé durante o balanço, o deslocamento vertical da bacia e a movimentação da bacia no plano sagital e frontal.

Outros métodos por outro lado vão valer-se de novas tecnologias computacionais, tais como inteligência artificial para tentar gerar relações causais entre características mecânicas da passada humana. (DEY et al., 2020) utiliza-se do algoritmo de machine learning Random Forest para prever a cinemática do joelho através da angulação, velocidade angular e aceleração angular da coxa. Ainda em (DELIS, A. et al., 2008) é proposto um algoritmo de controle e estimação do ângulo do joelho protético a partir da análise de sinais mioelétricos e da aplicação de uma rede neural. No momento em que as fases do ciclo de marcha podem ser identificadas, as estratégias de controle podem ser utilizadas, tais como: Q-Learning, máquina de estados e modelagem matemática.

A modelagem matemática pode ser aplicada escolhendo-se uma angulação para o joelho e conhecendo-se a equação dinâmica do sistema. Desse modo, pode-se definir o nível do coeficiente de amortecimento mecânico necessário para atingir-se a angulação estipulada (KIM; OH, 2001). Para a utilização de uma máquina de estados, primeiro realiza-se a sub-divisão do ciclo de marcha e cada uma das fases geradas vai representar um estado

na máquina de estados do sistema. Para cada estado é associada uma constante de resistência mecânica ao pistão magneto-relógico de modo a aproximar o movimento da prótese do movimento natural humano (HERR, H.; WILKENFELD, 2003). Por outro lado, a aplicação de Q-learning é baseada em gerar políticas de controle a partir de um ambiente que recebe informações de performance e treinamento em relação ao sucesso ou falha da atuação de controle (HUTABARAT et al., 2020). Ainda é possível como descrito em (PARK et al., 2016) realizar um controle PD da prótese, utilizando-se como referência uma aproximação polinomial para a curva de angulação do joelho no decorrer da deambulação.

2.3 Prótese RLEG

2.3.1 Histórico do Projeto

O projeto RLEG é uma iniciativa do LARA (Laboratório de Robótica e Automação) da Universidade de Brasília e visa a criação de uma prótese inteligente para amputados transfemorais. A prótese tem passado por diversas mudanças ao longo de seu desenvolvimento, desde ligeiras alterações até completas mudanças funcionais e estruturais. As primeiras tentativas de implementação da prótese ocorreram no ano de 2005 com o seu primeiro projeto mecânico. Na Figura 2.11 é possível visualizar esse dimensionamento. Inicialmente o projeto propunha uma prótese ativa com a utilização de motores DC controlando as juntas em todos os três graus de liberdade descritos na Figura 2.11.

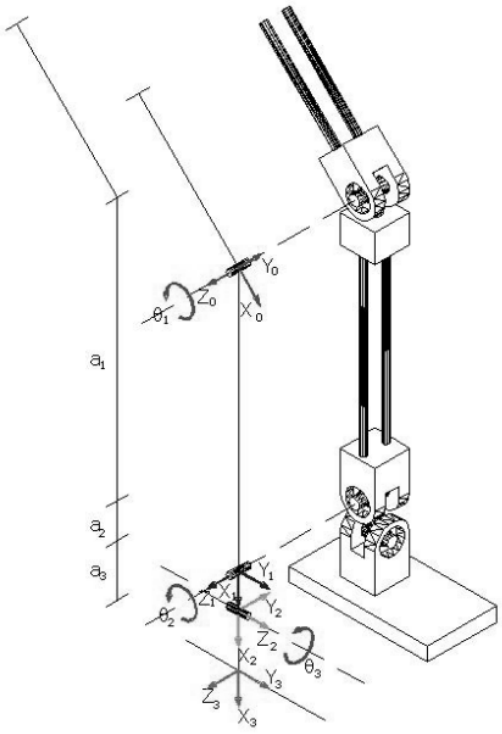

Figura 2.11 – Primeiro desenho mecânico da prótese RLEG

Fonte: (CASCÃO JR et al., 2005)

Um sistema embarcado com capacidade de comunicação serial por protocolo RS485 foi implementado no projeto, além de sensores de ultrassom para verificação da proximidade do pé com o solo e acelerômetros para controle de torque empregado pelos motores. A concepção inicial da prótese buscava a implementação de controle por sinais mioelétricos, sendo que algoritmos para tratamentos desses sinais e atuação sobre a prótese chegaram a ser propostos em (FERREIRA et al., 2005).

A prótese também já teve um controlador PI analógico implementado com sucesso como visto em (JÚNIOR, 2005). Na figura 2.12 é visível o diagrama de blocos da primeira arquitetura de implementação e controle da prótese. No ano de 2006 novas atualizações foram realizadas na eletrônica da prótese e também novas tentativas de controle (BECKMANN; SANTOS, 2006). As alterações realizadas resumem-se em novas placas para controle digital dos motores das juntas utilizando o microcontrolador ATMEGA8. O controle digital realizado nesse trabalho não obteve o êxito esperado em parte devido a folgas no acoplamento mecânico do motor. Também foi construído um suporte para fixação e testagem da prótese, como visto na Figura 2.13.

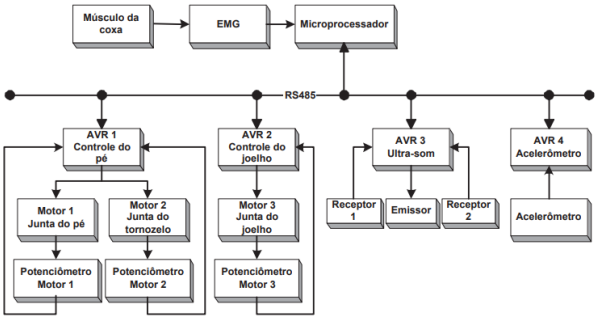

Figura 2.12 – Diagrama de blocos primeiro controle da prótese RLEG

Fonte: (JÚNIOR, 2005)

Figura 2.13 – Suporte construido para teste da prótese antiga

Fonte: (BECKMANN; SANTOS, 2006)

Em 2007 o trabalho (SCANDAROLI, 2007) realizou novas alterações na prótese visando a implementação de controle PID e adaptativo. Novas placas de acionamento dos motores foram contruídas e também um circuito de interfaceamento entre a prótese e o computador. No mesmo ano o trabalho (SILVA ALVES, 2007) introduziu métodos de estimação da atitude do pé com relação ao solo utilizando-se sensores infravermelho. No ano seguinte, 2008, o trabalho (BRASIL, 2008) deu continuidade ao anterior implementando um controle PI para ajuste do posicionamento do pé com relação ao solo. Foi também a partir de 2008 que iniciaram-se as tentativas de obtenção de sinais de referência para o controle da prótese a partir do tratamento de sinais mioelétricos. Os trabalhos (DELIS, A. L.; ROCHA, A. F. D. et al., 2008), (DELIS, A. et al., 2008), (FÉLIX; VASCONCELLOS, 2008), (DELIS, A. L.; CARVALHO, J. L. A. de et al., 2009), (DELIS, A. L.; DE CARVALHO et al., 2010) e (DELIS, A. L., 2010) ilustram essas tentativas e foram o foco do trabalho com a prótese entre os anos de 2008 e 2010.

No ano de 2011 o trabalho (MARTINS; CHAURAIS; ROCHA, T. S., 2011) buscou novamente construir controladores para cada junta da prótese. Foi feita a modelagem matemática do sistema e a identificação de parâmetros para o modelo construído. Na Figura 2.14 é mostrada uma foto da prótese durante a execução desse trabalho. O controle realizado enfrentou diversas dificuldades, principalmente devido a problemas mecânicos e elétricos. Foi relatada a ocorrência de atrito excessivo no motor do joelho, excentricidade considerável do eixo da caixa de redução, falha mecânica das engrenagens da caixa de redução e folga nos acoplamentos dos motores. Ainda é citada a ocorrência de mal contato entre parte dos circuitos montados na prótese.

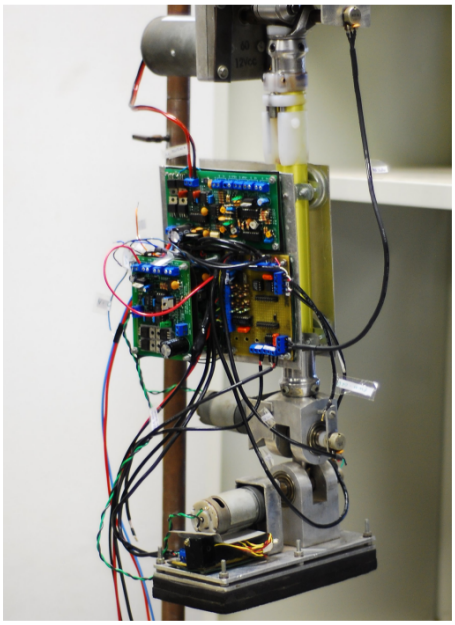

Figura 2.14 – Prótese RLEG no ano de 2011

Fonte: (MARTINS; CHAURAIS; ROCHA, T. S., 2011)

No período de 2012 a 2014 não foram realizados trabalhos significativos com a prótese RLEG. O projeto só foi retomado no ano de 2015 com o trabalho (ROCHA, T. S., 2015) que representou uma revolução completa na proposta do projeto. Foi realizado um novo desenho mecânico e foi alterado o tipo de prótese e o tipo de atuação sobre o joelho. Esse trabalho substituiu o modelo de joelho protético anterior por um mecanismo de quatro barras que é mecanicamente mais próximo de uma articulação natural. A Figura 2.15 representa o projeto mecânico realizado. Além disso, a prótese passou a ser semi-ativa tendo como atuador apenas um pistão magneto-relógico capaz de ajustar a resistência mecânica no joelho durante sua extensão e flexão. Ainda foi implementado um sistema embarcado com controle de corrente capaz de ajustar manualmente a amperagem entregue ao atuador.

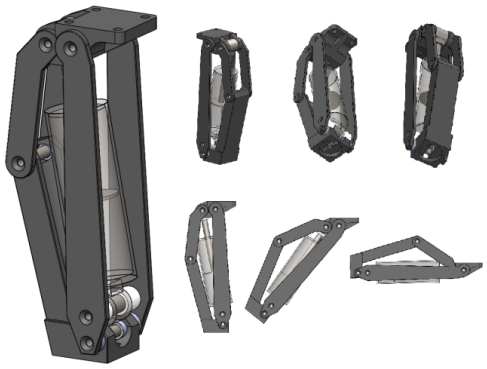

Figura 2.15 – Projeto mecânico do joelho RLEG

Fonte: (ROCHA, T. S., 2015)

Entre os anos de 2016 e 2018 novamente não foram realizados trabalhos significativos com a prótese. Apenas em 2019 o trabalho (BENTO, 2019) trouxe atualizações para o projeto com a implementação de um novo sistema embarcado. A nova placa eletrônica faz a substituição do microcontrolador utilizado e inclui um sensor de corrente para aferição da potência entregue ao atuador magneto-relógico. Essa placa é a que compõe a eletrônica atual do projeto e seus componentes serão descritos em mais detalhes a seguir. Na Figura 2.16 é visível uma imagem atual da prótese.

Figura 2.16 – Prótese RLEG Atualmente

2.3.2 Revisão de Hardware

No projeto atual da prótese RLEG é utilizado o microcontrolador $STM32F103C8T6$ também chamado de "Bluepill". Esse é um microcontrolador de $32bits$. A programação é feita em linguagem C por meio da biblioteca HAL de alta abstração, desenvolvida pelo próprio fabricante $STMicroeletronics$. Por meio desse componente é possível realizar comunicação Serial-Uart, I2C, SPI e USB com vários dispositivos e com um computador convencional. Também é possível realizar conversão analógico digital, ativar portas digitais de propósito geral, gerar sinais PWM e configurar temporizadores. Os sensores utilizados são um encoder absoluto angular AMT-20, um módulo inercial e um sensor de corrente. O atuador localizado na junta do joelho é um pistão magneto-relógico LORD-RD-8040-1. A Figura 2.17 representa o design de PCB da placa eletrônica em questão.

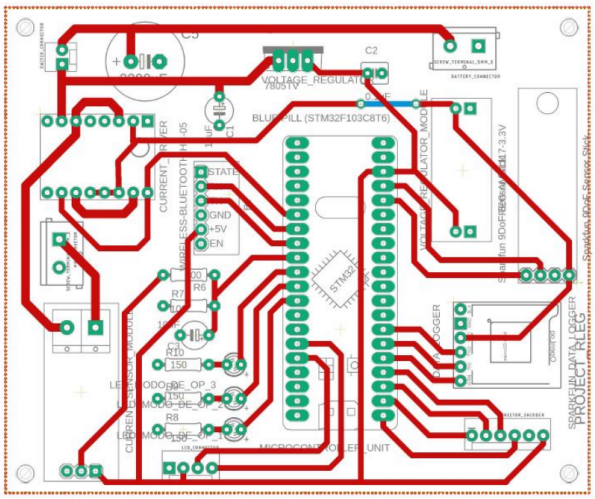

Figura 2.17 – Esquemático PCB - RLEG

Os componentes eletrônicos da prótese exigem alimentação de $12V$, $5V$ e $3{,}3V$ e para atender a esses requisitos são necessários dois reguladores de tensão e uma bateria de $12V$. Além disso, existem capacitores de $2200\mu F$ e $10\mu F$ responsáveis pela filtragem de ruídos e picos de tensão na alimentação do circuito. A placa possui ainda terminais molex e barras de pino fêmea, que permitem a conexão de dispositivos externos com o microcontrolador através de comunicação SPI, I2C e Serial-Uart. Os terminais de comunicação SPI são utilizados para acessar dados do encoder angular absoluto que é externo a placa do projeto.

O driver de corrente empregado na placa é o CI TB6612FNG, ele é utilizado para regular a corrente fornecida ao atuador LORD RD-8040-1. O funcionamento desse CI baseia-se na topologia de uma ponte H. É suportado por esse componente uma tensão de $6V$ a $13{,}5V$ nos terminais de alimentação da carga. Sua lógica funciona por meio de três canais de entrada que suportam até $6V$. Desses três canais, dois operam indicando o estado da ponte H e o último fornece o sinal PWM que regula o nível de potência entregue a carga. A Figura 2.18 relaciona as combinações de nível lógico dos canais com os estados da ponte H.

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Figura 2.18 – Relação Canais X Saída TB6612FNG

O sensor de corrente presente no projeto visa mensurar o nível de potência entregue ao atuador magneto-relógico. É utilizado para isso um módulo que contém o CI ACS712 que consiste em um sensor por efeito hall. Esse componente é um sensor analógico alimentado com uma tensão de $5V$ que possui capacidade de medir amperagem de $-30A$ até $30A$. A tensão analógica do pino de saída pode variar de $0V$ a $5V$, sendo o valor intermediário de $2{,}5V$ equivalente à corrente de $0A$. Seu terminal analógico gera um acréscimo de $83mV$ para cada ampér mensurado.

No contexto do projeto RLEG o microcontrolador é capaz de realizar conversões analógico-digital de até 12bits. Entretanto suas portas ADC operam sob tensão nominal de $3{,}3V$ o que exige um divisor de tensão na saída analógica do sensor. Na placa do projeto é aplicado dois resistores de 100Ω dividindo o sinal pela metade. Desse modo é possível descrever a mínima variação de corrente observada pelo microcontrolador através da relação 2.1.

$$\Delta Imin = \frac{2 * Tp}{Ss * 2^{Nbits}} = \frac{2 * 3{,}3}{0{,}083 * 2^{12}} = 19{,}41mA \tag{2.1}$$

- ΔImin - Variação de corrente perceptível
- Tp - Tensão nominal do pino ADC
- Nbits - Número de bits do conversor ADC
- Ss - Sensibilidade do sensor de corrente

Também é empregado no projeto o módulo inercial MPU-6050. Esse módulo é capaz de realizar medições de aceleração linear e velocidade angular em todos os três eixos espaciais e também aferir a temperatura ambiente. Suporta tensão de operação de até $6V$ e se comunica com o microcontrolador por meio de protocolo I2C. Os dados coletados possuem 16 bits de tamanho. Na Figura 2.19 é mostrado o posicionamento dos eixos de orientação do módulo.

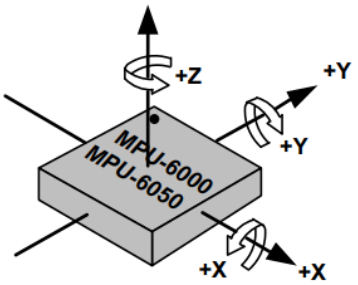

Figura 2.19 – Orientação dos eixos do módulo inercial

Para captura da angulação do joelho é empregado o encoder absoluto AMT-20. Esse componente é alimentado com $5V$ e se comunica com o microcontrolador por meio de protocolo SPI. Os dados coletados possuem 12 bits de tamanho e a forma de identificação de angulação utilizada por esse sensor é a quadratura de pulsos. Esse método se baseia em dois canais de sinais que geram pulsos quadrados defasados 90° conforme o eixo de rotação do sensor gira. A Figura 2.20 demonstra o funcionamento desse mecanismo. Interessante perceber que devido a defasagem dos sinais é possível perceber o sentido de rotação do eixo. Considerando-se a resolução de 12 bits para os dados de angulação, uma rotação completa do eixo é dividida em 4096 intervalos resultando em uma variação perceptível de angulação de aproximadamente 0,088°.

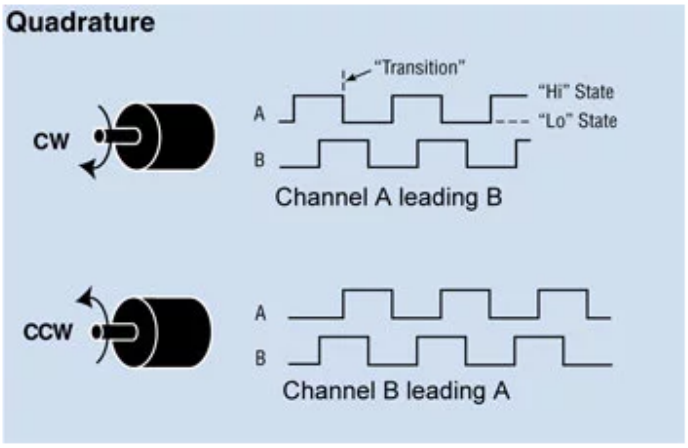

Figura 2.20 – Pulso de sinais em quadratura

O atuador magneto-relógico Lord RD-8040-1 empregado no projeto possui tensão nominal de $12V$ e em temperatura ambiente possui resistência interna próxima a 5Ω. A sua capacidade de corrente máxima é de $2A$ e pode operar entre as temperaturas de $-40°C$ e $100°C$. Vale lembrar que o pistão magneto-relógico possui ferrofluido em seu interior e devido ao campo magnético gerado pela corrente elétrica esse líquido aumenta sua viscosidade, o que aumenta por consequência a resistência mecânica do dispositivo.

3 Desenvolvimento

Com o intuito de manter uma base de dados definitiva sobre os avanços no projeto, a primeira atividade realizada foi a criação de um ambiente exclusivo para o desenvolvimento da prótese RLEG e registro de informações envolvendo o projeto, desde relatórios de teste dos componentes eletrônicos até artigos, trabalhos e datasheets utilizados como base para as atividades realizadas. No laboratório reservou-se uma bancada, um computador e um monitor exclusivamente para o uso do projeto. Nesse computador foram instalados os softwares utilizados para programação e depuração do microcontrolador $STM32F103C8T6$. Além de outros programas úteis como, Github, Python3 e Putty. Desse modo, todo o resultado e conteúdo do trabalho realizado pode ser acessado por quem continuar a atividade de desenvolvimento da prótese RLEG.

3.1 Ferramentas de Programação e Depuração

Para a construção dos algoritmos aplicados na testagem e comunicação da prótese foi fundamental a utilização de softwares que facilitam o trabalho de desenvolvimento, considerando a complexidade de pinagem e configuração do microcontrolador utilizado no projeto. As principais ferramentas computacionais utilizadas foram as IDEs $SystemWorkbench(SW4STM32)$ e $STM32CubeIDE$, a ferramenta gráfica de configuração de microcontroladores $STM32CubeMX$ e o emulador de terminais $Putty$.

3.1.1 System Workbench

O $SystemWorkbench$ também chamado de $SW4STM32$ é uma interface de desenvolvimento de software compatível com Línux e Windows e que possui suporte a toda gama de microcontroladores da $STMicroeletronics$. Treinamentos, tutoriais e fóruns sobre a ferramenta podem ser encontrados no site $www.openstm32.org$. Essa ferramenta possui soluções para depuração, carregamento de instruções, compiladores C e C++ e terminal para comunicação Serial.

3.1.2 STM32CubeIDE

O $STM32CubeIDE$ é uma ferramenta alternativa ao $SystemWorkbench$, pois também consiste em uma plataforma de desenvolvimento de software com várias outras aplicações agregadas. Possui suporte para compilação e geração de código em C e C++, depuração de microcontroladores STM32 e configuração de periféricos.

3.1.3 STM32CubeMX

O *STM32CubeMX* é uma ferramenta gráfica para visualização e configuração de periféricos dos microcontroladores STM32. É possível por meio desse software gerar os códigos iniciais de configuração da pinagem e processos do *STM32F103C8T6*. A utilização do programa baseia-se em escolher inicialmente o modelo de microcontrolador do projeto e então uma janela com as opções de configuração deste dispositivo é gerada. É possível definir os pinos de entrada e saída digitais, pinos para conversores analógico-digital, temporizadores e protocolos de comunicação. Além de configurar o clock do microcontrolador e a utilização de ferramentas para processamento em tempo real como FreeRTOS. A Figura 3.21 mostra a tela do *STM32CubeMX* para configuração de periféricos.

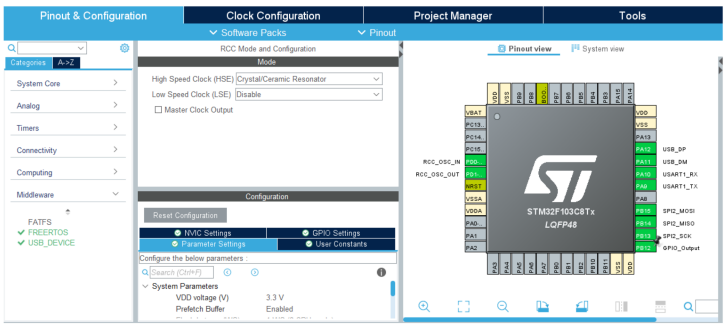

Figura 3.21 – Janela do *STM32CubeMX* para configuração de microcontroladores

3.1.4 Putty

Putty é um emulador de terminais gratuito e de código livre. Suporta diversos protocolos de comunicação como SSH, Telnet e Serial. Seu propósito é atuar como cliente para conexão entre dispositivos, possuindo diversas opções de configuração dessas conexões. É um programa simples que não exige instalação, o executável baixado é suficiente para o funcionamento do programa. No contexto do projeto RLEG o propósito do Putty é servir como um terminal de comunicação entre o sistema embarcado da prótese e o computador, permitindo a visualização dos dados coletados pelos sensores do projeto e a inserção de comandos para o microcontrolador.

3.2 Software Implementado

O objetivo do programa construído é realizar a testagem dos componentes da placa eletrônica da prótese e permitir que qualquer pessoa possa realizar esse teste enviando comandos simples por meio de um terminal conectado via protocolo Serial com o microcontrolador do projeto. O terminal utilizado durante o desenvolvimento e verificação do software foi o Putty. Foram implementados os comandos listados na Tabela 3.3. Tais comandos são enviados ao microcontrolador digitando-se as letras correspondentes no terminal.

Tabela 3.3 – Lista de Comandos

Comando	Caracter Correspondente	Descrição
Menu	"c"	Lista os modos de operação e indica o comando para inicialização de cada modo
Sair	"q"	Finaliza o procedimento de teste em execução
Pausar	"p"	Interrompe o procedimento de teste em execução
Retomar	"r"	Retoma o teste em execução

3.2.1 Configuração do Microcontrolador

De início foi realizada a configuração dos periféricos do microcontrolador através da ferramenta *STM32CubeMX*. Na placa do projeto estão presentes três leds conectados a pinos GPIO, um sensor de corrente conectado a um pino conversor analógico-digital, um driver de corrente comandado por sinal PWM, um módulo inercial com comunicação I2C e um encoder angular com comunicação SPI. Para cada um desses componentes o microcontrolador teve sua pinagem configurada de acordo a necessidade dos dispositivos. Na Tabela 3.4 os pinos utilizados e suas respectivas funcionalidades podem ser visualizados. Também foram configurados mais três temporizadores internos ao microcontrolador, um servindo como clock do sistema e mais dois para contagem de tempo em diferentes intervalos mínimos. Foi configurado ainda por meio do *STM32CubeMX* a comunicação USB-Serial do microcontrolador e a utilização de FreeRTOS para processamento em tempo real dos dados coletados. Foram criadas três threads para o funcionamento do sistema, uma para administração do procedimento de teste, outra para a operação padrão da prótese e a última para tratamento de uma fila circular de transmissão de dados via USB-Serial.

Tabela 3.4 – Configuração de Pinos *STM32F103C8T6*

Dispositivo	Configuração	Pinagem
Leds	GPIO	A5, A6, A7
Driver de Corrente	Saída PWM	A1
Sensor de Corrente	Conversor ADC	A4
Módulo Inercial	Comunicação I2C	B8, B9
Encoder Angular	Comunicação SPI	B12, B13, B14, B15

3.2.2 Fila Circular

Os dados lidos pelos sensores são transformados em caracteres e armazenados dentro de uma lista circular antes de serem enviados ao computador. Esse procedimento é realizado para que a tarefa de envio de dados não atrase a leitura dos sensores. A fila circular implementada funciona por meio de um vetor com tamanho fixo de 500 bytes e dois índices que representam o final e o início dos dados armazenados.

Toda vez que um caracter é inserido na fila de transmissão ele é posicionado no elemento correspondente ao índice final da fila e esse índice é então incrementado em uma unidade passando para o elemento seguinte do vetor. O byte enviado ao computador é sempre aquele para o qual aponta o índice de início da fila, uma vez realizada a transmissão, esse índice também é incrementado em uma unidade passando para o elemento seguinte.

Toda vez que um dos índices aponta para o quingentésimo elemento sua próxima iteração o levará de volta para o primeiro elemento do vetor, de modo a preservar a memória do microcontrolador. A Figura 3.22 exemplifica a estratégia utilizada para implementação da fila circular, nesse exemplo os elementos azuis do vetor representam dados armazenados na fila enquanto os elementos brancos representam o espaço ainda disponível para armazenamento.

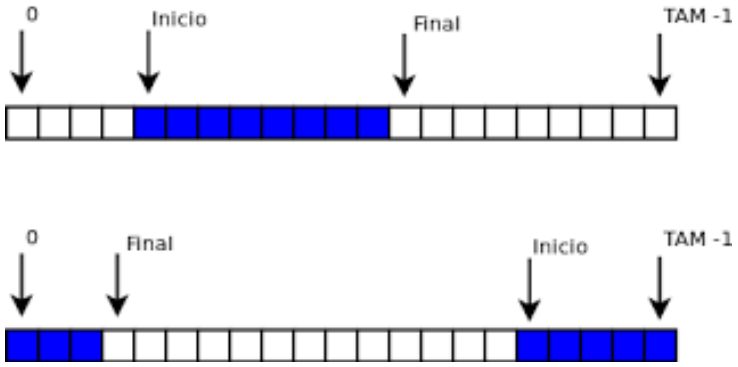

Figura 3.22 – Modelo de fila circular implementado

Enquanto o procedimento de testes é realizado, os dados transformados em caracteres e outros bytes de informação como quebras de linha e retorno de carro são introduzidos na fila circular. Esse procedimento reduz o tempo de amostragem dos dados, uma vez que o processo de atribuição de valor a uma variável é mais veloz do que a transmissão via USB.

Todo procedimento de teste implementa um tempo de espera para configuração do período de amostragem, logo existe um intervalo de tempo, entre cada medição realizada, em que a thread de testes interrompe seu funcionamento e passa a prioridade de execução para outra tarefa. Nesse momento, a thread de transmissão de dados inicia sua execução, esvaziando a fila enquanto o procedimento de teste espera o momento de rotomada da atividade de aferição.

Existe a possibilidade de perda de dados, caso a taxa de recebimento de dados exceda a taxa de envio, já que caracteres podem ser sobrescritos antes de enviados ao computador. Para prevenir esse problema a thread de envio de dados tem sua prioridade aumentada quando o nível de preenchimento da fila está muito elevado. Vale ressaltar que a fila circular é um recurso compartilhado entre as threads, o que exige a utilização de mutex para impedir o acesso simultâneo a este recurso.

3.2.3 Descrição das Threads

Das threads criadas a destinada para o funcionamento padrão da prótese é onde futuramente devem ser implementadas as estratégias de controle para o joelho protético. Já a thread de administração dos procedimentos de teste é a responsável por definir os dispositivos que serão testados por meio dos comandos inseridos no terminal do computador. Seu funcionamento baseia-se em um grupo de rotinas que constantemente sondam um buffer de dados recebidos por meio da comunicação USB com o computador, para então tomar decisões. Toda vez que o conteúdo do buffer é aferido, em seguida seus dados são apagados para evitar a leitura repetida do mesmo caracter.

A primeira rotina da thread apenas sonda o buffer USB a espera do comando de abertura do menu, descrito na Tabela 3.3. No instante em que esse comando é percebido uma string contendo o conteúdo do menu é enviado ao computador e escrito na tela do terminal, o que revela as opções de funcionamento da prótese. Foram idealizadas duas funcionalidades: Teste de componentes e operação padrão que consiste no algoritmo de controle a ser implementado. Vale ressaltar que no momento em que a prótese é ligada ou reiniciada seu modo de funcionamento é configurado como operação padrão automaticamente. Para iniciar o modo de teste de componentes basta teclar o caracter correspondente a essa opção descrita no menu. Uma imagem do menu pode ser vista na Figura 3.23.

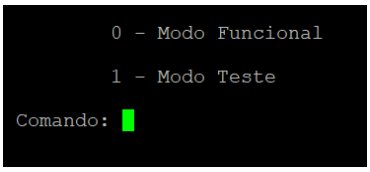

Figura 3.23 – Menu de modos de operação

Uma vez selecionado o modo de teste, aparecerá no terminal o menu de testes, que consiste nas opções de teste implementadas no programa. Foram implementadas cinco opções: Teste das portas GPIO, teste do driver de corrente, teste do sensor de corrente, teste do módulo inercial e teste do encoder angular. Novamente cada opção é associada a um caracter e teclar esse caracter carregará a opção escolhida. Na imagem 3.24 é possível visualizar o menu de testes implementado. Cada uma das opções de teste corresponde a um dispositivo da placa do projeto descrito na Tabela 3.4.

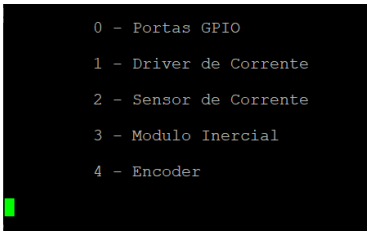

Figura 3.24 – Menu de teste

A opção selecionada iniciará uma rotina para configurar e ativar o dispositivo em questão. No caso das portas GPIO os LEDs piscarão em sequência, no caso do driver de corrente a potência entregue ao pistão magneto-relógico será gradativamente aumentada até seu valor máximo e então zerada novamente, já nos casos do sensor de corrente, do módulo inercial e do encoder as leituras realizadas serão mostradas no terminal, assim como o intervalo de tempo entre aferições.

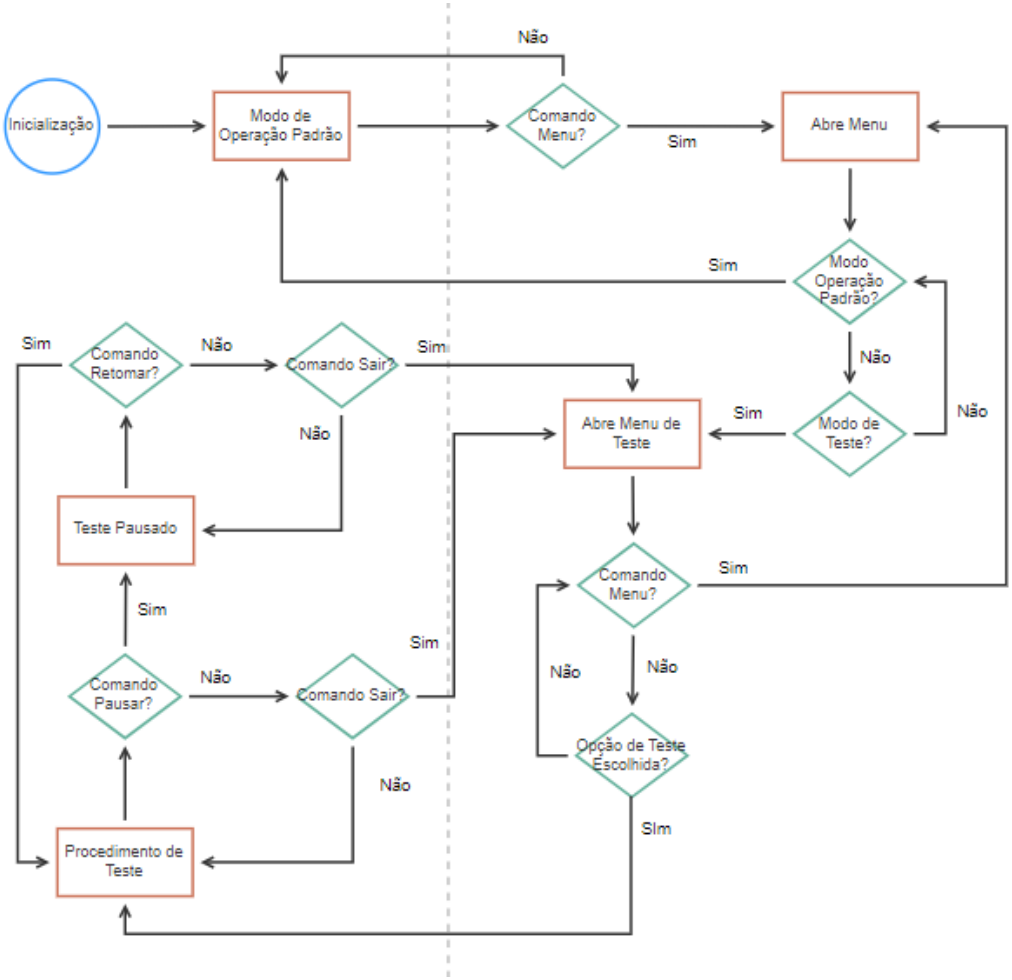

Figura 3.25 – Fluxograma da thread de testes

As rotinas de ativação dos dispositivos da placa também sondam continuamente o buffer de dados advindos do USB a procura dos comandos descritos na Tabela 3.3 e também limpam o buffer a cada sondagem. Identificar o comando de pausa criará uma rotina ociosa que interrompe a rotina anterior e apenas verifica se o buffer contém o comando de retomada. Isso causa a parada temporária do procedimento de teste.

Quando o comando de retomada é identificado a rotina ociosa é finalizada e o teste continua de onde parou. Identificar o comando de saída dentro de qualquer das duas rotinas vai finalizar o procedimento de teste, terminar de enviar os dados ao computador e abrir novamente o menu de testes para que uma nova opção de teste seja selecionada. O fluxograma da Figura 3.25 motra o algoritmo implementado na thread de administração de testes.

Cabe à thread de transmissão de dados cuidar para que a fila circular não fique sem espaço, resultando em perda de informação. No início da thread sua prioridade é configurada para um nível mais baixo que o resto das threads do programa, esse procedimento é realizado para garantir que a transmissão de dados em condições normais não vai interromper as demais threads. O próximo procedimento é verificar se a lista circular está vazia, caso esteja, a prioridade é passada para outra thread.

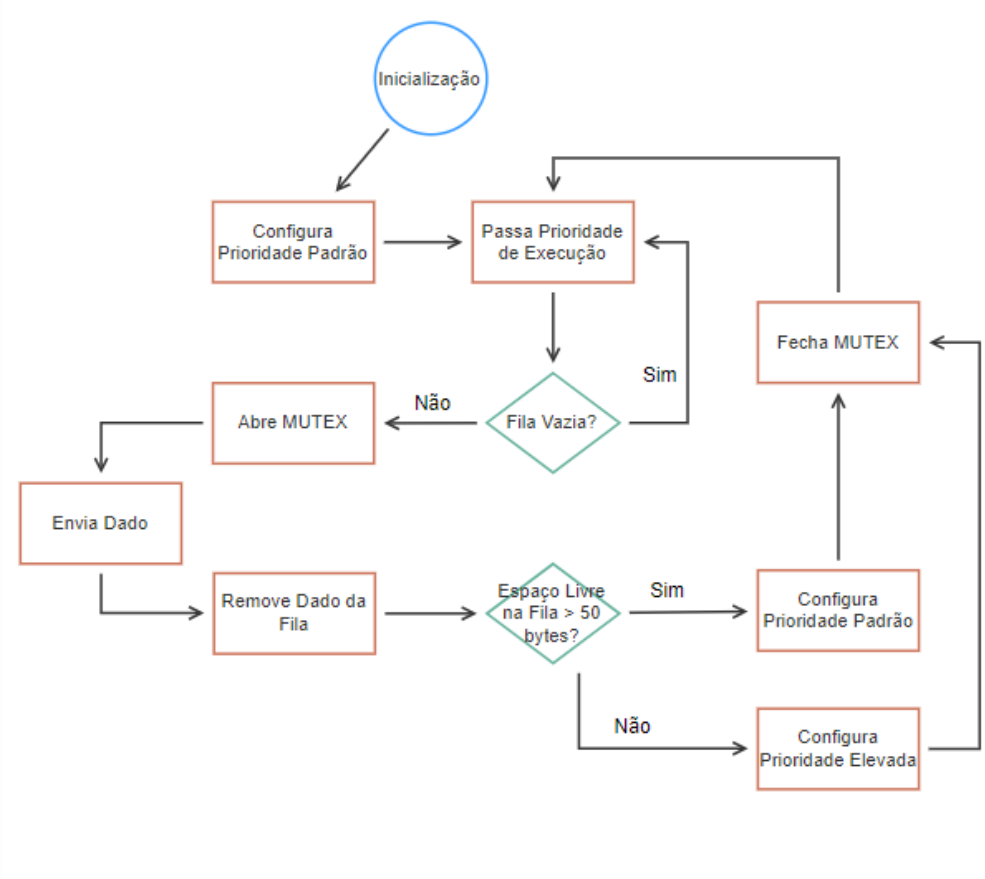

Figura 3.26 – Fluxograma da thread de transmissão de dados

Por outro lado, caso hajam caracteres a serem enviados é aberta uma rotina que se inicia ativando um mutex já que a lista circular é um recurso compartilhado entre a thread de teste e a thread de transmissão de dados. Em seguida o procedimento de transmissão de caracteres e iteração dos índices da fila é realizado. Por fim, como medida de segurança e garantia de envio dos dados, é verificado o que resta de espaço livre na fila circular. Caso haja menos de 50 bytes livres a prioridade da thread é alterada para o mesmo patamar das outras threads. Nessa situação não ocorre mais interrupção da transmissão, o que aumenta a taxa de envio e reduz o risco de perda de dados. No momento em que o espaço livre da fila ultrapassa 50 bytes a prioridade da thread é novamente configurada para seu valor inicial. Para fechar a rotina o mutex é desativado indicando que o procedimento de alteração da fila circular finalizou. O fluxograma de funcionamento da thread de transmissão de dados por ser visto na Figura 3.26.

3.3 Funções Implementadas

A ferramenta *STM32CubeMX* utilizada para configuração do microcontrolador gera blocos de código próprios para a configuração de cada periférico requisitado. No apêndice, na seção A.4, é possível verificar as funções geradas para esse fim. Além dessa funcionalidade, mais funções precisaram ser desenvolvidas para atender aos requisitos de implementação da aplicação. É necessário ler os sensores da prótese, comandar o driver de corrente e as portas GPIO, possibilitar a contagem de tempo para verificação do intervalo entre leituras e garantir o envio de dados por meio da comunicação USB com o computador. A seguir cada solução em código implementada para atingir os objetivos descritos será mostrada e explicada.

3.3.1 Registro de Intervalos de Aferição

Um dos problemas encontrados durante o desenvolvimento da aplicação foi a necessidade de utilizar uma base de tempo com o intuito de posicionar as medições realizadas pelos sensores da prótese em um instante de tempo específico. A solução encontrada foi a configuração de um temporizador exclusivamente para a contagem de tempo.

Contudo, isso não é suficiente para resolver a problemática, uma vez que a contagem de pulsos do temporizador é limitada pelo tamanho máximo de inteiro capaz de ser computado pelo microcontrolador. Para o microcontrolador *STM32F103C8T6* o maior número de pulsos que pode ser registrado no contador de seus temporizadores é 65535 e para o contexto da passada humana o intervalo de $1ms$ é uma discretização temporal suficientemente para aproximar o sistema para uma descrição contínua.

Dado esses fatores a frequência de $1kHz$ foi escolhida como frequência do temporizador em questão, o que caracteriza um pulso a cada $1ms$. Isso implica que o contador do temporizador será zerado a cada 65,535 segundos, uma vez que, seu valor limite será atingido nesse intervalo de tempo. Por sua vez, procedimentos de teste que durem mais de 1 minuto passarão a registrar leituras temporais inconsistentes devido à reinicialização do contador.

Para resolver esse problema uma nova estratégia foi utilizada. Ao invés de registrar o tempo absoluto com relação ao instante de inicialização do temporizador, o contador do temporizador é zerado a cada nova aferição dos sensores e cada vez que os sensores são aferidos é registrado então o tempo transcorrido desde a última medição. Desse modo, o contador só atinge seu limite de contagem se a leitura do sensor demorar mais de 1 minuto para ser realizada novamente, o que é incoerente com o propósito da prótese, garantindo que não haverá inconsistência no registro temporal.

```
uint16_t timer_set(uint8_t cmd){
        uint16_t ticks;
        if(cmd == 0){
```

```
                HAL_TIM_Base_Start(&htim4);
                return 0;
        }
        else if(cmd == 1){
                return __HAL_TIM_GET_COUNTER(&htim4);
        }
        else if(cmd == 2){
                ticks = __HAL_TIM_GET_COUNTER(&htim4);
                htim4.Instance->CNT = 0;
                return ticks;
        }
        else if(cmd == 3){
                HAL_TIM_Base_Stop(&htim4);
                return 0;
        }
        return 1;
}
```

Código Fonte 3.1 – Função de manipulação do temporizador

O Código Fonte 3.1 foi construído para permitir a manipulação do temporizador em questão. A função exige um comando como parâmetro de entrada e por meio dele escolhe-se a operação desejada, seja iniciar o temporizador, interromper o temporizador, retornar o valor do contador ou retornar o valor do contador e depois zerá-lo.

3.3.2 Atrasos de Microsegundos

Durante o desenvolvimento da aplicação observou-se a necessidade de introdução de atrasos no algoritmo, principalmente durante a comunicação com o computador e com o encoder por meio de protocolo SPI. É utilizada a biblioteca HAL desenvolvida pela *STMicroeletronis* para construção do programa de testes e essa biblioteca possui funções específicas para transmissão e recepção de dados via USB e SPI.

Entretanto, foi observada a falha dessas funções quando nenhum atraso é aplicado ao microcontrolador depois de chamá-las. Além desse problema, a biblioteca HAL não implementa funções que aplicam tempo de espera menores que $1ms$, sendo esse intervalo de tempo muito longo para a taxa de transmissão dos protocolos de comunicação utilizados, o que prejudica consideravelmente a velocidade de comunicação do microcontrolador com outros dispositivos.

```
void us_delay(uint16_t tempo){
        uint16_t temp_init, temp_tt;
```

```
        HAL_TIM_Base_Start(&htim3);
        temp_init = __HAL_TIM_GET_COUNTER(&htim3);
        do{
                temp_tt = __HAL_TIM_GET_COUNTER(&htim3) - temp_init;
        }while(temp_tt < tempo);
        HAL_TIM_Base_Stop(&htim3);
}
```

Código Fonte 3.2 – Função de delay em microsegundos

Devido a essa problemática o algoritmo descrito pelo Código Fonte 3.2 foi implementado, a fim de introduzir atrasos de microsegundos em instantes específicos da comunicação com o computador e o encoder. A função criada foi capaz de resolver a falha de comunicação percebida e tornar a troca de dados mais eficiente. Seu funcionamento baseia-se em uma rotina que verifica o número de pulsos do temporizador transcorridos desde a chamada da função.

Quando o número de pulsos ultrapassa o parâmetro passado para a função, a rotina é encerrada e o temporizador é desativado. Para que a função descrita no código 3.2 funcione foi necessário configurar mais um temporizador por meio da ferramenta $STM32CubeMX$ e realizar a calibração de sua frequência para $1MHz$, o que caracteriza um pulso do temporizador a cada $1\mu s$.

3.3.3 Recebimento e Transmissão de Dados Via SPI

O protocolo SPI necessita de quatro canais de comunicação entre os dispositivos mestre e escravo. Um canal transmite dados do mestre para o escravo, outro transmite do escravo para o mestre, o terceiro canal é o clock e o último é o seletor de escravos. Na Tabela 3.5 é possível ver a definição de cada canal do protocolo. No contexto da prótese RLEG o microcontrolador é o dispositivo mestre e o encoder angular é o dispositivo escravo.

Tabela 3.5 – Canais do Protocolo SPI

Canais	Descrição	Funcionalidade
MISO	Master in slave out	Transmite dados do dispositivo mestre para o escravo
MOSI	Master out slave in	Transmite dados do escravo para o mestre
CLK	Clock	Define a ocorrência de um bit nos canais de transmissão
SS	Slave Select	Ativa o escravo para envio ou recepção de dados

Para o encoder angular os canais de comunicação são ativados configurando o canal *SS* para o nível lógico baixo e é definido no datasheet do componente a necessidade de aplicação de 20μs de espera entre comandos enviados ao encoder. Por isso nas funções 3.3 e 3.4 são aplicados os atrasos de 20μs utilizando-se a função descrita no código 3.2.

O datasheet do componente ainda explica a necessidade de liberação do canal *SS* ao fim de todo dado transmitido ou recebido, o que implica em levar o canal para o nível lógico alto. Esse procedimento pode ser observado nos códigos 3.3 e 3.4, a partir da função *HAL_GPIO_WritePin*() que configura o nível lógico de pinos GPIO e nesse caso do canal *SS* do encoder.

```
uint8_t SPI_Receive(){
        uint8_t rp;

        us_delay(20);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_SPI_Receive(&hspi2, (uint8_t*)&rp, 1, 100);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET);
        us_delay(20);
        return rp;

}
```

Código Fonte 3.3 – Função de Recepção de dados SPI do encoder

Todo comando enviado ao encoder é seguido por uma resposta que indica o recebimento da mensagem por parte do escravo. Existem três comandos aplicáveis ao dispositivo, o primeiro é o comando de não operação que gera apenas a confirmação de recebimento pelo dispositivo, o segundo é o comando de leitura que retorna a posição angular medida pelo encoder e o terceiro é o comando de configuração da referência zero.

A leitura de dados e a configuração da referência do sensor não são necessariamente realizadas no instante do recebimento desses comandos e por isso o encoder pode enviar ao microcontrolador várias confirmações de recebimento do comando até que ele seja realizado. Por isso, é necessário sondar os dados enviados ao microcontrolador até que os resultados do comando pedido sejam enviados. Por esse motivo a função 3.3 é implementada. Por outro lado, visando receber imediatamente a confirmação de recebimento dos comandos por parte do encoder a função 3.4 é implementada.

```
uint8_t SPI_Send_Receive(uint8_t cmd){
        uint8_t rp;
```

```
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_SPI_Transmit(&hspi2, (uint8_t*)&cmd, 1, 100);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET);
        us_delay(20);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_SPI_Receive(&hspi2, (uint8_t*)&rp, 1, 100);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET);
        return rp;
}
```

Código Fonte 3.4 – Função de Transmissão e Recepção de dados SPI do encoder

3.3.4 Pausa e Retomada

A thread de administração de testes possui comandos para realizar a pausa e a retomada do teste em execução. A função 3.5 é responsável em implementar esses comandos. Nessa função, é verificado se o buffer de dados da comunicação USB possui o comando de pause. Caso haja, uma rotina é iniciada atrás do comando de retomada ou de saída. Quando algum desses comandos é identificado a rotina é concluída e a função finalizada. A função manda mensagens ao computador indicando a pausa e a retomada do sistema quando esses comandos são identificados. Essas mensagens são escritas no terminal em uso.

```
void pause_resume(){
        if(n_buf[0] == 'p'){
                const char pause[15] = "Pausado...\n\r";
                const char retorno[15] = "Retomada...\n\r";
                memset(n_buf, '\0', sizeof(n_buf));
                SendUSB((uint8_t*)pause, strlen(pause));
                while(n_buf[0] != 'r' && n_buf[0] != 'q'){
                        memset(n_buf, '\0', sizeof(n_buf));
                        HAL_Delay(100);
                }
                if(n_buf[0] == 'r'){
                        SendUSB((uint8_t*)retorno, strlen(retorno));
                        memset(n_buf, '\0', sizeof(n_buf));
                        osDelay(1000);
                }
        }
}
```

Código Fonte 3.5 – Função de Pausa e Retomada

3.3.5 Conversão de Ponto Flutuante em Vetor de Caracteres

Diferente dos dados lidos pelos outros sensores o Módulo inercial precisa ter seus dados transformados para ponto flutuante, uma vez que, alterações em casas decimais das leituras já são relevantes para descrever a mecânica da prótese. Devido a arquitetura de $32bits$ do microcontrolador utilizado a função da biblioteca padrão de C que realiza a conversão de valores em ponto flutuante para strings não é compatível e não pode ser utilizada. Por esse motivo foi implementada a função 3.6, a fim de realizar essa conversão.

O funcionamento do código se baseia em transformar o ponto flutuante em um dado do tipo inteiro, nesse processo as casas decimais são perdidas. Em seguida o dado é transformado novamente em ponto flutuante e subtraído do dado original restando apenas as casas decimais, então o resultado é multiplicado n vezes por dez, sendo n o número de casas decimais desejadas na conversão. Desse modo tem-se um valor inteiro que representa a parte inteira do dado e outro valor inteiro que representa a parte decimal do dado e ambos podem ser mais facilmente transformados para um vetor de caracteres.

```
void f_conv_trans(char c_res[20], int nc, float num){
        char mas[20];
        float num_aux, base10;
        uint16_t resul, rest;
        num_aux = num;
        resul = (uint16_t)num_aux;
        num_aux = num_aux - (float)resul;
        base10 = exp_int(10, nc);
        num_aux = num_aux*base10;
        rest = (uint16_t)num_aux;
        sprintf(mas, "%d.%d", resul, rest);
        strcpy(c_res, mas);
}
```

Código Fonte 3.6 – Função conversão ponto flutuante para String

3.3.6 Transmissão de Dados Via USB

A thread de transmissão de dados remove caracteres da lista circular e os envia para o computador. Para possibilitar esse funcionamento foram implementadas as funções 3.7 e 3.8. A primeira, permite a transmissão diretamente ao computador sem a necessidade de envio de dados para a lista. Essa funcionalidade é importante pois em algumas situações caracteres são enviados ao terminal do computador enquanto o processo de teste não está em execução. Nessa situação não há a necessidade de utilização da fila circular. A segunda

função, por outro lado, é o meio de inserção de dados na lista. Quando os processos de teste estão em execução é através dessa função que a lista é preenchida.

```
void SendUSB(uint8_t* txt, uint16_t tam){
        uint8_t usb_ok = USBD_OK;
        uint8_t count = 0;
        do{
                usb_ok = CDC_Transmit_FS(txt, tam);
                count++;
                us_delay(100);
        }while(usb_ok != USBD_OK && count < 10);
}
```

Código Fonte 3.7 – Função de transmissão direta via USB

Pode-se perceber na função 3.7 a chamada da função 3.2. Foi observada a necessidade da aplicação desse tempo de espera depois da tentativa de envio de dados por meio do canal USB. Sem esse tempo de espera a transmissão não era realizada. Além disso, nem sempre o envio de dados é bem sucedido, o que exige a aplicação de uma rotina para repetir a tentativa de transmissão uma quantidade finita de vezes. A função 3.8 necessita da utilização de um mutex, uma vez que a fila circular é um recurso compartilhado com a thread de transmissão de dados. É enviado como parâmetro dessa função um vetor de caracteres, a função então acrescenta cada caracter a um elemento da fila circular e incrementa o índice de fim da fila a cada inserção.

```
void Send_to_USBq(uint8_t* txt, uint16_t tam){
        int i = 0;

        xSemaphoreTake(listMutex, portMAX_DELAY);
        for(i = 0; i < tam; i++){
                USBq[USBq_end_index] = txt[i];
                if(USBq_end_index < sizeof(USBq) - 1){
                        USBq_end_index++;
                }
                else{
                        USBq_end_index = 0;
                }
        }
        xSemaphoreGive(listMutex);
}
```

Código Fonte 3.8 – Função de inserção de dados na fila circular

3.3.7 Procedimentos de Teste

Para cada procedimento de teste foi implementada uma função. Cada uma dessas funções aplica uma rotina que ativa cada um dos periféricos de interesse repetidamente, até que o comando de saída seja enviado via terminal. O primeiro procedimento implementado foi o teste das portas GPIO. Esse procedimento intercala o nível lógico de cada uma das portas. Cada porta é conectada a um led, desse modo o que se observa na placa do projeto são os leds piscando em sequência de modo periódico. Quando a rotina é finalizada, todas as portas são configuradas para o nível lógico baixo.

O segundo procedimento implementado foi o teste do driver de corrente. O pulso PWM gerado pelo microcontrolador pode ter seu ciclo de trabalho configurado por meio do código. Portanto, o teste do driver se baseia em gerar uma rotina que a cada repetição incrementa o ciclo de trabalho do PWM. Quando o ciclo de trabalho de 100% é atingido, a rotina é reiniciada, o que zera novamente o ciclo de trabalho. Caso os terminais do driver sejam conectados a um led o efeito visual será o gradativo aumento da intensidade luminosa seguido por um repentino desligamento do led.

O terceiro procedimento realizado é a aferição do sensor de corrente. Tal sensor é um dispositivo analógico, o que exige a configuração de um conversor analógico digital no microcontrolador. O $STM32F103C8T6$ possui conversores de 12 bits, logo o valor recebido pelo microcontrolador consiste em um inteiro entre 0 e 4096 que representa proporcionalmente o valor de tensão do pino analógico do sensor entre $0V$ e $3{,}3V$.

O sensor em questão amostra a corrente que percorre os terminais do driver, por isso é necessário repetir o procedimento de teste do driver, a fim de verificar a variação de corrente gerada por sua atuação. Desse modo, o procedimento de teste do sensor de corrente é igual ao teste do driver de corrente com o acréscimo da leitura analógica do sensor após cada incremento do ciclo de trabalho do sinal PWM. Cada vez que a leitura é realizada o resultado obtido é transcrito para um vetor de caracteres e enviado para fila circular por meio da função 3.8.

O próximo procedimento realizado foi o teste do módulo inercial por meio do protocolo de comunicação I2C. A comunicação com o módulo inercial é realizada por meio de leitura e escrita de registradores presentes no dispositivo. Antes de realizar leituras com o módulo inercial é necessário configurar o valor de alguns registradores. Sem essas configurações o módulo não funcionará, ou os valores lidos não serão corretamente interpretados. Inicialmente é necessário escrever no registrador de gerenciamento de energia e retirar o módulo do modo de repouso. Em seguida é necessário escrever nos registradores de configuração do acelerômetro e do giroscópio, a fim de escolher a resolução das leituras realizadas. Na Tabela 3.6 é possível observar o endereço dos registradores em questão e as configurações realizadas.

Tabela 3.6 – Configuração dos registradores do módulo inercial

Nome do Registrador	Endereço do Registrador	Configuração Enviada	Descrição da Configuração
Gerenciamento de Energia	0x6b	0x00	Tira o módulo do modo de repouso
Configuração do giroscópio	0x1b	0x08	Define faixa de leituras: 500°/s a -500°/s
Configuração do acelerômetro	0x1c	0x08	Define faixa de leituras: 4g a -4g

Depois de realizadas as configurações da tabela 3.6 uma rotina é iniciada para leitura periódica dos dados do módulo. As medições realizadas pelo acelerômetro e pelo giroscópio são armazenadas dentro de registradores. O módulo é capaz de realizar medições de velocidade angular e aceleração em todos os três eixos do espaço. Como cada um dos dados coletados possui 2 bytes de informação, são necessários 12 registradores apenas para armazenamento das leituras de interesse.

Existe ainda um sensor de temperatura no módulo que também gera dados com o mesmo tamanho. Desse modo, existem 14 registradores de onde pode-se ler dados medidos pelo módulo inercial. A rotina criada lê todos os 14 registradores e converte os inteiros retornados em ponto flutuante. Por meio da função 3.6 o ponto flutuante é convertido em vetor de caracteres. Em seguida esse vetor é introduzido na fila circular, sendo esse processo repetido para cada dois registradores. A Tabela 3.7 descreve os registradores lidos.

Tabela 3.7 – Registradores de leitura - Módulo Inercial

Registradores	Informação
0x3b, 0x3c	Aceleração no eixo X
0x3d, 0x3e	Aceleração no eixo Y
0x3f, 0x40	Aceleração no eixo Z
0x41, 0x42	Temperatura
0x43, 0x44	Velocidade angular eixo X
0x45, 0x46	Velocidade angular eixo Y
0x47, 0x48	Velocidade angular eixo Z

O último procedimento de teste implementado consiste na leitura da angulação do joelho protético por meio do encoder angular. O funcionamento do encoder se baseia no envio de comandos por meio do canal MOSI descrito na Tabela 3.5. A definição dos comandos do encoder pode ser visualizada na Tabela 3.8. O teste desse dispositivo se inicia verificando a conexão do mesmo com o microcontrolador. Um comando de não operação é enviado ao encoder, por meio da função 3.4, e a resposta descrita na Tabela 3.8 é esperada.

Caso a resposta correta não seja recebida o comando é enviado novamente, até um total de 20 tentativas. Depois da vigésima tentativa, se a resposta esperada não foi recebida,

o teste é finalizado e uma mensagem de erro é enviada ao terminal. Quando alguma das tentativas recebe a confirmação de recebimento do comando uma nova instrução é enviada ao encoder, dessa vez o comando de configuração do ponto zero. Quando o encoder finalizar a configuração, enviará a resposta descrita na Tabela 3.8. Por isso, o microcontrolador sonda continuamente o canal MISO por meio da função 3.3.

Tabela 3.8 – Descrição dos comandos do encoder

Comando	Código	Resposta	Funcionalidade
Não operação	0x00	0xa5	———–
Leitura	0x10	Angulação atual do encoder	Transmitir ao mestre a informação de angulação
Configuração do ponto zero	0x70	0x80	Configura a posição atual do encoder como referência zero

Uma vez recebida a resposta para o comando de configuração do ponto zero, uma nova rotina é criada. Nessa rotina o comando de leitura é enviado e enquanto a informação de angulação não está disponível o encoder envia ao microcontrolador a resposta $0xa5$. Antes de transmitir o dado requisitado, o encoder envia o próprio comando de leitura, para que se identifique que os próximos dados serão referentes a esse comando.

A informação de angulação é passada para o microcontrolador na forma de inteiros de 12 bits, logo é necessária a transmissão de dois bytes de dados para o microcontrolador. Isso exige que a função 3.3 seja chamada duas vezes depois da retransmissão do comando de leitura. Os bytes recebidos são primeiro convertidos para um inteiro de 16 bits e depois para um vetor de caracteres.

Em seguida são adicionados na fila circular para transmissão via USB para o computador. Verificou-se, que com certa frequência os dados de angulação recebidos não eram condizentes. Percebeu-se, que ao invés de transmitir a informação de angulação, em alguns instantes era retransmitido pelo encoder, ou a resposta $0xa5$ ou o comando de leitura. Para evitar que esse problema interfira na aferição de angulação da prótese é verificado se os dados recebidos equivalem a algum comando ou resposta do encoder.

Caso o problema seja identificado, as leituras falhas são descartadas e uma nova leitura é realizada. O código 3.9 apresenta o trecho do procedimento de teste do encoder responsável em mandar o comando de leitura e converter o dado recebido em inteiro de 32 bits e vetor de caracteres.

```
resp = SPI_Send_Receive(command[1]);
if(resp == ack){
    do{
        us_delay(20);
        if(i > 10){
           resp = SPI_Send_Receive(command[1]);
```

```
            i = 0;
        }
        resp = SPI_Receive();
        i++;
    }while(resp != command[1] && n_buf[0] != 'q');}
us_delay(20);
dado[0] = SPI_Receive(command[0]);
us_delay(20);
dado[1] = SPI_Receive(command[0]);
dt = timer_set(2);
deg = 256*dado[0] + dado[1];
deg = 360 - deg*360/4096;
if(dado[0] != ack && dado[1] != command[1] &&
dado[1] != ack && dado[1] != command[1]){
    utoa(deg, msg, dec);
    Send_to_USBq((uint8_t*)msg, strlen(msg));
    Send_to_USBq((uint8_t*)space, strlen(space));
    utoa(dt, msg, dec);
    Send_to_USBq((uint8_t*)msg, strlen(msg));
    Send_to_USBq((uint8_t*)enter, strlen(enter));}
```

Código Fonte 3.9 – Trecho do procedimento de teste do encoder

4 Resultados

O sistema embarcado da prótese RLEG foi testado, comprovando o funcionamento tanto do programa de teste construído nesse trabalho, quanto dos componentes eletrônicos que compõem o projeto. O terminal *Putty* foi utilizado e um arquivo de texto contendo os resultados de cada um dos procedimentos de teste foi gerado. Por meio deles, é possível analisar a aplicabilidade dos dispositivos da prótese e desenhar os gráficos mostrados a seguir. Qualquer pessoa pode repetir os procedimentos de teste realizados, bastando conectar o sistema embarcado da prótese a um computador e utilizar um terminal para abrir a comunicação USB com o dispositivo.

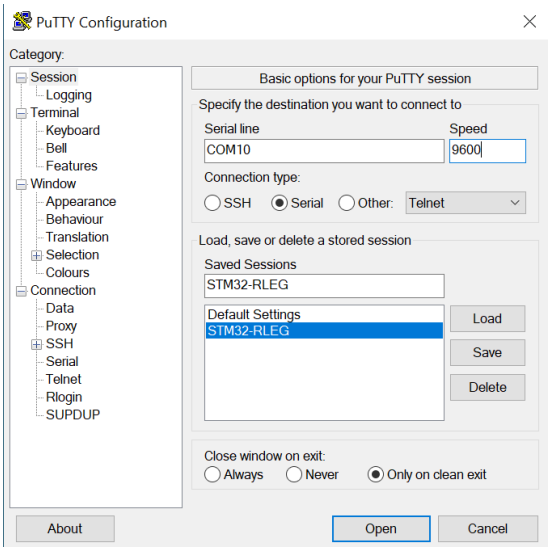

Figura 4.27 – Configuração de conexão

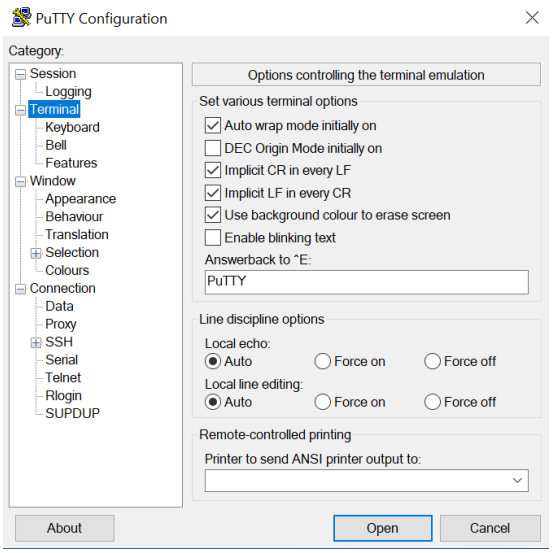

Figura 4.28 – Configuação de terminal

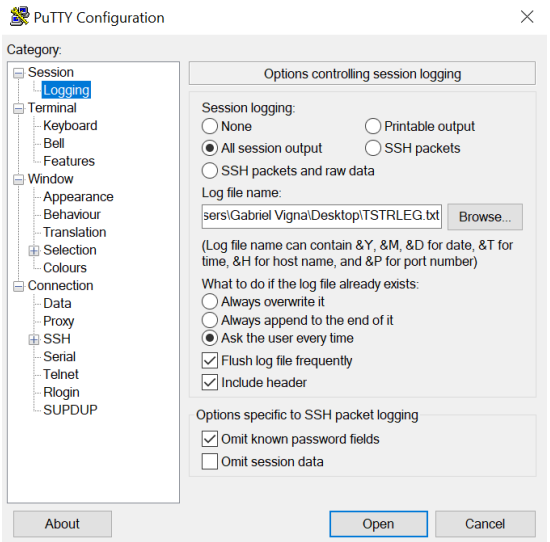

Figura 4.29 – Configuração de registro de dados

A primeira atividade necessária para utilização do software é realizar a conexão via cabo USB entre o computador e o microcontrolador $STM32F103C8T6$. Entre as portas seriais em uso no computador, uma será identificada como $STMicroeletronicsVirtualCOMPort$, essa é a porta serial disponível para comunicação com a prótese RLEG. Com essa informação é possível configurar o terminal *Putty* para ler e enviar dados ao microcontrolador. As imagens 4.27, 4.28 e 4.29 mostram as configurações necessárias para o ideal funcionamento do sistema. É necessário selecionar a conexão serial, inserir o endereço da porta serial e escolher a velocidade da conexão. Outra configuração importante é feita na aba *Terminal* onde os caracteres de quebra de linha e retorno de carro são ativados. Desse modo, eles são incluídos de modo automático no terminal, o que permite a legibilidade dos dados recebidos.

Outra configuração importante é realizada na aba *logging*, onde são estabelecidos os parâmetros para registro dos dados recebidos via terminal. É necessário ativar o registro de dados e escolher qual tipo de informação deve ser registrada. Os dados coletados podem ser armazenados dentro de um arquivo de texto, por isso, é necessário escolher um nome para o arquivo e o diretório de criação. Finalizadas as configurações, basta acionar o botão *Open* posicionado no canto inferior direito. Desse modo, o terminal será aberto e a comunicação USB estabelecida. Quando iniciado, o sistema embarcado da prótese é automaticamente posto em modo padrão de operação. Para realização dos testes é necessário ativar o modo de teste teclando-se o comando *Menu* descrito na Tabela 3.3. Depois que o menu é aberto basta escolher a opção *Modo Teste* como visto na Figura 3.23.

4.1 Teste das Portas GPIO

O procedimento descrito anteriormente abrirá o menu de testes mostrado na Figura 3.24. O primeiro procedimento de teste é o das portas GPIO, seu acionamento ativará os leds da placa eletrônica do projeto, fazendo-os piscar em sequência. O teste foi realizado e verificou-se o correto funcionamento das portas como visto na figura 4.30.

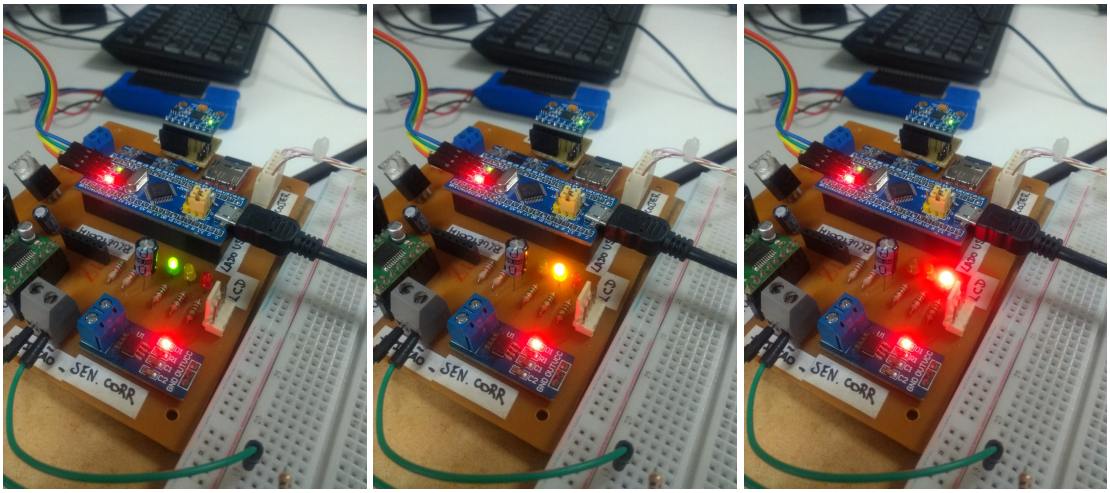

Figura 4.30 – Teste das portas GPIO

4.2 Teste Driver de Corrente

Para finalizar qualquer procedimento de teste tecla-se o comando *Saida* descrito na Tabela 3.3, o que abrirá novamente no terminal o menu de testes. O segundo procedimento implementado é o teste do driver de corrente. Para percepção visual do funcionamento do driver, decidiu-se conectar a saída de alimentação do pistão magneto-relógico a um led por meio de uma matriz de contatos. O teste em questão realiza o aumento gradual do ciclo de trabalho do sinal PWM que aciona o driver de corrente, logo, espera-se visualizar o gradual aumento de brilho do led. O comportamento observado foi condizente com o esperado, na figura 4.31 é possível observar o teste realizado.

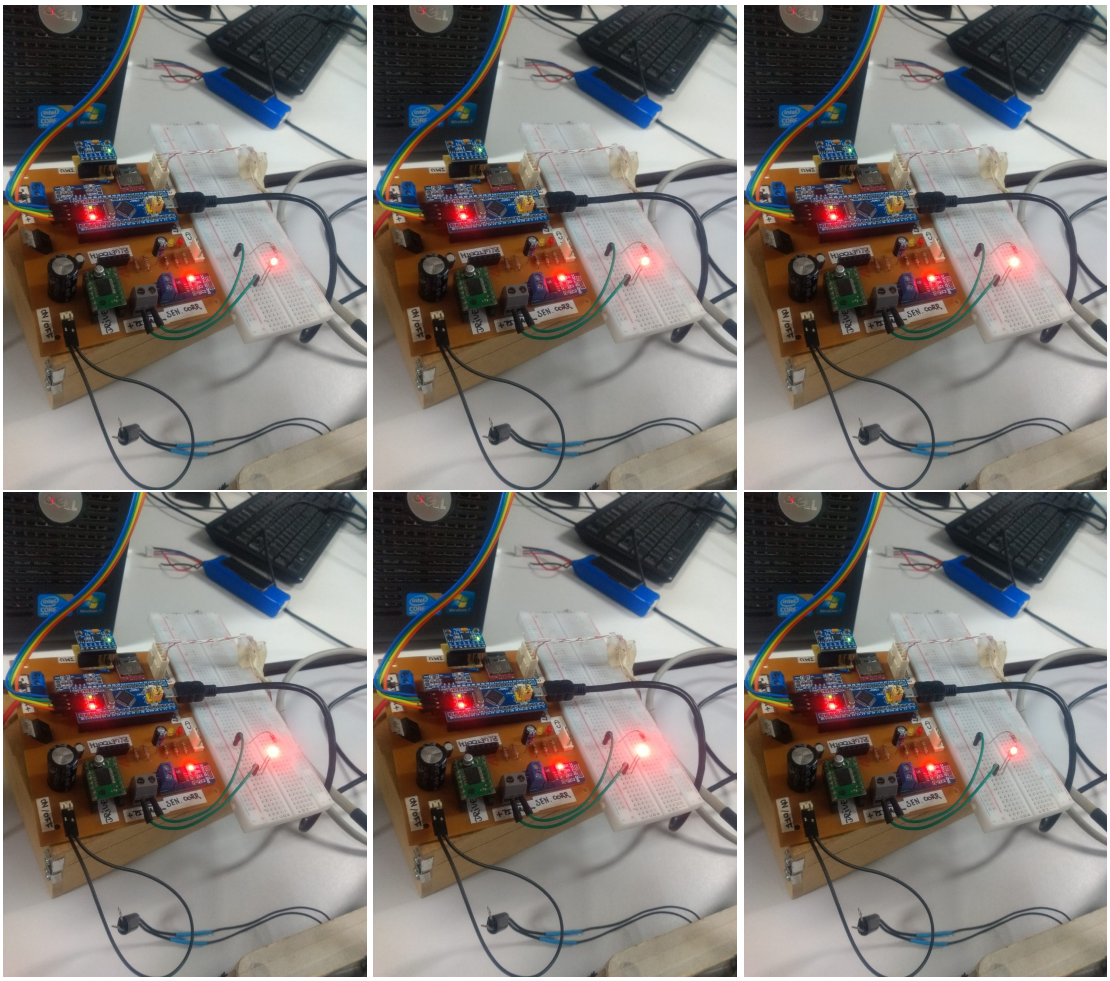

Figura 4.31 – Teste do driver de corrente

4.3 Teste Sensor de Corrente

O teste do sensor de corrente necessita da utilização do driver de corrente. Como visto na relação 2.1 a variação mínima de corrente que pode ser percebida pelo microcontrolador é de aproximadamente $19mA$. Por isso, esse teste não pode ser realizado com um led, já que a corrente máxima suportada por esse dispositivo é $20mA$. Para a realização desse teste a

placa do projeto foi conectada a uma fonte de bancada fornecendo tensão de $12V$. O pistão magneto-relógico foi conectado à placa para servir de carga ao driver, uma vez que sua corrente nominal é de $2A$. A Figura 4.32 evidencia o teste realizado. Com os dados coletados o gráfico da figura 4.33 foi montado.

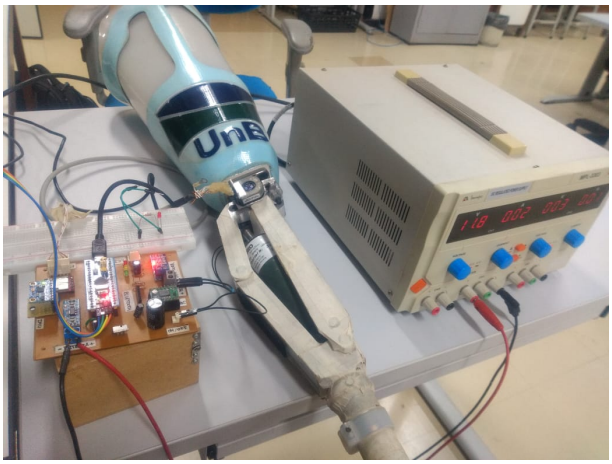

Figura 4.32 – Teste do sensor de corrente

Por meio do gráfico da Figura 4.33 é possível fazer algumas observações acerca do sensor de corrente. O gráfico em questão apresenta dois ciclos de aumento gradativo do sinal PWM entregue ao driver de corrente. Desse modo, a primeira observação a ser realizada é a presença de um offset na leitura do sensor. De início a potência entregue ao atuador é nula, contudo é registrado pelo sensor uma corrente de aproximadamente $4{,}5A$. Percebe-se com o passar do tempo que a corrente é reduzida até atingir aproximadamente $2{,}2A$. Essa redução demonstra que, para o referencial do sensor, a corrente possui sentido negativo. Ajustando-se o offset, percebe-se uma corrente de aproximadamente $2{,}3A$ atravessando o pistão magneto-relógico, durante o instante de maior aplicação de potência. Sabe-se, que o atuador possui resistência de aproximadamente 5Ω a temperatura ambiente. Desse modo, sob a tensão de $12V$, espera-se uma corrente próxima de $2{,}4A$ no pistão magneto-relógico, o que está de acordo com o observado no gráfico 4.33.

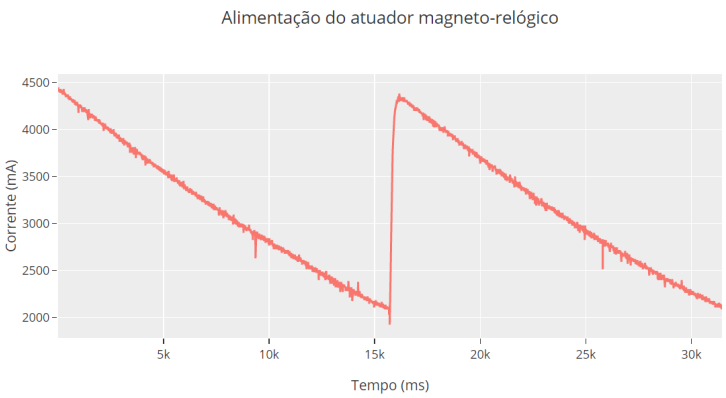

Figura 4.33 – Gráfico corrente x tempo - Alimentação atuador magneto-relógico

4.4 Teste Módulo Inercial

Para realização do teste do módulo inercial a placa eletrônica do projeto foi rotacionada no sentido anti-horário com relação ao eixo Z, depois no sentido anti-horário com relação ao eixo Y, em seguida no sentido anti-horário com relação ao eixo X e por fim no sentido horário com relação ao eixo Y. Com isso, os gráficos 4.34 e 4.35 foram gerados a partir dos dados de aceleração e velocidade angular recebidos no terminal.

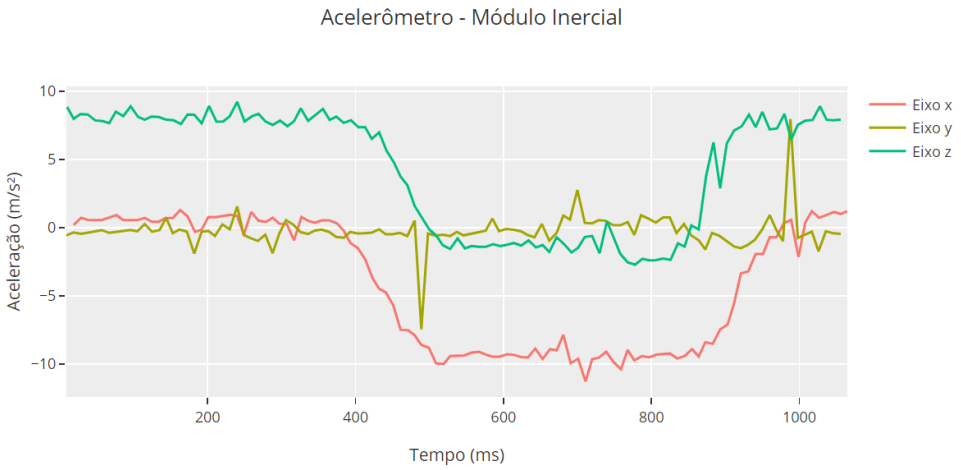

Figura 4.34 – Ação da força peso sobre o módulo inercial

Considerando-se o gráfico 4.34, percebe-se inicialmente a disposição da força peso paralelamente ao eixo Z e a mudança de ação dessa força conforme diferentes eixos do módulo inercial tornam-se paralelos à aceleração da gravidade. No gráfico 4.35 é possível observar a ocorrência de todas as rotações realizadas. Os giros são traduzidos em extremos no gráfico de velocidade angular. Os extremos positivos são rotações no sentido anti-horário e os extremos negativos são rotações no sentido horário. Assim percebe-se, que a movimentação realizada na placa pode ser descrita por meio dos dados coletados com o módulo inercial.

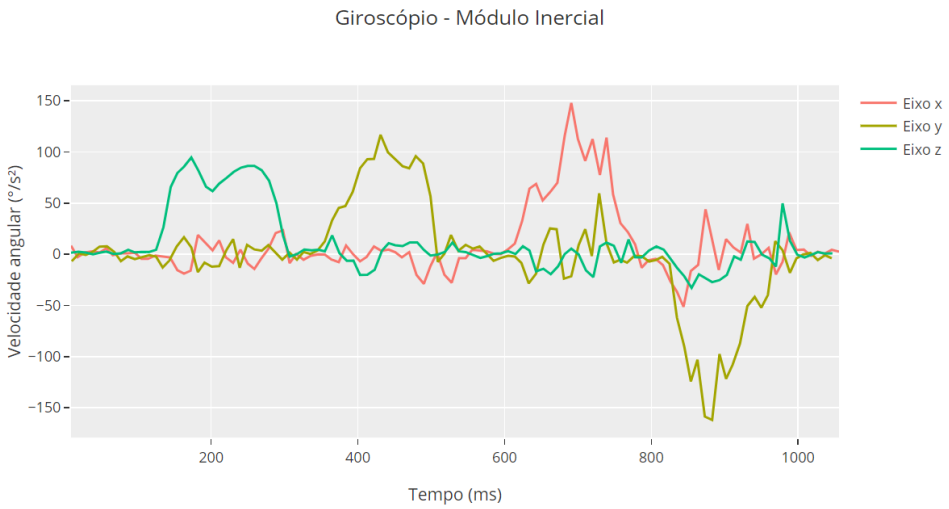

Figura 4.35 – Velocidade angular devido rotação da placa

4.5 Teste Encoder Angular

O teste do encoder angular foi iniciado com o joelho protético totalmente estendido. Essa posição foi configurada como posição de referência zero para a angulação do joelho. Enquanto os dados do encoder eram lidos, o joelho foi flexionado primeiro até 60°, depois estendido de volta para 30°, em seguida flexionado para 90° e por fim, totalmente estendido para 0°. Os resultados obtidos estão representados no gráfico 4.36. Pode-se perceber, que o comportamento do encoder está de acordo com o esperado uma vez que as angulações medidas condizem com o movimento realizado na prótese pelo autor.

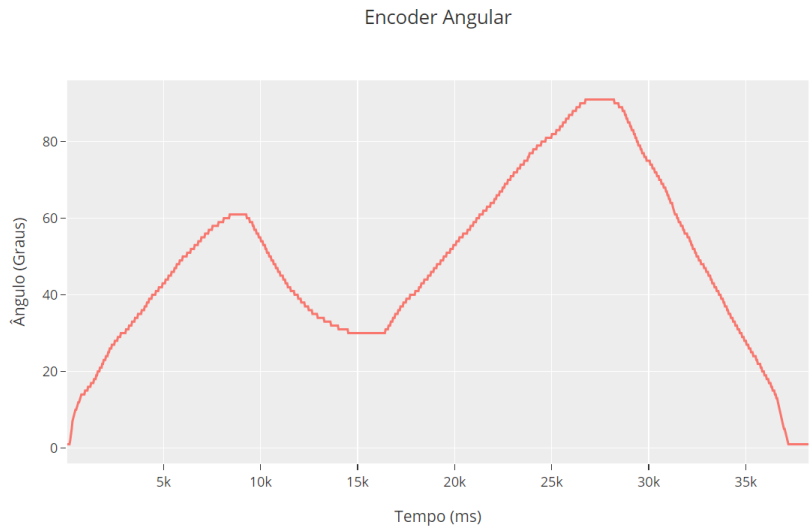

Figura 4.36 – Angulação medida com o encoder angular

4.6 Prioridade da Fila Circular

Foi realizada uma análise com o intuito de verificar a eficácia da solução implementada para impedir a perda de dados na fila circular. Como explicitado na Figura 3.26 a prioridade da thread de transmissão de dados é aumentada caso a fila circular fique muito cheia. O teste realizado buscou aferir a influência que essa mudança de prioridade gera no tempo de amostragem e no espaço livre da fila circular. O tempo para transmissão de dados via USB pode ser alterado na função 3.7 por meio do tempo de espera gerado pela função 3.2.

Desse modo, é possível reduzir a transmissão de dados ao computador aumentando a espera para o envio, isso pode gerar superlotação da fila circular, pois mais dados serão introduzidos do que retirados. Para o teste da fila circular escolheu-se o procedimento de teste do sensor de corrente, uma vez que esse procedimento não envolve protocolos de comunicação, o que possibilita maior velocidade na aferição, o tempo de espera introduzido na função 3.7 foi de 400μs. A evolução do intervalo de aferição e do preenchimento da fila circular pode ser vista no gráfico 4.37.

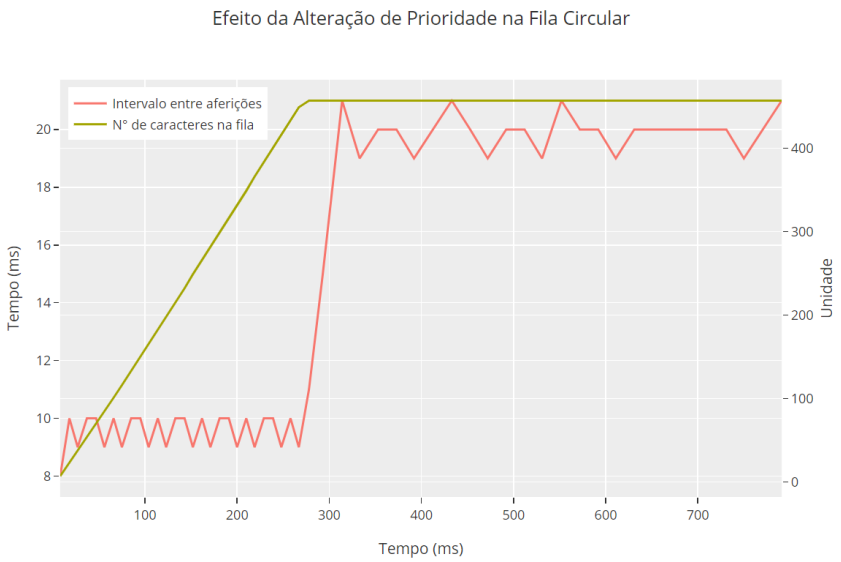

Figura 4.37 – Teste de perda de dados na fila circular

Nota-se pelo gráfico 4.37 que o atraso de 400μs foi suficiente para causar o crescimento descontrolado da quantidade de caracteres na fila circular. É perceptível a alteração de prioridade na thread de transmissão de dados por volta do tempo 270mA, pois é nesse instante que ocorre a interrupção do aumento de caracteres na fila e o crescimento abrupto do período de amostragem. Como mostrado no gráfico, essa estratégia foi capaz de estabilizar a quantidade de caracteres na fila, atingindo o equilíbrio em 457 unidades.

Além disso, fica comprovado que essa estabilização ocorre devido à redução da frequência de amostragem. Uma vez que a taxa de transmissão via USB foi limitada, o meio de estabilizar a quantidade de dados na fila é restringir a taxa de entrada de informação, o que é atingido com a redução da frequência de amostragem. Nessa situação o tempo entre aferições é aumentado devido a equidade de prioridade entre as threads de transmissão e de teste, o que impede a interrupção de uma pela outra. Desse modo, a thread de teste é obrigada a esperar o término de execução da thread de transmissão.

5 Conclusões

O presente trabalho retomou a atividade de desenvolvimento da prótese RLEG após um período de três anos sem avanços. Foi realizado um levantamento histórico do projeto, desde seu início em 2005 até a última atividade realizada em 2019. Além disso, foi implementada uma solução em software em tempo real capaz de testar o sistema embarcado do projeto e gerar relatórios com o envio de dados de todos os sensores do sistema ao computador.

Essa solução facilita consideravelmente a realização de testes da prótese por qualquer pessoa que futuramente trabalhe com o projeto e contribui para a atividade de calibração dos sensores e demais dispositivos do sistema. Foram realizados testes da eletrônica do projeto por meio do programa implementado e verificou-se que o funcionamento dos componentes está de acordo com o esperado.

As pesquisas e códigos realizados, assim como, as ferramentas utilizadas durante esse trabalho estão instaladas e arquivadas no computador do LARA (Laboratório de Robótica e Automação) exclusivo para uso desse projeto, o que também facilita a continuação da atividade de desenvolvimento da prótese.

Como trabalhos futuros destaca-se inicialmente a necessidade de calibração e tratamento dos dados lidos com os sensores do projeto. Percebe-se pelos gráficos 4.32, 4.34 e 4.35 que há uma alta frequência de oscilação do sinal do módulo inercial, o que demonstra presença de ruído nessas leituras. Com relação ao sensor de corrente, ocorre também oscilação do sinal, porém com menor amplitude. No caso desse sensor, é necessária a compensação de seu valor de offset e a verificação de uma proporção mais precisa entre corrente medida e tensão da saída analógica.

Outro trabalho que ainda precisa ser realizado é o desenvolvimento e teste de um algoritmo para predição de movimento do joelho utilizando-se os dados de aceleração, velocidade angular e angulação da junta. Por fim, sendo possível fazer a previsão de intenção de movimento do joelho, uma estratégia de controle precisa ser implementada para a conquista do objetivo final da prótese: Aproximar a movimentação do joelho protético à cinemática de um membro saudável.

Referências

ADAMCZYK, P. G. Chapter 9 - Semi-active prostheses for low-power gait adaptation. In: DALLALI, H.; DEMIRCAN, E.; RASTGAAR, M. (Ed.). **Powered Prostheses**. Academic Press, 2020. P. 201–259. ISBN 978-0-12-817450-0. DOI: https://doi.org/10.1016/B978-0-12-817450-0.00009-2. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780128174500000092>. Citado na p. 23.

ASIF, M.; TIWANA, M. I.; KHAN, U. S.; QURESHI, W. S.; IQBAL, J.; RASHID, N.; NASEER, N. Advancements, Trends and Future Prospects of Lower Limb Prosthesis. **IEEE Access**, Institute of Electrical e Electronics Engineers Inc., v. 9, p. 85956–85977, 2021. ISSN 21693536. DOI: 10.1109/ACCESS.2021.3086807. Citado na p. 22.

BECKMANN, E. D.; SANTOS, G. F. dos. **DESENVOLVIMENTO DE CONTROLADORES DE JUNTAS PARA PRÓTESE ROBÓTICA DE PERNA**. 2006. Trabalho de conclusão de curso de graduação em Eng. Mecatrônica – Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado na p. 26.

BENTO, J. L. **VERIFICAÇÃO E REMODELAGEM DE UM SISTEMA DE PRÓTESE ROBÓTICA PARA AMPUTADOS DE MEMBRO INFERIOR**. 2019. Trabalho de conclusão de curso de graduação em Eng. Mecatrônica – Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado na p. 28.

BRASIL, D. A. **Controle de orientação do pé de uma prótese robótica para amputados acima do joelho**. 2008. Trabalho de conclusão de curso de graduação em Eng. Mecatrônica – Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado na p. 27.

CAMPOS, H. O.; DRUMMOND, L. R.; PAULA, E. Q. de. **ANÁLISE DO GASTO ENERGÉTICO EM INDIVÍDUOS COM AMPUTAÇÃO DURANTE ATIVIDADES FÍSICAS: REVISÃO SISTEMÁTICA**. v. 23. 2022. P. 05–20. Citado na p. 14.

CARVALHO, J. A. Amputações de membros inferiores: em busca da plena reabilitação. In: AMPUTACOES de membros inferiores: em busca da plena reabilitacao. 2003. P. xix–365. Citado na p. 14.

CASCÃO JR, C. A.; FERREIRA, R. U.; BECKMANN, E. D.; BORGES, G. A.; ISHIHARA, J. Y.; ROCHA, A. F. da. Estudo e desenvolvimento de uma prótese ativa de perna comandada por sinais eletromiográficos. **VII Simpósio Brasileiro de Automação Inteligente**, 2005. Citado na p. 25.

DELIS, A. L.; ROCHA, A. F. D.; CARVALHO, J. L. A.; NASCIMENTO, F. A. O.; RODRIGUES, S. S.; BORGES, G. A. **SISTEMA PARA ESTIMAÇÃO DE INTENÇÃO DE MOVIMENTO EM PRÓTESE DE PERNA**. 2008. Citado na p. 27.

DELIS, A.; NASCIMENTO, F.; CARVALHO, J.; ROCHA, A. da; BORGES, G. Algoritmo de estimação do ângulo do joelho para controle mioelétrico. **Universidade de Brasılia–Brasılia-DF. Brasil**, 2008. Citado nas pp. 24, 27.

DELIS, A. L. **METODOLOGIAS DE ESTIMAÇÃO DO ÂNGULO DO JOELHO PARA DETECÇÃO DA INTENÇÃO DE MOVIMENTO**. 2010. Tese de Mestrado – Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado na p. 27.

DELIS, A. L.; CARVALHO, J. L. A. de; ROCHA, A. F. da; OLIVEIRA NASCIMENTO, F. A. de; ARAÚJO BORGES, G. de. Development of a Myoelectric Controller based on Knee Angle Estimation. In: BIODEVICES. 2009. P. 97–103. Citado na p. 27.

DELIS, A. L.; DE CARVALHO, J. L. A.; SEISDEDOS, C. V.; BORGES, G. A.; DA ROCHA, A. F. Myoelectric control algorithms for leg prostheses based on data fusion with proprioceptive sensors. In: PROCEEDINGS ISSNIP biosignals and biorobotics conference. 2010. P. 137–42. Citado na p. 27.

DEY, S.; YOSHIDA, T.; FOERSTER, R. H.; ERNST, M.; SCHMALZ, T.; SCHILLING, A. F. Continuous Prediction of Joint Angular Positions and Moments: A Potential Control Strategy for Active Knee-Ankle Prostheses. **IEEE Transactions on Medical Robotics and Bionics**, Institute of Electrical e Electronics Engineers Inc., v. 2, p. 347–355, 3 ago. 2020. ISSN 25763202. DOI: 10.1109/TMRB.2020.3011841. Citado na p. 24.

DIAZ, C. P. O. **CHARACTERIZATION OF AMPUTEE GAIT USING A BIOMECHANICAL APPROACH**. 2015. Tese (Doutorado) – Faculdade de Tecnologia, Universidade de Brasília, Brasília, Brasil. Citado na p. 19.

ELERY, T.; REZAZADEH, S.; REZNICK, E.; GRAY, L.; GREGG, R. D. Effects of a Powered Knee-Ankle Prosthesis on Amputee Hip Compensations: A Case Series. **IEEE Transactions on Neural Systems and Rehabilitation Engineering**, v. 28, n. 12, p. 2944–2954, 2020. DOI: 10.1109/TNSRE.2020.3040260. Citado na p. 23.

FÉLIX, A. C. C.; VASCONCELLOS, H. A. S. **INTERPRETAÇÃO DE SINAIS EMG PARA JOELHO DE PRÓTESE ROBÓTICA**. 2008. Trabalho de conclusão de curso de graduação em Eng. Mecatrônica – Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado na p. 27.

FERREIRA, R. U.; ROCHA, A. F. da; CASCAO JR, C.; BORGES, G. A.; NASCIMENTO, F. A. O.; VENEZIANO, W. H. Reconhecimento de Padrões de Sinais de EMG para Controle de Prótese de Perna. In: SN. XI Congresso Brasileiro de Biomecânica. 2005. P. 1–5. Citado na p. 26.

HERR, H.; WILKENFELD, A. User-adaptive control of a magnetorheological prosthetic knee. **Industrial Robot**, v. 30, p. 42–55, 1 2003. ISSN 0143991X. DOI: 10.1108/01439910310457706. Citado na p. 25.

HERR, H. M.; GRABOWSKI, A. M. Bionic ankle–foot prosthesis normalizes walking gait for persons with leg amputation. **Proceedings of the Royal Society B: Biological Sciences**, v. 279, n. 1728, p. 457–464, 2012. DOI: 10.1098/rspb.2011.1194. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspb.2011.1194. Disponível em: <https://royalsocietypublishing.org/doi/abs/10.1098/rspb.2011.1194>. Citado na p. 21.

HUANG, Q.; YANG, J.; YU, Z.; XU, W.; LI, J.; LI, K. Measurement of Human Walking and Generation of Humanoid Walking Pattern, 2007. Citado na p. 24.

HUTABARAT, Y.; EKKACHAI, K.; HAYASHIBE, M.; KONGPRAWECHNON, W. Reinforcement Q-Learning Control With Reward Shaping Function for Swing Phase Control in a Semi-active Prosthetic Knee. **Frontiers in Neurorobotics**, Frontiers Media S.A., v. 14, nov. 2020. ISSN 16625218. DOI: 10.3389/fnbot.2020.565702. Citado na p. 25.

JACQUELIN PERRY, M. **GAIT ANALYSIS, Normal and Pathological Function**. Slack, 1992. Citado nas pp. 14, 16.

JÚNIOR, C. A. C. **PRÓTESE MECÂNICA PARA REABILITAÇÃO ROBÓTICA**. 2005. Trabalho de conclusão de curso de graduação em Eng. Mecatrônica – Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado na p. 26.

KAUFMAN, K. R.; FRITTOLI, S.; FRIGO, C. A. Gait asymmetry of transfemoral amputees using mechanical and microprocessor-controlled prosthetic knees. **Clinical biomechanics**, Elsevier, v. 27, n. 5, p. 460–465, 2012. Citado na p. 22.

KIM, J. H.; OH, J. H. Development of an above knee prosthesis using MR damper and leg simulator. In: v. 4, p. 3686–3691. ISBN 0780365763. DOI: 10.1109/ROBOT.2001.933191. Citado na p. 24.

MARTINS, C. P.; CHAURAIS, J. R.; ROCHA, T. S. **CONTROLE DE IMPEDÂNCIA ADAPTATIVO DO JOELHO DE UMA PRÓTESE DE PERNA PARA AMPUTAÇÕES TRANSFEMURAIS**. 2011. Trabalho de conclusão de curso de graduação em Eng. Mecatrônica – Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado na p. 27.

MCGIMPSEY, G.; BRADFORD, T. C. Limb prosthetics services and devices. **Bioengineering Institute Center for Neuroprosthetics Worcester Polytechnic Institution**, 2008. Citado na p. 20.

MCGUIRE, L. C.; STRINE, T. W.; OKORO, C. A.; AHLUWALIA, I. B.; FORD, E. S. Peer reviewed: Healthy lifestyle behaviors among older US adults with and without disabilities, behavioral risk factor surveillance system, 2003. **Preventing chronic disease**, Centers for Disease Control e Prevention, v. 4, n. 1, 2007. Citado na p. 14.

NEUMANN, D. A. **Kinesiology of the Musculoeskeletal System**. Mosby, 2002. Citado nas pp. 16, 18, 19.

PARK, J.; YOON, G. H.; KANG, J. W.; CHOI, S. B. Design and control of a prosthetic leg for above-knee amputees operated in semi-active and active modes. **Smart Materials and Structures**, Institute of Physics Publishing, v. 25, 8 jul. 2016. ISSN 1361665X. DOI: 10.1088/0964-1726/25/8/085009. Citado na p. 25.

ROCHA, T. S. **DESENVOLVIMENTO DE PRÓTESE TRANSFEMURAL ROBÓTICA: PROJETO MECÂNICO E DE ATUAÇÃO**. 2015. Tese de Mestrado – Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado nas pp. 20, 28.

SCANDAROLI, G. G. **Controle adaptativo de juntas para uma prótese robótica de perna**. 2007. Trabalho de conclusão de curso de graduação em Eng. Mecatrônica – Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado na p. 27.

SILVA ALVES, E. da. **Proposta de um módulo sensorial para estimação de postura com relação ao solo de uma prótese robótica de perna**. 2007. Trabalho de conclusão de curso de graduação em Eng. Mecatrônica – Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado na p. 27.

SILVA GOMES, A. I. da; SANTOS VIGÁRIO, P. dos; MAINENTI, M. R. M. M.; FIGUEIREDO FERREIRA, M. d. F. F. de; RIBEIRO, B. G.; ABREU SOARES, E. de. Basal and resting metabolic rates of physically disabled adult subjects: a systematic review of controlled cross-sectional studies. **Annals of Nutrition and Metabolism**, Karger Publishers, v. 65, n. 4, p. 243–252, 2014. Citado na p. 14.

SUP, F.; BOHARA, A.; GOLDFARB, M. Design and control of a powered transfemoral prosthesis. **The International journal of robotics research**, Sage Publications Sage UK: London, England, v. 27, n. 2, p. 263–273, 2008. Citado na p. 23.

TROWER, T. A. Changes in lower extremity prosthetic practice. **Physical Medicine and Rehabilitation Clinics**, Elsevier, v. 17, n. 1, p. 23–30, 2006. Citado na p. 20.

TURCOT, K.; SAGAWA JR, Y.; LACRAZ, A.; LENOIR, J.; ASSAL, M.; ARMAND, S. Comparison of the International Committee of the Red Cross foot with the solid ankle cushion heel foot during gait: a randomized double-blind study. **Archives of Physical Medicine and Rehabilitation**, Elsevier, v. 94, n. 8, p. 1490–1497, 2013. Citado na p. 21.

UNAL, R.; CARLONI, R.; HEKMAN, E. E.; STRAMIGIOLI, S.; KOOPMAN, H. F. Biomechanical conceptual design of a passive transfemoral prosthesis. In: p. 515–518. ISBN 9781424441235. DOI: 10.1109/IEMBS.2010.5626020. Citado na p. 22.

XIE, H. L.; LIANG, Z. Z.; LI, F.; GUO, L. X. The Knee Joint Design and Control of Above-knee Intelligent Bionic Leg Based on Magneto-rheological Damper. **International Journal of Automation and Computing**, v. 7, p. 277–282, 3 2010. ISSN 14768186. DOI: 10.1007/s11633-010-0503-y. Citado na p. 23.

ZHANG, Z.; YU, H.; CAO, W.; WANG, X.; MENG, Q.; CHEN, C. Design of a semi-active prosthetic knee for transfemoral amputees: Gait symmetry research by simulation. **Applied Sciences (Switzerland)**, MDPI AG, v. 11, 12 jun. 2021. ISSN 20763417. DOI: 10.3390/app11125328. Citado na p. 24.

Anexos

Anexo A – Programa Implementado

A.1 Bibliotecas Inclusas

```
#include "main.h"
#include "cmsis_os.h"
#include "usb_device.h"
#include "usbd_cdc_if.h"
#include "string.h"
#include "stdio.h"
#include "stdlib.h"
#include "usbd_cdc.h"
#include "stm32f1xx_hal.h"
#include "stm32f1xx_hal_adc.h"
#include "semphr.h"
```

Código Fonte A.1 – Bibliotecas

A.2 Declarações

A.2.1 Constantes

```
const int dec = 10;
const char opt[120] = "\n\n\r\t 0 - Portas GPIO\n\r\t 1 - Driver de Corrente\n\r\t 2 - Sensor de Corrente\n\r\t"
"3 - Modulo Inercial\n\r\t 4 - Encoder\n\r";
const char quit[30] = "Pressione q para sair...\n\r";
const char ps_rsm[50] = "Pressione r para continuar e p para pausar...\n\r";
const char tncd[25] = "\n\rTeste Encoder...\n\r";
const char tmi[31] = "\n\rTeste Modulo Inercial...\n\r";
const char tdc[31] = "\n\rTeste Driver de Corrente...\n\r";
const char tsc[31] = "\n\rTeste Sensor de Corrente...\n\r";
const char tgpio[31] = "\n\rTeste Portas GPIO...\n\r";
const char msg[20] = "Conectado\n\r";
const char texto[20] = "Operando...";
const char space[2] ="  ";
const char enter[2] = "\n\r";
const char titulo[100] = "\n\n\t 0 - Modo Funcional\n\r \t 1 - Modo Teste\n\r Comando: ";
```

Código Fonte A.2 – Constantes globais

A.2.2 Variáveis

```
ADC_HandleTypeDef hadc1;
I2C_HandleTypeDef hi2c1;
SPI_HandleTypeDef hspi2;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
uint8_t n_buf[64];
char USBq[500];
int USBq_init_index = 0;
```

```
int USBq_end_index = 0;
SemaphoreHandle_t listMutex;
```

Código Fonte A.3 – Variáveis globais

A.2.3 Threads

```
osThreadId_t defaultTaskHandle;
const osThreadAttr_t defaultTask_attributes = {
  .name = "defaultTask",
  .priority = (osPriority_t) osPriorityBelowNormal,
  .stack_size = 1024};
osThreadId_t TaskTesteHardwaHandle;
const osThreadAttr_t TaskTesteHardwa_attributes = {
  .name = "TaskTesteHardwa",
  .priority = (osPriority_t) osPriorityAboveNormal,
  .stack_size = 1024};
osThreadId_t TaskOprtHandle;
const osThreadAttr_t TaskOprt_attributes = {
  .name = "TaskOprt",
  .priority = (osPriority_t) osPriorityNormal,
  .stack_size = 1024};
```

Código Fonte A.4 – Threads implementadas

A.2.4 Protótipos de Funções

```
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void);
static void MX_SPI2_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);
void StartDefaultTask(void *argument);
void StartTaskTstHrd(void *argument);
void StartTaskOprt(void *argument);
void testeEncoder();
void testeI2C();
void testeADC();
void testeGPIO();
void testePWM();
void setPWM(TIM_HandleTypeDef, uint32_t, uint16_t);
uint8_t SendUSB(uint8_t*, uint16_t);
uint8_t SPI_Send_Receive(uint8_t);
uint8_t SPI_Receive();
int exp_int(int, int);
void f_conv_trans(char* , int , float );
void pause_resume();
void us_delay(uint16_t);
void Send_to_USBq(uint8_t*, uint16_t);
uint16_t timer_set(uint8_t);
```

Código Fonte A.5 – Funçoes

A.3 Função Main

```
int main(void){
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_ADC1_Init();
  MX_I2C1_Init();
  MX_SPI2_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();
  osKernelInitialize();
  listMutex = xSemaphoreCreateMutex();
  defaultTaskHandle = osThreadNew(StartDefaultTask, NULL,
  &defaultTask_attributes);
  TaskTesteHardwaHandle = osThreadNew(StartTaskTstHrd, NULL,
  &TaskTesteHardwa_attributes);
  TaskOprtHandle = osThreadNew(StartTaskOprt, NULL, &TaskOprt_attributes);
  osKernelStart();
  while (1){}}
```

Código Fonte A.6 – Main

A.4 Funções de Configuração

A.4.1 System Clock

```
void SystemClock_Config(void){
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL6;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK){
    Error_Handler();}
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  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_1) != HAL_OK){
    Error_Handler();  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC|RCC_PERIPHCLK_USB;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV4;
  PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLL;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK){
    Error_Handler();}}
```

Código Fonte A.7 – Configuração do clock

A.4.2 Conversor Analógico Digital

```
static void MX_ADC1_Init(void){
  ADC_ChannelConfTypeDef sConfig = {0};
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK){
    Error_Handler();}
  sConfig.Channel = ADC_CHANNEL_4;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK){
    Error_Handler();}}
```

Código Fonte A.8 – Configuração conversor analógico digital

A.4.3 Comunicação I2C

```
static void MX_I2C1_Init(void){
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK){
    Error_Handler();}}
```

Código Fonte A.9 – Configuração comunicação I2C

A.4.4 Comunicação SPI

```
static void MX_SPI2_Init(void){
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_2EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi2) != HAL_OK){
    Error_Handler();}}
```

Código Fonte A.10 – Configuração comunicação SPI

A.4.5 Temporizadores

```
static void MX_TIM2_Init(void){
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 16384;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK){
    Error_Handler();}
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK){
    Error_Handler();}
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK){
    Error_Handler();}
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK){
    Error_Handler();}
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_2) != HAL_OK){
    Error_Handler();}
  HAL_TIM_MspPostInit(&htim2);}

static void MX_TIM3_Init(void){
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 71;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 65535;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK){
    Error_Handler();}
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK){
    Error_Handler();}
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK){
    Error_Handler();}}

static void MX_TIM4_Init(void){
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 50000;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 65535;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK){
```

```
    Error_Handler();}
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK){
    Error_Handler();}
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK){
    Error_Handler();}}
```

Código Fonte A.11 – Configuração dos temporizadores

A.4.6 Portas GPIO

```
static void MX_GPIO_Init(void){
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
  GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  GPIO_InitStruct.Pin = GPIO_PIN_12;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);}
```

Código Fonte A.12 – Configuração portas GPIO

A.5 Funções Auxiliares

```
uint16_t timer_set(uint8_t cmd){
        uint16_t ticks;
        if(cmd == 0){
                HAL_TIM_Base_Start(&htim4);
                return 0;}
        else if(cmd == 1){
                return __HAL_TIM_GET_COUNTER(&htim4);}
        else if(cmd == 2){
                ticks = __HAL_TIM_GET_COUNTER(&htim4);
                htim4.Instance->CNT = 0;
                return ticks;}
        else if(cmd == 3){
                HAL_TIM_Base_Stop(&htim4);
                return 0;}
```

```
        return 1;}

void us_delay(uint16_t tempo){
        uint16_t temp_init, temp_tt;
        HAL_TIM_Base_Start(&htim3);
        temp_init = __HAL_TIM_GET_COUNTER(&htim3);
        do{
                temp_tt = __HAL_TIM_GET_COUNTER(&htim3) - temp_init;
        }while(temp_tt < tempo);
        HAL_TIM_Base_Stop(&htim3);}

uint8_t SPI_Receive(){
        uint8_t rp;
        us_delay(20);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_SPI_Receive(&hspi2, (uint8_t*)&rp, 1, 100);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET);
        us_delay(20);
        return rp;}

uint8_t SPI_Send_Receive(uint8_t cmd){
        uint8_t rp;
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_SPI_Transmit(&hspi2, (uint8_t*)&cmd, 1, 100);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET);
        us_delay(20);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_SPI_Receive(&hspi2, (uint8_t*)&rp, 1, 100);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET);
        return rp;}

void pause_resume(){
         if(n_buf[0] == 'p'){
                 const char pause[15] = "Pausado...\n\r";
                 const char retorno[15] = "Retomada...\n\r";
                 memset(n_buf, '\0', sizeof(n_buf));
                 SendUSB((uint8_t*)pause, strlen(pause));
                 while(n_buf[0] != 'r' && n_buf[0] != 'q'){
                         memset(n_buf, '\0', sizeof(n_buf));
                         HAL_Delay(100);}
                 if(n_buf[0] == 'r'){
                         SendUSB((uint8_t*)retorno, strlen(retorno));
                         memset(n_buf, '\0', sizeof(n_buf));
                         osDelay(1000);}}}

int exp_int(int base, int expt){
        int i;
        float res = 1;
        for (i = 0; i < expt; i++){
                res = res*base;}
        return res;}

void f_conv_trans(char c_res[20], int nc, float num){
        char mas[20];
        float num_aux, base10;
        uint16_t resul, rest;
        num_aux = num;
        resul = (uint16_t)num_aux;
        num_aux = num_aux - (float)resul;
        base10 = exp_int(10, nc);
```

```
        num_aux = num_aux*base10;
        rest = (uint16_t)num_aux;
        sprintf(mas, "%d.%d", resul, rest);
        strcpy(c_res, mas);}

void setPWM(TIM_HandleTypeDef timer, uint32_t channel, uint16_t pulse){
         HAL_TIM_PWM_Stop(&timer, channel);
         TIM_OC_InitTypeDef sConfigOC;
         sConfigOC.OCMode = TIM_OCMODE_PWM1;
         sConfigOC.Pulse = pulse;
         sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
         sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
         HAL_TIM_PWM_ConfigChannel(&timer, &sConfigOC, channel);
         HAL_TIM_PWM_Start(&timer, channel);}

uint8_t SendUSB(uint8_t* txt, uint16_t tam){
        uint8_t usb_ok = USBD_OK;
        uint8_t count = 0;
        do{
                    usb_ok = CDC_Transmit_FS(txt, tam);
                    count++;
                    us_delay(100);
        }while(usb_ok != USBD_OK && count < 10);
        if(usb_ok == USBD_OK){
                  return 0;}
        else{
                  return 1;}}

void Send_to_USBq(uint8_t* txt, uint16_t tam){
        int i = 0;
        xSemaphoreTake(listMutex, portMAX_DELAY);
        for(i = 0; i < tam; i++){
                    USBq[USBq_end_index] = txt[i];
                    if(USBq_end_index < sizeof(USBq) - 1){
                            USBq_end_index++;}
                    else{
                            USBq_end_index = 0;}}
        xSemaphoreGive(listMutex);}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  if (htim->Instance == TIM1) {
    HAL_IncTick();}}
```

Código Fonte A.13 – Funções auxiliares

A.6 Funções de Teste

A.6.1 Teste GPIO

```
void testeGPIO(){
        int i = 0;
        uint16_t gpioP;
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
        i = 0;
        gpioP = GPIO_PIN_5;
        do{
                memset(n_buf, '\0', sizeof(n_buf));
```

```
                HAL_GPIO_WritePin(GPIOA, gpioP, GPIO_PIN_SET);
                HAL_Delay(200);
                pause_resume();
                if(n_buf[0] == 'q'){
                        break;}
                HAL_GPIO_WritePin(GPIOA, gpioP, GPIO_PIN_RESET);
                HAL_Delay(200);
                i++;
                if(i == 1){
                        gpioP = GPIO_PIN_6;}
                else if(i == 2){
                        gpioP = GPIO_PIN_7;
                        i = -1;}
                else if(i == 0){
                        gpioP = GPIO_PIN_5;}
                pause_resume();
        }while(n_buf[0] != 'q');
        i = 0;
        memset(n_buf, '\0', sizeof(n_buf));
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);}
```

Código Fonte A.14 – Teste portas GPIO

A.6.2 Teste Driver de Corrente - (PWM)

```
void testePWM(){
        int i = 0;
        do{
                for(i = 0; i < 1638; i++){
                        pause_resume();
                        if(n_buf[0] == 'q'){
                                setPWM(htim2, TIM_CHANNEL_2, 0);
                                break;}
                        memset(n_buf, '\0', sizeof(n_buf));
                        setPWM(htim2, TIM_CHANNEL_2, i*10);
                        osDelay(1);}
                }while(n_buf[0] != 'q');
                memset(n_buf, '\0', sizeof(n_buf));}
```

Código Fonte A.15 – Teste driver de corrente

A.6.3 Teste Sensor de Corrente - (Conversor ADC)

```
void testeADC(){
        int i = 0;
        unsigned int adc_read;
        uint16_t delt;
        char str_msg[20];
        memset(str_msg, '\0', sizeof(str_msg));
        timer_set(0);
        do{
                memset(n_buf, '\0', sizeof(n_buf));
                for(i = 0; i < 1638; i++){
                        pause_resume();
```

```
                    if(n_buf[0] == 'q'){
                            setPWM(htim2, TIM_CHANNEL_2, 0);
                            break;}
                    memset(n_buf, '\0', sizeof(n_buf));
                    HAL_ADC_Start(&hadc1);
                    setPWM(htim2, TIM_CHANNEL_2, 10*i);
                    utoa(i, str_msg, dec);
                    Send_to_USBq((uint8_t*)str_msg, strlen(str_msg));
                    Send_to_USBq((uint8_t*)space, strlen(space));
                    osDelay(10);
                    adc_read = HAL_ADC_GetValue(&hadc1);
                    delt = timer_set(2);
                    utoa(adc_read, str_msg, dec);
                    Send_to_USBq((uint8_t*)str_msg, strlen(str_msg));
                    Send_to_USBq((uint8_t*)space, strlen(space));
                    utoa(delt, str_msg, dec);
                    Send_to_USBq((uint8_t*)str_msg, strlen(str_msg));
                    Send_to_USBq((uint8_t*)enter, strlen(enter));
                    HAL_ADC_Stop(&hadc1);}
        }while(n_buf[0] != 'q');
        memset(n_buf, '\0', sizeof(n_buf));
        timer_set(3);}
```

Código Fonte A.16 – Teste sensor de corrente

A.6.4 Teste Módulo Inercial - (I2C)

```
void testeI2C(){
    int neg = 0, i = 0;
    HAL_StatusTypeDef ret;
    char tag[10], sendmsg[20];
    char n_sig[2] = " -";
    char erro[20] = "Erro\n\r";
    uint8_t init[2] = {0x6b, 0x00};
    uint8_t Acconf[2] = {0x1c, 0x08};
    uint8_t AcWconf[2] = {0x1b, 0x08};
    uint8_t regaddr[7] = {0x3b, 0x3d, 0x3f, 0x41, 0x43, 0x45, 0x47};
    uint8_t buf[2] = {0x00, 0x00};
    uint16_t read0, read1;
    uint16_t dt;
    float g_ac = 0;
    const float g = 9.8;
    static const uint8_t ADDR = 0x68 << 1; // Use 8-bit address
    ret = HAL_I2C_IsDeviceReady(&hi2c1, ADDR, 3, 1000);
    if(ret == HAL_OK){
                    ret = HAL_I2C_Master_Transmit(&hi2c1, ADDR, init, 2, 100);
        if(ret == HAL_OK){
                    ret = HAL_I2C_Master_Transmit(&hi2c1, ADDR, Acconf, 2, 100);}
        if(ret == HAL_OK){
                    ret = HAL_I2C_Master_Transmit(&hi2c1, ADDR, AcWconf, 2, 100);}}
    else{
        if(ret == HAL_BUSY){
            strcpy(erro, "HAL_BUSY\n\r");}
        else if(ret == HAL_TIMEOUT){
            strcpy(erro, "HAL_TIMEOUT\n\r");}
        else{
            strcpy(erro, "HAL_ERROR\n\r");}}
    timer_set(0);
    do{
```

```
memset(n_buf, '\0', sizeof(n_buf));
if(ret == HAL_OK){
          for(i = 0; i < 7; i++){
        osDelay(10);
        pause_resume();
        if(n_buf[0] == 'q'){
                            break;}
        memset(n_buf, '\0', sizeof(n_buf));
        HAL_I2C_Mem_Read(&hi2c1, ADDR, regaddr[i],
        I2C_MEMADD_SIZE_8BIT, buf, 2, 100);
        dt = timer_set(2);
        if(i == 0){
            strcpy(tag, "x : \0");}
        else if(i == 1){
            strcpy(tag, "y : \0");}
        else if(i == 2){
            strcpy(tag, "z : \0");}
        else if(i == 3){
            strcpy(tag, "t : \0");}
        else if(i == 4){
            strcpy(tag, "Wx : \0");}
        else if(i == 5){
            strcpy(tag, "Wy : \0");}
        else{
            strcpy(tag, "Wz : \0");}
        Send_to_USBq((uint8_t*)tag, strlen(tag));
        read0 = (uint16_t)buf[0];
        read1 = (uint16_t)buf[1];
        read0 = read0 << 8;
        read0 = read0 | read1;
        if(read0 <= 32768){
            neg = 0;}
        else{
            read0 = 65536 - read0;
            neg = 1;}
        utoa(read0, sendmsg, dec);
        if(i < 3){
            g_ac = (float)read0/8192*g;}
        if(i == 3){
            g_ac = (float)read0/340;
            if(neg == 1){
                g_ac = g_ac*(-1);}
            g_ac = g_ac + 36.53;
            if(g_ac < 0){
                neg = 1;
                g_ac = g_ac*(-1);}
            else{
                neg = 0;}}
        if(i > 3){
            g_ac = (float)read0/65.5;}
        if(neg == 1){
            Send_to_USBq((uint8_t*)n_sig, strlen(n_sig));}
        f_conv_trans(sendmsg, 3, g_ac);
        Send_to_USBq((uint8_t*)sendmsg, strlen(sendmsg));
        Send_to_USBq((uint8_t*)space, strlen(space));
        utoa(dt, sendmsg, dec);
        Send_to_USBq((uint8_t*)sendmsg, strlen(sendmsg));
        Send_to_USBq((uint8_t*)enter, strlen(enter));}}
else{
    SendUSB((uint8_t*)erro, strlen(erro));}
```

```
    }while(n_buf[0] != 'q');
    memset(n_buf, '\0', sizeof(n_buf));
    timer_set(3);}
```

Código Fonte A.17 – Teste módulo inercial

A.6.5 Teste Encoder Angular - (SPI)

```
void testeEncoder(){
    uint8_t resp = 0x00;
    uint8_t  dado[2];
    uint16_t deg;
    uint16_t dt;
    char msg[20];
    const uint8_t command[3] = {0x00, 0x10, 0x70};
    const char erro[10] = "Erro\n\r";
    const uint8_t ack = 0xa5, stp = 0x80;
    int i = 0;
    int count = 0;
    do{
                memset(n_buf, '\0', sizeof(n_buf));
                resp = SPI_Send_Receive(command[0]);
                utoa(resp, msg, dec);
                Send_to_USBq((uint8_t*)msg, strlen(msg));
                Send_to_USBq((uint8_t*)enter, strlen(enter));
        i++;
    }while(resp != ack && i < 20);
    if(resp == ack){
                resp = SPI_Send_Receive(command[2]);
                i = 0;
                do{
                  memset(n_buf, '\0', sizeof(n_buf));
                  us_delay(20);
                  resp = SPI_Receive();
                  if(i > 5){
                resp = SPI_Send_Receive(command[2]);
                      i = 0;}
            utoa(resp, msg, dec);
            Send_to_USBq((uint8_t*)msg, strlen(msg));
            Send_to_USBq((uint8_t*)enter, strlen(enter));
            i++;
                }while(resp != stp && n_buf[0] != 'q');
                timer_set(0);
                do{
                  memset(n_buf, '\0', sizeof(n_buf));
                  i = 0;
                  resp = 0;
                  resp = SPI_Send_Receive(command[1]);
                  if(resp == ack){
                do{
                    us_delay(20);
                    if(i > 10){
                        resp = SPI_Send_Receive(command[1]);
                        i = 0;
                    }
                    resp = SPI_Receive();
                    i++;
                }while(resp != command[1]);}
            us_delay(20);
```

```
            dado[0] = SPI_Receive(command[0]);
            us_delay(20);
            dado[1] = SPI_Receive(command[0]);
            dt = timer_set(2);
            deg = 256*dado[0] + dado[1];
            deg = 360 - deg*360/4096;
            if(dado[0] != ack && dado[1] != command[1] && dado[1] != ack &&
            dado[1] != command[1]){
                utoa(deg, msg, dec);
                Send_to_USBq((uint8_t*)msg, strlen(msg));
                Send_to_USBq((uint8_t*)space, strlen(space));
                utoa(dt, msg, dec);
                Send_to_USBq((uint8_t*)msg, strlen(msg));
                Send_to_USBq((uint8_t*)enter, strlen(enter));}
            else{
                for(i = 0; i < 10; i++){
                    resp = SPI_Send_Receive(command[0]);}}
            osDelay(40);
            pause_resume();
        }while(n_buf[0] != 'q');
        memset(n_buf, '\0', sizeof(n_buf));
        timer_set(3);}
    else{
        SendUSB((uint8_t*)erro, strlen(erro));}}
```

Código Fonte A.18 – Teste encoder

A.7 Threads

A.7.1 Thread de Transmissão de Dados

```
void StartDefaultTask(void *argument){
    MX_USB_DEVICE_Init();
    osThreadId id_task = osThreadGetId();
    char data;
    uint8_t usb_ok = USBD_OK;
    int USBq_depth;
    int prty;
    osThreadSetPriority(id_task, osPriorityBelowNormal);
    prty = 0;
    for(;;){
        while(USBq_init_index != USBq_end_index){
            xSemaphoreTake(listMutex, portMAX_DELAY);
            data = USBq[USBq_init_index];
            usb_ok = SendUSB((uint8_t*)&data, sizeof(data));
            if(!usb_ok){
                if(USBq_init_index < sizeof(USBq) - 1){
                    USBq_init_index++;}
                else{
                    USBq_init_index = 0;}}
            id_task = osThreadGetId();
            if(USBq_init_index <= USBq_end_index){
                USBq_depth = USBq_end_index - USBq_init_index;}
            else{
                USBq_depth = sizeof(USBq) - USBq_init_index + USBq_end_index;}
            if(id_task != NULL){
                if(USBq_depth >= 450 && prty == 0){
```

```
                    osThreadSetPriority(id_task, osPriorityAboveNormal);
                    prty = 1;}
                else if(USBq_depth < 450 && prty == 1){
                    osThreadSetPriority(id_task, osPriorityBelowNormal);
                    prty = 0;}}
        xSemaphoreGive(listMutex);}
    osDelay(1);}}
```

Código Fonte A.19 – Thread de transmissão de dados

A.7.2 Thread de Testes

```
void StartTaskTstHrd(void *argument){
    uint8_t tst = 0, opr = 1;
    for(;;){
        if(n_buf[0] == 'c'){
            SendUSB((uint8_t*)titulo, strlen(titulo));
            memset(n_buf, '\0', sizeof(n_buf));
            while(n_buf[0] != '0' && n_buf[0] != '1'){
                memset(n_buf, '\0', sizeof(n_buf));
                osDelay(100);}
            SendUSB((uint8_t*)enter, strlen(enter));
            if(n_buf[0] == '0'){
                opr = 1;
                tst = 0;}
            else if(n_buf[0] == '1'){
                      opr = 0;
                tst = 1;}
            memset(n_buf, '\0', sizeof(n_buf));}
            if(opr){
                //Operação do sistema
                SendUSB((uint8_t*)texto, strlen(texto));
                while(n_buf[0] != 'c'){
                    memset(n_buf, '\0', sizeof(n_buf));
                    osDelay(100);}
                         SendUSB((uint8_t*)enter, strlen(enter));}
            if(tst){
                //Modo de teste
                SendUSB((uint8_t*)opt, strlen(opt));
                while((n_buf[0] < '0' || n_buf[0] > '4') && n_buf[0] != 'c'){
                    memset(n_buf, '\0', sizeof(n_buf));
                    osDelay(100);}
                    if(n_buf[0] == '0'){
                        memset(n_buf, '\0', sizeof(n_buf));
                        SendUSB((uint8_t*)tgpio, strlen(tgpio));
                        SendUSB((uint8_t*)quit, strlen(quit));
                        SendUSB((uint8_t*)ps_rsm, strlen(ps_rsm));
                        testeGPIO();}
                    else if(n_buf[0] == '1'){
                        //Teste drive de corrente
                        memset(n_buf, '\0', sizeof(n_buf));
                        SendUSB((uint8_t*)tdc, strlen(tdc));
                        SendUSB((uint8_t*)quit, strlen(quit));
                        SendUSB((uint8_t*)ps_rsm, strlen(ps_rsm));
                        testePWM();}
                    else if(n_buf[0] == '2'){
                        //Teste sensor de corrente
                        memset(n_buf, '\0', sizeof(n_buf));
                        SendUSB((uint8_t*)tsc, strlen(tsc));
```

```
                    SendUSB((uint8_t*)quit, strlen(quit));
                    SendUSB((uint8_t*)ps_rsm, strlen(ps_rsm));
                    testeADC();}
                else if(n_buf[0] == '3'){
                    //Teste modulo inercial
                    memset(n_buf, '\0', sizeof(n_buf));
                    SendUSB((uint8_t*)tmi, strlen(tmi));
                    SendUSB((uint8_t*)quit, strlen(quit));
                    SendUSB((uint8_t*)ps_rsm, strlen(ps_rsm));
                    testeI2C();}
                else if(n_buf[0] == '4'){
                    //Teste encoder
                    memset(n_buf, '\0', sizeof(n_buf));
                    SendUSB((uint8_t*)tncd, strlen(tncd));
                    SendUSB((uint8_t*)quit, strlen(quit));
                    SendUSB((uint8_t*)ps_rsm, strlen(ps_rsm));
                    testeEncoder();}}
    while(USBq_init_index != USBq_end_index){
        osDelay(10);}}}
```

Código Fonte A.20 – Thread de testes

A.7.3 Thread de Operação Padrão

```
void StartTaskOprt(void *argument){
    for(;;){
        osDelay(500);}}
```

Código Fonte A.21 – Thread de operação padrão