GUILHERME DE AGRELA LOPES
JOÃO VITOR SANCHES

AQUISIÇÃO, CONDICIONAMENTO E TRANSMISSÃO DE DADOS COM CÉLULAS DE CARGA PARA APLICAÇÕES EM ROBÓTICA

São Paulo
2020

GUILHERME DE AGRELA LOPES
JOÃO VITOR SANCHES

AQUISIÇÃO, CONDICIONAMENTO E TRANSMISSÃO DE DADOS COM CÉLULAS DE CARGA PARA APLICAÇÕES EM ROBÓTICA

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecatrônico.

São Paulo
2020

GUILHERME DE AGRELA LOPES
JOÃO VITOR SANCHES

AQUISIÇÃO, CONDICIONAMENTO E TRANSMISSÃO DE DADOS COM CÉLULAS DE CARGA PARA APLICAÇÕES EM ROBÓTICA

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecatrônico.

Orientador:
Rafael Traldi Moura

Co-orientador:
Jun Okamoto Jr.

São Paulo
2020

AGRADECIMENTOS

Ao Fundo Patrimonial Amigos da Poli fica nossa profunda gratidão pela confiança e pelo apoio ao projeto desenvolvido nesse trabalho. Não somente ao viabilizar a implementação nos laboratórios da Escola Politécnica, mas também por despertar em nós o sentimento que eleva muito o que todo esse trabalho representa e assim, motivar a sua completude. Aproveitamos para reafirmar nosso compromisso em devolver para a Escola uma pequena fração de tudo que nela aprendemos.

Agradecemos também ao Prof. Rafael Traldi Moura, bem como ao Prof. Jun Okamoto Jr. por primeiro nos escutarem e, ao longo de todo o processo, ajudarem com o possível.

RESUMO

O ambiente da Robótica possui diversos equipamentos, componentes e ferramentas que precisam ser instrumentados, de modo a garantir seu correto funcionamento e precisão adequada. Nesse contexto, busca-se implementar uma solução para aquisição e condicionamento de dados provenientes de extensômetros que seja versátil, precisa e de baixo custo com relação aos produtos do mercado, além de prover uma plataforma para que equipamentos que usam diferentes protocolos de comunicação possam interagir entre si. Dessa forma, foi criada a PANDA (Placa de Aquisição e Condicionamento de Dados), um projeto que auxilia em oito aplicações diferentes da Escola Politécnica da USP, sobretudo nas áreas de Engenharia Mecatrônica e Mecânica, trazendo dados em tempo real relativos a deformações, forças, tensões e potências em células de carga; leitura de *encoders* e termistores; leitura de torques em exoesqueletos; caracterização de motores; e controle de módulos motorizados. Junto com essas tarefas, também há a comunicação com outros equipamentos através dos protocolos USB, CAN, I^2C e RS485. O protótipo desenvolvido, financiado pelo Fundo Patrimonial Amigos da Poli, obteve resultados satisfatórios de acordo com os requisitos levantados e, sendo uma plataforma *open-source* que ajuda a Escola, além de ajudar alunos no seu aprendizado e pesquisadores em seus Laboratórios, permite também que no futuro o projeto possa ser otimizado e assim abranger cada vez mais aplicações.

Palavras-Chave – Célula de carga; Aquisição de sinais; Medição; Força; Torque; Extensômetros; Protocolos de dados; Controle.

ABSTRACT

The Robotics environment has several equipments, components and tools that need to be instrumented, in order to guarantee its correct functioning and adequate precision. In this context, this project seeks to implement a solution for the acquisition and conditioning of data from strain gauges that is versatile, accurate and inexpensive in relation to the products on the market, in addition to providing a platform for equipments that use different communication protocols, so they are able to interact with each other. Thus, PANDA (Data Acquisition and Conditioning Board) was created, a project that assists in eight different applications of the Polytechnic School of USP, especially in the areas of Mechatronic and Mechanical Engineering, bringing real-time data on deformations, forces, stresses and powers in load cells; reading of encoders and thermistors; reading of torques in exoskeletons; characterization of motors; and control of motorized modules. Along with these tasks, there is also communication with other equipment through USB, CAN, I^2C and RS485 protocols. The developed prototype, financed by Patrimonial Fund Amigos da Poli, obtained satisfactory results according to the requirements raised and, being an open-source platform that helps the School, in addition to helping students in their learning and researchers in their Laboratories, also allows the project to be optimized in the future and thus cover more and more applications.

Keywords – Load cell; Data acquisition; Measurement; Force; Torque; Strain gauges; Data protocols; Control.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE LISTAGEMS

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Já com um grande número de robôs, o cenário industrial ainda apresenta grandes tendências de crescimento. Estimulada pelo mercado chinês, com evolução tecnológica tardia, frentes de automação fabril demandam grande número de recursos. Como tendência mundial, a interação humano-máquina vem tomando cada vez mais espaço, como forma de potencializar os métodos produtivos. [2]

Tal interação é denominada *(*"operator in the loop") e, nos estudos sobre o futuro dessa área, compartilha espaço com temas como "Service Robotics". Esse último, com viés de maior alcance fora do ambiente produtivo, consiste da prestação de serviços e por isso também infere a necessidade de aprimorar a relação dos humanos com robôs. [3]

De modo geral, as máquinas devem ser mais eficientes em ler o ambiente ao seu redor e tomar decisões mais assertivas. Com a evolução da capacidade de processamento, traduzida nos algoritmos de controle, cresce também a demanda por medidas, na forma de sinais.

Há tempos sistemas de medição permitem a obtenção de dados para muitas tarefas da sociedade e embora muitas vezes errôneas, o elemento humano nos processos atua como filtro, minimizando problemas. Com a demanda por dados mais acurados e em frequências maiores, no âmbito da robótica os mecanismos de aquisição ficam mais rebuscados e, para operarem de forma autônoma, devem minimizar erros. [4]

Tendo isso em mente, é válido notar um aumento do esforço no aprimoramento da microeletrônica e de circuitos eletrônicos voltados para a precisão no condicionamento do sinal adquirido. O avanço na utilização de sensores ou transdutores inteligentes, compostos por microcontroladores, conversores e processadores de sinais digitais e analógicos, além de outros circuitos integrados, vem transformando esses sensores em sistemas de medição cada vez mais robustos e confiáveis. [5]

O ambiente onde o processo ocorre também interfere no modo com que o sinal será

tratado. Devido às várias fontes de ruído que podem comprometer um sistema e à dificuldade de se separar de onde vem cada parte desse ruído, torna-se mais complicada a remoção dessa interferência sem causar alterações ao sinal que se pretende captar [6]. Portanto, deve-se atentar à construção de filtros que atendam às especificações requeridas pelo projeto. Para o auxílio na construção desse e de outros elementos, destaca-se a importância do uso de *softwares* de simulação de circuitos, para a garantia do correto funcionamento do que está sendo proposto.

Além de melhores sinais de entrada, os sistemas robóticos demandam a boa caracterização das plantas e processos. Tomando os atuadores, interfaces entre um controlador e a planta, fica claro que é de extrema importância também conhecer com precisão as características do mesmo. Motores elétricos, por exemplo, devem ter curvas de torque e velocidade bem conhecidas. Para tal, sistemas de medição também são relevantes.

Para sistemas de cooperação homem-máquina, o cuidado à atuação é especialmente importante, uma vez que contatos devem ser previstos e tratados com segurança. Para tal propósito existem métodos como o controle de impedância ou controle de força direta [7]. Nessa categoria de máquina, é comum o emprego de motores BLDC[1] devido suas características construtivas e performance [8].

Para a aplicação dos controles citados, é necessária a realimentação dos esforços exercidos pelo robô. Tal grandeza, na ausência de sensores dedicados, pode ser estimada por observadores de corrente [9]. Entretanto, quando os mecanismos e transmissões são complexos, a estimação requer grande esforço de calibração e ainda assim pode apresentar resultados falhos. Nesse contexto se faz necessário o uso de sensores de torque e força em aplicações robóticas.

1.2 Revisão Bibliográfica Histórica e Estado da Arte

Quando a captação de forças é necessária, a alternativa comum na indústria é o uso de sensores de 6 graus de liberdade (3 forças e 3 momentos) [10]. Um exemplo desse dispositivo é mostrado na figura 1, da empresa HBM:

[1] Brushless Direct Current

Figura 1: Transdutor industrial do tipo F/T (força/torque) *6dof*.

Fonte: HBM [11].

Tal equipamento não costuma ser viável para aplicações em robôs de serviço, devido ao seu elevado custo. Nesse caso, pode-se optar por medir torques nas juntas, com células de carga especializadas [12].

Com mecanismos de articulação com apoios de maneira que a célula de carga tenha apenas esforço torsional, é possível realizar a medição sem o fenômeno de *crosstalk* (presença de componentes na medida advindos de momentos ou forças que não os desejados [13]) [14].

Soluções de mercado nesse formato são poucas, pois é difícil adequar formatos padrão à mecanismos normais (frequentemente com limitações de espaço). Medições em grandes mecanismos podem utilizar transdutores no formato de flanges, como ilustrado na figura 2, da empresa Honeywell.

Figura 2: Sensor de torque reativo do tipo flange.

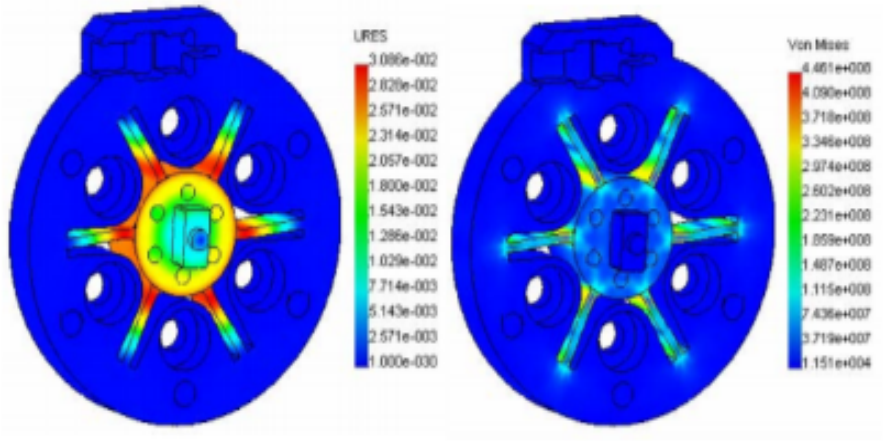
Fonte: Honeywell [15].

Para maximização da sensibilidade, projetos específicos devem ser feitos, com a possibilidade de se utilizar, inclusive, materiais com menor módulo de elasticidade. Atenção deve ser dada para outros critérios, como rigidez do mecanismo e resistência à fadiga.

Junto a esses cuidados, é importante que as deformações em decorrência de fatores como variação de temperatura e esforços que não o objeto da medição, sejam consideradas na alocação dos extensômetros [16].

Como em [12], [13] e [14], a célula de carga pode ser simulada numericamente. Esses três trabalhos concordam ao empregar análise por elementos finitos, ferramenta capaz de impulsionar a performance do sensor (ao permitir otimização das regiões de medição) e garantir distanciamento de falhas e efeitos indesejados. Um exemplo de análise numérica é apresentado na figura 3:

Figura 3: Análise de deformações em célula de carga com método de elementos finitos.



Fonte: extraído de [12].

Um software adequado para realização dessas simulações é o *Abaqus*, no qual a modelagem de materiais dúcteis é bastante completa e as ferramentas de análise permitem boa estimação do comportamento do dispositivo [17].

As deformações mecânicas nos pontos adequados devem ser transformadas em sinais elétricos. Para tal, extensômetros são colados na superfície do elemento elástico. Este componente é construído de modo que a resistência elétrica entre seus terminais varia linearmente com a deformação mecânica, multiplicada por um fator característico (*gauge factor*). Assim sendo, é comum montar o arranjo da ponte de Wheatstone [4].

Esse circuito pode ser compreendido como um comparador entre dois divisores de tensão resistivos, de modo que variações em qualquer uma das resistências faz variar a diferença de tensão medida.

Pontos negativos dessa técnica de medição são associados a variações da saída com mudanças de temperatura no sensor e instalação incorreta dos extensômetros [18]. Tais pontos podem ser mitigados com a correta calibração do sensor.

Nesse processo, deve-se comparar a saída do sistema de medição com um equipamento equivalente, de precisão conhecida. Essencialmente, para um sensor extensométrico digital, é válido impor entradas conhecidas, nulas e não-nulas, para que o sistema inteligente do equipamento componha a curva de calibração. Para a validade desse método, deve-se garantir no projeto da célula de carga, que o comportamento para qualquer entrada respeitará a linearidade, base para o funcionamento do dispositivo [4].

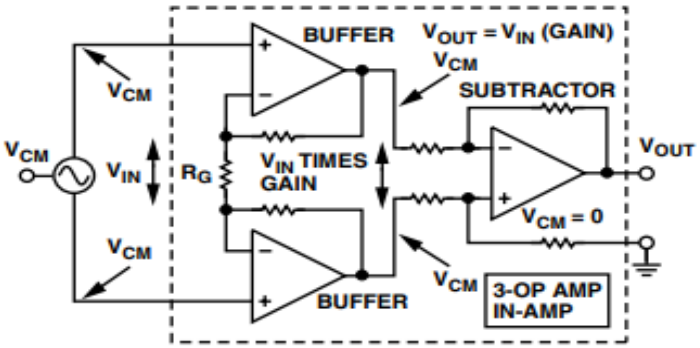
Durante a vida útil, o mecanismo de medida em questão deverá sempre ser calibrado, com um método bem definido durante a fase de projeto, e com frequência compatível com a tolerância da aplicação.

Essas células de carga, de modo geral, utilizam da teoria de extensometria, como explicado acima. Portanto, eletronicamente, é necessário o uso de um circuito que consiga captar o sinal adquirido da Ponte de Wheatstone e o interprete com precisão [19]. Para isto, exige-se não só componentes passivos de alta precisão, como também um projeto que siga à risca as especificações técnicas.

O sinal da ponte, analógico, precisa passar por um filtro e ser amplificado, para então ocorrer sua conversão para sinal digital e ser manuseado por um microcontrolador. O projeto do filtro, como já dito, depende do ambiente em que o sensor será utilizado, já que isto interfere nos tipos de ruído que estarão presentes. Dessa forma, não há como se ter um consenso sobre qual o melhor modo para a eliminação desses ruídos. O que se pode afirmar é que, para que se mantenha a alta precisão, a filtragem e a amplificação demandam o uso de in-amps[2] [20].

[2]Amplificadores Operacionais de instrumentação

Figura 4: Circuito equivalente de um *in-amp*.



Fonte: extraído de [20].

A conversão e o tratamento do sinal podem ser realizados com um conjunto de circuitos integrados, ou diretamente em um microcontrolador. O avanço nas tecnologias atuais de processamento permite que seja adicionada cada vez mais complexidade nos firmwares de placas eletrônicas, garantindo uma melhor leitura por parte dos sensores de força e torque. É válido destacar que os núcleos de processamento ARM[3] correspondem a parte dessas tecnologias mais utilizadas no quesito. Microcontroladores de 32 bits com essa estrutura são bastante vistos em projetos atuais e abrem portas para uma ampla gama de possibilidades de integração de novas propriedades aos sensores [21].

Uma vez que os dados foram tratados, outra preocupação é para onde transmiti-los e de que modo. A comunicação dos circuitos integrados com o microcontrolador é na maioria dos casos feita por meio de protocolos de comunicação serial, como I^2C[4], SPI[5] e CAN[6]. Onde cada um é utilizado depende da aplicação, já que todos possuem vantagens e desvantagens, sendo necessária uma comparação mais minuciosa, mostrada em [22]. Assim, é válido considerar que, na construção de um sensor com o objetivo de versatilidade no seu uso, sejam implementados os três protocolos mencionados. Também se faz necessária a presença do protocolo USB[7] não só para a transferência de dados, mas também pensando na entrada de alimentação do circuito, já que se trata de um projeto acessível.

Por fim, resta a interface com o usuário. As soluções industriais existentes no mercado disponibilizam, além de manuais orientando a utilização, também os firmwares e softwares usados nos produtos e suas atualizações. Por possuir uma característica comercial, os

[3]Advanced RISC Machine
[4]Inter-Integrated Circuit
[5]Serial Peripheral Interface
[6]Controller Area Network
[7]Universal Serial Bus

programas costumam focar em serem *user friendly*. É algo a que se deve atentar, para que o projeto abranja um número maior de usuários por requerer um conhecimento técnico menor.

Analisando projetos acadêmicos voltados para o tema de aquisição de forças e torques, vê-se grande variedade de ideias para a obtenção dos resultados. Um deles, com o objetivo de implementar um controle multivariável para motores de indução, usa um estimador que se baseia no erro referente à velocidade do motor para obter o valor do torque, de modo a não se ter sensores no sistema, tornando o processo não invasivo [23].

O estimador inclui uma ação integral no controle, aumentando a robustez. No trabalho, os resultados obtidos se aproximam bastante em comparação aos obtidos com uso de sensores, de modo que o projeto é uma alternativa a ser considerada dependendo da aplicação em que se deseja ler torques.

Outro projeto propõe a criação de um sistema virtual de instrumentação para determinar automaticamente a curva tensão-deformação de um teste de tração [24]. É dividido em três módulos: dos sensores, com a célula de carga e os extensômetros; de aquisição de dados, com a eletrônica *Compact FieldPoint*, da National Instruments [25]; e por fim o de software, que utiliza algoritmos relacionados com cálculos e apresentação de dados, no programa LabVIEW, também da National Instruments.

A comparação do sistema virtual com os resultados de um equipamento analógico comercial foi feita determinando a curva tensão-deformação através de testes de tração. Os resultados foram próximos, com erro máximo de 4,6%.

Na temática *low-cost*, há bons trabalhos que merecem ser citados, como [26], que elaborou um hardware a partir da plataforma Arduino para a aquisição de dados provenientes de extensômetros presos a lâminas metálicas e vigas de concreto armado, associando a um software para *smartphones*, que se comunica com o Arduino por *Bluetooth*.

A análise dos resultados foi realizada comparando os valores analíticos dos experimentos, os obtidos por meio de um aquisitor comercial e os obtidos utilizando o projeto. O erro relativo aos valores analíticos foi aceitável para a leitura de deformações e forças, mas não para a leitura de deslocamentos. Este erro percentual foi um pouco maior que o do aquisitor comercial, mas ainda assim foi próximo. A maioria das leituras de deformações e forças se distanciou das leituras do aquisitor em torno de 1%.

Outro trabalho, voltado à área médica, foca em facilitar o acesso do público a esse tipo de solução [27], criando um sensor de força de baixo custo, *open-source*, compatível

com sinais de ressonância magnética. Possui quatro células de carga com *strain gauges* e realiza o tratamento dos dados e a comunicação com cabo Ethernet, Arduino e um módulo ADC[8] HX711, de modo que estas ferramentas são acessíveis por não serem caras.

O trabalho trouxe resultados satisfatórios considerando o contexto: o foco em redução de custo diminui a precisão; além disso, o ambiente onde se pretende ler o sinal é cheio de altas frequências e interferências.

Por fim, focando ainda em viabilidade econômica, há outro projeto [28] que ousa na complexidade ao mesmo tempo que utiliza impressão 3D e Arduino: um sensor de forças e torques de 6 eixos que consiga lê-los em formas complexas, sendo composto por duas peças de PLA[9] com oito pernas, formando uma Plataforma de Stewart, com cada perna servindo como uma viga *cantilever* e permitindo uma leitura de deslocamento por meio de sensores optoeletrônicos.

O máximo erro detectado para cada eixo varia em torno de 15 a 20%. A discrepância é alta, e os autores acreditam que uma das causas principais seja o material da célula (PLA). Ainda assim, dependendo da aplicação, o projeto pode ser útil para estimar valores em ocasiões onde seja difícil de se utilizar uma solução comercial.

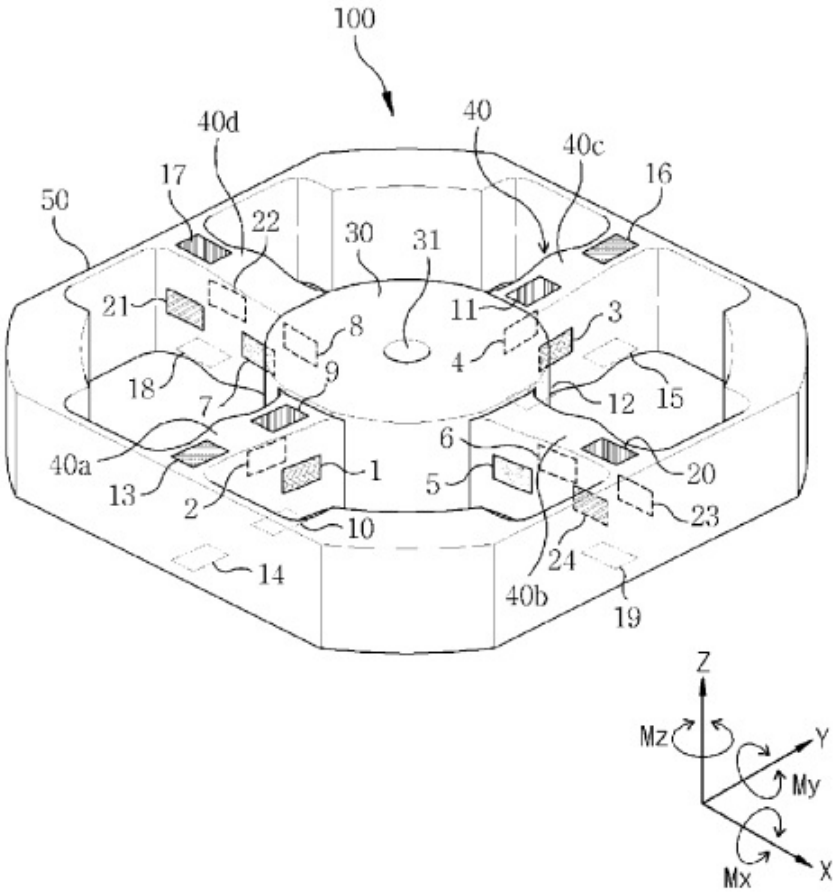
Esses projetos procuram inovar na forma como descobrem os torques e/ou forças. A maioria possui em comum o pensamento de que as soluções comerciais são consideradas caras para aplicações não industriais, de modo a ter como uma de suas características o baixo custo. Ainda, também se destaca a dificuldade em adquirir o sinal por meio da incorporação de uma célula de carga no sistema, por conta das diferentes formas que este pode ter, ou da inviabilidade de serem realizadas alterações para adicionar algum elemento extra a esse sistema. Todas são preocupações válidas, mas há ainda a necessidade de garantir a robustez do hardware. O projeto desenvolvido nesta monografia, portanto, tem como um dos objetivos focar no desenvolvimento de um hardware robusto que possa ser utilizado por uma grande variedade de aplicações, pensando também em baixo custo e voltado para o meio acadêmico.

No campo das patentes, foi encontrada uma célula de carga [29] que parece encaixar bem na ideia do projeto que aqui será descrito, e pode ser vista na figura 5. A patente em si engloba o formato da célula, o funcionamento dela e o método de leitura das deformações. O sensor possui um furo no centro para que seja acoplado um eixo, e do centro para a borda da célula há 4 vigas. Após uma pré-carga, é realizada a medição da deformação

[8] Analog to Digital Converter
[9] Poliácido Láctico

dessas vigas a partir do momento em que é aplicado um torque na célula proveniente do eixo acoplado. Em cada uma das vigas estão posicionados *strain gauges* para realizar a leitura da deformação nos eixos X, Y e Z. Assim, é possível realizar leituras de um a até todos os seis eixos. As vigas possuem uma forma de gradiente para que se tente reduzir ou até mesmo anular a diferença da leitura de deformação referente à posição do *strain gauge*. Se a viga é reta ao longo do seu comprimento, ocorrerá uma concentração das tensões próximo ao centro da célula, de modo que a deformação será maior nessa região. Nessa situação, o local onde o *strain gauge* seria colocado teria que ser calculado e perfeitamente posicionado.

Figura 5: Desenho em perspectiva da célula de carga patenteada.



Fonte: Retirado de [29].

2 OBJETIVOS GERAIS

Medir forças e torques em baixa latência e com precisão em aplicações robóticas variadas.

2.1 Objetivos Principais

- Melhorar laboratórios e aulas da Poli com hardware funcional;
- Projetar circuito de condicionamento de sinal de extensômetros;
- Produzir PCI[1] protótipo com os circuitos projetados.

2.2 Extensões

- Definir método/rotina de calibração do sensor;
- Projetar experimentos de validação do projeto;
- Elaborar manual de uso do sensor;
- Projetar e fabricar célula de carga para medição de forças e torques;
- Implementar interface em alto nível para usuários.

[1] Placa de Circuito Impresso

3 APLICAÇÕES PREVISTAS

Com base nos objetivos levantados e na ideia do projeto, faz-se necessário pesquisar mais sobre as áreas onde se pretende aplicá-lo, entrevistando Professores e alunos que usufruiriam dele. Assim, foi criada uma lista com disciplinas, Laboratórios e até mesmo grupos de extensão onde o projeto poderia ser aplicado, sendo levantados requisitos e analisados os pontos em comum às aplicações.

3.1 Disciplina: PMR3408 - Instrumentação

Victor Pacheco Bartholomeu

- Necessidade de medição de torque: Realização de experimentos didáticos para familiarização dos estudantes com tecnologias de medição. No oferecimento da disciplina são formados pequenos grupos de alunos que compartilham bancadas experimentais e revezam as atividades. Sendo assim, cada grupo poderia se valer do dispositivo de aquisição para propósitos específicos de acordo com o estudo em andamento.

  Alguns dos testes feitos na disciplina também incluem atuação, necessitando de outras placas eletrônicas que controlem os atuadores. Assim, é interessante que o projeto possa interagir com essas interfaces de atuação, de modo a simplificar as montagens experimentais. Ainda, seria proveitosa a função de interface com sensores diversos (com acesso por SPI ou I^2C, por exemplo) e um computador, via USB.

  Finalmente, é importante que o dispositivo não demande hardware ou software muito sofisticado, que torne seu uso proibitivo com computadores mais simples, tais quais os alunos têm acesso;

- Faixa de força / torque: Médias, de modo que serão produzidas com força manual dos alunos. 10 N ou 1 N.m;

- Precisão: 1% , desde que haja atenuação de ruídos;

- Frequência de aquisição: 10 Hz;

- Interface: Computadores, via USB ou com Arduinos, via I^2C;
- Versatilidade: O dispositivo deve se adequar aos diversos experimentos e à futuros projetos da disciplina;
- Célula de carga: Preparadas pelos alunos durante as aulas do curso;
- Custo esperado: Valores até R$500,00 são aceitáveis para os dispositivos;
- Solução atual: Módulos comerciais ou placas de aquisição de alto custo.

3.2 Disciplina: PME3516 - Evolução Tecnológica e Histórica dos Sistemas Termo Fluido Mecânicos

Professor Maurício Silva Ferreira

- Necessidade de medição de torque: Instrumentação de modelos em escala reduzida, com interface para Arduinos utilizados pelos alunos, na construção de protótipos.

  Devido a natureza da disciplina, proteção contra respingos de água é um atrativo, assim como a existência de interface para *encoders* rotacionais (sensores de velocidade).

  É desejável que o dispositivo seja versátil, de modo a continuar funcional em oferecimentos posteriores da disciplina. Um projeto de hardware e software aberto devem facilitar sua customização.
- Faixa de força / torque: Torques muito baixos – aproximadamente 0.04 N.m;
- Precisão: Não crítica – 5 % de erro relativo ao medido;
- Frequência de aquisição: Não crítica – 10Hz;
- Interface: Arduino / PC;
- Versatilidade: Uso na instrumentação de modelos em escala reduzida. Preferencialmente com software e hardware aberto e bem documentado;
- Célula de carga: Tração, para interface com freios de prony e similares;
- Custo esperado: R$200,00 incluindo célula de carga – Tomando como referência sensores para Arduino;

- Solução atual: Não existe.

Recursos disponíveis para o desenvolvimento do projeto: fresadora CNC[1] do laboratório LETE.

3.3 Disciplina: PMR3404 e PMR3409 - Controle I e Controle II

Professor Eduardo Aoun Tannuri

- Necessidade de medição de torque: Desenvolvimento de plataforma didática para uso nas aulas de laboratório das disciplinas, seja por meio de modificações na plataforma atual, ou pela criação de um novo sistema.

  É interessante que o dispositivo seja capaz de realizar a aquisição de dados de torque e rotação e também prover a saída do sinal de controle (*driver*) para o motor;

- Faixa de força / torque: 0.2 N.m;
- Precisão: 1%;
- Frequência de aquisição: 1 KHz;
- Interface: Placas micro controladas (Arduino) e computador, com softwares como Matlab;
- Versatilidade: Adequação à solução atual e a uma possível nova plataforma;
- Célula de carga: Na fixação do motor, de modo a medir o torque reativo;
- Custo esperado: R$ 300;
- Solução atual: Módulo de controle de motor DC (LJ Create 207-15).

3.4 Disciplina: PMR3411 - Projeto de Máquinas

Professor Gilberto Francisco Martha de Souza

- Necessidade de medição de torque: Caracterização e levantamento de curva de motores utilizados nas máquinas construídas na disciplina.

[1] Comando numérico computadorizado

- Faixa de força / torque: 5 N.m, sendo que esse valor é maior que o produzido pelos motores e, portanto, garantem margem para os teste;
- Precisão: 1%;
- Frequência de aquisição: 10Hz;
- Interface: USB, para um software que exiba os gráficos de torque;
- Versatilidade: Todos os motores medidos tem características similares;
- Célula de carga: A ser desenvolvida pela disciplina;
- Custo esperado: Até R$ 250,00;
- Solução atual: Estimação com o uso de um disco de inércia, mas sem obtenção de valores absolutos.

3.5 Laboratório de Biomecatrônica

Victor Pacheco Bartholomeu

- Necessidade de medição de torque: Controle de exoesqueletos e aparelhos para fisioterapia;
- Faixa de força / torque: Torques de até 40Nm; trações de até 1000N;
- Precisão: Alta - 0.1 %;
- Frequência de aquisição: 100Hz;
- Interface: CAN;
- Versatilidade: Uso em motores com rotação limitada;
- Célula de carga: Medição de torque reativo; medição de força em cabos Bowden;
- Custo esperado: R$300,00, excluindo célula de carga;
- Solução atual: Extensômetros colocados diretamente no mecanismo;

Recursos disponíveis para o desenvolvimento do projeto: componentes e ferramentas eletrônicos.

3.6 Laboratório de Engenharia Térmica e Ambiental (LETE)

Professor Guenther Carlos Krieger Filho

- Necessidade de medição de torque: Instrumentação de bancada de motor a vapor, utilizada na disciplina de Termodinâmica. Adequação ao sistema de dinamômetro mecânico da bancada e interface de leitura de termopares. É desejável incluir sensores de velocidade para o motor;
- Faixa de força / torque: 5N;
- Precisão: 5%;
- Frequência de aquisição: 10 Hz;
- Interface: Que possibilite registro das medições pelos alunos: Computador;
- Versatilidade: Instalação única na bancada em questão;
- Célula de carga: Localizada entre os dinamômetros e estrutura, não devem substituir o sistema analógico;
- Custo esperado: R$200;
- Solução atual: Dinamômetros analógicos, com freio de corda. Para temperaturas, é necessário conectar um multímetro nos terminais de cada termopar.

3.7 Grupo de Extensão: Equipe ThundeRatz de Robótica

Gustavo Oliveira Barranova

- Necessidade de medição de torque: Caracterização de motores, principalmente *brushless* de corrente contínua.
- Faixa de força / torque: Torques baixos para motores pequenos (mais frequente) – 0.2 N.m

  Torques altos para motores grandes - 10 N.m;
- Precisão: 2 % de erro relativo ao medido;

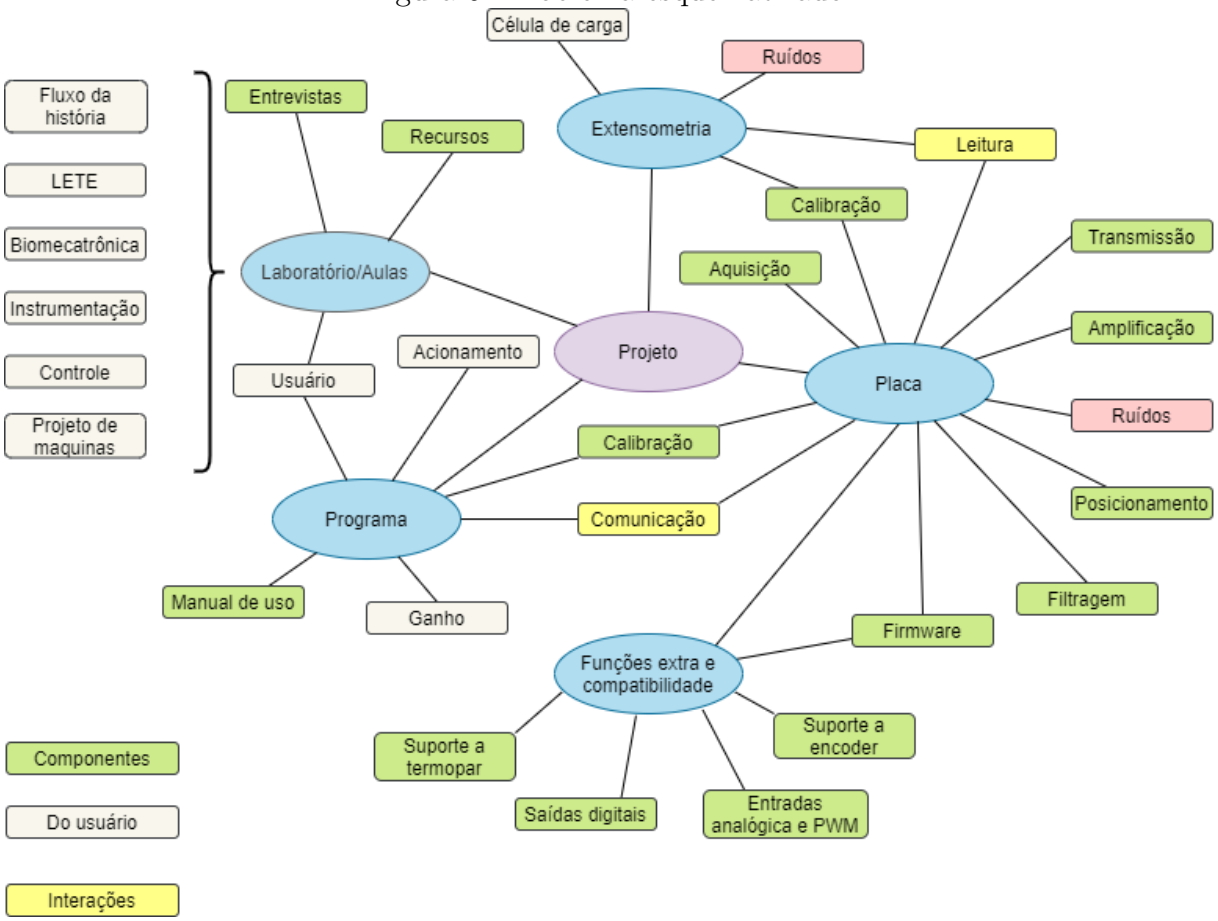
- Frequência de aquisição: Não crítica – 10Hz;
- Interface: Pc;
- Versatilidade: Uso com motores de diversos tamanhos;
- Célula de carga: Para medição de torque reativo - fixação no estator do motor;
- Custo esperado: R$150,00, excluindo célula de carga ;
- Solução atual: Não existe;

Recursos disponíveis para o desenvolvimento do projeto: ferramentas e equipamentos da oficina da equipe - Ferros de solda, estação de retrabalho, fonte de tensão, osciloscópio, ferramentas de usinagem e furadeira de bancada.

4 PROJETO BÁSICO

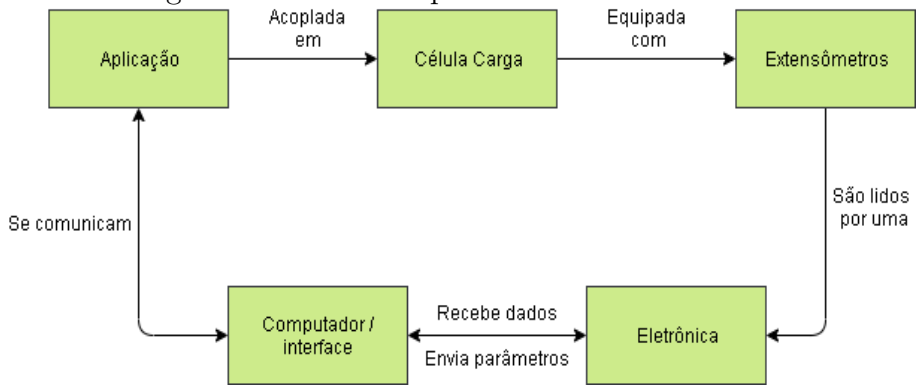
A pesquisa e reflexão sobre o Projeto Básico iniciou com uma esquematização do problema, mostrado na figura 6. Para entendimento claro do problema e levantamento de todos os requisitos, o projeto foi dividido em subsistemas (figura 7) e então foi realizada uma análise mecatrônica sobre cada um deles, ou seja, foram destacados os pontos relacionados com Mecânica, Computação, Elétrica e Controle de cada subsistema.

Figura 6: Problema esquematizado.



Fonte: elaborado pelos autores.

Figura 7: Divisão do problema em subsistemas.



Fonte: elaborado pelos autores.

4.1 Análise Mecatrônica

4.1.1 Subsistema: Célula de carga

4.1.1.1 Mecânica

Construída sob os princípios da resistência dos materiais, as células de carga devem ter comportamento de deformação muito bem definido, de modo que se possa traduzir um esforço mecânico em uma escala de deformação linear. Vigoram princípios de escolha de materiais, análises numéricas de resistência estática e dinâmica, projeto de mecanismos (com função de integrar o elemento de medição aos sistemas mecânicos) e manufatura de precisão.

4.1.1.2 Computação

Não se aplica.

4.1.1.3 Elétrica

Não se aplica.

4.1.1.4 Controle

Não se aplica.

4.1.2 Subsistema: Extensômetros

4.1.2.1 Mecânica

Enquanto colado em um material operando em regime elástico, o extensômetro é sujeito às mesmas deformações do corpo. Nesse caso, uma célula de carga. Na colagem, deve-se garantir a adesão correta no componente. Além disso, é importante garantir que a deformação no extensômetro não ultrapasse seus limites de resistência.

4.1.2.2 Computação

Não se aplica.

4.1.2.3 Elétrica

De acordo com a segunda Lei de Ohm, com a deformação da grade do extensômetro, sua resistência varia. Embora muito pequena, essa variação pode ser medida com um arranjo de ponte de Wheatstone e um circuito compatível.

4.1.2.4 Controle

Não se aplica.

4.1.3 Subsistema: Eletrônica

4.1.3.1 Mecânica

De modo a cumprir os requisitos do projeto, é importante prever a necessidade de encapsulamento da placa eletrônica. De modo a garantir a proteção dos circuitos, quanto a interferências e/ou respingos de água, é necessário definir métricas de teste para esse subsistema.

4.1.3.2 Computação

Para utilização do hardware, é vital a boa implementação do firmware do projeto. Com o suporte do estudo em microcontroladores, deve existir um grande esforço na elaboração dos programas que permitem a aquisição, processamento e transmissão dos dados, pelas diversas interfaces que serão implementadas.

4.1.3.3 Elétrica

Tomando importantes conceitos de eletrônica analógica e digital, nesse subsistema destaca-se o desenvolvimento de uma placa de circuito impresso contendo sistemas de amplificação e filtragem de sinais, conversão analógica-digital e transmissão de informações por diversas interfaces de comunicação.

4.1.3.4 Controle

Não se aplica.

4.1.4 Subsistema: Computador/Interface

4.1.4.1 Mecânica

Não se aplica.

4.1.4.2 Computação

Deve-se preparar o computador/interface onde será conectado o dispositivo. O projeto exige programação de software e criação de modelos em Matlab compatíveis com o hardware.

4.1.4.3 Elétrica

Não se aplica.

4.1.4.4 Controle

Não se aplica.

4.1.5 Subsistema: Aplicação

4.1.5.1 Mecânica

Como os sistemas onde o projeto será aplicado já estão prontos, é necessário pensar na forma de acoplamento do sensor, de modo a se obter fácil adaptação a cada caso. Por exemplo, algumas aplicações envolvem ler o torque em um sistema com rotação contínua, enquanto em outras a rotação é limitada, podendo a fixação ser realizada de outra maneira.

4.1.5.2 Computação

A programação do sensor deve ser realizada pensando na possibilidade de integração com outros dispositivos.

4.1.5.3 Elétrica

Algumas aplicações podem exigir a interação do sensor com outros sensores, como um *Encoder*, por exemplo. Desta forma, é necessário pensar na conexão do projeto com outros dispositivos, e em como eles se comunicarão, com que protocolo e se haverá algum tipo de interferência e/ou incompatibilidade entre eles.

4.1.5.4 Controle

Há aplicações que trabalharão também com o controle do sistema, e não só a leitura. Assim, o sensor deve ser capaz de transmitir os dados em baixa latência e levar em consideração fatores que permitam o funcionamento correto de um controlador.

4.2 Requisitos de cada subsistema

Com base na divisão dos subsistemas e nas entrevistas com responsáveis dos locais onde o projeto seria usado, foi criada uma lista de requisitos que devem ser atendidos para o bom funcionamento nas aplicações desejadas.

4.2.1 Célula de Carga

- Para torques de até 40Nm;
- Para forças de até 1000N;
- Facilidade e versatilidade no acoplamento;
- Material com tensão de escoamento maior ou igual à do alumínio (por volta de 300MPa);
- Formato que permita a colagem de quatro extensômetros;

4.2.2 Extensômetros

- Utilização de cabos blindados para comunicação com a placa, para diminuir interferência;
- Proteção contra água;
- Extensômetros que aguentem uma deformação maior que a deformação elástica do material onde serão colados;
- Extensômetros devem ser pequenos (menor que 10mm);

4.2.3 Eletrônica

- Frequência de aquisição do sinal de pelo menos 100Hz;
- Uso das seguintes interfaces de comunicação de baixa latência: I^2C, USART[1], USB e CAN;
- Conexão para *Encoder*;
- Circuito de amplificação com ganho variável;
- Circuito de filtragem de sinais que não modifique o ganho do sistema;
- Variação da tensão na Ponte de Wheatstone de no máximo 20mV;
- Auto balanceamento da Ponte de Wheatstone e possibilidade de controle de *offset*;
- Custo de componentes menor que U$50,00;
- Precisão do sinal de 1% da máxima deformação;
- Proteção contra respingos de água;
- hardware e software abertos;

4.2.4 Computador e Interface

- Configurações da placa ajustáveis via software;
- Manual de uso do programa para usuários com conhecimento técnico;
- Opção de implementar Controle ou não;

[1]Universal Asynchronous Receiver/Transmitter

4.3 Definições de cada subsistema

4.3.1 Célula de Carga

4.3.1.1 Entrada

Força e/ou torque provenientes do sistema onde está acoplada.

4.3.1.2 Saída

Deformação resultante dessa força e/ou torque.

4.3.1.3 Possíveis Soluções

Células de carga industriais são ofertadas por empresas como Honeywell, HBM e Omega, de forma que aplicações comerciais sejam satisfeitas, incluindo fatores além da medição, como compatibilidade química, resistência a sobrecarga e em uma grande gama de formatos. De modo geral, no entanto, não existem muitas opções para aplicações com espaço limitado, uma vez que nesse caso é adequado projetar o corpo de medição segundo os requisitos da aplicação.

Também é usual realizar a instrumentação em elos estruturais já existentes na aplicação, de modo que não ocorra interferência em sistemas já operantes.

4.3.1.4 Ponderação

Com as aplicações esperadas, é prevista a necessidade de células de carga bastante específicas, pois as faixas de medição são muito diferentes e na maioria dos casos, a planta onde o dispositivo será utilizado não aceita grandes alterações mecânicas para a inclusão de uma peça de padrão industrial.

Assim, prevê-se o projeto de células de carga com geometrias específicas e materiais apropriados às cargas em cada caso. É prudente então, que seja escolhido um caso de uso para que na primeira iteração do projeto, sirva de plataforma de testes do dispositivo.

4.3.1.5 Solução Escolhida

Suprindo a necessidade inicial do Laboratório de Biomecatrônica, deve-se projetar e fabricar uma célula de carga de torção, compatível com torques de 20 N.m. A peça em

questão será acoplada a mecanismos do laboratório e baseará a definição dos processos de calibração e configuração do dispositivo. A execução do projeto deve envolver o levantamento do formato da célula de carga (de acordo com a instalação), dimensionamento da secção de medida, simulação por elementos finitos e fabricação das peças necessárias.

4.3.2 Extensômetros

4.3.2.1 Entrada

Deformação da célula de carga.

4.3.2.2 Saída

Variação no valor da resistência dos extensômetros.

4.3.2.3 Possíveis Soluções

Uma vez analisados os requisitos que os extensômetros devem obedecer, foram pesquisadas alternativas a serem compradas. Como versões mais simples já satisfazem o que é pedido para o projeto, são muitas as opções aceitáveis. Assim, optou-se por focar no preço e na facilidade de acesso ao produto como fatores mais importantes. Alguns destaques foram os das empresas HBM, Micro-Measurements (Vishay) e Excel Sensores.

O extensômetro escolhido deveria ser linear (mede apenas uma direção), com compensação de temperatura e menos suscetível a interferências.

4.3.2.4 Ponderação

Já que o objetivo do projeto é ser acessível, o preço do sensor é muito relevante. Comparando os sensores das três empresas citadas, observou-se que os da Vishay eram mais caros. Os da HBM variavam bastante de acordo com o tipo. Entre ela e a Excel Sensores, os valores eram similares, mas a Excel se destacou por ser mais próxima, além do grupo já ter utilizado seus extensômetros.

4.3.2.5 Solução Escolhida

Foram escolhidos os extensômetros da Excel Sensores, pelo preço, proximidade da empresa e familiaridade do grupo com seus produtos. Trata-se de um extensômetro linear,

com comprimento total de 8mm e resistência de 350Ω .

4.3.3 Eletrônica

4.3.3.1 Entrada

Variação no valor de tensão na saída da Ponte de Wheatstone dos extensômetros.

4.3.3.2 Saída

Valor da variável desejada (i.e: força, torque), com o protocolo exigido para a aplicação.

4.3.3.3 Possíveis Soluções

As soluções envolvendo a eletrônica dividem-se em seis partes: alimentação, aquisição, filtragem, amplificação, interpretação e interface.

A alimentação da placa pode ser pensada em duas alternativas: fixa, conectada a um carregador, ou através do uso de baterias. Ambas as opções possuem prós e contras e exigem cuidados específicos. Para o primeiro caso, é preciso pensar no tipo de carregador e qual a entrada na placa. Para o segundo, a escolha da solução envolve o estudo de baterias para verificar o tipo que se adequa melhor. Também é necessário verificar, para qualquer dos casos, a tensão e a corrente que entrarão no sistema e a necessidade de regulação da tensão para determinados componentes da placa.

A parte de aquisição envolve pensar na forma com que o sinal dos extensômetros serão lidos. O mais comum é a utilização de uma Ponte de Wheatstone, apesar de haverem outras soluções, como por exemplo a Ponte de Chevron. Dentro da solução da Ponte de Wheatstone, ainda existe a possibilidade de usá-la como 1/4, meia ponte ou ponte completa.

Outro fator importante para o correto funcionamento da Eletrônica é a filtragem do sinal. Ela pode ser feita analogicamente, digitalmente ou por uma mistura das duas. A forma digital é feita por meio de códigos, enquanto a analógica geralmente utiliza circuitos com amplificadores operacionais, capacitores e resistores. Além dessas formas, também deve-se escolher o tipo de filtro e qual a frequência de corte dele.

Quanto à parte de amplificação, existem diversos circuitos padronizados de acordo com sua função. Ela é usualmente feita com o uso de Amplificadores Operacionais. Deve-se

prestar atenção ao ganho que será utilizado. Um ganho grande torna mais fácil trabalhar com o sinal adquirido, mas também amplia os ruídos.

A interpretação do sinal é feita digitalmente, através de um microcontrolador. Para isto, primeiramente é necessário transformar o sinal analógico em digital. Assim, demanda-se a escolha de um conversor ADC e de um microcontrolador que seja suficiente para o projeto.

Por fim, a parte da interface refere-se a como o sinal adquirido e interpretado se comunicará com as aplicações previstas anteriormente. Trata-se dos protocolos que serão utilizados e da conexão com as aplicações, de forma ainda a manter a placa barata, versátil e eficiente.

4.3.3.4 Ponderação

A alimentação por bateria permite maior mobilidade do sistema por não depender de tomadas. Contudo, este não é um fator muito importante já que as aplicações do sistema são fixas. A alimentação utilizando um carregador, por outro lado, é vantajosa por eliminar o problema de ter que recarregar a bateria, além da preocupação com sua vida útil. Se decidido o uso de carregador, é interessante usar os que sejam mais comuns, como os de celular. Pensando assim, os tipos de entrada mais comuns para a placa variam entre Micro-USB, Lightning e USB-C. O Micro-USB é utilizado na maioria dos celulares Android, enquanto o Lightning é utilizado nos celulares iOS. Contudo, o USB-C tem crescido bastante e apresenta muitas melhorias com relação aos outros dois, sendo provável que ele substitua os dois em um futuro próximo.

No que diz respeito à aquisição, é muito comum o uso de uma Ponte de Wheatstone. Esta pode ser utilizada com apenas um (1/4 de ponte), dois (meia ponte) ou quatro extensômetros (ponte completa). A última configuração apresenta vantagens com relação às outras duas, como por exemplo uma melhor compensação da variação dos valores de resistência devido à variação de temperatura e uma maior sensibilidade à deformação.

Na filtragem do sinal, tanto os filtros analógicos quanto os digitais possuem vantagens e desvantagens, portanto deve-se ponderar qual deles se adequa melhor ao projeto. Os filtros digitais são flexíveis, podendo ser alterados muito mais facilmente. Também são mais baratos. Porém, introduzem erros provenientes da digitalização do sinal e requerem uma frequência de aquisição. Os analógicos são mais rápidos e não precisam do uso de microcontroladores e nem de códigos, além de não precisarem do sinal discretizado. Em compensação, introduzem erros provenientes da mudança de temperatura dos componen-

tes eletrônicos e da precisão dos valores destes componentes.

Já na amplificação, o tipo de circuito depende de alguns fatores, como a região de frequência de trabalho, a tolerância a ruídos, o ganho, a estabilidade do sinal de saída, entre outros. Para o projeto em questão, é importante que o sinal seja amplificado com a menor quantidade de ruídos possível e com ganho suficiente para que se consiga trabalhar com o sinal sem maiores problemas. Uma solução que atende aos requisitos é o uso de Amplificadores de Instrumentação.

Quanto à interpretação do sinal, existem microcontroladores que possuem ADCs inclusos suficientes para o que o projeto exige. Os mais utilizados atualmente são os da família da ST. Como o projeto pretende se comunicar com diversos protocolos, é interessante que o microcontrolador consiga se comunicar com o maior número possível deles. Alguns modelos atendem a todos os requisitos exigidos.

Por fim, quanto à interface, é necessário que hajam saídas para todos os protocolos já mencionados. Uma placa só que possua todas elas é interessante por permitir a fácil modificação na comunicação com a aplicação caso seja necessário. Por outro lado, para aplicações que exijam só uma saída em um protocolo específico, o projeto é encarecido desnecessariamente. Dessa forma, outra abordagem possível é a criação de uma placa base que possua a função de aquisição do sinal e sua interpretação, como foi explicado nos parágrafos anteriores, mas com *shields* diferentes para cada protocolo e função pedidos. Assim, cada aplicação usaria o *shield* que quisesse.

4.3.3.5 Solução Escolhida

Como as aplicações são fixas e se deseja que o projeto seja de uso simples pelo usuário, além de mais barato, foi escolhido utilizar um carregador para a alimentação. A saída escolhida é a Micro USB, por suas vantagens como a disponibilidade de cabos e dispositivos compatíveis. Essa porta também será utilizada na comunicação USB com computadores.

A aquisição será feita pelo uso de uma Ponte de Wheatstone completa, para que o sinal seja mais preciso e a temperatura influencie menos. O balanceamento da ponte será feito eletronicamente com o uso de um conversor digital/analógico.

Na entrada, será usado um filtro analógico ativo, ou seja, com Amplificadores Operacionais, junto de capacitores e resistores. Além das vantagens já ditas, o filtro analógico auxilia a reduzir problemas como distorções no sinal por *aliasing*. Ao mesmo tempo, como se deseja precisão na resposta, o tipo de filtro ideal para o caso é o Butterworth, por ter

uma resposta em frequência o mais plana possível.

Na amplificação, o amplificador de instrumentação será usado. Trata-se de um amplificador diferencial com um *buffer* em cada entrada. Ele deve possuir uma impedância maior de entrada, alto CMR[2] e baixa tensão de *offset*. Sendo assim, ele traz mais precisão para o circuito. O ganho a princípio é fixo, mas existe a alternativa de adicionar um sistema para alteração desse valor.

Para a interpretação do sinal, será usado um microcontrolador com arquitetura ARM, por este normalmente possuir todas as funções necessárias para o projeto, além de sua popularidade.

Por fim, para a interface com as aplicações, acredita-se ser mais vantajoso particionar a mesma placa, de modo que cada sistema seja facilmente isolado dos demais e, com isso, atividades de manutenção sejam facilitadas.

4.3.4 Computador e Interface

4.3.4.1 Entrada

Variáveis calculadas pela Eletrônica e originadas da Aplicação.

4.3.4.2 Saída

Dados recalculados enviados para a Eletrônica e novos parâmetros para controle da Aplicação.

4.3.4.3 Possíveis Soluções

A interface entre a Eletrônica e a Aplicação acaba por ser variável, dependendo da aplicação a ser tratada. Há aplicações onde o interesse é puramente a leitura do valor da força e/ou torque, enquanto em outras é interessante que haja o seu controle. Ainda, para cada situação, será usado um protocolo diferente. Dessa forma, o programa a ser criado para interação com o usuário deve permitir as opções de utilização de diversas funções, com a possibilidade de ligá-las e desligá-las. Como algumas das aplicações são previstas para serem utilizadas em sala de aula, é válido ponderar a construção de um programa *open source*, onde o aluno possa programar parte das funções, ou se seria melhor construir uma interface mais alto nível com um manual de utilização.

[2]Common-Mode Rejection

4.3.4.4 Ponderação

Ainda que a ideia de *open source* seja interessante para auxiliar no ensino, pode acabar por causar problemas como o mau funcionamento do projeto. Como nenhuma das aulas teria como foco a programação em si, mas sim o aprendizado do funcionamento do projeto, uma interface alto nível chama mais a atenção, além de também ser mais simples para o uso nos laboratórios.

4.3.4.5 Solução Escolhida

Pensando na simplicidade de utilização pelos Laboratórios e em sala de aula, a criação de um programa com interface em alto nível acaba por se destacar mais. Nesta interface, há a possibilidade de alternar entre os módulos disponíveis e as opções de leitura.

4.3.5 Aplicação

4.3.5.1 Entrada

Parâmetros vindos do Computador/Interface.

4.3.5.2 Saída

Forças e/ou torques gerados na célula de carga.

4.3.5.3 Ponderação

Aqui cada aplicação já possui a sua solução, de modo que cabe ao projeto apenas integrar-se a elas.

5 PROJETO DETALHADO

Considerando o uso do projeto por muitos usuários, decidiu-se por escolher um nome para ele que fosse fácil de ser lembrado: PANDA[1].

5.1 Projeto do Hardware

5.1.1 Extensômetros em ponte de Wheatstone

Tendo em mente o projeto da porção analógica do circuito, parte-se das especificações dos extensômetros comerciais. Avaliando catálogos da HBM [30], Excel [31] e Omega [32], nota-se a prevalência de sensores com resistência nominal de 120Ω e 350Ω, ambos com *gauge factor* aproximado de 2. Feitas escolhas quanto às dimensões, ligas, material base e padrão do gage, de acordo com os requisitos citados anteriormente, é adequado optar pela maior resistência [33], de modo a minimizar aumentos de temperatura que interferem no sensor.

É natural a necessidade de condutores elétricos entre o ponto de instalação dos extensômetros e o dispositivo. Nesse caso deve-se utilizar cabos adequados aos ambientes de aplicação, com consciência da presença de potenciais fontes de interferência para a conexão.

Enquanto arranjos mais simples, com pares trançados de cabos, são comuns na transmissão de sinais digitais, sua efetividade para sinais analógicos não é suficiente nas situações de maior incidência de ruído [34]. O emprego de blindagem com folha (*foil*) visa proteção contra campos elétricos, na forma de acoplamento capacitivo, já o envolvimento dos condutores em tranças de cobre é normalmente efetivo na presença de fontes de baixa frequência(associadas ao chaveamento de cargas). A associação dessas três medidas é obtida, dentre outras opções, com o uso de cabos manga: Comuns em equipamentos de automação e informática, são fabricados com diferentes números de vias. Um modelo desse cabo é exemplificado na figura 8.

[1]Placa de Aquisição e Condicionamento de Dados

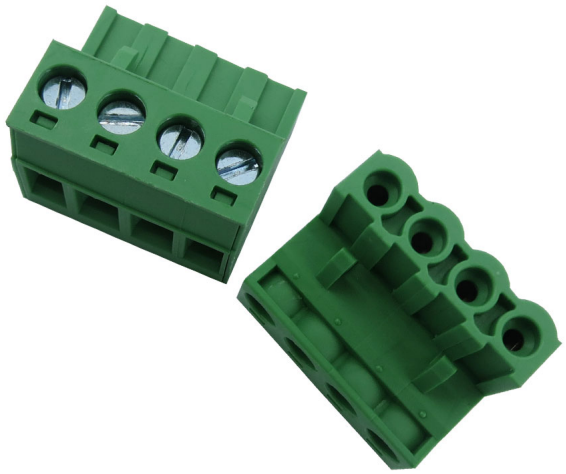
Figura 8: Cabo manga, como fornecido por fabricantes nacionais.

Fonte: extraído de [35].

Com fornecimento nacional já é possível cumprir as sugestões quanto a construção do cabo, e obter valores que podem ser usados no refinamento do projeto.

Antes de concluir o esquema de entrada do sinal no sistema de medição (PCI), vale realizar a escolha dos conectores que permitirão a montagem e desmontagem de todo o conjunto. Seguindo os requisitos de projeto e retomando a proposta didática do dispositivo, deve-se atribuir maior peso à flexibilidade do dispositivo ao interagir com o ambiente e conectar-se às interfaces de saída. Sendo assim, faz-se cômodo o uso de conectores tipo borne (terminal com parafuso de pressão), que não exigem crimpagem ou soldagem dos cabos e ainda facilitam a instalação e remoção dos componentes do sistema de aquisição. A figura 9 apresenta o componente descrito.

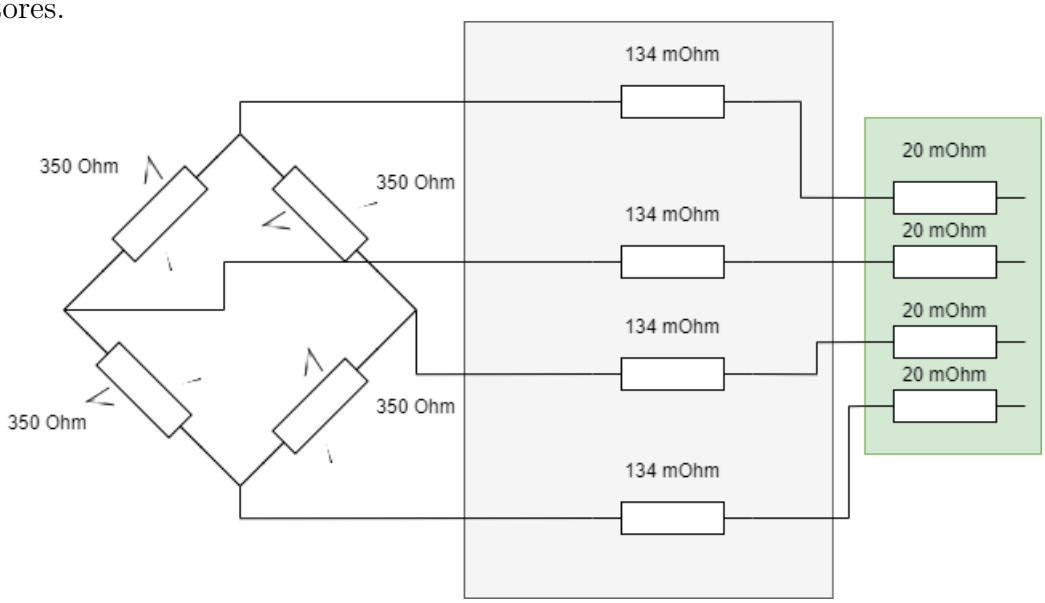
Figura 9: Conector Borne 2EDGK-4vias.



Fonte: extraído de [36].

Também com fornecimento nacional, é possível contar com suas características técnicas na elaboração do modelo apresentado na figura 10 que sintetiza a entrada dos sinais analógicos no dispositivo.

Figura 10: Diagrama esquemático da ponte de Wheatstone e resistências dos cabos e conectores.



Fonte: elaborado pelos autores.

Com tais valores, segue o cálculo da corrente elétrica que deverá alimentar a ponte de Wheatstone composta com os extensômetros (tabela 1). O valor é calculado para

Tabela 1: Levantamento da corrente elétrica na ponte de Wheatstone, para diferentes tensões de referência.

Tensão (V)	Corrente (mA)	Observações
3.3	9.4	Valor frequente em circuitos integrados modernos, visando redução de potência
4.096	11.7	Valor amigável aos cálculos com conversão 12-bit, permitindo divisões de 1mV
5	14.3	Valor de tensão popularizado com as tecnologias TTL[2]CMOS[3]

Fonte: elaborado pelos autores.

diferentes tensões, uma vez que esse valor ainda será definido.

Valores mais elevados contribuiriam com o aquecimento do sensor, ao fazer circular correntes demasiadamente altas.

5.1.2 Filtro

Como primeiro item da cadeia de condicionamento do sinal da ponte de Wheatstone, é adequado atenuar frequências indesejadas, provenientes do ambiente de instalação do dispositivo. Tomando a máxima taxa de aquisição do sistema, de acordo com os requisitos, em 1kHz, é proveitoso filtrar frequências maiores que esse valor. Assim, espera-se limpar o sinal de saída da ponte de extensômetros, em canais independentes e antes da amplificação.

As fontes de ruído consideradas até então geram majoritariamente interferência eletromagnética (EMI) e podem ser divididas entre dispositivos que intencionalmente emitem sinais (telefones, roteadores sem-fio. etc.) e as que o fazem acidentalmente (equipamentos eletrônicos em geral) [37].

Em laboratórios, por exemplo, vale destacar a presença de diversos computadores, luzes fluorescentes e motores-normalmente chaveados em altas frequências. Tal característica (altas tensões oscilando com frequências altas) favorecem o acoplamento indutivo [4]. Esse fenômeno pode gerar tensões da ordem de milivolts no conjunto de extensômetro e cabos do dispositivo. Uma vez que os sinais de entrada são dessa mesma ordem de grandeza, é essencial tomar medidas de proteção.

Exposta então a necessidade de atenuar os sinais de entrada no circuito amplificador, busca-se o tipo adequado de filtro. Enquanto o filtro passa baixa ideal elimina todas as componentes de frequência maior que a frequência de corte e não atenua as demais, as

[2]Transistor-Transistor Logic
[3]Complementary Metal-Oxide Semiconductor

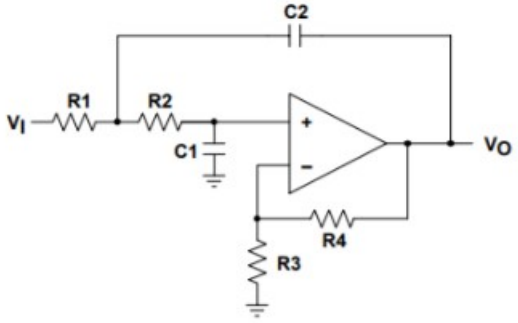
implementações reais têm limitações de distinguem os circuitos em: adaptado de [38]:
Butterworth: Otimizado para o ganho unitário na faixa de passagem e atenuação constante de -20dB/dec;
Bessel: Otimizado para resposta de fase linear em detrimento estabilidade da banda de passagem;
Chebyshev: Apresenta *ripple* na banda de passagem, entretanto, responde de modo mais acentuado à frequências maiores que a de corte.

Confrontando essas opções, é claro optar pela consistência da banda de passagem, uma vez que a introdução de ganhos diferentes nos sinais de entrada é catastrófica para o funcionamento do circuito.

Tomando a frequência alvo da captação e sabendo das possíveis interferências de maior relevância (Chaveamento de grandes cargas, por exemplo, normalmente em taxas maiores que 10kHz), a atenuação de -20dB/dec/ordem demanda o filtro de segunda ordem para reduzir à 1% a amplitude de ruídos de 10KHz.

O filtro Butterworth pode ser constituído de um circuito integrado específico para esse propósito (TLC14ID [39] da Texas Instruments ou MAX291 [40] da Maxim Integrated por exemplo) ou montado com um amplificador operacional e componentes passivos. A primeira abordagem tende a eliminar erros construtivos da PCI e contribui com a redução do número de componentes, entretanto, tais benefícios não justificam o maior custo, frente à simplicidade do circuito implementado com a arquitetura Sallen-Key [41] ilustrada na figura 11.

Figura 11: Esquemático da arquitetura Sallen-Key.



Fonte: extraído de [41].

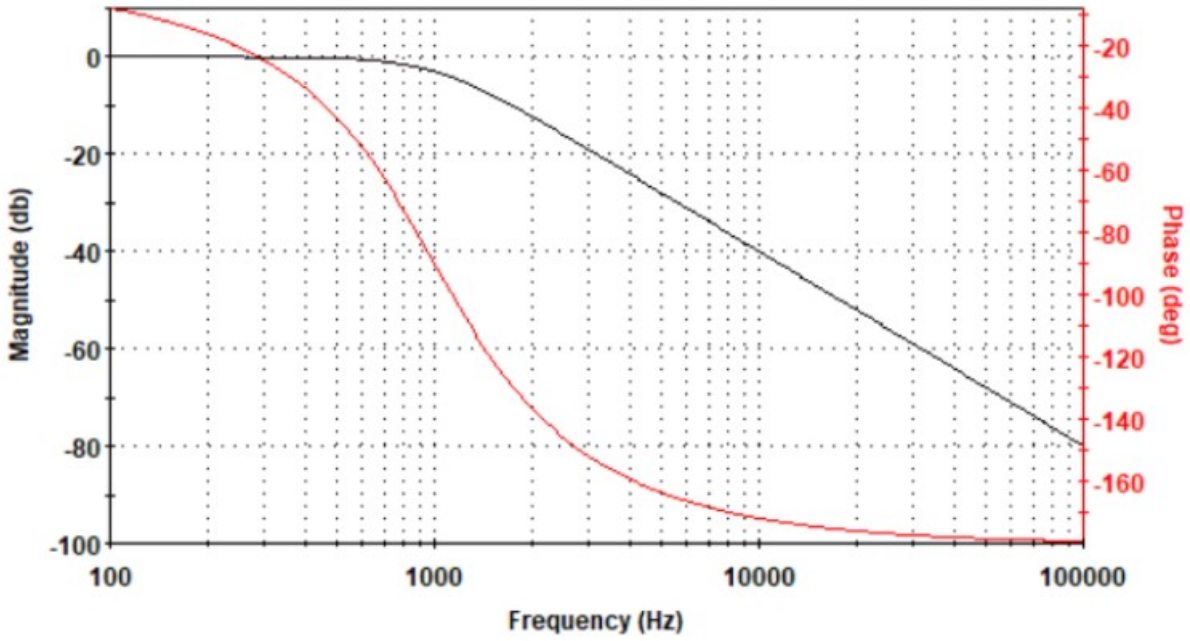
Eliminando os resistores R3 e R4 e a ligação com o GND[4] obtém-se o filtro Butterworth de segunda ordem desejado.

[4]Ground

Uma forma eficiente de obter os valores dos componentes passivos é utilizar um software comercial, que ao levar em conta a disponibilidade de capacitores e resistores do mercado, evita combinações que, embora cumpram os requisitos de performance, não podem ser fabricadas.

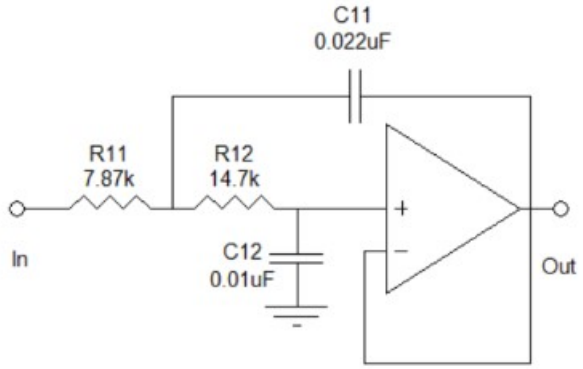
A tarefa realizada com o FilterLab, disponibilizado pela Microchip [42], ao selecionar-se o tipo de filtro, ordem e frequência de corte. São apresentados o diagrama de bode do circuito(figura 12) e seu desenho esquemático de circuito (figura 13).

Figura 12: Diagrama de bode associado ao filtro Butterworth de segunda ordem sugerido.



Fonte: extraído do Filterlab.

Figura 13: Circuito esquemático do filtro Butterworth de segunda ordem sugerido.



Fonte: extraído do Filterlab.

Ainda, utilizando a equação da frequência de corte e definindo a faixa tolerável de variação desse valor, é possível restringir a tolerância dos componentes. Impondo o limite de erro em 2% para a frequência de corte e assumindo que todos os componentes terão a mesma precisão, calcula-se:

$$\omega_{corte} = \sqrt{\frac{1}{R_1 R_2 C_1 C_1}}$$

e, no caso com frequência defasada em 2%:

$$98\%\omega_{corte} = \sqrt{\frac{1}{R_{1max} R_{2max} C_{1max} C_{1max}}}$$

com $R_{nmax} = R_n \times (1 + err)$ e $C_{nmax} = C_n \times (1 + err)$

Portanto:

$$98\% = \sqrt{\frac{1}{(1 + err)^4}}$$

$$err = 0.01$$

Esse resultado coincide com o padrão utilizado em componentes passivos SMD[5], eliminando a necessidade de componentes de precisão, cujos preços são consideravelmente mais altos.

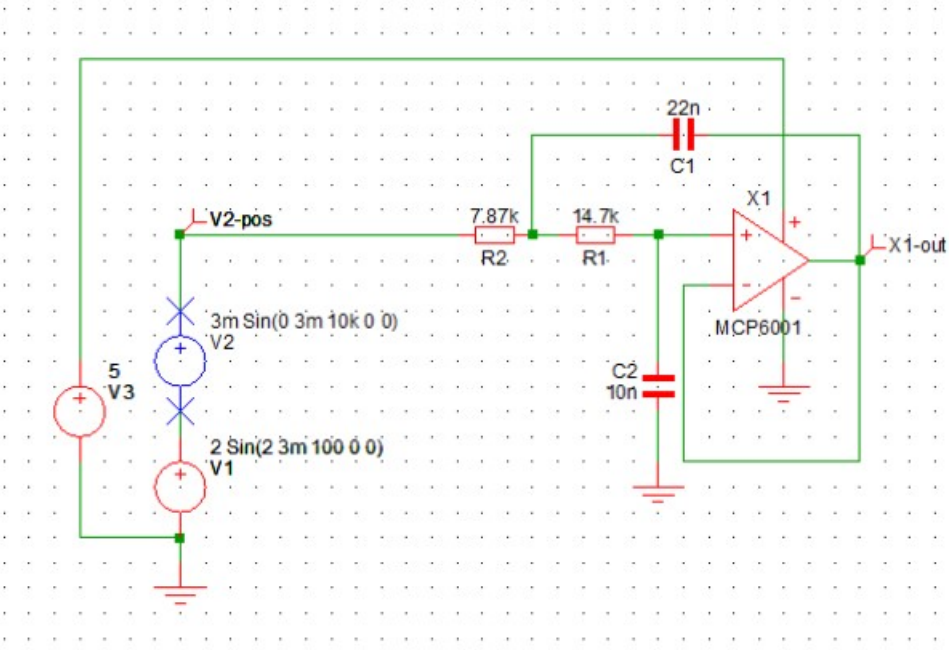
Finalmente, deve-se selecionar o amplificador operacional para o circuito. Uma vez que as tensões de entrada serão em torno de metade da tensão de referência da ponte de Wheatstone, não é crítica a faixa de operação linear do amplificador (a faixa central de tensões é naturalmente linear). O parâmetro de escolha seguinte é a largura de banda do amplificador (*bandwidth*), que deve superar 100 vezes a frequência de corte do filtro pretendido [43]. No caso em discussão, esse valor deve ser, no mínimo 100KHz. Ainda, vale observar a *bias current*, garantindo que seu valor não seja significativo quando comparado à corrente da ponte.

Com os requisitos relaxados, o CI[6] MCP6002 [44] da Microchip foi cogitado para o projeto. Encapsulando dois amplificadores independentes, o componente deve facilitar a confecção da PCI enquanto mantém boa performance a preço compatível com o dispositivo.

[5]Surface Mount Display
[6]Circuito Integrado

Outro software da Microchip se faz útil nesse momento de escolha. O MPLab Mindi [45] realiza simulações de circuitos analógicos com base em modelos bastante sofisticados dos componentes da própria fabricante. Sendo assim, todo o circuito de filtragem pode ser descrito e testado virtualmente (figura 14).

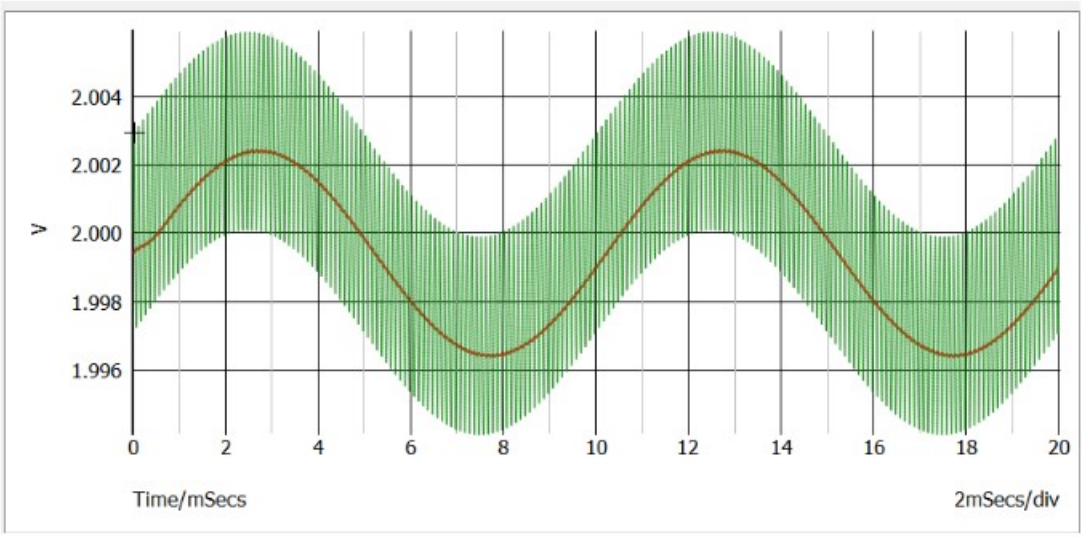
Figura 14: Circuito do filtro modelado para simulação.



Fonte: extraído do MPLab Mindi.

Foram conectadas em série duas fontes de tensão, uma cujo valor varia entre 1.997V e 2.003V senoidalmente com frequência de 100Hz e outra, com variação entre -0.003V e 0.003V na frequência de 10kHz. Dessa maneira,simula-se um sinal de interesse (100 Hz) e um ruído de alta frequência. O sinal de entrada no circuito é medido no ponto V2-pos e é representado em verde no gráfico (figura 15), enquanto a saída filtrada, obtida em X1-out é a curva vermelha. Observa-se claramente a rejeição da componente de alta frequência no sinal. É notável também a existência de um deslocamento horizontal da curva de saída, com valor de 0.6mV. Tal defasagem foi mantida ao refazer a simulação com outras taxas de variação e outros valores de tensão.

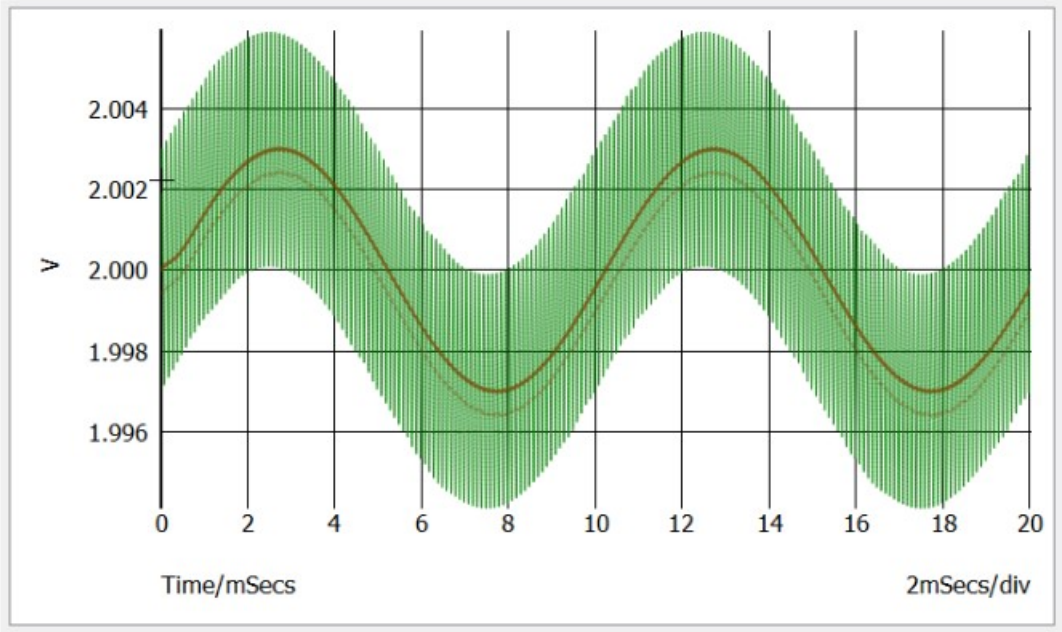
Figura 15: Análise dos sinais na entrada e saída do circuito de filtragem com o MCP6002.



Fonte: extraído do MPLab Mindi.

Avaliando o *datasheet* do amplificador operacional, vem que a *input offset voltage* do componente é de até 4.5 mV, ou seja, abrange o valor obtido. Como confirmação de que a defasagem encontrada é fruto do CI escolhido, a simulação é atualizada com o MCP6V01 [46], cujo o módulo do parâmetro em discussão é de 2. O resultado, em comparação com o valor anterior é visível na figura 16, comprovando a hipótese levantada, ao praticamente eliminar o erro na saída. O custo desse comportamento mais próximo ao ideal para o componente é refletido em seu preço, que é aproximadamente 20 vezes mais alto que o praticado para a opção original.

Figura 16: Análise dos sinais na entrada e saída do circuito de filtragem com o MCP6v02.



Fonte: extraído do MPLab Mindi.

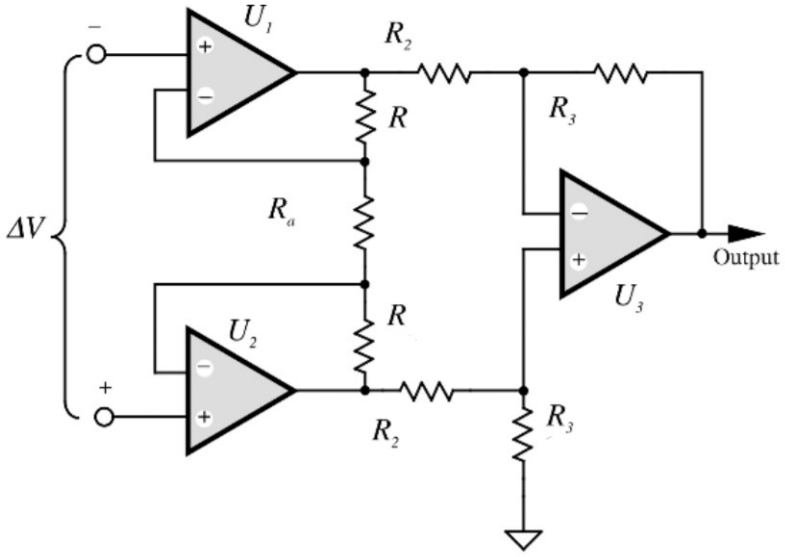
Avalia-se a influência dessa defasagem de tensão tendo em mente a amplificação do sinal e supondo que o ganho do sistema deve ser configurado em torno de 100 vezes. É necessário retomar que a precisão desejada para o dispositivo (1%) é obtida com a distribuição da faixa de medição em 100 patamares, que, para os valores cogitados como tensão de referência, abrangem até 50mV. Impondo que, caso não houvesse correção de *offset* para o circuito, o erro advindo do fenômeno avaliado acima deve ser mantido no limite da precisão desejada, define-se que ao dobrar o *input offset voltage* (devido a característica da amplificação diferencial dos sinais sujeitos ao erro) e aplicar o ganho, não se exceda o limiar de 50mV. Com tal métrica somada às condições postas anteriormente, é possível encontrar o CI MCP6022-E [47], também da Microchip, cujo parâmetro em análise é tipicamente menor que 250. É importante notar que, ainda que seja admitido esse erro, o mecanismo de correção de *offset* do amplificador deverá anular tal valor, assim como eventuais desequilíbrios não intencionais da ponte de Wheatstone.

5.1.3 Amplificador

Assim como o filtro, é possível construir o amplificador de instrumentação com amplificadores operacionais ou optar por um CI específico com essa função. A escolha por montar o circuito como na figura 17, incorre na necessidade de componentes ativos de precisão uma vez que, erros como o discutido na seção anterior não podem ser compensa-

dos, por afetar de forma imprevisível a diferenciação das tensões de entrada. A tolerância para os valores dos componentes passivos também deve ser minimizada para garantia do ganho linear e simetria na amplificação.

Figura 17: Amplificador de instrumentação com três Amplificadores Operacionais. Valores iguais de resistências atribuídos segundo a simetria do circuito.



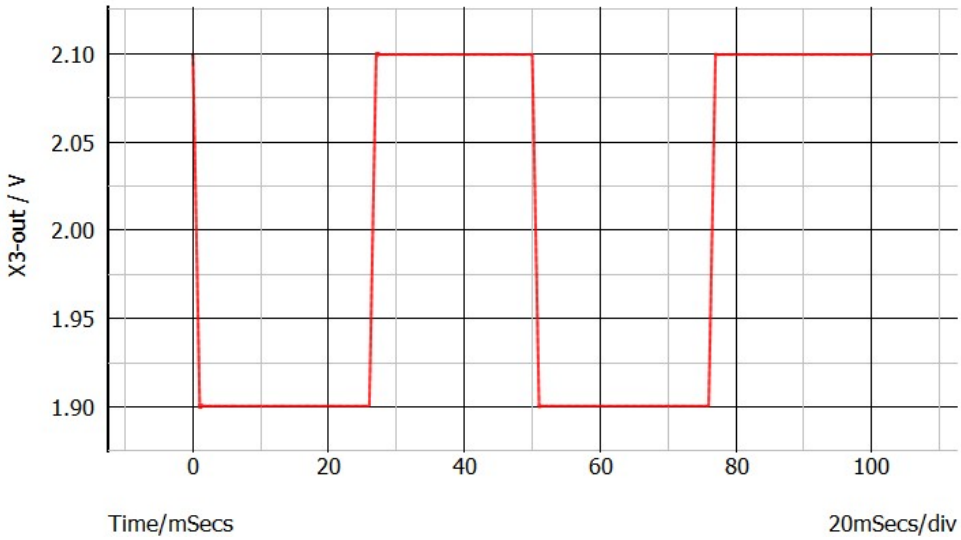
Fonte: Adaptado de [18].

Uma possível implementação desse circuito é desenvolvida no software de simulação, com o CI MCP6V02 citado anteriormente. Os valores das resistências foram determinados a partir da equação:

$$G = 1 + \frac{2R}{R_G}$$

de modo a amplificar uma onda quadrada de -1mV a 1mV em 100 vezes. É importante notar que, pensando na implementação real do circuito, utiliza-se uma fonte simples de tensão (de 0V à +4V). Portanto, a saída do circuito, apresentada na figura 18 oscila em torno de 2V, devido ao *offset* introduzido.

Figura 18: Saída do amplificador com ganho 100 para uma onda quadrada de amplitude 1mV.



Fonte: extraído do MPLab Mindi.

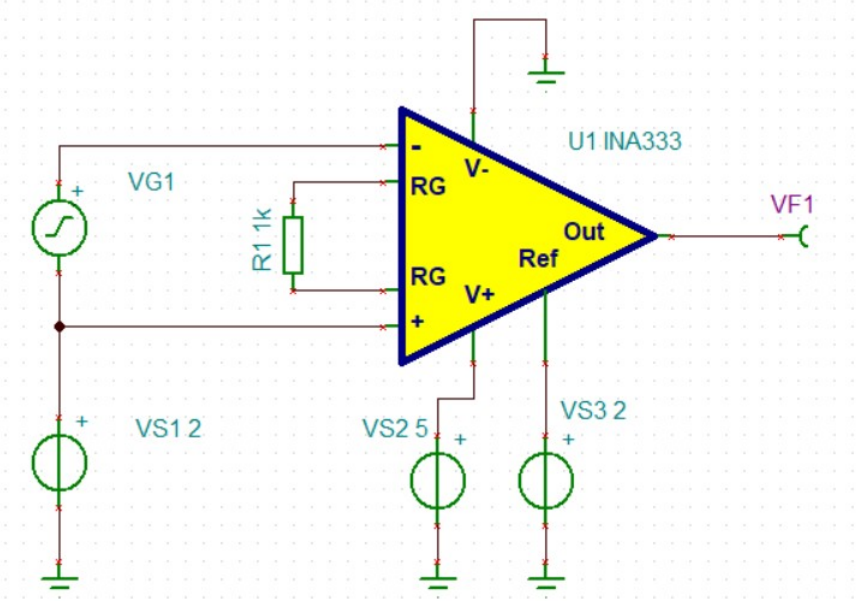
No simulador não é possível observar todo o erro introduzido com o uso de componentes menos precisos. Isso ocorre por conta da modelagem dos CIs, que possuem *input offset voltage* iguais e que se cancelam antes da amplificação. Esse fenômeno não é realista, pois o valor de *offset* segue uma distribuição estatística decorrente dos processos de fabricação [48] e por isso o valor máximo deve ser considerado. Nesse caso, utilizando o MCP6002, por exemplo, deveria ser considerado um erro de 9mV antes da amplificação. Seriam indispensáveis componentes mais próximos do MCP6V02, com o qual o erro seria limitado à 4 . Assim, mesmo com ganhos de 1000 vezes, o sistema se manteria funcional. Também vale destacar a dificuldade associada ao roteamento dos sinais e a posição dos componentes na PCI - A natureza do circuito não permite grandes distâncias entre os componentes [49], como seria adequado para manutenção do equipamento.

Considerando agora CIs amplificadores com a arquitetura apresentada (amplificadores monolíticos), deve ser destacado o ganho de performance em função da consolidação de todos os componentes do circuito no mesmo *die*. Essa característica, apoiada pelas técnicas correntes de fabricação, permite melhor associação dos amplificadores operacionais e resistores do circuito, maximizando a rejeição de modo comum. O fato desses CIs englobarem praticamente todo o circuito também favorece o projeto ao permitir sua extensa caracterização e a listagem dos parâmetros de funcionamento pelo próprio fabricante. Além disso, atualmente existe uma grande variedade desse tipo de amplificador e embora eles tenham surgido com a necessidade de melhor performance, a corrida tecnológica trouxe ao mercado dispositivos cada vez mais baratos.

Projetando os parâmetros discutidos na seleção de Amplificadores operacionais sobre os catálogos das grandes fabricantes de semicondutores (Microchip, Texas Instruments, Maxim Integrated e Analog Devices) um bom compromisso entre preço e requisitos é atingido pelo INA333, da Texas Instruments [50]. Em geral, os requisitos são facilmente atendidos após filtrar as opções por áreas de uso e eliminar componentes especiais, nos quais um parâmetro é melhorado em detrimento de outros (Operação com alta tensão, por exemplo).

Assim como a Microchip, a Texas Instruments disponibiliza um simulador que possui seus componentes modelados. O TINA-TI [51] permite experimentação do componente cogitado, como exibido na figura 19.

Figura 19: Circuito esquemático com o INA333, implementado no TINA.



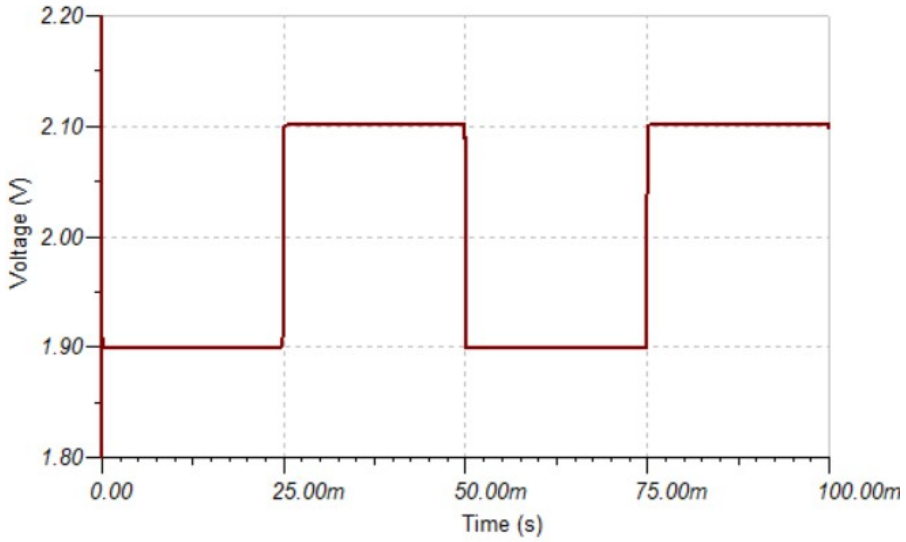
Fonte: extraído do TINA-TI.

O ganho do amplificador é calculado com:

$$G = 1 + \frac{100k\Omega}{R_G}$$

Assim, com um resistor de ganho de 1Ke aplicando na entrada uma onda quadrada de amplitude 1mV em torno de 2V, obtém-se a saída esperada amplificada e exibida na figura 20.

Figura 20: Saída do amplificador com ganho 101 para uma onda quadrada de amplitude 1mV.



Fonte: extraído do TINA-TI.

Com a possibilidade de ajustar o ganho simplesmente variando o valor de R_G, uma alternativa para cumprimento do requisito de ganho variável é a adição de um potenciômetro digital, que configurado como reostato pode ser acionado pelo microcontrolador para diversos valores de ganho [52].

Com um potenciômetro de 10K, é possível cobrir todas as faixas de ganho normalmente utilizadas com células de carga comerciais (com sensibilidade entre 1mV/V e 3mV/V). A escolha por um modelo específico leva em conta a quantidade de *taps*, associada aos níveis de saída. Também existem opções quanto à interface de controle, com destaque para I^2C e SPI.

Sendo assim, o MCP41010 [53] da microchip se faz uma opção interessante ao possuir 256 passos e interface SPI. Nessa configuração, o ganho máximo possível seria em torno de 1000 vezes, enquanto o mínimo, 11 vezes.

Nessa configuração, para todo o modulo amplificar, além dos circuitos integrados, bastam alguns capacitores, com função de *decoupling*.

5.1.4 Microcontrolador

Com o intuito de reduzir o número de componentes ativos e com isso, diminuir a complexidade da placa, a escolha de microcontrolador (MCU) deve levar em conta as interfaces de comunicação pretendidas para o dispositivo: USB, CAN, RS485 e I2C. Se-

Tabela 2: Periféricos necessários no Microcontrolador.

Requisito	Valor
Canais ADC	≥ 3
Canais DAC[7]	≥ 1
Interface CAN	≥ 1
Interface I2C *Slave*	≥ 1
Interface SPI *Master*	≥ 1
USART	≥ 1
Interface USB	≥ 1
Porta I/O	≥ 4
Porta I/O (interrupções)	≥ 2

Fonte: elaborado pelos autores.

guindo a arquitetura apresentada anteriormente, o conversor A/D utilizado com o circuito de amplificação também será contido no microcontrolador e dever atender aos requisitos de resolução e velocidade de conversão listados. O *offset* do amplificador será ajustado com a tensão aplicada por um conversor D/A e, portanto, esse periférico deve concordar com a resolução do ADC. Deverão estar disponíveis portas digitais para uso geral, interface SPI para comunicação com o potenciômetro de ajuste do ganho e serviços de interrupção para leitura de *encoders* . Além disso, é desejável a presença de alguma memória não volátil que possa armazenar configurações do dispositivo. Uma relação de periféricos necessários é apresentada na tabela 2

Com esses parâmetros, foram filtrados catálogos das principais fabricantes de MCUs resultando em componentes das famílias Kinetis® K (NXP) [54] e STM32F0 (ST Microeletronics) [55]. Ambos são baseados na arquitetura ARM-Cortex, de 32 bits. Enquanto a primeira é da linha *mid-end* da fabricante, a segunda apresenta modelos de entrada(*mainstream*). Considerando os diversos modelos das duas famílias, a opção ideal é o STM32F072x8/xB, uma vez que possui todos os periféricos necessários, permite implementação extremamente simplificada e tem custo menor que as outras opções disponíveis. Destaca-se a possibilidade de uso da interface USB sem necessidade de oscilador externo ao MCU, alimentação analógica independente(evita ruídos digitais das interfaces de comunicação sobre os módulos analógicos), memória flash e certa variedade de encapsulamentos(com quantidades diferentes de pinos, aumentando o leque de escolhas) [56].

Analisando a tabela 3 e as opções listadas na tabela 1, opta-se por utilizar 3.3V como tensão de alimentação e referência para o circuito analógico.

[7]Digital to Analog Converter

Tabela 3: Levantamento de faixas tensão para os componentes selecionados.

Componente	Limite[V]
MCU I/O (VDD)	≤ 3.6
MCU Analógico (VDDA)	$\geq VDD; \leq 3.6$
Amplificador	$\geq 1.8; \leq 5.5$
Filtro	$\geq 2.5; \leq 5.5$
Potenciômetro	$\geq 2.7; \leq 5.5$

Fonte: elaborado pelos autores.

Tabela 4: Corrente de operação para os componentes selecionados.

Componente	Corrente(max)[mA]
MCU I/O	120
Amplificador	55
Filtro	30
Potenciômetro	0.5
Driver CAN	48
Driver RS485	11
Total	264.5

Fonte: elaborado pelos autores.

5.1.5 Drivers de comunicação

A integração das interfaces CAN e RS485 exigem circuitos que convertem sinais lógicos entre 0 e 3.3V para as tensões diferenciais que definem os protocolos citados. Devido a especificidade desse tipo de componente, a escolha é feita aplicando restrições simples (tensão de operação, por exemplo) e de forma a minimizar custos e componentes adicionais. Sendo assim, o SN65HVD230DR [57] da Texas Instruments foi escolhido para a CAN e o SP3485 [58] da Exar Corporation. Ambos apresentam o circuito recomendado para compatibilidade do dispositivo com as redes de comunicação.

5.1.6 Regulador de tensão

Sabendo da alimentação geral do dispositivo em 5V, devido ao uso do USB para tal função, a tensão regulada em 3.3V será fornecida por um regulador linear. O componente escolhido deverá prover corrente suficiente (A estimativa de corrente é calculada na tabela 4) e possuir *dropout voltage* baixa o suficiente para o circuito.

A determinação da máxima *dropout voltage* V_{do} leva em conta a tensão de alimentação V_{in}, a tensão regulada V_{reg} e a queda de tensão no diodo de proteção V_{dd}:

$V_{do} < V_{in} - V_{dd} - V_{reg}$

O valor de V_{dd} é tomado considerando o diodo Schottky SS34 [59], e superdimensionando a corrente como 1A. Nesse caso, $V_{dd} = 0.4V$ e tem-se:

$V_{do} < 5 - 0.4 - 3.3$; $V_{do} < 1.3V$

Com esses requisitos e assumindo que o regulador está sujeito a defeitos devidos ao mau uso da placa, a escolha do componente é feita com base no custo e na simplificação do circuito associado à regulação. O LM1117 [60], embora relativamente antigo, é muito comum e configura uma opção viável. Uma opção mais moderna é o TLV1117 [61], que com o mesmo encapsulamento, é uma substituição direta ao modelo mais velho.

5.1.7 Referência de tensão

Responsável pela alimentação analógica do MCU (que incluí seus módulos ADC e DAC) e provedor da tensão de referência da ponte de Wheatstone e possíveis equipamentos nas entradas livres, um circuito de referência é dimensionado com os requisitos de precisão e corrente. Para operação do MCU o valor da tensão deve $VDDA$, tal que $VDD \leq VDDA \leq 3.6V$. Dos valores elencados na tabela 1, toma-se $VDDA = 3.3V$ e, portanto, $I_{bridge} = 9.4mA$. Além disso, a corrente de alimentação analógica do MCU é estimada em $IDDA = 0.2mA$. Nesse cenário, seria possível utilizar um CI referência de tensão, como o REF3033 [62], que lida, por exemplo, com variações de temperatura. Esse tipo de componente deve ser empregado quando é necessário um valor absoluto de tensão. Para o dispositivo em questão, uma vez que a tensão de excitação da ponte é a mesma da referência dos módulos ADC e DAC, deve-se avaliar a possibilidade de utilizar diretamente a tensão regulada nos circuitos analógicos. Ainda que seja necessário adicionar mais um regulador para separação dos lados digital e analógico, esse componente é mais barato e mais resiliente. Além disso, de acordo com a caracterização do CI TLV740 [63], correntes baixas reduzem o erro em relação a tensão de saída. Dessa maneira é possível esperar estabilidade nos circuitos. Vale também ressaltar que a maior disponibilidade de corrente em relação ao CI referência de tensão, será possível utilizar pontes de Wheatstone com resistência equivalente menor. Com isso, células de carga de 120Ω (valor comum nos catálogos de fabricantes) poderão ser conectadas diretamente ao dispositivo.

5.2 Projeto do Software

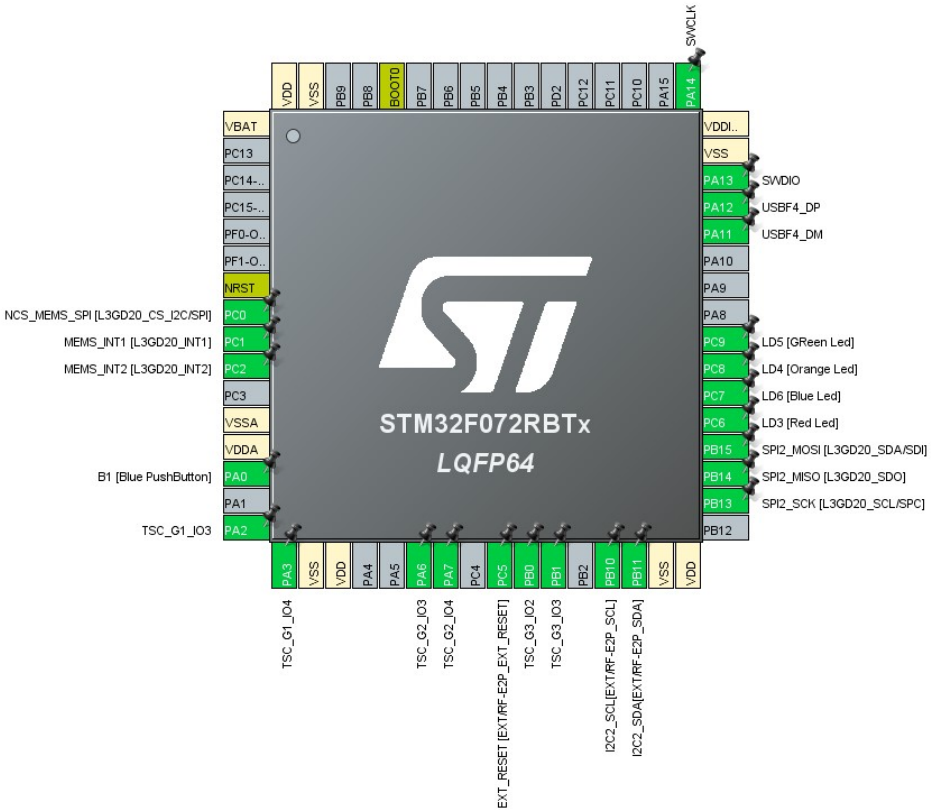
5.2.1 Preparação do ambiente de programação do MCU

Os microcontroladores STM32 são programados com auxilio de ferramentas simples que, no caso desse trabalho, foram instaladas em computadores com Windows. Enquanto não se tem acesso ao dispositivo descrito nas seções anteriores, a programação de firmware será ensaiada em uma placa de desenvolvimento munida de MCU muito similar ao selecionado para o projeto. A 32F072BDISCOVERY [64] (figura 21) possui o STM32F072RBT6, com pinos disponíveis para o usuário e já inclui o circuito de gravação ST-Link integrado. A atribuição de funções aos pinos do MCU é feita de forma bastante intuitiva com o software STM32Cube [65]. Configurada para o hardware da placa Discovery, a interface é mostrada na figura 22.

Figura 21: Kit de desenvolvimento 32F072BDISCOVERY.

Fonte: extraído de [64].

Figura 22: STM32Cube configurado para a placa Discovery.



Fonte: extraído do STM32Cube.

A partir da configuração feita no software é possível gerar os arquivos base para o dispositivo, incluindo a biblioteca HAL (*Hardware Abstraction Layer*) que facilita o acesso aos periféricos e funções do MCU. Esses arquivos podem ser compilados com a *GNU Toolchain* [66] e o arquivo binário gerado nesse processo, é gravado do MCU com o STM32CubeProgrammer [67] através do módulo ST-link.

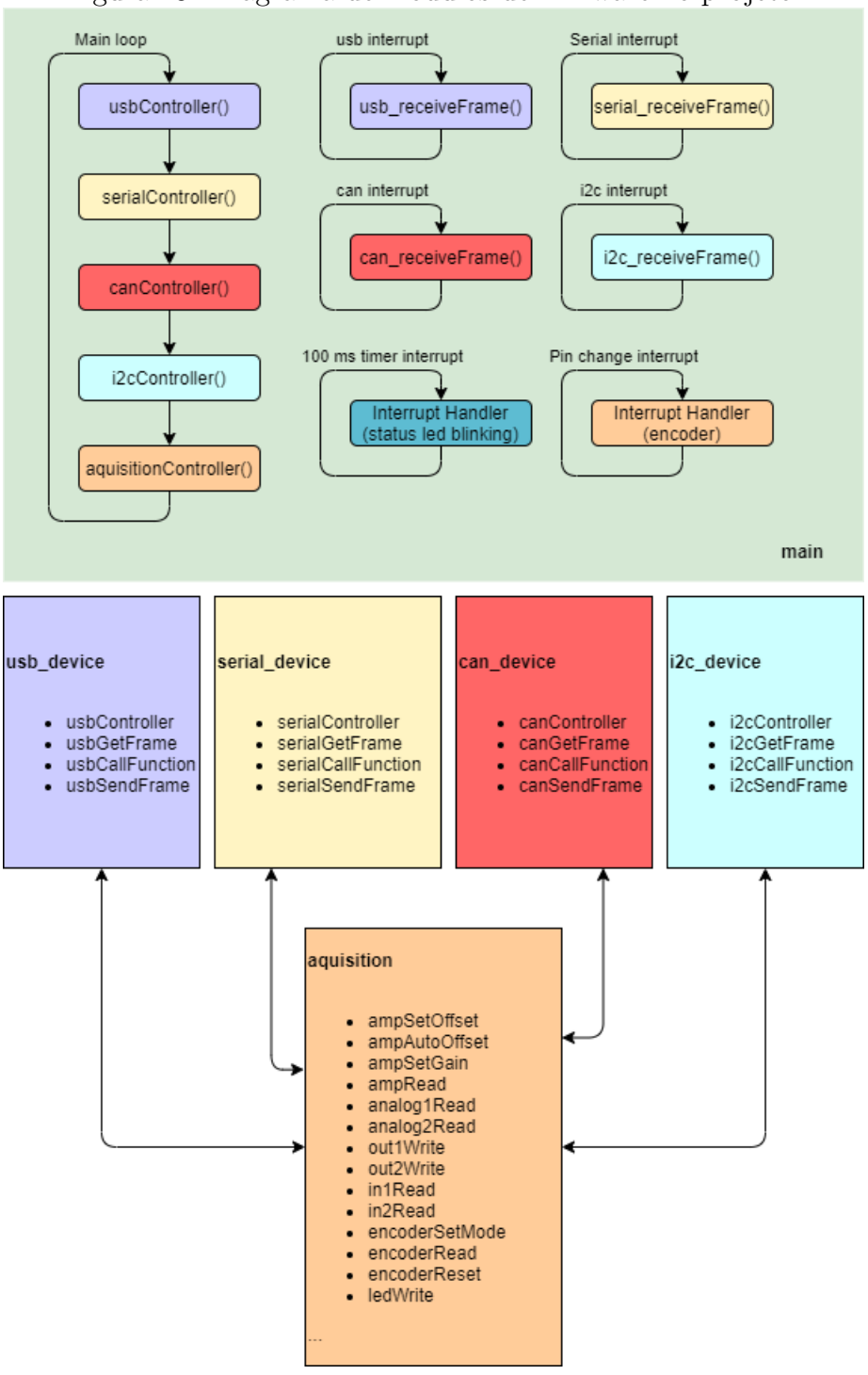
Alguns programas foram criados e gravados com sucesso na placa Discovery, como forma de familiarização. Nesses, foram implementadas algumas das funcionalidades previstas para o dispositivo, como uso do ADC, DAC, portas digitais, interrupções, comunicação serial e comunicação USB. O código utilizado, bem como os arquivos de configuração estão disponíveis em [1].

5.2.2 Módulos de firmware

Dada a pluralidade de interfaces de comunicação e a necessidade de interações rápidas por cada uma delas, o projeto de firmware leva em conta certo nível de isolamento entre cada módulo. De maneira geral, mensagens vindas por qualquer uma das interfaces gerará

interrupções e o conteúdo das mensagens será adicionado a uma fila de comandos. No *loop* principal do programa, serão chamados os controladores das interfaces que, independentemente deverão ser capazes de configurar e acessar o módulo de aquisição. Tal proposta é descrita na figura 23.

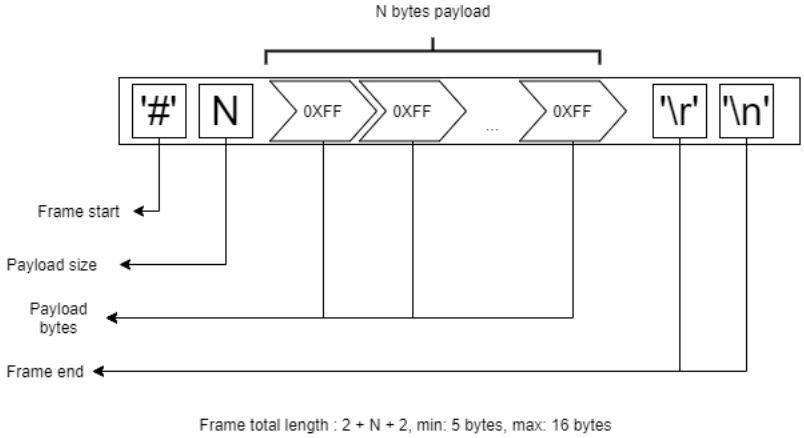
Figura 23: Diagrama de módulos de firmware no projeto.



Fonte: elaborado pelos autores

Essa figura aborda um nível bastante alto de programação e, para cada módulo, uma boa quantidade de código é necessária. O desenvolvimento, portanto, será iniciado pelo módulo USB, uma vez que esse é o mecanismo mais simples de ser testado.

Figura 24: Frame proposto para a comunicação USB.



Fonte: elaborado pelos autores.

Um primeiro passo adequado é a definição do *frame* de informações que será usado pelo módulo. Utilizando a implementação do protocolo MODBUS [68] como referência, propõe-se uma estrutura como a mostrada na figura 24. Nota-se que não é necessário incluir o endereço do dispositivo no *frame*, uma vez que a comunicação é apenas entre o host e um único dispositivo. Também destaca-se que o *frame* tem comprimento variável, especificado a partir do valor do segundo byte. Sendo assim, no recebimento de informações deve ocorrer a validação do comprimento com os bytes sinalizadores do fim do *frame*. Optou-se limitar o comprimento do *frame* em 16 bytes de modo que são acomodados entre 1 e 12 bytes de informações práticas. A implementação deverá também definir um tamanho de *buffer* adequado para garantir o processamento das requisições e assim não perder informações. A cada iteração do *loop* principal, o controlador da interface USB deve processar uma mensagem, ou seja, chamar a função definida, com o respectivo argumento da mensagem. Caso as funções chamadas retornem valores, esses deverão compôr a mensagem de resposta. Mensagens de configuração devem retornar o valor configurado como modo de validação para o *host*.

Já quanto à interface serial (lembrando que foi adotado o padrão RS-485 para o hardware) opta-se por implementar o protocolo MODBUS, fazendo assim o dispositivo compatível com muitas aplicações industriais típicas. É necessário portanto, adequar as chamadas de funções. como descritas no parágrafo anterior, para os tipos de funções previstas no protocolo [69]. É mandatório que a transmissão seja feita em modo RTU, no

formato da figura 25, além de que sejam cumpridos vários apontamentos de configurações iniciais que garantem a operabilidade do dispositivo por um mestre da rede. Vale destacar que, em comparação com o *frame* definido para a comunicação USB, passamos a ter campos de endereço e o chamado CRC (*ciclical redundancy check*) como forma de detecção de erros. Interessantemente, o microcontrolador escolhido para o projeto eletrônico possui um módulo dedicado para o cálculo desse verificador.

Figura 25: Frame para o protocolo MODBUS RTU.

Slave Address	Function Code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low, CRC Hi

Fonte: [70].

Para a interface CAN, o protocolo (levantado como requisito do projeto) será o CANopen, uma implementação de nível alto destinada ao uso industrial [71]. Em relação ao protocolo MODBUS, agora existem ainda mais restrições que devem guiar a programação do dispositivo, como a obrigatoriedade da implementação de uma máquina de estados que controla a interação do dispositivo com os demais em uma rede CANopen. Nota-se que embora a rede CAN não faça distinção entre mestres e escravos (o meio físico é igualmente acessível para todos os dispositivos), o protocolo CANopen diferencia mestres (controladores da rede que têm prioridade na comunicação e poder sobre o estado de funcionamento dos demais dispositivos) de escravos.

Pensando na utilização simultânea de mais de uma interface de comunicação, não podemos atrelar o funcionamento "real"do dispositivo com a máquina de estados da interface CAN. Dessa forma, tratamos a interface como um dispositivo "virtual"que acessa informações do dispositivo real.

A última interface de comunicação, I^2C, tem uma proposta relativamente diferente das anteriores: Esse protocolo foi originalmente criado para comunicação mais direta entre circuitos integrados e posteriormente expandido para dispositivos diferente [72]. A interface será, portanto, destinada à integração do dispositivo com outras placas microcontroladas, como Arduinos. Pela natureza do protocolo, o controlador I^2C será implementado para mensagens mais simples (limitadas a uma função de configuração ou de leitura). A maior distinção de papeis no protocolo I^2C ocorre pelo dispositivo mestre ser o controlador da linha de *clock*, não havendo necessidade de diferenciação no hardware.

Finalmente, como foi proposto que as interfaces serão máscaras para as funções do dispositivo, é essencial planejar tais funções que serão implementadas sob um chamado controlador de aquisição de dados. Destacam-se as funções de configuração (de endereço, *offset* do amplificador, ganho do amplificador, parâmetros do filtro), de escrita (Leds ou saídas PWM[8]) e de leitura (do amplificador, do *encoder* ou das outras entradas).

A leitura do *encoder* será realizada com interrupções, seja para o caso do *encoder* de quadratura, seja para o *encoder* PWM. No controlador deverá existir a função que recupera o valor de rotação, bem como uma que define o tipo de *encoder* conectado. As leituras analógicas serão realizadas continuamente, com o advento do DMA[9] que armazena de forma cíclica os valores convertidos pelo ADC. No controlador deve existir o método de recuperação, tratamento e transmissão dos valores. Os demais acessos e leituras serão realizados de forma mais simples e somente quando houver uma requisição para tal.

5.2.3 Pré-Processamento Digital

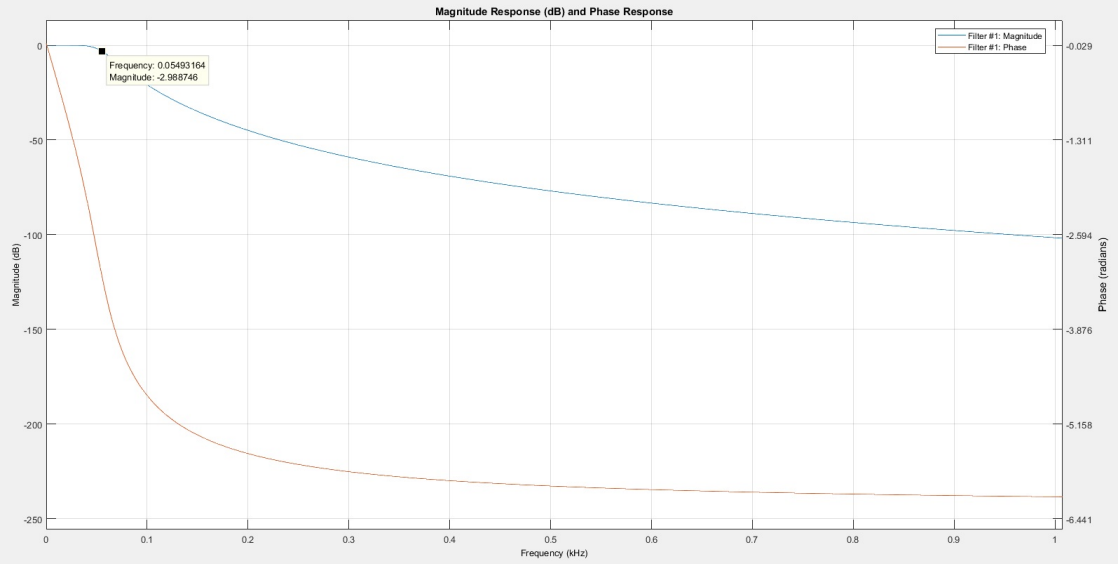
O pré-processamento digital cuida da filtragem digital e preparação do sinal para a análise requerida. Os filtros analógicos com amplificadores operacionais utilizados no hardware possuem a função de fazer o corte de frequências altas provenientes de ruídos de diferentes origens, sendo um passa-baixa. Esses filtros, aliados à alta taxa de amostragem do sinal, fazem com que o *aliasing* tenda a zero, algo importante para que o sinal não seja destruído em sua aquisição. Contudo, ainda restam ruídos no sinal que poderiam atrapalhar uma análise correta, como por exemplo o proveniente da rede elétrica, com frequência de 60 Hz. Assim, faz-se necessária a filtragem digital.

A banda utilizada nas aplicações é baixa, não ultrapassando 50 Hz. A taxa de amostragem, por sua vez, é da ordem de kHz. Dessa forma, não ocorrerá *aliasing*. Para que também não hajam frequências da rede elétrica, é proposto um filtro digital com frequência de corte de 55 Hz e frequência de amostragem de 12 kHz. Trata-se de um filtro passa-baixa do tipo Butterworth, como o utilizado no hardware, mas de 4ª ordem.

Na figura 26 é possível ver o módulo e a fase do filtro. O ponto marcado mostra a frequência de corte, ou seja, a frequência na qual o módulo se aproxima de -3 dB.

[8]Pulse Width Modulation
[9]Direct Memory Access

Figura 26: Módulo e fase do filtro digital utilizado, com frequência de corte de 55 Hz.



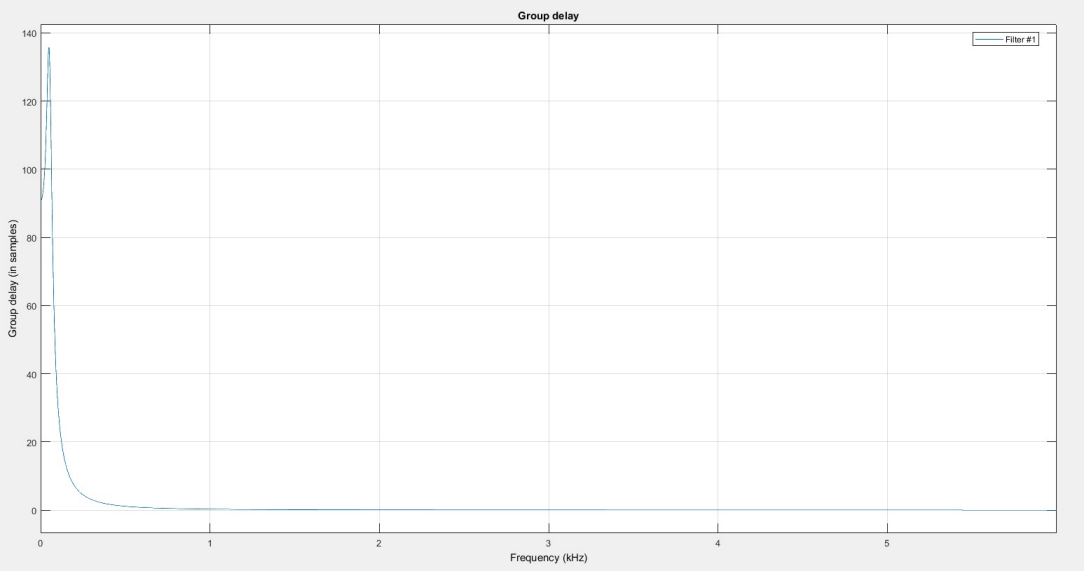
Fonte: elaborado pelos autores.

A função de transferência do filtro, mostrada em coordenada Z, foi calculada por meio de software e é a seguinte:

$$G(z) = \frac{(4.141e-8)z^4 + (1.656e-7)z^3 + (2.485e-7)z^2 + (1.656e-7)z + (4.141e-8)}{z^4 - 3.925z^3 + 5.777z^2 - 3.78z + 0.9275}$$

Também é importante salientar que a implementação de um filtro resulta em um atraso na resposta do sinal, que pode ser significativo quando se trata de um sistema com baixa latência. Portanto, é necessário verificar qual é esse atraso. Na figura 27 é possível ver que o maior atraso do sinal é de em torno de 135 amostras, que a uma frequência de amostragem de 12 kHz significa aproximadamente 11,2 ms. Assim, é válido dizer que, nas circunstâncias do projeto, o filtro não representa problema significativo no tempo de resposta.

Figura 27: Atraso de grupo do filtro passa-baixa.



Fonte: elaborado pelos autores.

Ainda, como algumas das aplicações envolvem a implementação do Controle do sistema, é interessante que sejam estimados os valores da derivada de cada ponto e/ou que seja aplicada uma ação integrativa no sinal para suavização da curva e aumento da robustez, podendo essas informações servirem também para a obtenção de outros valores de interesse, como a potência mecânica do sistema.

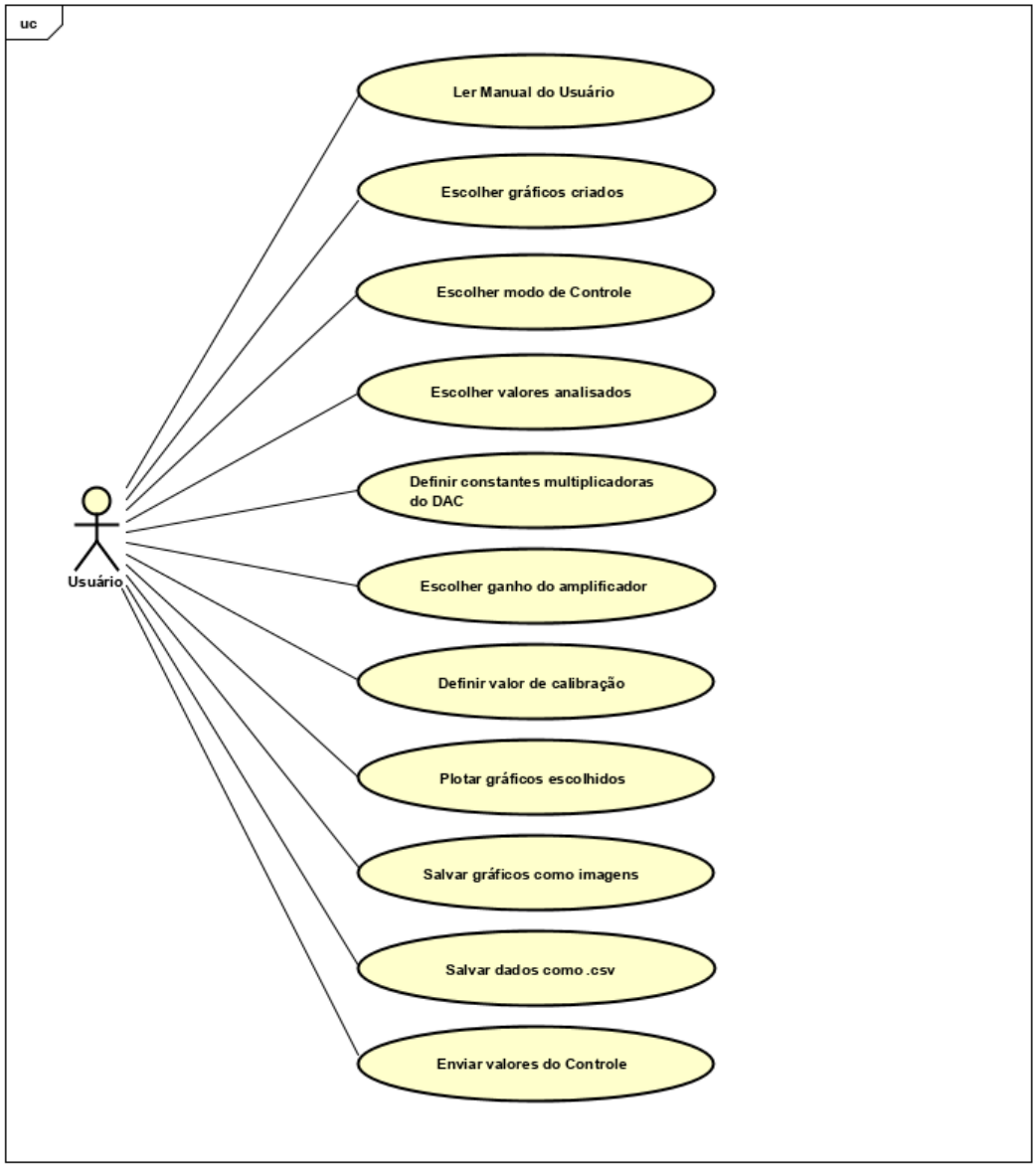
5.2.4 Análise e Processamento Digital

As informações referentes ao sinal pré-processado chegam até o software para que o usuário possa interagir com elas. São apresentadas a ele algumas opções, como mostra a figura 28. É válido ressaltar que as opções contemplam os requisitos de projeto.

Ler Manual do Usuário: abre o README na página do Github do projeto, contendo informações sobre a placa, as aplicações onde ela é utilizada e como usar o programa. É válido destacar que o manual oferece informações a usuários que já possuam algum conhecimento técnico na área, não sendo explicados conceitos de Controle ou de Instrumentação, por exemplo, mas sim como utilizar o programa como ferramenta para as aplicações.

Escolher gráficos criados: na janela de configurações são apresentadas as opções de gráficos que podem ser gerados pelo programa. As caixas que forem marcadas pelo usuário indicarão quais gráficos aparecerão quando for clicado o botão "Mostrar Gráficos".

Figura 28: Diagrama de Casos de Uso do projeto.



Fonte: elaborado pelos autores.

Escolher modo de Controle: a janela de configurações possui um botão *toggle* por meio do qual o usuário pode ativar o modo de Controle. Uma vez que esse botão foi apertado, é possível selecionar qual tipo de Controle será utilizado: P, PD, PI ou PID.

Escolher valores analisados: o usuário pode selecionar *checkboxes* para identificar de quais valores ele quer os gráficos e dados. As opções são: deformação, força, torque e potência.

Definir constantes multiplicadoras do DAC: o software recebe da placa os valores referentes à tensão de saída da ponte de Wheatstone amplificada, interpretados pelo DAC. Cabe ao usuário informar qual constante deve multiplicar essa tensão para que se tenha

o valor referente à deformação, à força, ao torque e à potência.

Escolher ganho do amplificador: ainda na janela de configurações, o usuário pode digitar qual é o ganho utilizado pelo amplificador na saída da ponte de Wheatstone.

Definir valor de calibração: quando a janela do modo de calibração é aberta, o usuário pode informar qual o valor da deformação, da força ou do torque que está sendo exercido naquele momento, para que a placa realize sua rotina de auto-calibração.

Plotar gráficos escolhidos: serão abertas as janelas com os gráficos pedidos conforme configuração do usuário, como: informações relativas a Controle como polos e zeros e resposta a entrada degrau e rampa; curva de torque do motor; e gráficos da deformação, força, torque e potência no decorrer do tempo.

Salvar gráficos como imagens: no menu das janelas de gráficos abertas, há a opção de salvar esses gráficos como imagens. Será salva a tela do gráfico no instante em que o botão for pressionado. Outra opção é o usuário pressionar 'Ctrl-X' antes de salvar, que fará com que o programa pare de gerar novos pontos do gráfico.

Salvar dados como *.csv*: outra opção no mesmo menu é salvar os dados do gráfico como um arquivo *.csv*. O arquivo gerado terá as colunas relativas ao tempo e ao valor do eixo Y.

Enviar valores do Controle: ao apertar esse botão na janela principal, será aberta outra janela pedindo que o usuário informe o valor do *setpoint* e indicando que é de responsabilidade do usuário informar um valor que esteja dentro da faixa possível da aplicação. Serão mostrados os valores das constantes aplicadas pelo Controle e será pedida a confirmação do envio.

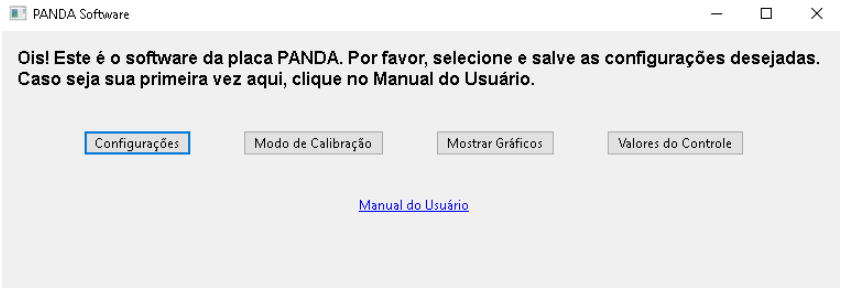
O software é feito na linguagem Python, utilizando a interface gráfica gerada com a biblioteca *wxPython*. Além disso, também são usadas as bibliotecas *matplotlib* (plotagem de gráficos), *os* (interação com o sistema operacional para salvar imagens e arquivos), *time* (informações relativas ao tempo) e *pyinstaller* (criação do arquivo executável). O resultado final é um executável pensado para funcionar em Windows, contudo, para usuários de Linux ou de Mac, bastaria rodar o comando do *pyinstaller* para gerar o executável para o sistema operacional específico. Todos os códigos estão apresentados e comentados no apêndice B, e eles e o link para download do executável encontram-se no Github do projeto [1].

5.2.5 O Programa

Todas as partes do programa foram testadas para garantir que não houvessem erros durante seu uso, havendo também janelas de diálogo que são abertas quando o usuário tenta realizar alguma ação que está impossibilitada devido a problemas como por exemplo não ter conectado a placa ao computador. A seguir é descrito um possível conjunto de interações entre o usuário e o software, relativo a uma das aplicações. É válido lembrar que no Manual do Usuário constam as instruções para uso do programa.

A primeira ação do usuário é abrir o executável do programa. Uma vez clicado, a janela principal demora até 30 segundos para aparecer (figura 29). Nela, como é a primeira vez do usuário no programa, é clicado em Manual do Usuário (figura 30), e então é aberta a página do README (figura 31). No README, o usuário obtém mais informações a respeito do projeto e as formas como ele pode usar o programa.

Figura 29: Janela principal do software.



Fonte: elaborado pelos autores.

Figura 30: Hyperlink para o README.

Fonte: elaborado pelos autores.

Figura 31: Arquivo README no Github do projeto.

PANDA V1.0

PANDA

Placa de Aquisição e coNdicionamento de DAdos

Escola Politécnica da USP - Engenharia Mecatrônica

Sobre

Trabalho de Conclusão de Curso dos alunos Guilherme de Agrela Lopes e João Vitor Sanches, orientados pelo Prof. Dr. Rafael Traldi Moura, para a obtenção do diploma de Engenheiros Mecatrônicos da Escola Politécnica da USP.

amigos da poli

O foco do projeto é a PANDA (Placa de Aquisição e Condicionamento de Dados), uma placa construída primeiramente com o objetivo de realizar a aquisição de sinais provenientes de extensômetros, para a medição de deformações, forças e torques em aplicações de robótica. Contudo, ela evoluiu para mais do que isso. Ela também é uma plataforma de comunicação para diversos protocolos: I2C, CAN, USB e RS-485, além de possuir entradas analógicas e portas digitais.

Seu propósito é auxiliar em aulas e laboratórios da Escola Politécnica, sobretudo nas áreas de Engenharia Mecatrônica e Mecânica, nas mais diversas aplicações aqui citadas, pretendendo-se no futuro expandir para outras aplicações em conjunto com outras Engenharias.

O projeto é apoiado e financiado pelo Fundo Patrimonial Amigos da Poli, fundo este criado com o propósito de desenvolver o potencial dos alunos da Poli, financiando projetos de politécnicos para politécnicos, auxiliando a Escola a crescer e se desenvolver cada vez mais. Este projeto não seria possível sem o seu apoio, e por isso somos muito gratos.

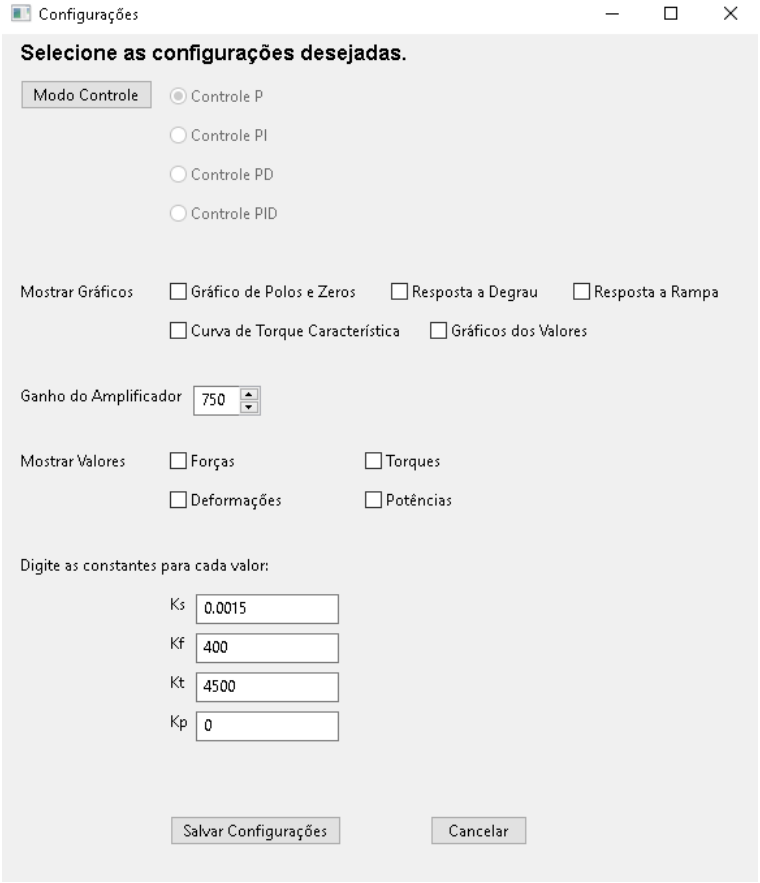
Tabela de conteúdos

- Sobre
- Tabela de Conteúdos
- Download
- Como Usar
  - Pré-Requisitos
  - Configurações
  - Calibração

Fonte: elaborado pelos autores.

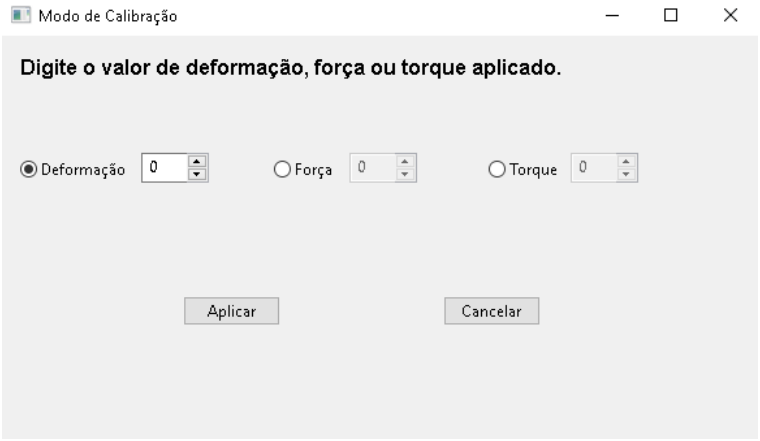
Uma vez que o usuário conhece mais sobre o funcionamento do projeto, ele clica em Configurações (figura 32), para definir o uso do programa. Nessa situação hipotética, quer-se medir a deformação de um corpo. Portanto, o usuário clicaria nas opções "Gráficos dos Valores", "Deformações"e digitaria o valor de Ks. Além disso, colocaria o ganho desejado para o amplificador (1-1000). Uma vez que as configurações estão salvas, o passo seguinte seria abrir o modo de calibração (figura 33). Naquele momento a deformação no corpo é nula, então seria selecionada a opção "Deformação"e colocado o valor 0.

Figura 32: Janela de configurações do software.



Fonte: elaborado pelos autores.

Figura 33: Janela de calibração do software.

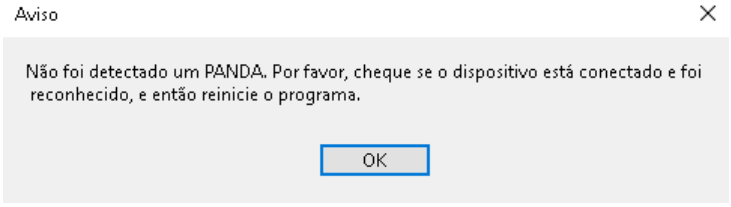


Fonte: elaborado pelos autores.

Em seguida chega a hora de abrir os gráficos. Se o usuário clicar no botão, mas não houver PANDA conectada, aparecerá um aviso solicitando que a conecte (figura 34). Se há PANDA conectada, uma janela com o gráfico da deformação sendo atualizado em baixa

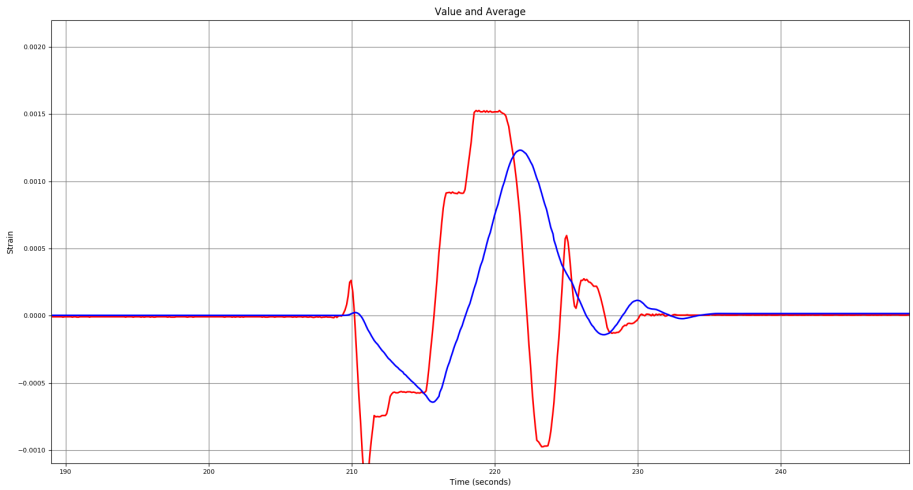
latência deve aparecer (figura 35). Nesta janela, é possível clicar no menu "Arquivo" (figura 36) e escolher dentre três opções: "Salvar dados em .csv", "Salvar imagem" e "Parar aquisição". A primeira opção gera uma planilha em formato *.csv* no diretório escolhido. A segunda grava a imagem do gráfico no momento em que o botão foi clicado. Por fim, a terceira para a aquisição dos dados, no caso a deformação. A janela do gráfico continua aberta, porém estática. Para retornar à aquisição, deve-se clicar de novo no botão "Mostrar Gráficos" na janela principal.

Figura 34: Aviso indicando que a placa não está conectada.



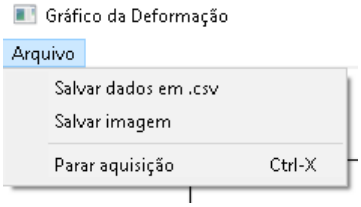
Fonte: elaborado pelos autores.

Figura 35: Janela com o(s) gráfico(s) gerado(s).



Fonte: elaborado pelos autores.

Figura 36: Menu da janela do gráfico.



Fonte: elaborado pelos autores.

Por fim, a figura 37 mostra um trecho do arquivo gerado com os dados de deformação no decorrer do tempo. O usuário então pode usá-los para o que desejar. O modo de Controle será implementado no futuro, junto com a aplicação direcionada às aulas de Controle.

Figura 37: Arquivo *.csv* gerado.

	A	B
139	97.28572	0.000597
140	97.37372	0.000595
141	97.49772	0.000624
142	97.57472	0.000616
143	97.69472	0.000633
144	97.81172	0.000641
145	97.89672	0.000645
146	98.01371	0.000648
147	98.12671	0.000655
148	98.25371	0.000665
149	98.37171	0.000674
150	98.45571	0.000694
151	98.59471	0.000725
152	98.6827	0.000744
153	98.8117	0.000759
154	98.8937	0.000793

Fonte: elaborado pelos autores.

5.3 Projeto Mecânico

5.3.1 Célula de carga

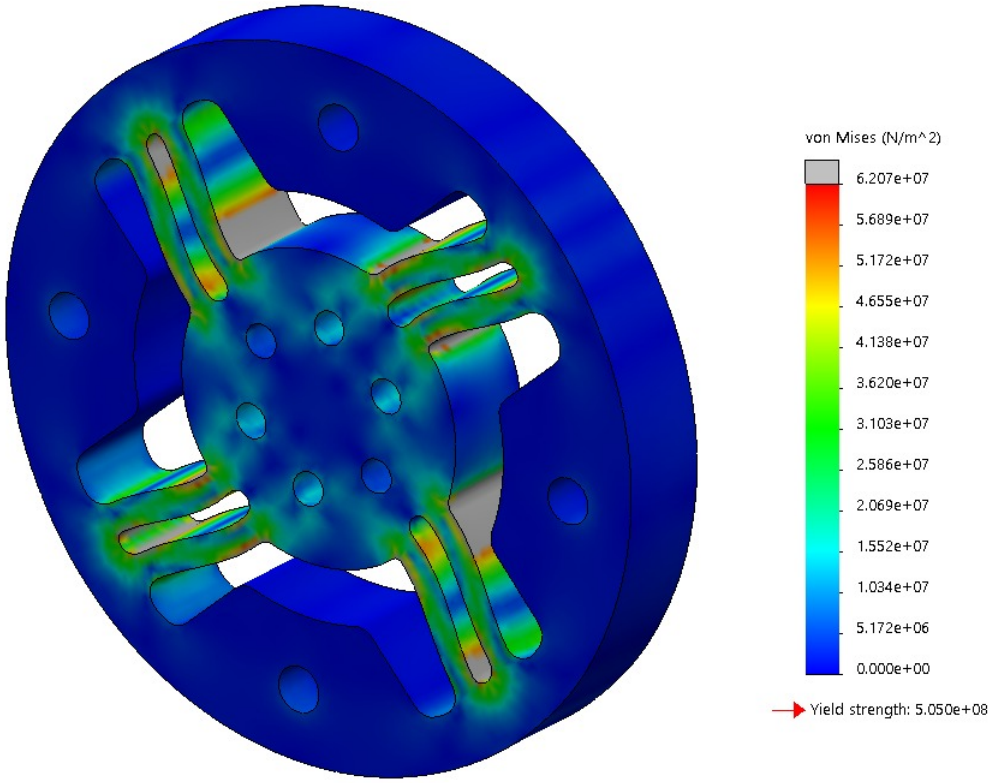
A parte mecânica do projeto possui como principal componente a célula de carga, onde serão colados os extensômetros e onde ocorrerá o acoplamento com o sistema da aplicação. A célula, apresentada na figura 38, possui como inspiração formatos encontrados em referências como [29].

Foram realizadas alterações no formato do sensor com o auxílio de software de elementos finitos, de modo a reduzir a quantidade de material e ter maior controle do fluxo de tensões ao longo da célula. Na figura é possível ver onde ocorre a concentração de tensões, em cinza. São nessas regiões que serão colados os extensômetros, um em cada braço, dois nos lados que sofrem tração e dois nos lados que sofrem compressão, totalizando quatro conforme requisitado.

A célula de carga é de alumínio 7075 (alumínio aeronáutico), material que apresenta alta resistência, sendo comparável à de alguns aços, mas com 1/3 do peso deles, além de aguentar os requisitos do projeto. Planeja-se que a fabricação do sensor seja realizada por

meio de corte a água, devido ao formato complexo da peça e para que ela seja feita com precisão.

Figura 38: Célula de carga projetada simulada em software de elementos finitos.



Fonte: elaborado pelos autores.

5.3.2 Encapsulamento da Placa

Pensando na diversidade de aplicações, é difícil construir um invólucro que se ajuste a todos os casos. Sendo assim, opta-se por modelar pensando em manufatura aditiva, de modo que sejam possíveis personalizações no projeto. Uma caixa simples é mostrada na figura 39 e seu modelo 3D está disponível no repositório do projeto.

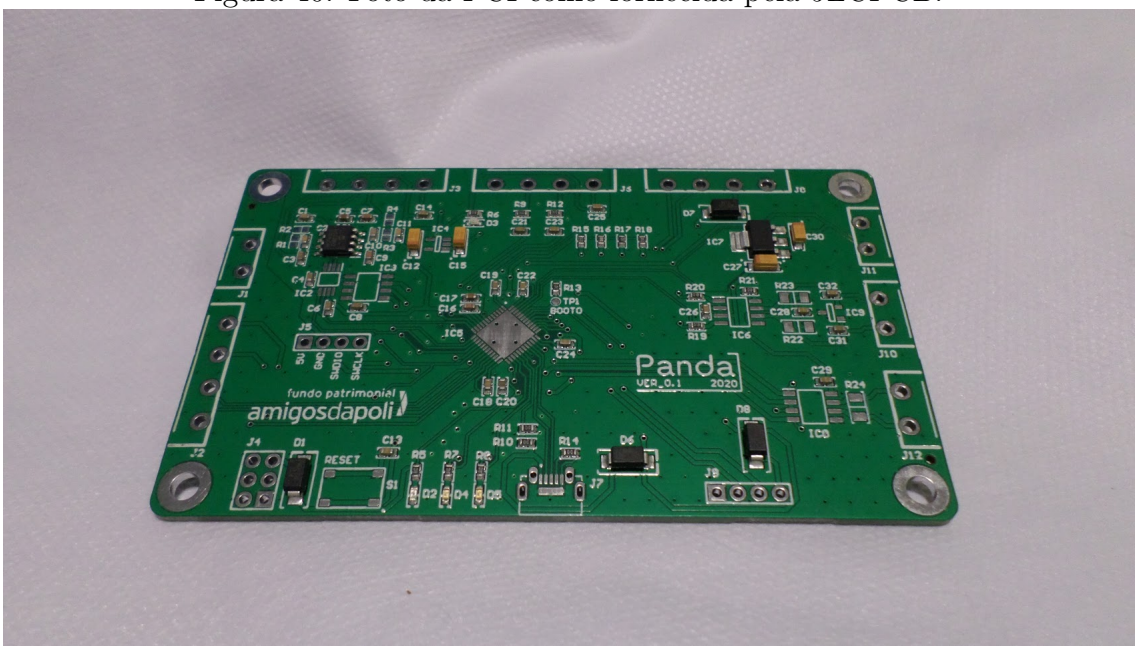
Figura 39: Modelo 3D de caixa para o dispositivo.

Fonte: elaborado pelos autores.

6 FABRICAÇÃO DOS PROTÓTIPOS

As placas foram fornecidas pela JLCPCB [73]. O serviço contratado incluiu a fabricação de 5 PCIs e a soldagem de alguns componentes (praticamente todos os componentes passivos, Amplificador Operacional do filtro e regulador de tensão) como exibido na figura 40. Os demais componentes foram comprados principalmente na Mouser [74]. Para efeitos de avaliação, o custo por dispositivo é calculado a partir do custo estendido para 20 unidades, de modo a evitar sobrepreço por conta do baixo volume. Os valores são apresentados na tabela 5.

Figura 40: Foto da PCI como fornecida pela JLCPCB.



Fonte: elaborado pelos autores.

Tabela 5: Custos de fabricação das placas protótipo.

	Componentes	PCI	Frete (placas + componentes)
Preço 20 Un. (U$)	431,60	15,50	69,00
Preço 1 Un. (U$)	21,58	0,78	3,45

Fonte: elaborado pelos autores.

6.1 Ferramentas

O processo de soldagem se valeu das ferramentas:

- Suporte de PCI
- Estação de retrabalho SMD (ferro de solda e soprador de ar quente)
- Luminária
- Pinça
- Escova anti-estática
- Sugador de solda
- Multímetro
- Microscópio USB

e insumos:

- Estanho para solda (Sn63/Pb37 0,5mm)
- Fluxo de solda em pasta
- Álcool isopropílico (Isopropanol)
- Malha dessoldadora
- Escova anti-estática
- Sugador de solda
- Hastes flexíveis com algodão
- Papel para limpeza

6.2 Procedimento

As placas foram recebidas com uma camada protetora que foi removida antes de qualquer processo de solda. Os conectores de gravação foram soldados nos devidos *pads* e ficaram disponíveis por todo o processo como pontos de alimentação do circuito. Nesse

primeiro momento, ao alimentar o dispositivo com 5V, observa-se que o LED[1] vermelho é aceso, representando o funcionamento do regulador de tensão (3.3V). Deve-se então soldar a referência de tensão que, ao ser alimentada com 5V, acenderá o segundo LED vermelho. É necessário notar que, antes de cada teste, foi testada a continuidade entre as principais trilhas da placa, de modo a encontrar eventuais curto circuitos. O passo seguinte é consideravelmente mais sensível e complicado: Deve-se soldar o microcontrolador com o uso do soprador de ar quente. Aqui o microscópio se faz útil para verificar a qualidade das junções soldadas. O sucesso nessa etapa é atestado pela conexão correta entre o microcontrolador e o *debugger*, através das linhas de gravação. É útil então, carregar um programa simples na placa de modo a verificar o controle sobre os vários pinos, com o uso de um multímetro.

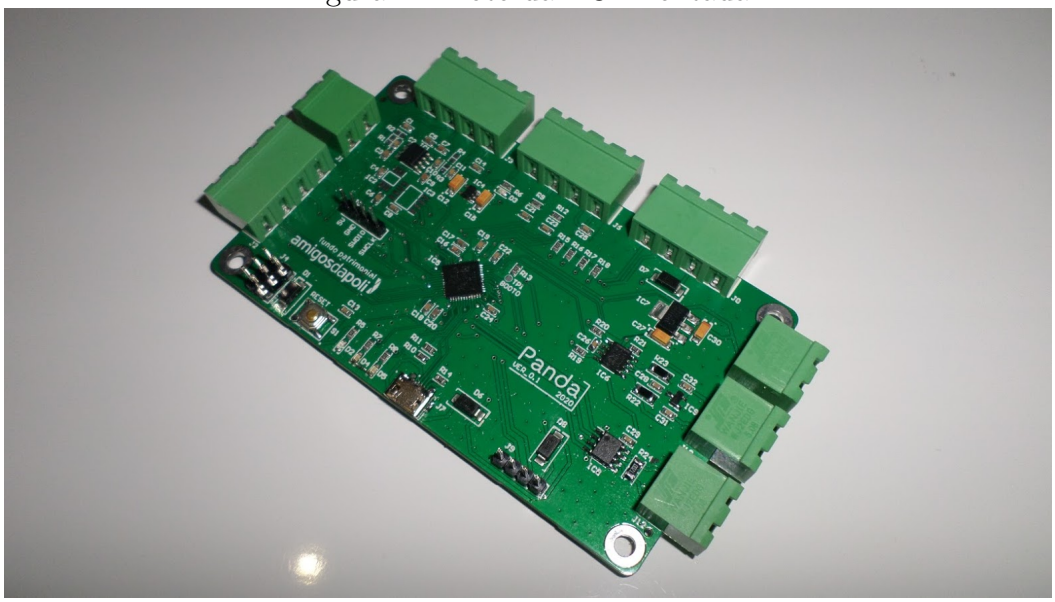
Olhando para o circuito de amplificação (já com o filtro soldado) deve-se soldar os resistores faltantes. Vale alimentar o circuito de amplificação e impôr tensões conhecidas nas entradas diferenciais. Essas mesmas tensões devem ser medidas dos pinos de saída do filtro. O circuito pode receber o CI potenciômetro digital (ajuste de ganho), que deve ter seu funcionamento verificado medindo-se a resistência entre os pinos corretos. É necessário carregar um programa simples no microcontrolador para alterar os valores de resistência através da interface SPI. Finalmente, o amplificador de instrumentação é soldado e assim já é possível realizar medidas com o ADC do microcontrolador.

O conector micro-USB pode ser soldado em seguida, tornando o dispositivo funcional através dessa interface. O mesmo vale para os pinos da interface I^2C. Para o CAN e Serial, no entanto, os respectivos *drivers* e resistores de terminação precisam ser soldados. Com todas as conexões bem feitas, o dispositivo será plenamente capaz em suas funcionalidades. Finalmente, os conectores faltantes são instalados na placa.

Destaca-se a atenção aos pontos de solda para garantir o bom funcionamento dos circuitos, bem como as frequentes limpezas com álcool, escova e papel. O resultado do processo de montagem é exibido na figura 41.

[1]Light Emitting Diode

Figura 41: Foto da PCI montada.



Fonte: elaborado pelos autores.

6.3 Testes Básicos

Ainda que os testes elétricos corram bem durante a montagem, é adequado verificar as funcionalidades da placa de forma mais prática:

- Entradas analógicas: Antes mesmo de avaliar o amplificador é adequado realizar testes de leitura analógica uma vez que assim, com o sucesso, é garantido que a atuação do ADC será correta no teste seguinte. Para tal, conecta-se um potenciômetro (como reostato) entre os pinos AIN1 e Vref. No microcontrolador deve estar carregado um programa simples, apenas com a funcionalidade de leitura do ADC. A obtenção dos valores convertidos é facilmente feita por meio do *debugger*. O mesmo procedimento vale para a segunda entrada analógica.

- Amplificador: Sendo o circuito mais complexo do dispositivo, o amplificador diferencial será testado gradualmente, a começar pela leitura do valor convertido pelo ADC, como descrito no item acima. Parte-se para a condição nula, com ambas entradas diferenciais no mesmo potencial, que pode ser gerado com um divisor resistivo ou com o uso de uma fonte de tensão. É importante lembrar que, devido a construção do amplificador de instrumentação, a diferença de tensão entre as entradas é amplificada e então somada a um *offset*. No caso do dispositivo em questão, tal *offset* é provido pelo conversor digital/analógico. Portanto, o microcontrolador deve ser

gravado com um programa que configura o valor do DAC para o meio da escala de conversão. Nessa situação, espera-se observar um valor em torno de 2048 na saída do ADC. Desvios desse valor de referência são frutos das características construtivas dos componentes utilizados e nesse caso, o valor configurado pelo programa pode ser ajustado para cancelar a defasagem observada.

O próximo passo consiste de efetivamente aplicar tensões diferentes às entradas do amplificador (50mV, por exemplo) pois, conhecendo esse valor, pode-se avaliar numericamente o valor amplificado. No caso do potenciômetro digital recém inicializado, o ganho do amplificador deve ser em torno de 21 vezes e esse valor deve ser compatível com o obtido pelo ADC (para a entrada de 50mV, a leitura digital será de aproximadamente 3351).

Nesse momento também é proveitoso testar a função de variação do ganho, por exemplo, pela configuração do potenciômetro para máxima resistência, de modo a reduzir o ganho do amplificador para o o mínimo de 11 vezes. Vale ainda, inverter as tensões na entrada para observar a coerência das leituras.

- Saídas digitais: O teste das saídas digitais é realizado simplesmente com a conexão de LEDs nos pinos designados. Além desses, as luzes da placa podem ser controladas pelo programa de modo a verificar o funcionamento das portas.

- Entradas digitais: Quando configuradas com resistor de *pull-up*, as entradas podem ser facilmente testadas conectando os pinos em questão a pinos de GND da placa. Nesse caso, a leitura digital feita pelo programa será 0, enquanto no caso contrário, o valor será 1. Caso a entrada fique flutuando, o mesmo teste pode ser feito utilizando as saídas que foram avaliadas no teste acima.

- Entradas para *encoder* : Primeiro, com um *encoder* de quadratura conectado aos pinos corretos, deve-se observar os valores digitais lidos nos pinos e garantir que os mesmos oscilem. Em seguida, com as interrupções habilitadas no microcontrolador e com o programa adequado, a contagem de pulsos deve operar normalmente.

- Interface USB: A conexão USB não depende de qualquer componente senão do microcontrolador e do próprio receptáculo e, portanto, se mostra funcional ou não quando o dispositivo é conectado à um computador, tendo gravado o programa com a interface em questão habilitada. O sucesso na instalação dos drivers e enumeração (reconhecimento do dispositivo pelo sistema operacional do computador) atesta a operação da conexão.

- Interface I2C: Assim como a USB, a comunicação I^2C depende apenas do microcontrolador e basta conectar um dispositivo compatível, como um Arduino. Para evitar erros de programação, é interessante usar programas tão simples quanto possíveis nas duas placas, prezando por mensagens de comprimento fixado. Em caso de sucesso, basta habilitar o controlador da comunicação e a interface funcionará integralmente.

- Interface CAN: Dessa vez o funcionamento da comunicação é pautado pela operação do *driver* do barramento CAN. Para validar a montagem desse componente é útil configurar as portas digitais no microcontrolador associadas ao *driver* como saída e entrada digital. Ao configurar o pino TX em nível lógico alto, deve-se obter alta impedância nas linha CAN HIGH e CAN LOW, denotando o estado recessivo do barramento. Escrevendo nível lógico baixo no pino, o barramento estará em modo dominante e uma tensão diferencial existirá entre os pinos do barramento. Nos dois estados descritos acima, o estado do pino RX pode ser lido e deve coincidir com o do pino TX.

- Interface RS-485: Uma vez que vale o mesmo descrito para a interface CAN, o teste da RS-485 deve seguir os mesmos passos. No entanto, além dos pinos TX e RX também existe o controle DE que habilita o *driver* permitindo controle do barramento. Mantendo esse sinal em nível lógico alto e alternando o valor do pino TX, é possível verificar a reversão da tensão diferencial entre as linhas A e B da interface RS-485. Mantendo o sinal DE em nível lógico baixo e impondo tensão entre as linhas A e B é possível ler o nível do pino RX. Alternando a tensão aplicada, a leitura do pino RX deve mudar por consequência.

No caso das placas protótipo fabricadas, observou-se que o LED D4 não respondia ao controle do microcontrolador. Após alguma inspeção, notou-se que o catodo do mesmo, embora soldado ao *pad*, não estava conectado ao plano de GND. Tal falha resultou de uma linha faltante no desenho esquemático utilizado para a fabricação da PCI e, portanto, se repetiu em todas as placas fabricadas. Uma simples solução para tal defeito foi acrescentar um pouco de solda ao terminal de modo a conecta-lo diretamente ao plano. Para tal foi necessário remover mecanicamente uma parte da máscara de solda próxima ao componente.

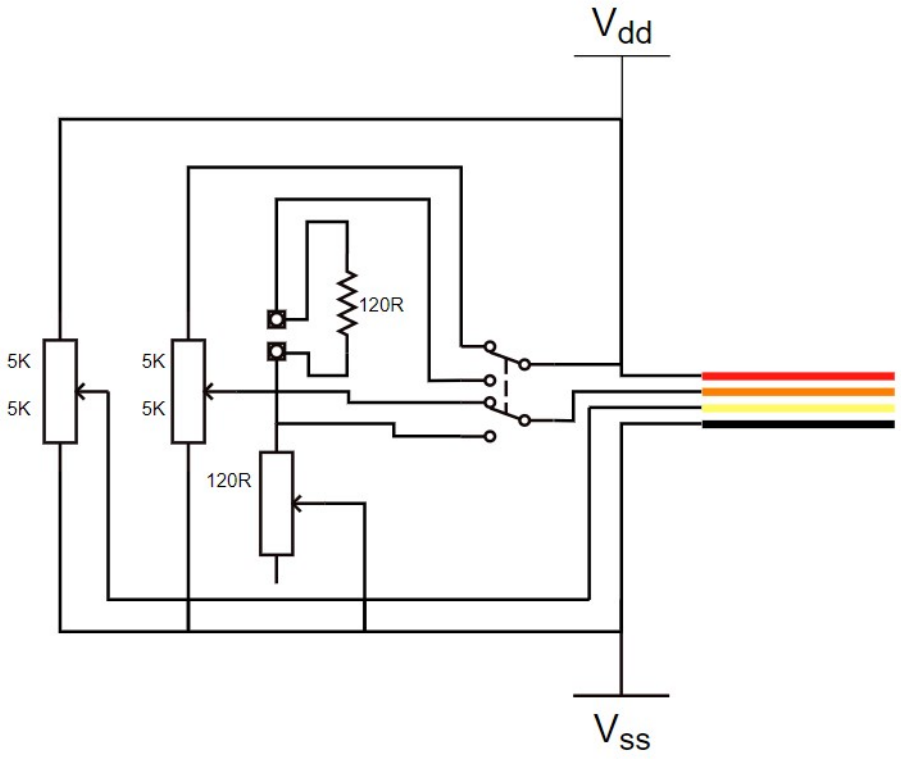
6.4 Testes Funcionais

Com a placa completamente montada e com o programa integral gravado no microcontrolador, os seguintes testes podem ser feitos:

- Testador da ponte de Wheatstone:

  Com a necessidade de testes em situações controladas, optou-se por criar o protótipo de um dispositivo para simulação de uma ponte de Wheatstone, com circuito demonstrado na figura 42.

Figura 42: Esquemático do simulador de ponte de Wheatstone.

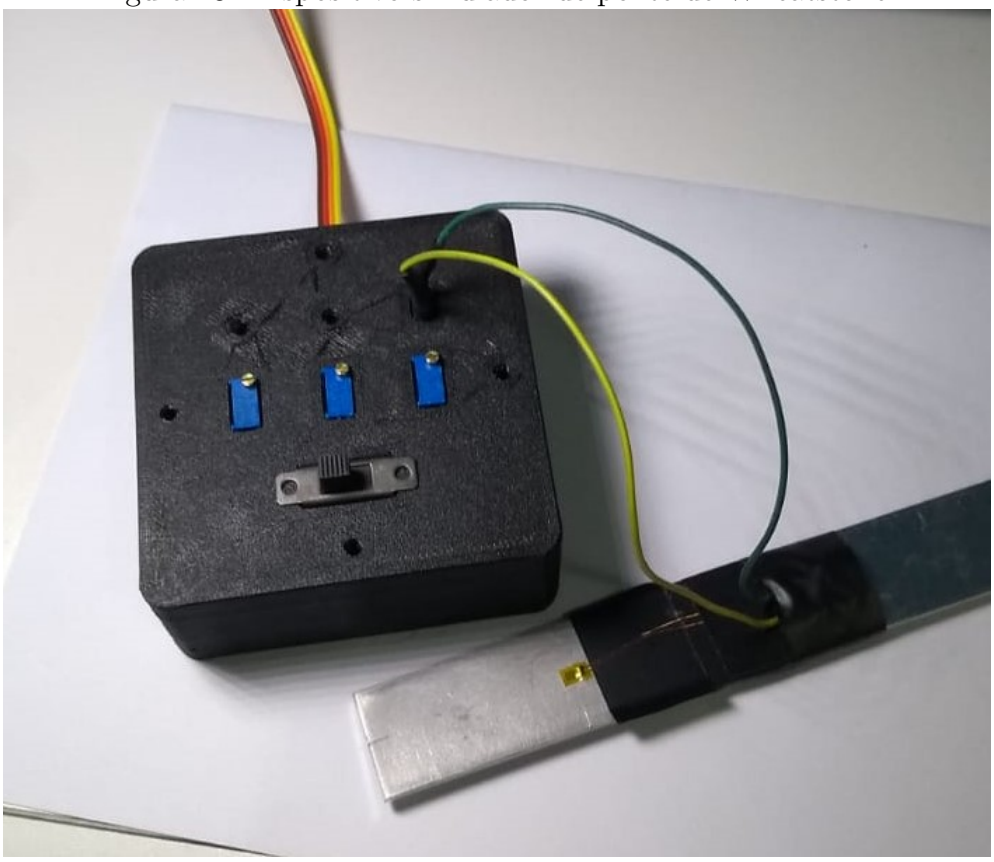


Fonte: elaborado pelos autores.

Quando conectado à placa de aquisição, com Vdd = 3.3V e Vss = 0V, tem-se no fio amarelo o valor de tensão dado pelo divisor resistivo regulável (potenciômetro). Com a chave seletora na posição 1 (representada na figura) o fio laranja também representa um valor de tensão regulado pelo divisor resistivo. Com a chave na posição 2, o fio laranja tem tensão dada pelo divisor resistivo formado por um extensômetro (externo, conectado ao testador por fios *jumper*) e por um reostato que deve ser regulado para balancear a ponte.

A construção do testador foi feita com uma caixa produzida em impressora 3D na qual foram montados os componentes. Na imagem 43 é possível observar o testador montado e com uma célula de carga com um extensômetro conectado. Quatro pontos de teste foram adicionados à caixa para que fosse possível medir as tensões com o multímetro.

Figura 43: Dispositivo simulador de ponte de Wheatstone.



Fonte: elaborado pelos autores.

- Comunicação CAN entre Pandas:

  Embora essa interface tenha sido proposta para conectar o dispositivo a redes CANOpen, o funcionamento da placa pode ser atestado ignorando esse protocolo e simplesmente trocando mensagens pré determinadas entre Pandas conectadas ao barramento. Para tal, o programa inclui uma função de testes que aciona o envio do *frame* de teste que, quando recebido por outras placas, faz piscar seus LEDs de usuário. Essa função pode ser ativada por um programa de teste ou por uma função da interface USB.

- Comunicação RS-485 entre Pandas:

  Assim como a interface anterior, mas operando com o protocolo MODBUS, a co-

municação serial também possui uma função de teste que pode ser chamada pela interface USB. Nesse caso, devido a necessidade de endereçar a mensagem, foi adicionada uma exceção no protocolo que permite qualquer placa receber *frames* com um endereço determinado para teste. No entanto, dado que o dispositivo escravo deve responder a mensagem de teste, deve-se realizar o teste apenas entre pares de placas.

O LED de usuário deve sinalizar tanto o recebimento da mensagem pelo escravo, como o recebimento da resposta, no mestre.

7 ESTUDOS DE CASO

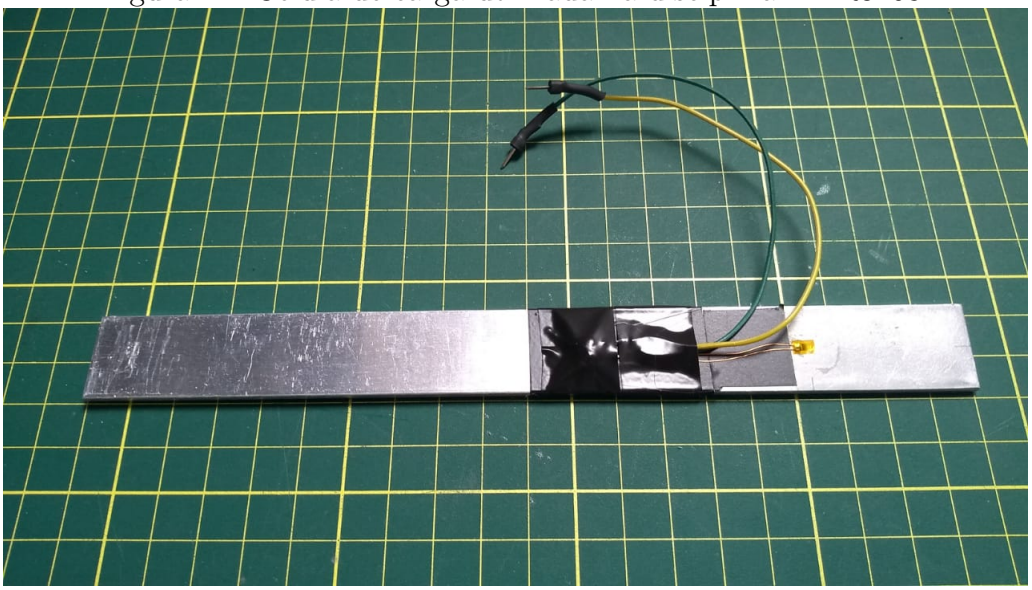
7.1 Balança Experimental

7.1.1 Descrição

Desenvolvida como experiência de primeiro contato dos alunos com extensômetros, a atividade "Balança"da disciplina PMR3408-Instrumentação permite que os estudantes construam uma célula de carga com um extensômetro de modo a mensurar a massa de um objeto.

A célula de carga consiste de uma barra de alumínio com dimensões aproximadas de 200mm x 20,5mm x 2,5mm, onde é colado um extensômetro de 120Ω, como na figura 44.

Figura 44: Célula de carga utilizada na disciplina PMR3408.



Fonte: elaborado pelos autores.

Originalmente, os terminais do extensômetro são montados em uma protoboard com resistores de precisão para formar uma ponte de Wheatstone completa, que por sua vez, é conectada à um Arduino com um módulo amplificador HX711. Com essa configuração, os alunos utilizam um programa simples que realiza leituras de deformação para calcular

a massa do objeto na balança. É notável, no entanto, que tal esquema elétrico é muito sensível à interferências do ambiente, o que prejudica a performance da balança e acaba por tomar tempo da atividade. Os valores medidos na atividade apresentam precisão em torno de 10% e a capacidade da célula de carga é de 2N.

7.1.2 Integração da Panda

Em substituição ao Arduino com módulo amplificador, a Panda será utilizada pela interface USB, conectada a um computador. Com isso, os alunos poderão elaborar softwares baseados na biblioteca Panda (disponibilizada no repositório do projeto) para cumprirem a tarefa de medição. Alternativamente, pode ser utilizado o software descrito anteriormente nesse trabalho. Em ambos os casos, será necessário obter os parâmetros que relacionam o carregamento a ser medido com a tensão elétrica amplificada. Um exemplo de programa em Python para realização de medidas de massa é apresentado no apêndice B.7. Utilizando tal programa, foi realizado o teste de precisão do módulo de amplificação: Um objeto de massa conhecida foi posicionado na extremidade da célula de carga, após o uso do comando de correção de *offset*. Com a função de calibração implementada no software, obteve-se o fator que relaciona a tensão elétrica com a medida de massa e o mesmo foi utilizado para as aquisições seguintes. Dois outros objetos foram pesados com esse arranjo e as medidas nos dois casos se mantiveram em faixas mais estreitas que 2g, em torno dos valores nominais.

7.2 Controle de Módulo Motorizado

7.2.1 Descrição

Funcionando como plataforma didática de Controle (disciplinas PMR3409 e PMR3404) atualmente é utilizado um módulo contendo um motor elétrico de corrente contínua e sensores de velocidade, no entanto, é prevista a substituição desse sistema por um desenvolvido na própria Escola Politécnica com as características ideais para a aplicação em questão. O intuito das atividades baseadas nessa plataforma é aplicar conceitos de controle discreto, projetando e testando controladores em um sistema mecânico.

7.2.2 Integração da Panda

A Panda, nesse caso, deve servir como interface entre a aplicação (incluindo um motor, seu *driver* e *encoder*) e um computador, no qual é executado o controlador. Com a planta definida, poderá ser preparado o programa que será utilizado pelos alunos nas atividades.

7.3 Leitura de temperatura

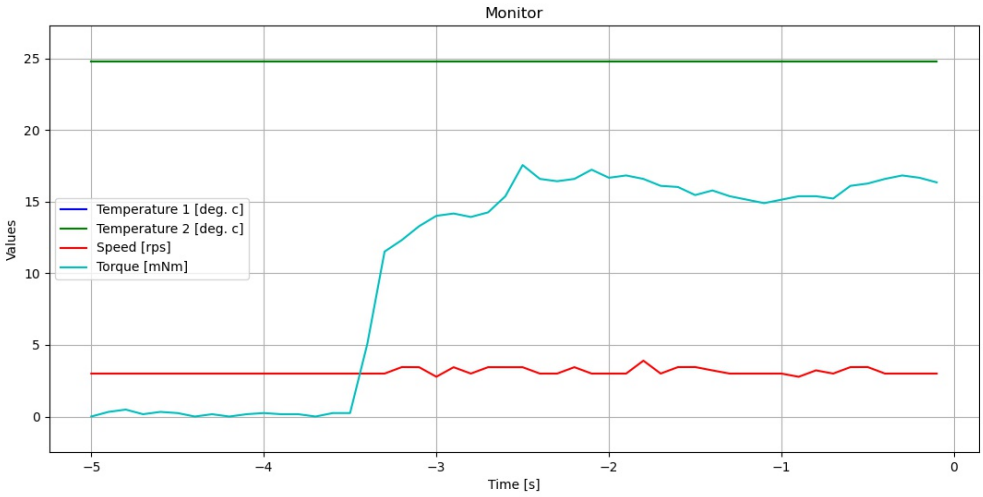
7.3.1 Descrição

A necessidade de medir temperaturas é associada à disciplina de Termodinâmica oferecida pelo PME. Nesta, estudantes utilizam uma máquina térmica como base de testes, fazendo medições de velocidade angular, torque e temperaturas. Cada uma dessas grandezas é medida com instrumentos independentes (dinamômetro, tacômetro e termopar) e posteriormente associadas em planilhas que baseiam cálculos pertinentes à disciplina.

7.3.2 Integração da Panda

Visto que o projeto da Panda levou em conta a possibilidade de medida de temperatura, essa aplicação poderá receber um sistema de instrumentação baseado no dispositivo desenvolvido. Com um freio de Prony equipado com extensômetros, um *encoder* acoplado ao eixo rotativo da máquina e termistores instalados nos pontos de medida de temperatura, uma Panda é capaz de fazer a interface digital do equipamento. Através da interface USB, os estudantes podem obter dados da máquina com um programa simples, como o apresentado no anexo B.8, e que produz visualizações como mostrado na figura 45.

Figura 45: Exemplo de gráfico gerado pelo programa exemplo de aquisição.



Fonte: elaborado pelos autores.

7.4 Caracterização de motores

7.4.1 Descrição

A necessidade de caracterização de motores é natural nas atividades da Mecatrônica e assim, como exposto anteriormente, tem sua importância em disciplinas (como Projeto de Máquinas e Controle I e II), laboratórios (Biomecatrônica) e equipes de competição (ThundeRatz). É pouco efetivo, no entanto, tomar como padrão um modelo de célula de carga para todos os usos de caracterização de motores, já que esses pertencem a diferentes classes de potência. Por outro lado, a parte eletrônica da instrumentação pode ser unificada.

7.4.2 Integração da Panda

A Panda é adequada para realizar a amplificação dos sinais de célula de carga, bem como a leitura de posição e velocidade angular com *encoders* de quadratura. Nesse caso, com a conexão USB e um software de aquisição similar ao apresentado no caso anterior, pode-se realizar a integração com o devido aparato para cada motor.

7.5 Controle de exoesqueletos

7.5.1 Descrição

Em matéria de controle de atuadores em tempo real, existem sistemas munidos de célula de carga. Nesse tipo de aplicação, se preza por resistência à ruído, precisão e rapidez na resposta. No laboratório de Biomecatrônica da Escola Politécnica, além desses, existe o desafio de integração entre os dispositivos de medição e os controladores.

7.5.2 Integração da Panda

Colhendo dados de célula de carga e de posição (com *encoder* PWM), uma Panda pode ser empregada nas aplicações de controle de juntas atuadas. Com o circuito de comunicação CAN contido na placa é possível integrar o dispositivo com as redes de controle já utilizadas no laboratório.

7.6 Aplicações móveis

7.6.1 Descrição

Quando a aplicação exige mobilidade, sistemas embarcados podem ser associados com a Panda de modo a desenvolver aplicações adequadas para coleta remota de dados. É comum nesse cenário, utilizar placas microcontroladas como Arduinos, no entanto, as mesmas não possuem todo o suporte para aquisição de sinais.

7.6.2 Integração da Panda

Sendo assim, a Panda poderá ser utilizada através da interface I^2C, com a possibilidade de transmitir os dados por meio da aplicação desenvolvida pelo usuário ou mesmo salvar as medições para recuperação posterior. A biblioteca de acesso da Panda para Arduino será disponibilizada no repositório do projeto.

8 DISCUSSÃO

8.1 Performance

A essência do dispositivo desenvolvido nesse trabalho é operar como interface de instrumentação para uma variedade de aplicações, principalmente de funções didáticas e de pesquisa. Dessa maneira, foi projetada uma placa de circuito impresso com circuitos para satisfazer as necessidades de professores e alunos, com requisitos levantados a partir desses possíveis usuários finais do sistema. Embora resultados definitivos só poderão ser obtidos com a efetiva aplicação do dispositivo nos ambientes citados ao longo desse trabalho, além de testagem mais extensiva, todas as funcionalidades básicas foram testadas, bem como o software com interface gráfica e a biblioteca de comunicação USB disponibilizada para futuras implementações.

A partir dos testes individuais, podemos fazer considerações de performance e comparar o funcionamento do dispositivo com outros, como o HX711, citado na revisão de estado da arte. Esse módulo é especializado para amplificação de células de carga e inclui um conversor analógico/digital. Dessa maneira, a frequência de amostragem é limitada (na transferência dos dados para o dispositivo mestre) a menos de 100Hz, o que impossibilita o processamento adicional do sinal. A Panda, por outro lado, ao integrar na mesma placa o circuito de amplificação e o microcontrolador, permite taxas de aquisição muito mais altas e, com isso, processamento do sinal. Além disso, o ganho do amplificador desenvolvido possui 256 configurações e atinge valores de até 1000 vezes. Ainda que se considere o conjunto HX711 com Arduino (uma vez que essa é a configuração utilizada em algumas das aplicações), certo esmero é necessário para a aquisição de dados com um computador, enquanto com a Panda, pouca ou nenhuma programação é necessária.

No que diz respeito à aquisição de forças e torques, o projeto pode ser comparado com as duas soluções comerciais referenciadas na revisão bibliográfica e do Estado da Arte. Tem-se um transdutor industrial F/t *6dof*, da HBM [11] e um sensor de torque reativo do tipo flange, da Honeywell [15]. O primeiro produto consegue captar forças e torques multiaxiais sem *crosstalk*, com precisão de 0,1%, além do grau de proteção ser IP67. Ele mede forças de 5kN até 200kN, e torques de 0,05kNm até 3,5kNm. Já o segundo

produto possui precisão de 0,15% e mede torques de até 2000 in-lb (aproximadamente 226 Nm). Por mais que o projeto aqui apresentado abranja uma faixa de valores abaixo da dessas soluções, pode-se ver que ele possui uma precisão competitiva, de 0,1% da máxima deformação. Ele é menos robusto, mas também é bem mais barato. Considerando a proposta para a qual a Panda foi criada, trata-se de uma placa que traz ótimos resultados para as funções que ela se propõe a realizar. No futuro, ela também poderia ser escalável para medir faixas de esforços maiores, com os devidos dimensionamentos.

Comparando com os projetos acadêmicos estudados, a Panda leva vantagem na precisão dos resultados. Ela é mais invasiva do que o aquisitor que estima valores com base no erro referente à velocidade do motor e do que o sistema virtual de instrumentação, mas é mais precisa e mais versátil. Seu ponto negativo com relação aos outros dois é a necessidade de células de carga, que podem variar de formato dependendo da aplicação. Ela também é mais robusta do que os projetos que utilizam Arduino, e pode até mesmo se comunicar com essas placas. Apesar de não possuir programa para celular, possui software com linguagem de alto nível para usuários menos experientes. Por fim, para o caso da aplicação médica em ambientes com altas frequências, ela também leva a vantagem por possuir filtro analógico e digital passa-baixas, trazendo um sinal resultante mais limpo.

8.2 Custos

Com o descrito na tabela 5 avaliamos que o custo do projeto se manteve no limite especificado de U$ 50,00. Ainda, parte dos componentes pode ser encomendada com a PCI, de forma que os preços são reduzidos ainda mais. Contando com os equipamentos (para montagem e testagem) já existentes nos laboratórios de aplicação do projeto, tem-se que a Panda é adequada, em termos de custo, para o uso didático pretendido.

9 CONCLUSÕES

Este trabalho teve como objetivo inicial trazer um estudo sobre equipamentos existentes para a captação de forças e torques dentro do contexto de robótica comentado, e propor um equipamento mais barato para a realização desta tarefa, mas que mantenha uma precisão alta, com o foco em ele ser utilizado dentro da universidade. Os sensores analisados foram principalmente das empresas HBM e Honeywell. Além disso foi realizado o estudo de trabalhos já existentes que buscam fazer essa medição de forma barata e inovadora, utilizando Arduino ou até mesmo um sistema virtual de instrumentação. Por fim, também buscou-se encontrar patentes que pudessem inspirar a criação de células de carga eficientes para as aplicações do projeto.

Os objetivos principais do trabalho focam em auxiliar a Escola Politécnica com a criação de um hardware que possa ser utilizado para a captação e o condicionamento de sinais de extensômetros. A partir dessa placa, os sinais adquiridos podem ser utilizados para obter informações sobre deformações, forças e torques exercidos por motores, exoesqueletos, balanças, entre outras aplicações. Esse objetivo foi alcançado com a criação dessa placa. Além disso, foi implementado um software para que os usuários possam interagir com a placa em alto nível e um Manual do Usuário associado a ele, facilitando o uso do projeto e trazendo maior acessibilidade para quem possui menor conhecimento em programação. Também foram projetados experimentos para validar o projeto, experimentos esses que podem ser replicados pelos alunos em disciplinas para entender melhor algumas matérias relacionadas com a Mecatrônica, além de entender melhor também o funcionamento do projeto. Como mais uma extensão, foi apresentada uma célula de carga que pode ser utilizada para algumas das aplicações citadas.

Foram propostas sete aplicações para o uso da placa, quatro delas em cinco disciplinas, duas em dois laboratórios e uma em um grupo de extensão. As aplicações para disciplinas envolvem o uso da placa para procedimentos experimentais, com o objetivo de consolidar o aprendizado dos alunos nas matérias. Em Instrumentação, é criada uma balança a partir da medição da deformação de uma barra. Para Sistemas Termofluidomecânicos, a construção de modelos em escala reduzida é beneficiada com o uso da placa para sua instrumentação e interação com o Arduino. No caso das duas disciplinas de Controle, a

placa pode ser utilizada para o controle de um módulo motorizado, ensinando aos alunos na prática como ocorre o equacionamento e o projeto de sistemas controlados. Em Projeto de Máquinas, a placa gera as curvas de torque dos motores trifásicos utilizados para a confecção de tornos e fresadoras CNC.

Para os laboratórios e o grupo de extensão, a placa possui como objetivo resolver problemas reais encontrados e facilitar a captação de sinais necessários para projetos. No laboratório de Biomecatrônica, a placa é utilizada na leitura do torque sendo exercido em exoesqueletos e aparelhos para fisioterapia, sendo possível realizar o controle deles. No laboratório de Engenharia Térmica e Ambiental, é feita a instrumentação de uma bancada de motor a vapor, para que seja possível ler termopares e informações sobre o motor. Por último, no grupo de extensão (Equipe ThundeRatz de Robótica), o projeto é importante para que seja feita a caracterização dos motores utilizados pela equipe.

O Projeto Básico consistiu na definição do projeto em cinco subsistemas: célula de carga, extensômetros, eletrônica, computador/interface e aplicação. Uma vez ocorrida essa divisão, tornou-se mais fácil analisar os requisitos de cada uma delas e a forma como cada parte poderia ser desenvolvida. Foi feita uma análise mecatrônica de cada subsistema, caracterizando-os nos tópicos de Mecânica, Computação, Elétrica e Controle. Em seguida foram levantados seus requisitos e por fim definiu-se a entrada e a saída dos subsistemas, possíveis soluções e qual delas foi escolhida. De forma resumida, pode-se dizer que o projeto percorre o seguinte ciclo: uma célula de carga é acoplada na aplicação e nela são colados extensômetros. Estes são conectados na eletrônica, que capta os sinais recebidos, filtra e amplifica, entregando para o computador um sinal tratado. O computador, por sua vez, serve de interface com o usuário, que pode acionar modos de Controle e gerar gráficos ou planilhas com os dados que deseja, comunicando-se com a placa e com a aplicação, que por sua vez continua realizando sua tarefa e interagindo com a célula de carga.

No momento do Projeto Detalhado, começou a determinação dos componentes e ferramentas necessários para a criação do que foi explicado. A placa, o programa e uma célula de carga foram projetados, analisando cada requisito que foi levantado. Conforme o andamento dessa criação, optou-se por adicionar outras funcionalidades à placa. Além de todos os pontos citados, ela também serve como uma plataforma para comunicação entre diversos protocolos: USB, I^2C, CAN e RS485. Assim, para as aplicações já consolidadas, não seria necessária a adaptação de seus protocolos para se comunicar com a placa. Além disso, ela poderia servir também como ponte entre dois equipamentos com protocolos diferentes. O resultado final do primeiro protótipo é visto na figura 41, sendo

também criado um *case* para sua alocação.

Enfim, foram fabricados os protótipos. Cinco placas foram feitas e cada parte dos circuitos foram testados. Cada conjunto e protocolo recebeu testes específicos para que fosse verificado seu correto funcionamento, e uma vez que todos foram validados, começaram os testes funcionais, ou seja, integrando duas ou mais funções em conjunto. Todos os testes foram documentados de modo que um usuário que queira replicar o projeto possa repetir os procedimentos para garantir todas as funcionalidades.

Para cada aplicação que foi proposta neste trabalho, os Estudos de Caso trataram de descrevê-la e esclarecer qual seria a melhor forma de desenvolvê-la. Para as aplicações que já foram desenvolvidas, também foram mostrados os resultados encontrados com os testes realizados e se o projeto funcionou como deveria.

Por fim, foram discutidos os resultados encontrados, comparando-os com os requisitos e com a ideia proposta. A discussão tratou de explicar como chegou-se a esses resultados e propor também formas de otimizar os processos utilizados.

9.1 Trabalhos Futuros

Devido ao contexto em que vive o mundo no momento desse trabalho, não foi possível testar o projeto implementando-o diretamente nas aplicações da Escola Politécnica, por mais que tenham sido criados protótipos que imitassem essas aplicações. Assim, um dos pontos para o futuro é o uso do projeto diretamente nos locais citados, para que dessa forma os professores e alunos possam já desfrutar do que foi criado. Além disso, foi projetada uma célula de carga que atende a algumas das aplicações, mas que não foi fabricada por restrições de acesso à oficina da faculdade no período de pandemia. Então outro ponto é a fabricação dessa célula de carga e também o projeto de outras, para que se tenha modelos que se apliquem a todos os casos citados.

Por fim, por mais que se tenha escolhido oito aplicações para a inclusão do projeto, a possibilidade de uso se estende a muitos outros exemplos. No futuro, pode ser que esse projeto ajude mais Laboratórios e disciplinas da Escola Politécnica, de outras Engenharias além da Mecatrônica e da Mecânica. Também, sendo um projeto *open-source*, os alunos podem estudar a forma como tudo foi criado e auxiliar na otimização de códigos e até mesmo na implementação de novas funcionalidades.

REFERÊNCIAS

[1] LOPES, G.; SANCHES, J. V. *Repositório do Projeto no Github*. 2020. Disponível em: <https://github.com/jvSanches/PANDA.git>.

[2] GRAU, A. et al. Industrial robotics in factory automation: From the early stage to the Internet of Things. In: *Proceedings IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*. [S.l.: s.n.], 2017. ISBN 9781538611272.

[3] GARCIA, E. et al. The evolution of robotics research. *IEEE Robotics and Automation Magazine*, v. 14, n. 1, p. 90–103, 2007. ISSN 10709932.

[4] MORRIS, A. S. *Measurement and Instrumentation Principles*. 3rd. ed. [S.l.]: Butterworth-Heinemann, 2001.

[5] FRANK, R. *Understanding Smart Sensors*. 2nd. ed. [S.l.]: Artech House, 2000. 320 p. ISSN 02602288. ISBN 0890063117.

[6] HERNANDEZ, W. Improving the response of a load cell by using optimal filtering. *Sensors*, v. 6, n. 7, p. 697–711, 2006. ISSN 14248220.

[7] MAGRINI, E.; FLACCO, F.; De Luca, A. Control of generalized contact motion and force in physical human-robot interaction. *Proceedings - IEEE International Conference on Robotics and Automation*, IEEE, v. 2015-June, n. June, p. 2298–2304, 2015. ISSN 10504729.

[8] AZAM, A. F. N. et al. High performance Torque Control of BLDC motor. *2013 International Conference on Electrical Machines and Systems, ICEMS 2013*, p. 1093–1098, 2013.

[9] ZHAO, L.; ZHANG, X.; JI, J. A torque control strategy of brushless direct current motor with current observer. *2015 IEEE International Conference on Mechatronics and Automation, ICMA 2015*, IEEE, p. 303–307, 2015.

[10] GRAVEL, D. P.; NEWMAN, W. S. Flexible robotic assembly efforts at Ford Motor Company. *IEEE International Symposium on Intelligent Control - Proceedings*, p. 173–182, 2001.

[11] HBM. *Sensor Multiaxial MCS10 - Medição de 6 Forças e Momentos*. 2019. Disponível em: <https://www.hbm.com/pt/5626/sensor-multiaxial-mcs10/>.

[12] TSETSERUKOU, D. et al. Development of a whole-sensitive teleoperated robot arm using torque sensing technique. *Proceedings - Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, World Haptics 2007*, p. 476–481, 2007.

[13] CHOI, S. Y. et al. Development of joint torque sensor applied to compensate crosstalk error. *IEEE International Conference on Automation Science and Engineering*, IEEE, p. 1086–1088, 2012. ISSN 21618070.

[14] KIM, I. M.; KIM, H. S.; SONG, J. B. Design of joint torque sensor and joint structure of a robot arm to minimize crosstalk and torque ripple. *2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence, URAI 2012*, IEEE, p. 404–407, 2012.

[15] HONEYWELL. *2110-2K Flanged Reaction Torque Transducers*. 2019. Disponível em: <https://sensing.honeywell.com/2110-2k-flanged-reaction-torque-transducers>.

[16] DOEBELIN, E. O. *Measurement Systems, Application and Design*. 4th. ed. [S.l.]: McGraw-Hill, 1990. ISBN 0-07-017338-9.

[17] Dassault Systems. *Abaqus CAE - SIMULA^{TM} by Dassault Systèmes®*. 2018. Disponível em: <https://www.3ds.com/products-services/simulia/products/abaqus/abaquscae/>.

[18] FRADEN, J. *Handbook of Modern Sensors*. 3rd. ed. [S.l.]: Springer, 1994. ISSN 0031-9228. ISBN 0387007504.

[19] HBM. *The Working Principle of a Compression Load Cell — HBM*. 2019. Disponível em: <https://www.hbm.com/en/7325/the-working-principle-of-a-compression-load-cell/>.

[20] KITCHIN, C.; COUNTS, L. *A Designer's Guide To Instrumentation Amplifiers*. 3rd. ed. [S.l.]: Analog Devices, 2006. ISSN 00127515.

[21] YIU, J.; JOHNSON, I. The Many Ways of Programming an ARM Cortex -M Microcontroller. *ARM White Papers*, p. 1–19, 2013.

[22] KHURANA, G.; GOYAL, U. An Insight Comparison of Serial Communication Protocols. *International Journal of Advanced Research in Computer Science and Electronics Engineering*, v. 2, n. 3, p. 308–313, 2013.

[23] Merabet, A.; Tanvir, A. A.; Beddek, K. Torque and state estimation for real-time implementation of multivariable control in sensorless induction motor drives. *IET Electric Power Applications*, v. 11, n. 4, p. 653–663, 2017.

[24] Jimenez, L. et al. Virtual instrumentation system to automatically determine the stress-strain curve of tensile test. In: *2018 IEEE International Conference on Automation/XXIII Congress of the Chilean Association of Automatic Control (ICA-ACCA)*. [S.l.: s.n.], 2018. p. 1–6.

[25] NATIONAL INSTRUMENTS. *Compact FieldPoint – Programmable Automation Controller*. 2020. Disponível em: <http://sine.ni.com/nips/cds/view/p/lang/pt/nid/11560>.

[26] SILVA, J. A. B. L. P. e. et al. Desenvolvimento de sistema de baixo custo para monitoramento de integridade estrutural. *MatÃ(Rio de Janeiro)*, scielo, v. 24, 00 2019. ISSN 1517-7076.

[27] SANTINI, F.; BIERI, O.; DELIGIANNI, X. Openforce mr: A low-cost open-source mr-compatible force sensor. *Concepts in Magnetic Resonance Part B: Magnetic Resonance Engineering*, v. 48B, p. e21404, 02 2019.

[28] Noh, Y. et al. A2-piece six-axis force/torque sensor capable of measuring loads applied to tools of complex shapes. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.: s.n.], 2019. p. 7976–7981.

[29] KIM, B. et al. *Force torque sensor, force torque sensor frame, and force torque measurement method*. 2016.

[30] HBM. *HBM strain gauges*. 2020. Disponível em: <https://www.hbm.com/pt/2073/catalogo-pdf-strain-gauge/>.

[31] EXCEL. *Extensômetros coláveis de resistência elétrica*. 2020. Disponível em: <https://excelsensor.com.br/extensometros-colaveis-de-resistencia-eletrica/modelos/>.

[32] OMEGA. *Extensômetros*. 2020. Disponível em: <https://br.omega.com/guides/straingages.html>.

[33] MICRO-MEASUREMENTS. *Tech Note TN-505-6*. [S.l.], 2018.

[34] IEEE Guide on Shielding Practice for Low Voltage Cables. *IEEE Std 1143-1994*, p. 1–88, 1995.

[35] MD POLICABOS. *Cabo Manga Blindado*. 2020. Disponível em: <https://www.mdpolicabos.com/cabo-manga-blindado/>.

[36] CROMATEK. *Conector Borne CTK 2EDGK 3*. 2020. Disponível em: <https://cromatek.com.br/conector-borne-ctk-2edgk-3/>.

[37] GABOIAN, J. A survey of common-mode noise. *Application Report*, Texas Instruments Incorporated, n. SLLLA057, 1999.

[38] KARKI, J. Active low-pass filter design. *Application Report*, Texas Instruments Incorporated, n. SLOA049B, 2002.

[39] TEXAS INSTRUMENTS. *TLC14ID*. 2020. Disponível em: <https://www.ti.com/store/ti/en/p/product/?p=TLC14ID>.

[40] MAXIM INNTEGRATED. *MAX291*. 2020. Disponível em: <https://www.maximintegrated.com/en/products/analog/analog-filters/MAX291.html>.

[41] ANALYSIS of the Sallen-Key Architecture. *Application Report*, Texas Instruments Incorporated, n. SLOA024B, 1999.

[42] Microchip Technology Inc. *FilterLab Filter Design Software*. 2020. Disponível em: <https://www.microchip.com/developmenttools/ProductDetails/filterlabdesignsoftware>.

[43] BAKER, B. C. Select the right operational amplifier for your filtering circuits. *ANALOG DESIGN NOTE*, Microchip Technology Inc., n. ADN003, 2003.

[44] MICROCHIP TECHNOLOGY INC. *MCP6002*. 2020. Disponível em: <https://www.microchip.com/wwwproducts/en/MCP6002>.

[45] Microchip Technology Inc. *MPLAB® Mindi™ Analog Simulator*. 2020. Disponível em: <https://www.microchip.com/mplab/mplab-mindi>.

[46] MICROCHIP TECHNOLOGY INC. *MCP6V02*. 2020. Disponível em: <https://www.microchip.com/wwwproducts/en/MCP6V02>.

[47] MICROCHIP TECHNOLOGY INC. *MCP6022*. 2020. Disponível em: <https://www.microchip.com/wwwproducts/en/MCP6022>.

[48] PALMER, R. Dc parameters: Input offset voltage (vio). *Application Report*, Texas Instruments Incorporated, n. SLOA059, 2001.

[49] WEILER, A.; PAKOSTA, A.; VERMA, A. High-speed layout guidelines. *Application Report*, Texas Instruments Incorporated, n. SCAA082A, 2006.

[50] TEXAS INSTRUMENTS. *INA333*. 2020. Disponível em: <https://www.ti.com/product/INA333>.

[51] Texas Instruments. *SPICE-based analog simulation program TINA-TI*. 2020. Disponível em: <http://www.ti.com/tool/TINA-TI>.

[52] WELLS, C.; BECKER, J. Low-cost digital programmable gain amplifier reference design. *TI Designs — Precision: Verified Design*, Texas Instruments Incorporated, n. TIDUB13-, 2015.

[53] MICROCHIP TECHNOLOGY INC. *MCP41010*. 2020. Disponível em: <https://www.microchip.com/wwwproducts/en/MCP41010>.

[54] NXP SEMICONDUCTORS. *Kinetis® K Series*. 2020. Disponível em: <https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/k-series-cortex-m4>.

[55] STMICROELECTRONICS. *STM32F0 Series*. 2020. Disponível em: <https://www.st.com/en/microcontrollers-microprocessors/stm32f0-series.html>.

[56] STMICROELECTRONICS. *STM32F072C8*. 2020. Disponível em: <https://www.st.com/en/microcontrollers-microprocessors/stm32f072c8.html>.

[57] TEXAS INSTRUMENTS. *SN65HVD230DR*. 2020. Disponível em: <https://www.ti.com/store/ti/en/p/product/?p=SN65HVD230DR>.

[58] EXAR CORPORATION. *SP3485*. 2020. Disponível em: <https://www.maxlinear.com/product/interface/serial-transceivers/rs485-422/sp3485>.

[59] VISHAY INTERTECNOLOGY. *SS32, SS33, SS34, SS35, SS36 PRODUCT INFORMATION*. 2020. Disponível em: <https://www.vishay.com/diodes/list/product-88751/>.

[60] TEXAS INSTRUMENTS. *LM1117*. 2020. Disponível em: <https://www.ti.com/product/LM1117>.

[61] TEXAS INSTRUMENTS. *TLV1117*. 2020. Disponível em: <https://www.ti.com/product/TLV1117>.

[62] TEXAS INSTRUMENTS. *REF3033*. 2020. Disponível em: <https://www.ti.com/product/REF3033>.

[63] TEXAS INSTRUMENTS. *TLV740P*. 2020. Disponível em: <https://www.ti.com/product/TLV740P>.

[64] STMICROELECTRONICS. *Discovery kit with STM32F072RB MCU*. 2020. Disponível em: <https://www.st.com/en/evaluation-tools/32f072bdiscovery.html>.

[65] STMICROELECTRONICS. *STM32Cube initialization code generator*. 2020. Disponível em: <https://www.st.com/en/development-tools/stm32cubemx.html>.

[66] ARM LIMITED. *GNU Arm Embedded Toolchain*. 2020. Disponível em: <https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm>.

[67] STMICROELECTRONICS. *STM32CubeProgrammer software for all STM32*. 2020. Disponível em: <https://www.st.com/en/development-tools/stm32cubeprog.html>.

[68] ORGANIZATION, M. Modbus application protocol specification. *Application Report*, MODBUS Organization, n. V1.1b3, 2012.

[69] THOMAS, G. Introduction to modbus serial and modbus tcp. *Contemporary Control Systems*, v. 9, n. 4, 2008.

[70] MODBUS.ORG. *MODBUS over serial line specification and implementation guide V1.02*. 2006. Disponível em: <https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf>.

[71] CANOPEN. *CANopen – The standardized embedded network*. 2020. Disponível em: <https://www.can-cia.org/canopen/>.

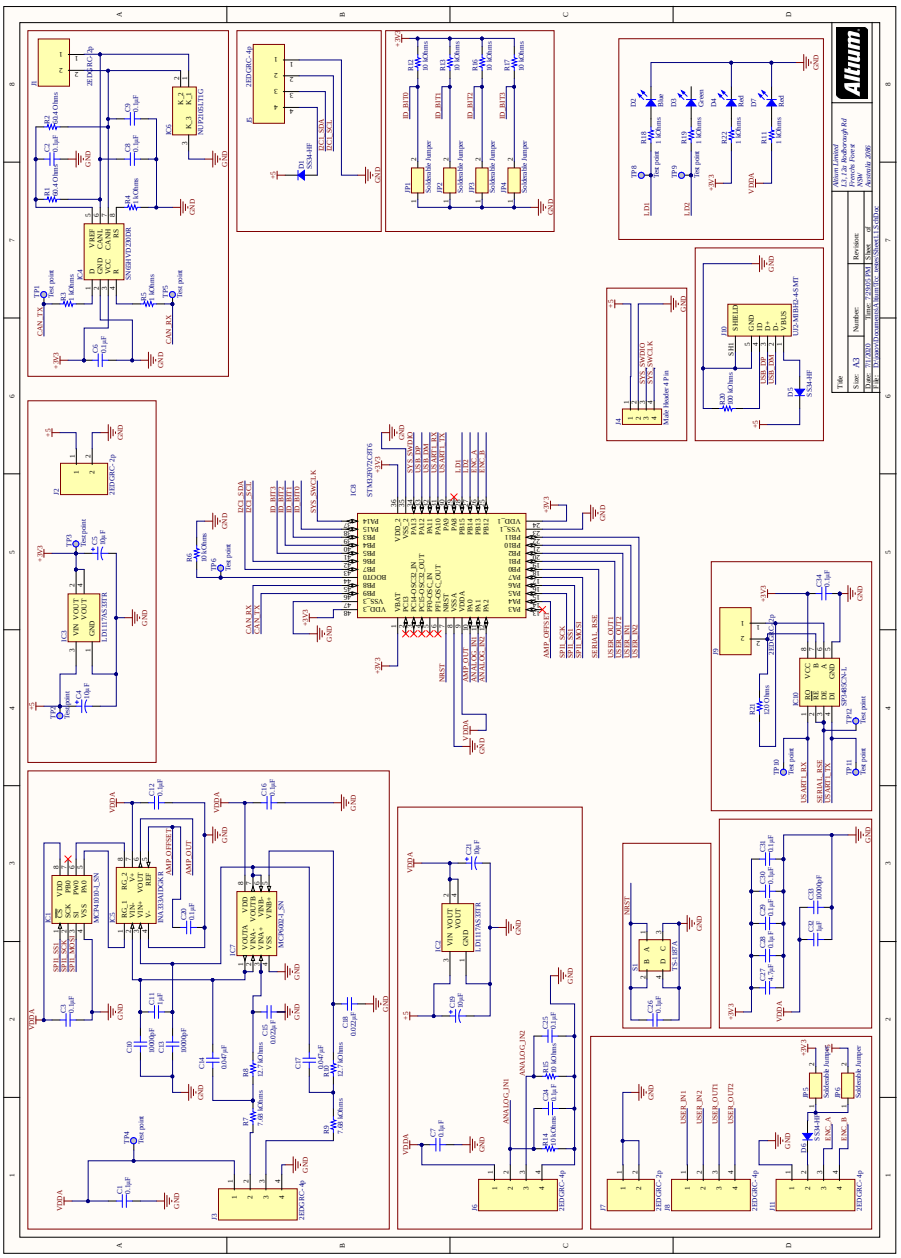
[72] MYERS, P. Interfacing using serial protocols using spi and i2c. *Proc. ESP*, Citeseer, v. 2005, p. 1–9, 2007.

[73] JLCPCB. *PCB Prototype PCB Fabrication Manufacturer*. 2020. Disponível em: <https://jlcpcb.com>.

[74] MOUSER ELECTRONICS, INC. *Electronic Components Distributor*. 2020. Disponível em: <https://br.mouser.com/>.

APÊNDICE A – ESQUEMÁTICO ELETRÔNICO

Figura 46: Desenho esquemático do circuito. Disponível em [1].



Fonte: elaborado pelos autores.

APÊNDICE B – CÓDIGOS DO SOFTWARE

```
#-------------------------------------------------------------------------

# Software da PANDA
# TCC - Engenharia Mecatrônica - Poli USP
# Guilherme de Agrela Lopes
# João Vitor Sanches
#
# MAIN FRAME
#
# Arquivos:
# MainFrame: Tela principal; chamada das outras classes
# CalibrationFrame: Tela para realizar a calibração da placa; criada ao clicar no botão 'Modo de Calibração' da tela principal
# PlotFrame: Telas com os gráficos gerados a partir das seleções na tela de Configuração; criadas ao clicar no botão 'Mostrar Gráficos'
# SettingsFrame: Tela para escolher as configurações desejadas do programa; criada ao clicar no botão 'Configurações'
# PandaDialogs: Diálogos de erro relacionados com a PANDA
# panda_board: Interação do software com o firmware da PANDA

#-------------------------------------------------------------------------

import wx
from wx.lib import plot as wxplot
from wx.lib.agw import hyperlink as wxhyperlink

from SettingsFrame import SettingsFrame
from CalibrationFrame import CalibrationFrame
from PlotFrame import PlotFrame
import PandaDialogs
from PandaDialogs import myPanda


class MainFrame(wx.Frame):
    """
    This is the Main Frame. It's the first screen the user will see, where
    they can decide what to do.
    """
    def __init__(self, parent, title):
        wx.Frame.__init__(self, parent, -1, title,
                          pos=(350, 350), size=(780, 280))

        # Creates the Panel to put the other controls on
        panel = wx.Panel(self)

        # Controls
        text = wx.StaticText(panel, -1, "Ois! Este é o software da placa PANDA. Por favor, selecione e salve as configurações desejadas. \nCaso seja sua primeira vez aqui, clique no Manual do Usuário.", pos=(15,15))
        text.SetFont(wx.Font(12, wx.SWISS, wx.NORMAL, wx.BOLD))
        text.SetSize(text.GetBestSize())
        lnk = wxhyperlink.HyperLinkCtrl(panel, -1, "Manual do Usuário", URL="https://github.com/jvSanches/PANDA/blob/master/README.md", pos=(325,150))
        btn_settings = wx.Button(panel, -1, "Configurações", pos=(75,90))
        btn_calibration = wx.Button(panel, -1, "Modo de Calibração", pos=(220,90))
        btn_graphics = wx.Button(panel, -1, "Mostrar Gráficos", pos=(395,90))
```

```
51          btn_sendControlValues = wx.Button(panel, -1, "Valores do Controle", pos=(550,90))
52
53          # Binds the controls to handlers
54          self.Bind(wx.EVT_BUTTON, self.OnSettingsButton, btn_settings)
55          self.Bind(wx.EVT_BUTTON, self.OnCalibrationButton, btn_calibration)
56          self.Bind(wx.EVT_BUTTON, self.onGraphicsButton, btn_graphics)
57          self.Bind(wx.EVT_BUTTON, self.OnSendControlButton, btn_sendControlValues)
58
59          panel.Layout()
60
61          # Initial value of arrays
62          self.settings = ['0', False, False, False, False, False, '750', False, False, False, False, 0.0015, 4000, 45,
     0, 0]
63          self.calibration = [True, False, False, '0', '0', '0', 0]
64
65
66      def OnTimeToClose(self, evt):
67          """Event handler for closing."""
68          myPanda.disconnect()
69          self.Close()
70
71
72      def OnSettingsButton(self, evt):
73          """Event handler for the Settings button."""
74          settings_frame = SettingsFrame(self.settings, parent=wx.GetTopLevelParent(self), title="Configurações")
75          settings_frame.Show()
76
77
78      def OnCalibrationButton(self, evt):
79          """Event handler for the Calibration button."""
80          cal = CalibrationFrame(self.calibration, parent=wx.GetTopLevelParent(self), title="Modo de Calibração")
81          res = cal.Show()
82
83
84      def onGraphicsButton(self, evt):
85          """Event handler for the Graphics button."""
86          if myPanda.exists():
87              if self.settings[1] or self.settings[2] or self.settings[3] or self.settings[4] or self.settings[5]:
88                  if self.settings[15] == wx.ID_OK:
89                      myPanda.setGain(self.settings[6])
90                      self.settings[15] = 0
91
92
93                  if self.calibration[6] == wx.ID_OK:
94                      if self.calibration[0]:
95                          myPanda.runAutoOffset(int(2048*(((1/3.3)*(self.calibration[3]/self.settings[11])) + 1)))
96                      if self.calibration[1]:
97                          myPanda.runAutoOffset(int(2048*(((1/3.3)*(self.calibration[4]/self.settings[12])) + 1)))
98                      if self.calibration[2]:
99                          myPanda.runAutoOffset(2048*(((1/3.3)*(self.calibration[5]/self.settings[13])) + 1))
100
101             if self.settings[5]:
102                 if self.settings[7]:
103                     plot = PlotFrame('force/torque', self.settings[11], self.settings[12], self.settings[13], self.
     settings[14], parent=wx.GetTopLevelParent(self), title="Gráfico da Força / Torque")
104                     res = plot.Show()
105                 if self.settings[8]:
106                     plot = PlotFrame('stress', self.settings[11], self.settings[12], self.settings[13], self.settings
     [14], parent=wx.GetTopLevelParent(self), title="Gráfico da Tensão")
107                     res = plot.Show()
108                 if self.settings[9]:
109                     plot = PlotFrame('strain', self.settings[11], self.settings[12], self.settings[13], self.settings
     [14], parent=wx.GetTopLevelParent(self), title="Gráfico da Deformação")
110                     res = plot.Show()
111                 if self.settings[10]:
112                     plot = PlotFrame('power', self.settings[11], self.settings[12], self.settings[13], self.settings
     [14], parent=wx.GetTopLevelParent(self), title="Gráfico da Potência")
113                     res = plot.Show()
114             if (not self.settings[1]) and (not self.settings[2]) and (not self.settings[3]) and (not self.settings[4])
     and (not self.settings[5]):
115                 noGraphicSelected_dialog = NoGraphicSelected(None, "Aviso")
116                 res = noGraphicSelected_dialog.ShowModal()
117                 if res == wx.ID_OK:
118                     noGraphicSelected_dialog.Destroy()
```

```
119         else:
120             pandaNotDetected_dialog = PandaDialogs.PandaNotDetected(None, "Aviso")
121             res = pandaNotDetected_dialog.ShowModal()
122             if res == wx.ID_OK:
123                 pandaNotDetected_dialog.Destroy()
124 
125 
126     def OnSendControlButton(self, evt):
127         """Event handler for the Control button."""
128         notImplemented_dialog = NotYetImplemented(None, "Aviso")
129         res = notImplemented_dialog.ShowModal()
130         if res == wx.ID_OK:
131             notImplemented_dialog.Destroy()
132 
133 
134 class NoGraphicSelected(wx.Dialog):
135     """
136     This is a Dialog. It's opened when the user tries to open graphics
137     but haven't checked any box in Settings.
138     """
139     def __init__(self, parent, title):
140         wx.Dialog.__init__(self, parent, -1, title,
141                            pos=(650, 450), size=(500, 150))
142 
143         self.panel = wx.Panel(self)
144         self.controlNotActiveText = wx.StaticText(self.panel, -1, label="Não foi selecionado nenhum gráfico. Por favor,
       selecione algum nas Configurações.", pos=(15,15))
145         self.button_ok = wx.Button(self.panel, label="OK", pos=(210, 70))
146         self.button_ok.Bind(wx.EVT_BUTTON, self.onOk)
147 
148         self.panel.Layout()
149 
150     def onOk(self, e):
151         """Event handler for the Ok button."""
152         self.EndModal(wx.ID_OK)
153 
154 
155 class NotYetImplemented(wx.Dialog):
156     """
157     This is a Dialog. It's opened when the user tries to open a frame
158     that hasn't yet been implemented.
159     """
160     def __init__(self, parent, title):
161         wx.Dialog.__init__(self, parent, -1, title,
162                            pos=(650, 450), size=(500, 150))
163 
164         self.panel = wx.Panel(self)
165         self.controlNotActiveText = wx.StaticText(self.panel, -1, label="Ainda será implementado!", pos=(15,15))
166         self.button_ok = wx.Button(self.panel, label="OK", pos=(210, 70))
167         self.button_ok.Bind(wx.EVT_BUTTON, self.onOk)
168 
169         self.panel.Layout()
170 
171     def onOk(self, e):
172         """Event handler for the Ok button."""
173         self.EndModal(wx.ID_OK)
174 
175 
176 class MyApp(wx.App):
177     """
178     Class to start the program.
179     """
180     def OnInit(self):
181         frame = MainFrame(None, "PANDA Software")
182         self.SetTopWindow(frame)
183         frame.Show(True)
184         return True
185 
186 
187 # Runs the program
188 app = MyApp(redirect=True)
189 app.MainLoop()
```

```
190 wx.Exit()
```

Listagem B.1: Software: MainFrame.py.

```
1  #-----------------------------------------------------------------------
2
3  # Software da PANDA
4  # TCC - Engenharia Mecatrônica - Poli USP
5  # Guilherme de Agrela Lopes
6  # João Vitor Sanches
7  #
8  # SETTINGS FRAME
9  #
10 # Arquivos:
11 # MainFrame: Tela principal; chamada das outras classes
12 # CalibrationFrame: Tela para realizar a calibração da placa; criada ao clicar no botão 'Modo de Calibração' da tela
       principal
13 # PlotFrame: Telas com os gráficos gerados a partir das seleções na tela de Configuração; criadas ao clicar no botão '
       Mostrar Gráficos'
14 # SettingsFrame: Tela para escolher as configurações desejadas do programa; criada ao clicar no botão 'Configurações'
15 # PandaDialogs: Diálogos de erro relacionados com a PANDA
16 # panda_board: Interação do software com o firmware da PANDA
17
18 #-----------------------------------------------------------------------
19
20 import wx
21
22 class SettingsFrame(wx.Frame):
23     """
24     This is the Settings Frame.  It shows all the options the user has:
25     generate graphics, set amplifier gain, recalibrate the board or
26     choose Control modes.
27     """
28     def __init__(self, settings, parent, title):
29         wx.Frame.__init__(self, parent, -1, title,
30                           pos=(450, 150), size=(600, 690))
31
32         # Creates the Panel to put the other controls on
33         self.panel = wx.Panel(self)
34
35         # Initializes settings array
36         self.settings = settings
37
38         # Controls
39         self.text = wx.StaticText(self.panel, -1, label="Selecione as configurações desejadas.", pos=(15, 5))
40         self.text.SetFont(wx.Font(12, wx.SWISS, wx.NORMAL, wx.BOLD))
41         self.text.SetSize(self.text.GetBestSize())
42
43         self.btn_controlMode = wx.ToggleButton(self.panel, -1, label="Modo Controle", pos=(15, 35))
44         self.btn_controlP = wx.RadioButton(self.panel, -1, label="Controle P", style=wx.RB_GROUP, pos=(130, 40))
45         self.btn_controlPI = wx.RadioButton(self.panel, -1, label="Controle PI", pos=(130, 70))
46         self.btn_controlPD = wx.RadioButton(self.panel, -1, label="Controle PD", pos=(130, 100))
47         self.btn_controlPID = wx.RadioButton(self.panel, -1, label="Controle PID", pos=(130, 130))
48
49         self.graphicsText = wx.StaticText(self.panel, -1, label="Mostrar Gráficos", pos=(15, 190))
50         self.btn_showPolesZeros = wx.CheckBox(self.panel, -1, label="Gráfico de Polos e Zeros", pos=(130, 190))
51         self.btn_showStepAnswer = wx.CheckBox(self.panel, -1, label="Resposta a Degrau", pos=(300, 190))
52         self.btn_showRampAnswer = wx.CheckBox(self.panel, -1, label="Resposta a Rampa", pos=(440, 190))
53         self.btn_showTorqueCurve = wx.CheckBox(self.panel, -1, label="Curva de Torque Caracter stica", pos=(130, 220))
54         self.btn_showValuesGraphic = wx.CheckBox(self.panel, -1, label="Gráficos dos Valores", pos=(330, 220))
55
56         self.amplifierGainText = wx.StaticText(self.panel, -1, label="Ganho do Amplificador", pos=(15, 270))
57         self.btn_amplifierGain = wx.SpinCtrl(self.panel, value='750', pos=(150, 270), min=0, max=1000)
58
59         self.valuesText = wx.StaticText(self.panel, -1, label="Mostrar Valores", pos=(15, 320))
60         self.btn_forces = wx.CheckBox(self.panel, -1, label="Forças/Torques", pos=(130, 320))
61         self.btn_stress = wx.CheckBox(self.panel, -1, label="Tensões", pos=(280, 320))
62         self.btn_strains = wx.CheckBox(self.panel, -1, label="Deformações", pos=(130, 350))
63         self.btn_powers = wx.CheckBox(self.panel, -1, label="Potências", pos=(280, 350))
64
65         self.constantValuesText = wx.StaticText(self.panel, -1, label="Digite as constantes para cada valor:", pos=(15,
       400))
66         self.strainConstantText = wx.StaticText(self.panel, -1, label="Ks", pos=(130, 430))
```

```
67          self.btn_strainConstant = wx.TextCtrl(self.panel, pos=(150, 430))
68          self.forceConstantText = wx.StaticText(self.panel, -1, label="Kf", pos=(130, 460))
69          self.btn_forceConstant = wx.TextCtrl(self.panel, pos=(150, 460))
70          self.stressConstantText = wx.StaticText(self.panel, -1, label="Kt", pos=(130, 490))
71          self.btn_stressConstant = wx.TextCtrl(self.panel, pos=(150, 490))
72          self.powerConstantText = wx.StaticText(self.panel, -1, label="Kp", pos=(130, 520))
73          self.btn_powerConstant = wx.TextCtrl(self.panel, pos=(150, 520))
74
75          self.btn_saveSettings = wx.Button(self.panel, -1, label="Salvar Configurações", pos=(130, 600))
76          self.btn = wx.Button(self.panel, -1, "Cancelar", pos=(330, 600))
77
78          # Binds the controls to handlers
79          self.Bind(wx.EVT_TOGGLEBUTTON, self.ToggleControl, self.btn_controlMode)
80          self.Bind(wx.EVT_RADIOBUTTON, self.OnSelectControlP, self.btn_controlP)
81          self.Bind(wx.EVT_RADIOBUTTON, self.OnSelectControlPI, self.btn_controlPI)
82          self.Bind(wx.EVT_RADIOBUTTON, self.OnSelectControlPD, self.btn_controlPD)
83          self.Bind(wx.EVT_RADIOBUTTON, self.OnSelectControlPID, self.btn_controlPID)
84
85          self.Bind(wx.EVT_CHECKBOX, self.OnCheckControlBoxes, self.btn_showPolesZeros)
86          self.Bind(wx.EVT_CHECKBOX, self.OnCheckControlBoxes, self.btn_showStepAnswer)
87          self.Bind(wx.EVT_CHECKBOX, self.OnCheckControlBoxes, self.btn_showRampAnswer)
88
89          self.Bind(wx.EVT_BUTTON, self.OnSaveSettingsButton, self.btn_saveSettings, id=wx.ID_OK)
90          self.Bind(wx.EVT_BUTTON, self.OnCancel, self.btn)
91
92          # Prepare Frame based on settings array values
93          if self.settings[0] == '0':
94              self.btn_controlMode.SetValue(False)
95              self.btn_controlP.Disable()
96              self.btn_controlPI.Disable()
97              self.btn_controlPD.Disable()
98              self.btn_controlPID.Disable()
99
100         elif self.settings[0] == 'P':
101             self.btn_controlMode.SetValue(True)
102             self.btn_controlP.SetValue(True)
103             self.btn_controlPD.SetValue(False)
104             self.btn_controlPI.SetValue(False)
105             self.btn_controlPID.SetValue(False)
106         elif self.settings[0] == 'PD':
107             self.btn_controlMode.SetValue(True)
108             self.btn_controlP.SetValue(False)
109             self.btn_controlPD.SetValue(True)
110             self.btn_controlPI.SetValue(False)
111             self.btn_controlPID.SetValue(False)
112         elif self.settings[0] == 'PI':
113             self.btn_controlMode.SetValue(True)
114             self.btn_controlP.SetValue(False)
115             self.btn_controlPD.SetValue(False)
116             self.btn_controlPI.SetValue(True)
117             self.btn_controlPID.SetValue(False)
118         elif self.settings[0] == 'PID':
119             self.btn_controlMode.SetValue(True)
120             self.btn_controlP.SetValue(False)
121             self.btn_controlPD.SetValue(False)
122             self.btn_controlPI.SetValue(False)
123             self.btn_controlPID.SetValue(True)
124
125         self.controlMode = self.settings[0]
126         self.btn_showPolesZeros.SetValue(self.settings[1])
127         self.btn_showStepAnswer.SetValue(self.settings[2])
128         self.btn_showRampAnswer.SetValue(self.settings[3])
129         self.btn_showTorqueCurve.SetValue(self.settings[4])
130         self.btn_showValuesGraphic.SetValue(self.settings[5])
131
132         self.btn_amplifierGain.SetValue(self.settings[6])
133
134         self.btn_forces.SetValue(self.settings[7])
135         self.btn_stress.SetValue(self.settings[8])
136         self.btn_strains.SetValue(self.settings[9])
137         self.btn_powers.SetValue(self.settings[10])
138
139         self.btn_strainConstant.SetValue(str(self.settings[11]))
140         self.btn_forceConstant.SetValue(str(self.settings[12]))
```

```
141         self.btn_stressConstant.SetValue(str(self.settings[13]))
142         self.btn_powerConstant.SetValue(str(self.settings[14]))
143 
144         self.panel.Layout()
145 
146 
147     def OnCancel(self, evt):
148         """Event handler for closing."""
149         self.settings[15] = wx.ID_CANCEL
150         self.Close()
151 
152     def ToggleControl(self, evt):
153         """Event handler for the Control Toggle Button."""
154         obj = evt.GetEventObject()
155         isPressed = obj.GetValue()
156 
157         if isPressed:
158             self.btn_controlP.Enable()
159             self.btn_controlPI.Enable()
160             self.btn_controlPD.Enable()
161             self.btn_controlPID.Enable()
162             if self.btn_controlP.GetValue():
163                 self.controlMode = 'P'
164             if self.btn_controlPD.GetValue():
165                 self.controlMode = 'PD'
166             if self.btn_controlPI.GetValue():
167                 self.controlMode = 'PI'
168             if self.btn_controlPID.GetValue():
169                 self.controlMode = 'PID'
170 
171         else:
172             self.btn_controlP.Disable()
173             self.btn_controlPI.Disable()
174             self.btn_controlPD.Disable()
175             self.btn_controlPID.Disable()
176             self.controlMode = '0'
177             self.btn_showPolesZeros.SetValue(False)
178             self.btn_showStepAnswer.SetValue(False)
179             self.btn_showRampAnswer.SetValue(False)
180 
181 
182     def OnSelectControlP(self, evt):
183         """Event handler for the P control Radio Button."""
184         self.controlMode = 'P'
185 
186     def OnSelectControlPD(self, evt):
187         """Event handler for the PD control Radio Button."""
188         self.controlMode = 'PD'
189 
190     def OnSelectControlPI(self, evt):
191         """Event handler for the PI control Radio Button."""
192         self.controlMode = 'PI'
193 
194     def OnSelectControlPID(self, evt):
195         """Event handler for the PID control Radio Button."""
196         self.controlMode = 'PID'
197 
198 
199     def OnCheckControlBoxes(self, evt):
200         """Event handler for the Checkboxes."""
201         if not self.btn_controlMode.GetValue():
202             controlNotActive_dialog = ControlNotActive(None, "Aviso")
203             res = controlNotActive_dialog.ShowModal()
204             if res == wx.ID_OK:
205                 controlNotActive_dialog.Destroy()
206 
207             self.btn_showPolesZeros.SetValue(False)
208             self.btn_showStepAnswer.SetValue(False)
209             self.btn_showRampAnswer.SetValue(False)
210 
211 
212     def OnSaveSettingsButton(self, evt):
213         """Event handler for the Save button."""
214         self.settings[0] = self.controlMode
```

```
215         self.settings[1] = self.btn_showPolesZeros.GetValue()
216         self.settings[2] = self.btn_showStepAnswer.GetValue()
217         self.settings[3] = self.btn_showRampAnswer.GetValue()
218         self.settings[4] = self.btn_showTorqueCurve.GetValue()
219         self.settings[5] = self.btn_showValuesGraphic.GetValue()
220         self.settings[6] = self.btn_amplifierGain.GetValue()
221         self.settings[7] = self.btn_forces.GetValue()
222         self.settings[8] = self.btn_stress.GetValue()
223         self.settings[9] = self.btn_strains.GetValue()
224         self.settings[10] = self.btn_powers.GetValue()
225         self.settings[11] = float(self.btn_strainConstant.GetValue())
226         self.settings[12] = float(self.btn_forceConstant.GetValue())
227         self.settings[13] = float(self.btn_stressConstant.GetValue())
228         self.settings[14] = float(self.btn_powerConstant.GetValue())
229         self.settings[15] = wx.ID_OK
230 
231         self.Close()
232 
233 
234     def GetSettings(self):
235         """Returns settings array when asked."""
236         return self.settings
237 
238 
239 class ControlNotActive(wx.Dialog):
240     """
241     This is a Dialog. It's opened when the user tries to select Control graphics
242     when Control mode is not active.
243     """
244     def __init__(self, parent, title):
245         wx.Dialog.__init__(self, parent, -1, title,
246                            pos=(650, 450), size=(500, 150))
247 
248         self.panel = wx.Panel(self)
249         self.controlNotActiveText = wx.StaticText(self.panel, -1, label="Não é poss vel selecionar gráficos
      relacionados com Controle se ele não estiver ativo.", pos=(15,15))
250         self.button_ok = wx.Button(self.panel, label="OK", pos=(210, 70))
251         self.button_ok.Bind(wx.EVT_BUTTON, self.onOk)
252 
253         self.panel.Layout()
254 
255     def onOk(self, e):
256         """Event handler for the Ok button."""
257         self.EndModal(wx.ID_OK)
```

Listagem B.2: Software: SettingsFrame.py.

```
1  #-----------------------------------------------------------------------
2  
3  # Software da PANDA
4  # TCC - Engenharia Mecatrônica - Poli USP
5  # Guilherme de Agrela Lopes
6  # João Vitor Sanches
7  #
8  # MAIN FRAME
9  #
10 # Arquivos:
11 # MainFrame: Tela principal; chamada das outras classes
12 # CalibrationFrame: Tela para realizar a calibração da placa; criada ao clicar no botão 'Modo de Calibração' da tela
       principal
13 # PlotFrame: Telas com os gráficos gerados a partir das seleções na tela de Configuração; criadas ao clicar no botão '
       Mostrar Gráficos'
14 # SettingsFrame: Tela para escolher as configurações desejadas do programa; criada ao clicar no botão 'Configurações'
15 # PandaDialogs: Diálogos de erro relacionados com a PANDA
16 # panda_board: Interação do software com o firmware da PANDA
17 
18 #-----------------------------------------------------------------------
19 
20 import wx
21 
22 class CalibrationFrame(wx.Frame):
23     """
24     This is the Calibration Frame.  It shows the options of calibration for the user to choose.
```

```
    The value is sent to the PANDA board and it's DAC does the calibration procedure.
    """
    def __init__(self, calibration, parent, title):
        wx.Frame.__init__(self, parent, -1, title,
                          pos=(350, 250), size=(600, 350))

        # Creates the Panel to put the other controls on
        self.panel = wx.Panel(self)

        # Initializes calibration array
        self.calibration = calibration

        # Controls
        self.text = wx.StaticText(self.panel, -1, label="Digite o valor de deformação, força ou torque aplicado.", pos
     =(15, 15))
        self.text.SetFont(wx.Font(12, wx.SWISS, wx.NORMAL, wx.BOLD))
        self.text.SetSize(self.text.GetBestSize())

        self.btn_optStrain = wx.RadioButton(self.panel, -1, label="Deformação", style=wx.RB_GROUP, pos=(15, 95))
        self.btn_optForce = wx.RadioButton(self.panel, -1, label="F/T", pos=(210, 95))
        self.btn_optStress = wx.RadioButton(self.panel, -1, label="Tensão", pos=(375, 95))

        self.btn_strainValue = wx.SpinCtrl(self.panel, value='0', pos=(110, 90), min=0, max=1000)
        self.btn_forceValue = wx.SpinCtrl(self.panel, value='0', pos=(270, 90), min=0, max=1000)
        self.btn_stressValue = wx.SpinCtrl(self.panel, value='0', pos=(440, 90), min=0, max=1000)

        self.btn_confirmChoices = wx.Button(self.panel, -1, label="Aplicar", pos=(140, 200))
        self.btn = wx.Button(self.panel, -1, "Cancelar", pos=(340, 200))

        # Binds the controls to handlers
        self.Bind(wx.EVT_BUTTON, self.OnTimeToClose, self.btn)
        self.Bind(wx.EVT_RADIOBUTTON, self.SelectStrain, self.btn_optStrain)
        self.Bind(wx.EVT_RADIOBUTTON, self.SelectForce, self.btn_optForce)
        self.Bind(wx.EVT_RADIOBUTTON, self.SelectStress, self.btn_optStress)
        self.Bind(wx.EVT_BUTTON, self.OnConfirmButton, self.btn_confirmChoices)

        # Prepares the Frame, based on calibration array values
        if self.calibration[0]:
            self.btn_strainValue.Enable()
        else:
            self.btn_strainValue.Disable()
        if self.calibration[1]:
            self.btn_forceValue.Enable()
        else:
            self.btn_forceValue.Disable()
        if self.calibration[2]:
            self.btn_stressValue.Enable()
        else:
            self.btn_stressValue.Disable()
        self.btn_optStrain.SetValue(self.calibration[0])
        self.btn_optForce.SetValue(self.calibration[1])
        self.btn_optStress.SetValue(self.calibration[2])
        self.btn_strainValue.SetValue(self.calibration[3])
        self.btn_forceValue.SetValue(self.calibration[4])
        self.btn_stressValue.SetValue(self.calibration[5])

        self.panel.Layout()


    def SelectStrain(self, evt):
        """Event handler for the Strain Radio Button."""
        self.btn_strainValue.Enable()
        self.btn_forceValue.Disable()
        self.btn_stressValue.Disable()


    def SelectForce(self, evt):
        """Event handler for the Force/Torque Radio Button."""
        self.btn_strainValue.Disable()
        self.btn_forceValue.Enable()
        self.btn_stressValue.Disable()


    def SelectStress(self, evt):
```

```
98          """Event handler for the Stress Radio Button."""
99          self.btn_strainValue.Disable()
100         self.btn_forceValue.Disable()
101         self.btn_stressValue.Enable()
102
103
104     def OnConfirmButton(self, evt):
105         """Event handler for the Confirm button."""
106         # Attributes new values to calibration array
107         self.calibration[0] = self.btn_optStrain.GetValue()
108         self.calibration[1] = self.btn_optForce.GetValue()
109         self.calibration[2] = self.btn_optStress.GetValue()
110         self.calibration[3] = self.btn_strainValue.GetValue()
111         self.calibration[4] = self.btn_forceValue.GetValue()
112         self.calibration[5] = self.btn_stressValue.GetValue()
113         self.calibration[6] = wx.ID_OK
114
115         self.Close()
116
117
118     def OnTimeToClose(self, evt):
119         """Event handler for closing."""
120         self.Close()
121
122
123     def GetCal(self):
124         """Returns calibration array when asked."""
125         return self.calibration
```

Listagem B.3: Software: CalibrationFrame.py.

```
1  #-----------------------------------------------------------------------
2
3  # Software da PANDA
4  # TCC - Engenharia Mecatrônica - Poli USP
5  # Guilherme de Agrela Lopes
6  # João Vitor Sanches
7  #
8  # PLOT FRAME
9  #
10 # Arquivos:
11 # MainFrame: Tela principal; chamada das outras classes
12 # CalibrationFrame: Tela para realizar a calibração da placa; criada ao clicar no botão 'Modo de Calibração' da tela
       principal
13 # PlotFrame: Telas com os gráficos gerados a partir das seleções na tela de Configuração; criadas ao clicar no botão '
       Mostrar Gráficos'
14 # SettingsFrame: Tela para escolher as configurações desejadas do programa; criada ao clicar no botão 'Configurações'
15 # PandaDialogs: Diálogos de erro relacionados com a PANDA
16 # panda_board: Interação do software com o firmware da PANDA
17
18 #-----------------------------------------------------------------------
19
20 import wx
21
22 import matplotlib
23 matplotlib.use('WXAgg')
24 from matplotlib.figure import Figure
25 from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigCanvas
26 from matplotlib import pyplot as plt
27
28 from time import time
29
30 from panda_board import panda
31
32 import os
33
34 import numpy as np
35
36 import PandaDialogs
37 from PandaDialogs import myPanda
38
39 # Globals
40 looped = False
```

```
start = time()


class PlotFrame(wx.Frame):
    """
    This is the Plot Frame.  It plots in real time the graphics that the user has chosen
    in the Settings Frame.  The user has the option to save the plot as an image or
    save the data in .csv format.
    """
    def __init__(self, chooseUnit, Ks, Kf, Kt, Kp, parent, title):
        self.dpi = 100
        self.height = 5
        self.width = 10

        wx.Frame.__init__(self, parent, -1, title,
                          pos=((self.width * self.dpi)-500, (self.height * self.dpi)-300), size=(1000,600))

        # Initializes unit and constants
        self.unit = chooseUnit
        self.Ks = Ks
        self.Kf = Kf
        self.Kt = Kt
        self.Kp = Kp

        # Defines maximum sizes
        self.maximumData = 5
        self.maximumArray = 3000

        # Gets initial value from function in PandaDialogs
        initialValue = PandaDialogs.dataConversion(self.unit, self.Ks, self.Kf, self.Kt, self.Kp)

        self.avgOver = 50

        self.data = [initialValue]
        self.time = [0]
        self.avgData = [initialValue]

        self.totalTime = 0

        self.fileTime = [0]
        self.fileValue = [initialValue]
        self.fileAvg = [initialValue]

        self.panel = wx.Panel(self, -1)

        self.initPlot()
        self.createMenu()

        self.canvas = FigCanvas(self.panel, -1, self.fig)

        # Creates timers
        self.redrawTimer = wx.Timer(self)
        self.updateValueTimer = wx.Timer(self)
        self.Bind(wx.EVT_TIMER, self.onRedrawTimer, self.redrawTimer)
        self.Bind(wx.EVT_TIMER, self.updateValue, self.updateValueTimer)
        self.Bind(wx.EVT_CLOSE, self.onClose, id=wx.ID_CLOSE)
        self.redrawTimer.Start(0.1)
        self.updateValueTimer.Start(0.1)

        self.graphBox = wx.BoxSizer(wx.VERTICAL)
        self.graphBox.Add(self.canvas, 1, flag=wx.LEFT | wx.TOP | wx.GROW)

        self.panel.SetSizer(self.graphBox)
        self.panel.Show()


    def onClose(self, event):
        """Event handler for closing."""
        self.redrawTimer.Stop()
        self.updateValueTimer.Stop()
        event.Skip()


    def updateValue(self, event):
```

```
115         """Gets current value and updates it in the arrays. Also blinks PANDA led."""
116         myPanda.setLedMode("ON")
117         value = PandaDialogs.dataConversion(self.unit, self.Ks, self.Kf, self.Kt, self.Kp)
118         now = time() - start
119 
120         avgSum = 0
121         window = self.avgOver if self.avgOver < len(self.data) else len(self.data)
122         for a in range(window):
123             avgSum = avgSum + self.data[(-1 * a)]
124         avgSum = avgSum / window
125         self.avgData.append(float(avgSum))
126 
127         self.data.append(float(value))
128         self.time.append(float(now))
129         self.totalTime = now
130 
131         if len(self.data) > self.maximumArray:
132             global looped
133             if not looped:
134                 print ("Starting re-use of graph arrays for efficiency...")
135                 looped = True
136             self.time.pop(0)
137             self.avgData.pop(0)
138             self.data.pop(0)
139 
140         self.fileTime.append(float(now))
141         self.fileValue.append(float(value))
142         self.fileAvg.append(float(avgSum))
143 
144         myPanda.setLedMode("OFF")
145 
146     def initPlot(self):
147         """Initializes plot."""
148         self.fig = Figure((self.width, (self.height)), dpi=self.dpi)
149 
150         self.axes = self.fig.add_subplot(111)
151         self.axes.set_facecolor('white')
152         self.axes.set_title('Value and Average', size=12)
153         self.axes.set_xlabel("Time (seconds)", size=10)
154         if self.unit == 'strain':
155             self.axes.set_ylabel("Strain", size=10)
156         if self.unit == 'force/torque':
157             self.axes.set_ylabel("Force (N) or Torque (N.m)", size=10)
158         if self.unit == 'stress':
159             self.axes.set_ylabel("Stress (N/m2)", size=10)
160 
161         plt.setp(self.axes.get_xticklabels(), fontsize=8)
162         plt.setp(self.axes.get_yticklabels(), fontsize=8)
163 
164         self.plotData = self.axes.plot(
165             self.data,
166             linewidth=2,
167             color=(1, 0, 0),
168             )[0]
169 
170 
171         self.plotAvg = self.axes.plot(
172             self.avgData,
173             linewidth=2,
174             color=(0, 0, 1),
175             )[0]
176 
177         self.Bind(wx.EVT_CLOSE, self.onClose)
178 
179     def drawPlot(self):
180         """Draws plot when called."""
181         gap = self.maximumData
182         xmax = self.totalTime if self.totalTime > gap else gap
183         xmin = xmax - gap
184 
185         if self.unit == 'strain':
186             ymin = -self.Ks*5
187             ymax = self.Ks*5
188         if self.unit == 'force/torque':
```

```
189             ymin = -self.Kf*5
190             ymax = self.Kf*5
191         if self.unit == 'stress':
192             ymin = -self.Kt*5
193             ymax = self.Kt*5
194         if self.unit == 'power':
195             ymin = -self.Kp*5
196             ymax = self.Kp*5
197         #ymin = round(min(self.data), 3) - (1 * abs(round(min(self.data), 3)))
198         #ymax = round(max(self.data), 3) + (1 * round(max(self.data), 3))
199         #if (ymin == 0 and ymax == 0) or ymin == ymax:
200             #ymax = self.Ks*10
201             #ymin = -self.Ks*10
202 
203         self.axes.set_xbound(lower=xmin, upper=xmax)
204         self.axes.set_ybound(lower=ymin, upper=ymax)
205 
206         self.axes.grid(True, color='gray')
207         plt.setp(self.axes.get_xticklabels(),
208             visible=True)
209 
210         self.plotAvg.set_data(np.array(self.time), np.array(self.avgData))
211         self.plotData.set_data(np.array(self.time), np.array(self.data))
212 
213         self.canvas.draw()
214 
215     def onRedrawTimer(self, event):
216         """Keeps calling drawPlot whenever the timer event is triggered."""
217         self.drawPlot()
218 
219     def createMenu(self):
220         """Creates menu where the user can choose to save an image or csv."""
221         self.menuBar = wx.MenuBar()
222 
223         self.menuFile = wx.Menu()
224         menuSaveCSV = self.menuFile.Append(-1, "&Salvar dados em .csv", "Save data to csv")
225         self.Bind(wx.EVT_MENU, self.onSaveCSV, menuSaveCSV)
226         menuSave = self.menuFile.Append(-1, "&Salvar imagem", "Save plot to image")
227         self.Bind(wx.EVT_MENU, self.onSavePlot, menuSave)
228         self.menuFile.AppendSeparator()
229         menuExit = self.menuFile.Append(-1, "Parar aquisição\tCtrl-X", "Stop")
230         self.Bind(wx.EVT_MENU, self.onClose, menuExit)
231 
232         self.menuBar.Append(self.menuFile, "&Arquivo")
233         self.SetMenuBar(self.menuBar)
234 
235     def onSavePlot(self, event):
236         """Saves the plot as an image."""
237         fileChoices = "PNG (*.png)|*.png"
238 
239         dlg = wx.FileDialog(
240             self,
241             message="Save plot as...",
242             defaultDir=os.getcwd(),
243             defaultFile="plot.png",
244             wildcard=fileChoices,
245             style=wx.FD_SAVE)
246 
247         if dlg.ShowModal() == wx.ID_OK:
248             path = dlg.GetPath()
249             self.canvas.print_figure(path, dpi=self.dpi)
250 
251     def onSaveCSV(self, event):
252         """Saves the data as a csv file."""
253         fileChoices = "CSV (*.csv)|*.csv"
254 
255         dlg = wx.FileDialog(
256             self,
257             message="Save data as...",
258             defaultDir=os.getcwd(),
259             defaultFile="plot.csv",
260             wildcard=fileChoices,
261             style=wx.FD_SAVE)
262 
```

```
263         if dlg.ShowModal() == wx.ID_OK:
264             path = dlg.GetPath()
265             outFile = open(path, 'w')
266 
267             if self.unit == 'strain':
268                 outFile.write("Time,Strain,Avg\n")
269             if self.unit == 'force/torque':
270                 outFile.write("Time,Force/Torque,Avg\n")
271             if self.unit == 'stress':
272                 outFile.write("Time,Stress,Avg\n")
273             if self.unit == 'power':
274                 outFile.write("Time,Power,Avg\n")
275 
276             for i in range(len(self.data)):
277                 outFile.write(str(self.fileTime[i]) + ",")
278                 outFile.write(str(self.fileValue[i]) + ",")
279                 outFile.write(str(self.fileAvg[i]) + "\n")
280             outFile.close()
```

Listagem B.4: Software: PlotFrame.py.

```
1  #-----------------------------------------------------------------------
2  
3  # Software da PANDA
4  # TCC - Engenharia Mecatrônica - Poli USP
5  # Guilherme de Agrela Lopes
6  # João Vitor Sanches
7  #
8  # PANDA DIALOGS
9  #
10 # Arquivos:
11 # MainFrame: Tela principal; chamada das outras classes
12 # CalibrationFrame: Tela para realizar a calibração da placa; criada ao clicar no botão 'Modo de Calibração' da tela
       principal
13 # PlotFrame: Telas com os gráficos gerados a partir das seleções na tela de Configuração; criadas ao clicar no botão '
       Mostrar Gráficos'
14 # SettingsFrame: Tela para escolher as configurações desejadas do programa; criada ao clicar no botão 'Configurações'
15 # PandaDialogs: Diálogos de erro relacionados com a PANDA
16 # panda_board: Interação do software com o firmware da PANDA
17 
18 #-----------------------------------------------------------------------
19 
20 from panda_board import panda
21 import wx
22 
23 # Connects to PANDA
24 myPanda = panda('auto')
25 
26 def dataConversion(unit, Ks, Kf, Kt, Kp):
27     """Receives user-defined constants and the unit desired, and returns the value."""
28 
29     value = 3.3*((myPanda.getAmpValue()/2048) - 1)   # DAC value converted to Vout
30     strain = Ks*value
31     force = Kf*value
32     stress = Kt*value
33     power = Kp*value
34 
35     if unit == 'strain':
36         return strain
37     elif unit == 'force/torque':
38         return force
39     elif unit == 'stress':
40         return stress
41     elif unit == 'power':
42         return power
43     else:
44         print("Invalid unit!")
45 
46 
47 
48 class PandaNotDetected(wx.Dialog):
49     """
50     This is a Dialog. It's opened when a PANDA board has not
```

```
    been detected.
    """
    def __init__(self, parent, title):
        wx.Dialog.__init__(self, parent, -1, title,
                           pos=(650, 450), size=(500, 150))

        self.panel = wx.Panel(self)
        self.controlNotActiveText = wx.StaticText(self.panel, -1, label="Não foi detectado um PANDA. Por favor, cheque
     se o dispositivo está conectado e foi\n reconhecido, e então reinicie o programa.", pos=(15,15))
        self.button_ok = wx.Button(self.panel, label="OK", pos=(210, 70))
        self.button_ok.Bind(wx.EVT_BUTTON, self.onOk)

        self.panel.Layout()

    def onOk(self, e):
        """Event handler for the Ok button."""
        self.EndModal(wx.ID_OK)
```

Listagem B.5: Software: PandaDialogs.py.

```
#-------------------------------------------------------------------

# Software da PANDA
# TCC - Engenharia Mecatrônica - Poli USP
# Guilherme de Agrela Lopes
# João Vitor Sanches
#
# PANDA BOARD INTERFACE
#
# Arquivos:
# MainFrame: Tela principal; chamada das outras classes
# CalibrationFrame: Tela para realizar a calibração da placa; criada ao clicar no botão 'Modo de Calibração' da tela
     principal
# PlotFrame: Telas com os gráficos gerados a partir das seleções na tela de Configuração; criadas ao clicar no botão '
     Mostrar Gráficos'
# SettingsFrame: Tela para escolher as configurações desejadas do programa; criada ao clicar no botão 'Configurações'
# PandaDialogs: Diálogos de erro relacionados com a PANDA
# panda_board: Interação do software com o firmware da PANDA

#-------------------------------------------------------------------

import serial
import serial.tools.list_ports
import struct
from time import sleep
from datetime import datetime
import matplotlib.pyplot as plt
import queue
import numpy as np
DEVICE_INQUIRE_TIMEOUT = 0.5


def getPandaPorts():
    ports = [str(i).split()[0] for i in serial.tools.list_ports.comports()]
    pandas = []
    for portname in ports:
        try:
            device = serial.Serial(portname, baudrate = 57600)
            sleep(0.5)
            device.write([35, 1, 1, 13, 10])
            sleep(0.1)
            answer = []
            scanstart = datetime.now()
            timeout = 0
            while device.in_waiting > 0 and not timeout:
                timeout = (datetime.now() - scanstart).total_seconds() > DEVICE_INQUIRE_TIMEOUT
                answer = (device.read(device.in_waiting))
            if(b'I am a Panda' in answer):
                pandas.append(portname)

            device.close()
        except:
            pass
```

```
    return pandas


class panda:
    def send(self, data):
        if (self.board):
            le = len(data)
            msg  = [35, le] + data + [13, 10]
            self.board.write(msg)
            #print("sent: ", [i for i in msg])
        else:
            print("Serial port not Configured")
    def parseFrame(self):
        frame = []
        timeout = 0.1
        start_byte = 0
        expected_len = 0
        actual_len = 0
        end_of_frame = 0
        state = 0 # 0 start / 1 check len / 2 listening / 3 eof
        parse_start = datetime.now()
        _timeout = 0
        while not end_of_frame and not _timeout and self.board.in_waiting != 0:
            _timeout = (datetime.now() - parse_start).total_seconds() > timeout
            data = int.from_bytes(self.board.read(), 'big')
            if data >= 0 and data <= 255:
                if state == 0:
                    if data == ord('#'):
                        state = 1
                        frame.append(data)
                    else:
                        break
                elif state == 1:
                    expected_len = data
                    state = 2
                    frame.append(data)
                elif state == 2:
                    actual_len += 1
                    frame.append(data)
                    if actual_len == expected_len:
                        state = 3
                elif state == 3:
                    if data == ord('\r'):
                        frame.append(data)
                        state = 4
                    else:
                        break
                elif state == 4:
                    if data == ord('\n'):
                        frame.append(data)
                        return frame
                    else:
                        break
        print("Reception fault")
        return None


    def receive(self):
        lines = []

        while self.board.in_waiting > 0:
            lines.append(self.parseFrame())
        if lines:
            if len(lines) > 1:
                print("Unexpected lines received")
                input()

            #for line in lines:
                #print("Received: ", [i for i in line])
            return lines[-1]
        else:
            return 0

    def waitAck(self, timeout = 0.1):
```

```
126         wait_start = datetime.now()
127         _timeout = 0
128         while self.board.in_waiting == 0 and not _timeout:
129             _timeout = (datetime.now() - wait_start).total_seconds() > timeout
130             if _timeout:
131                 print("Ack timeout")
132 
133         msg = self.receive()
134         if msg:
135             return msg
136         else:
137             print("Panda didn't answer last call")
138             return 0
139 
140     def __init__(self, port, id = None):
141         if port == 'auto':
142             available_pandas = getPandaPorts()
143             self.board = 0
144             if available_pandas:
145                 self.board = serial.Serial(available_pandas[0], baudrate = 57600, timeout = 0.2)
146                 self.send([10, 2])
147                 self.waitAck()
148                 self.send([12, 20])
149                 self.waitAck()
150                 self.send([5, 1])
151                 self.waitAck()
152                 sleep(0.5)
153                 self.send([10, 0])
154                 self.waitAck()
155                 print("Panda Connected")
156                 #print(available_pandas)
157 
158             else:
159                 print("No pandas available...")
160 
161     def exists(self):
162         if self.board != 0:
163             return True
164         else:
165             return False
166 
167     def disconnect(self):
168         if self.board:
169             self.send([5, 0])
170             self.waitAck()
171             self.send([10, 2])
172             self.waitAck()
173             self.send([12, 20])
174             self.waitAck()
175             sleep(0.5)
176             self.send([10, 0])
177             self.waitAck()
178             self.board.close()
179             print("Panda Disconnected")
180 
181     def setLedMode(self, state):
182         if state == "OFF" or state == 0:
183             self.send([10,0])
184         elif state == "ON" or state == 1:
185             self.send([10,1])
186         elif state == "BLINK" or state == 2:
187             self.send([10,2])
188         return bool(self.waitAck())
189 
190     def setLedFreq(self, freq):
191         if freq > 100 or freq < 1:
192             print("Frequency out of bounds")
193             return 0
194         freq = int(freq)
195         self.send([12,freq])
196         return bool(self.waitAck())
197 
198     def setOut1(self, state):
199         if state == "OFF" or state == 0:
```

```
200             self.send([14,0])
201         elif state == "ON" or state == 1:
202             self.send([14,1])
203         return bool(self.waitAck())
204 
205     def setOut2(self, state):
206         if state == "OFF" or state == 0:
207             self.send([16,0])
208         elif state == "ON" or state == 1:
209             self.send([16,1])
210         return bool(self.waitAck())
211 
212     def getAmpValue(self):
213         self.send([20])
214         response = self.waitAck()
215         if response[2] == 148:
216             value = (response[3]<<8) + (response[4])
217             return value
218         else:
219             print("wrong response code")
220             return None
221 
222     def getAnalog1(self):
223         self.send([21])
224         response = self.waitAck()
225         if response[2] == 149:
226             value = (response[3]<<8) + (response[4])
227             return value
228         else:
229             print("wrong response code")
230             return None
231 
232     def getAnalog2(self):
233         self.send([22])
234         response = self.waitAck()
235         if response[2] == 150:
236             value = (response[3]<<8) + (response[4])
237             return value
238         else:
239             print("wrong response code")
240             return None
241 
242     def getIn1(self):
243         self.send([30])
244         response = self.waitAck()
245         if response[2] == 158:
246             value = response[3]
247             return value
248         else:
249             print("wrong response code")
250             return None
251     def getIn2(self):
252         self.send([31])
253         response = self.waitAck()
254         if response[2] == 159:
255             value = response[3]
256             return value
257         else:
258             print("wrong response code")
259             return None
260 
261     def setEncoderMode(self, mode):
262         if mode == "QUADRATURE" or mode == 0:
263             self.send([40,0])
264         elif mode == "PWM" or mode == 1:
265             self.send([40,1])
266         return bool(self.waitAck())
267 
268     def setEncoderCount(self, count):
269         if count < 0 or count > 2**32:
270             print("Invalid count")
271             return 0
272         self.send([42]+list(struct.unpack('4B', struct.pack('>I', count))))
273         return bool(self.waitAck())
```

```
274
275     def getEncoderCount(self):
276         self.send([43])
277         response = self.waitAck()
278         if response[2] == 171:
279             value = (response[3]<<24) + (response[4]<<16) + (response[5]<<8) + response[6]
280             return value
281         else:
282             print("wrong response code")
283             return None
284
285     def getEncoderDuty(self):
286         self.send([44])
287         response = self.waitAck()
288         if response[2] == 172:
289             value = (response[3]<<8) + (response[4])
290             return value
291         else:
292             print("wrong response code")
293             return None
294
295     def runAutoOffset(self, zero_value = 2048):
296         if zero_value < 0 or zero_value  > 4095:
297             print("Invalid Offset")
298             return 0
299         self.send([62]+list(struct.unpack('2B', struct.pack('>h', zero_value))))
300         return bool(self.waitAck(2))#Reduzir timeout depois
301
302     def setGain(self, gain):
303         if gain < 1:
304             print("Invalid gain")
305             return 0
306         self.send([54]+list(struct.unpack('2B', struct.pack('>h', gain))))
307         ack = self.waitAck(1) #Reduzir timeout depois
308         #print((ack[3]<<8) + (ack[4]))
309         return bool(ack)
310
311 class monitor:
312     def __init__(self, timestep, samples = 50, y_label = 'Value'):
313         self.x_vec = np.linspace(-samples*timestep ,-timestep, samples)
314         self.timestep = timestep
315         self.samples = samples
316         self.curve_legends = []
317         self.curve_getters = []
318         self.curves = []
319         plt.ion()
320         self.fig = plt.figure(figsize=(13,6))
321         self.ax = self.fig.add_subplot(111)
322         self.hists = []
323         self.curve_colors = ['b','g','r','c','m','y']
324         self.y_label = y_label
325
326
327     def addCurve(self, c_name, c_getter):
328         self.curve_getters.append(c_getter)
329         self.curve_legends.append(c_name)
330
331         y_hist = queue.Queue()
332         for i in range(self.samples):
333             y_hist.put(0)
334         self.hists.append(y_hist)
335
336         y_vec = np.zeros(self.samples)
337         curve_, = self.ax.plot(self.x_vec, y_vec, self.curve_colors[len(self.curves)])
338         self.curves.append(curve_)
339
340     def start(self):
341         plt.xlabel('Time [s]')
342         plt.ylabel(self.y_label)
343         plt.title('Monitor')
344         plt.show()
345         plt.grid()
346         plt.legend(self.curve_legends)
347
```

```
348     def update(self):
349         for curve_n in range(len(self.curves)):
350             self.hists[curve_n].get()
351             n_value = self.curve_getters[curve_n]()
352             self.hists[curve_n].put(n_value)
353             y_vec = list(self.hists[curve_n].queue)
354             self.curves[curve_n].set_ydata(y_vec)
355 
356         min_y = min([min(i.queue) for i in self.hists])
357         max_y = max([max(i.queue) for i in self.hists])
358         marg = 0.1*(max_y - min_y)
359 
360         plt.ylim([min_y-marg, max_y+marg])
361         plt.pause(self.timestep)
362 
363     def stop(self):
364         plt.ioff()
365 
366     def resume(self):
367         plt.ion()
```

Listagem B.6: Library: panda_board.py.

```
1  '''
2  Panda scale Demo
3 
4  This program initializes a panda object (connects to the board) and continuously
5  requests the value from the amplifier (ranging from 0 to 4096). The main loop also
6  keeps toggling the board LED.
7  After a succesful connection the user will be prompted to press any key to start
8  the loop.
9  By pressing the ESC key, the loop stops and the board is disconnected.
10 '''
11 
12 from time import sleep
13 from panda_board import panda
14 import msvcrt
15 
16 #Constant to be used as converting factor from 12bit adc value to a weight measurement
17 Km = -0.7070365358592695 # g/bit
18 Gain = 600
19 
20 myPanda = panda('auto') # A serial port can be especified. "auto" makes everything easier
21 
22 print("Press ESC to stop, Z to zero or C to calibrate \n")
23 input("Hit Enter key to start")
24 
25 myPanda.setGain(Gain)
26 myPanda.setLedMode("ON") #Mode 1 = ON
27 samples = 10
28 while 1:
29     digital_value = 0
30     for i in range(samples):
31         digital_value += myPanda.getAmpValue()
32     digital_value /= samples
33     weight = (digital_value - 2048) * Km # corrects mid scale offset and apllies the linear factor
34     print("Weight measurement: %.1f g" %(weight))
35     sleep(0.1)
36 
37     # Scans for keypress
38     if msvcrt.kbhit():
39         key = ord(msvcrt.getch())
40         if key == 27: # ESC
41             break #Stops the program
42         elif key == 122: # Z Key
43             print("Zeroing the device")
44             myPanda.runAutoOffset() #Hardware offset performed by the board
45 
46         elif key == 99: #C Key
47             known_value = int(input("Enter the known value: "))
48             if known_value != None:
49                 Km = known_value / (digital_value-2048)
50                 print("New Km is: ", Km)
```

```
51
52 myPanda.setLedMode("OFF") #Mode 0 = OFF
53
54 myPanda.disconnect()
```

Listagem B.7: Demos: scale_demo.py.

```
1  #-----------------------------------------------------------------------
2
3  # Software da PANDA
4  # TCC - Engenharia Mecatrônica - Poli USP
5  # Guilherme de Agrela Lopes
6  # João Vitor Sanches
7  #
8  # PANDA THERMICS APPLICATION
9  #
10 # This example reads load cell data, temperature (2 channels) and rotationn,
11 # according to the needs of a steam engine testing activity
12 # Data is displayed with the monitor Class and some logging is made to a csv file
13 #
14 # Each of the 10k NTC thermistor should be connect to Vref and it's AIN. User must
15 # verify the board has resistors soldered to R9 and R12 (they are near User Analog
16 # connector and are soldered by default upon board assembly)
17 #-----------------------------------------------------------------------
18 from panda_board import panda, monitor
19 import msvcrt
20 from datetime import datetime
21 from math import log
22 from time import sleep
23
24
25 #Instancing of a Panda
26 myPanda=panda('auto')
27
28 #Set led frequiency to 15Hz
29 myPanda.setLedFreq(15)
30
31 # Configure amplifier gain to 500
32 myPanda.setGain(500)
33 sleep(1)
34
35 myPanda.runAutoOffset()
36
37 # Declaration of a getter function to read the voltage at Analog in 1
38 def voltage():
39     return 3.3/4096 * (myPanda.getAmpValue()-2048) #Calculation to map the 12bit adc value to a 0V~3.3V range
40
41 # Function to calculate the torque
42 def torque():
43     Kt = -100 # mNm/V     Depends on the load cell and gain used
44     return voltage()*Kt
45
46
47 # Declaration of a getter function to read the number of turns in the encoder
48 def encoderTurns():
49     ticks_per_rev = 40
50     ticks = myPanda.getEncoderCount()
51     if ticks > 2147483648: #Zero offset for the  32bit count
52         ticks -= 4294967296
53     return ticks/ticks_per_rev
54
55 #Variables to keep values between function calls
56 last_encoder_pos = 0
57 last_encoder_time = datetime.now()
58 # Getter function to calculate angular speed in rpm
59 def angSpeed():
60     global last_encoder_pos
61     global last_encoder_time
62     delta_time = (datetime.now() - last_encoder_time).total_seconds()
63     last_encoder_time = datetime.now()
64     delta_ang = encoderTurns() - last_encoder_pos
65     last_encoder_pos = encoderTurns()
66     return (delta_ang/delta_time)
```

```
67
68 # Returns the temperature in Kelvins, calulated from ADC value with SteinhartHart equation
69 # https://en.wikipedia.org/wiki/Steinhart%E2%80%93Hart_equation
70 # The voltage measured by the adc is Vadc = Vref * Rthermistor / (Rthermistor + 10 KOhm), and Vadc = (adc_value / 4096)
       * Vref
71 # thus, Rthermistor = ((10KOhm * Vadc) - Vref ) / (1 - Vadc)
72 def calc_temp(adc_value):
73     a = 1.009249522e-03
74     b = 2.378405444e-04
75     c = 2.019202697e-07
76     Vref = 3.3
77     Vadc = (adc_value/4096) * Vref
78     Rt = ((10000 * Vadc) - Vref ) / (1 - Vadc)
79     ##Rt = 10000   # Unncoment to avoid errors while testing. Keeps output at 25 deg.C
80     Temp_K = (1.0 / (a + b*log(Rt) + c*log(Rt)**3))
81     return Temp_K-273
82
83 #Getter functions to be passed to the monitor
84 def temp1():
85     return calc_temp(myPanda.getAnalog1())
86 def temp2():
87     return calc_temp(myPanda.getAnalog2())
88
89 #Instancing of a monitor with sampling time and number of samples to be shown at once
90 g = monitor(0.1, y_label='Values')
91
92 #Assign curves to the monitor. The curves will share the Y axis, watch out for the best unitis to get a better
       visualization
93 g.addCurve('Temperature 1 [deg. c]', temp1)
94 g.addCurve('Temperature 2 [deg. c]', temp2)
95
96 g.addCurve('Speed [rps]', angSpeed)
97
98 g.addCurve('Torque [mNm]', torque)
99
100 #Led will start blinking at 15 Hz
101 myPanda.setLedMode("BLINK")
102
103 #The monitor will pop-up
104 g.start()
105 print("Press ESC to stop or Space to pause")
106 while 1:
107     #Keeps updating the graph
108     g.update()
109
110     #Until user presses ESC on the console
111     if msvcrt.kbhit():
112         k = ord(msvcrt.getch())
113         if  k == 27:
114             print("Stopping")
115
116             break
117         elif k == 32:
118             g.stop()
119             input("Press Enter to resume...")
120             myPanda.setLedMode("OFF")
121             g.resume()
122             print("Press ESC to stop or Space to pause")
123             myPanda.setLedMode("BLINK")
124         else:
125             print("Press ESC to stop or Space to pause")
126
127 myPanda.setLedMode("OFF")
128 # Disconnects
129 myPanda.disconnect()
```

Listagem B.8: Demos: temperature_demo.py.