JHONATAN RODRIGUES

CONTROLE DE ESTABILIZAÇÃO HORIZONTAL DE UMA PLATAFORMA INERCIAL COM TRÊS GRAUS DE LIBERDADE

LAVRAS – MG

2020

JHONATAN RODRIGUES

CONTROLE DE ESTABILIZAÇÃO HORIZONTAL DE UMA PLATAFORMA INERCIAL COM TRÊS GRAUS DE LIBERDADE

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Engenharia de Controle e Automação, para a obtenção do título de Bacharel.

Prof. Dr. Leonardo Silveira Paiva
Orientador

LAVRAS – MG

2020

AGRADECIMENTOS

- A Deus, por me apoiar em todas as horas difíceis, ajudar a superar todos os desafios e propiciar o convívio com todas essas incríveis pessoas.

- Ao Prof. Dr. Leonardo Silveira Paiva, pela sua dedicação em me orientar e formar excelentes profissionais por meio do exercício da docência.

- Ao Prof. Dr. Fábio Domingues de Jesus, por conceder seu laboratório e ajudar na confecção da plataforma.

- À Universidade Federal de Lavras, por propiciar sua excelente estrutura e corpo docente, onde pude interagir com diversas áreas do conhecimento e vivenciar diversas experiências enriquecedoras para minha formação.

- Aos amigos que fiz durante o meu percurso em Lavras, por todo o suporte e companheirismo e por todo o crescimento pessoal e profissional que me proporcionaram.

MUITO OBRIGADO!

“Não se deve ir atrás de objetivos fáceis, é preciso buscar o que só pode ser alcançado por meio dos maiores esforços. ”
Albert Einstein.

RESUMO

O presente trabalho tem por objetivo apresentar o desenvolvimento de uma plataforma inercial autônoma com três graus de liberdade para aplicação de estabilização, como por exemplo, sistemas de navegação, veículos autônomos e estabilização de câmeras. O sistema desenvolvido consiste de um sensor micro eletromecânico, MPU-6050, que possui acelerômetro, giroscópio e magnetômetros, um microcontrolador para aquisição de dados, processamento e envio dos dados ao sistema de controle e aquisição de dados. Para o controle dos ângulos de inclinação e orientação da plataforma, será implementado um controlador PID utilizando o microcontrolador. Este receberá os dados no MPU-6050 e fornecer os sinais de controle utilizando as saídas PWM que acionam os servomotores, os quais controlam a posição da plataforma. Foram explorados conceitos de eletrônica, instrumentação, filtragem e tratamento de dados, desenvolvimento de *software*, utilização de ferramentas computacionais, redes de comunicação, teoria de controle e alguns conceitos de mecânica. Com base nesses conceitos foram realizados simulações e testes práticos de cada etapa separadamente do sistema. Desde a comunicação entre o arduino e um microcomputador, criação do sinal de correção dos ângulos de inclinação dos servomotores, filtragem e amplificação de sinal, e feito o dimensionamento de cada componente eletrônico utilizado.

Palavras-Chave: Plataforma inercial, Controle PID, Estabilização de câmeras, MPU-6050, *software*, Filtragem.

ABSTRACT

This work aims to present the development of an autonomous inertial platform with three degrees of freedom for stabilization applications, such as navigation systems, autonomous vehicles and camera stabilization. The developed system consists of a micro electromechanical sensor, MPU-6050, which has an accelerometer, gyroscope and magnetometers, a microcontroller for data acquisition, processing and sending data to the control and data acquisition system. For the control of the platform's inclination and orientation angles, a PID controller will be implemented using the microcontroller. It will receive the data on the MPU-6050 and supply the control signals using the PWM outputs that drive the servomotors, which control the position of the platform. Concepts of electronics, instrumentation, filtering and data treatment, software development, use of computational tools, communication networks, control theory and some mechanics concepts were explored. Based on these concepts, simulations and practical tests of each stage were carried out separately from the system. From the communication between the arduino and a microcomputer, creation of the signal for correcting the inclination angles of the servomotors, filtering and signal amplification, and the dimensioning of each electronic component used.

Keywords: Inertial platform, PID control, Camera stabilization, MPU-6050, *software*, Filtering.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

A inércia é uma propriedade física da matéria que foi estudada e formulada inicialmente por Galileu Galilei - 1564-1642 - e, acreditava-se que o estado natural de um corpo era o repouso e que um movimento só subsistiria mediante uma contínua aplicação de forças. Galileu concluiu que força externa a um corpo somente provoca variação em sua velocidade e para que um corpo permaneça em movimento retilíneo uniforme - MRU - ou em repouso, não será necessária a aplicação de força externa (FARIA. 2019).

Posteriormente, provada por Isaac Newton, tornando-se a base para a tão conhecida *"1ª Lei de Newton".* Em 1687, Newton enunciou em sua obra *"Philosophiae Naturalis Principia Mathematica"* que todo corpo tende a permanecer no estado de repouso ou de movimento retilíneo uniforme, a menos que haja sobre ele forças externas que o obrigue a mudar este estado. Essa lei, também conhecida como *"Princípio da Inércia"*, é um dos conceitos mais básicos da mecânica clássica (MECÂNICA. 2019).

Conhecendo-se as leis da dinâmica e sabendo-se como o movimento de corpos são descritos, sensores inerciais, que têm por objetivo perceber os efeitos da ação de forças que provoquem uma mudança do estado inercial de sistemas sobre os quais estas forças são exercidas, são utilizados na captação destes movimentos para tratamento e controle em sistemas eletrônicos. Devido ao grande número de aplicações, estes sensores se tornaram um dos sistemas microeletromecânicos mais populares.

O problema de estabilização de plataformas é recorrente em diferentes aplicações, e são objetos de estudos em diferentes áreas, como aviação comercial e militar, marinha e aeroespacial. Existem desde veículos modernos, que utilizam suspensões inteligentes para manter os passageiros sempre em sua posição horizontal, como também aeronaves não tripuladas, que possuem câmeras instaladas em sistemas inteligentes em suas bases, conhecidos como *gimbals*, para manter a câmera sempre estabilizada, o que torna a filmagem estável e muito mais atraente para os telespectadores.

A ideia de plataforma inercialmente estabilizada muito se assemelha com *gimbals*, trata-se de estruturas tipicamente mecatrônicas dotadas de sensores capazes de medir deslocamentos e velocidades angulares em relação ao referencial da Terra, e de conjuntos mecânicos capazes de corrigir a orientação do objeto a ser estabilizado.

Levando em conta a vasta aplicabilidade dos sistemas de controle e estabilização, bem como a real complexidade desses sistemas, que exigem diversos conceitos do curso

da Engenharia de Controle e Automação para o seu desenvolvimento como, instrumentação, aquisição e tratamento de dados, sistemas mecânicos e de controle robustos, entre outros, o tema foi escolhido como uma forma de revisar e aplicar os conhecimentos adquiridos durante o curso.

2. OBJETIVOS

Para o desenvolvimento do trabalho em questão são redigidos os seguintes objetivos:

2.1. Objetivo Geral

Como objetivo geral, o presente trabalho propõe-se a analisar e avaliar o controle de estabilidade de uma plataforma inercial com três graus de liberdade quanto a sua funcionalidade frente à praticidade, visando conceitos de robótica e controle de sistemas.

2.2. Objetivos Específico

Para alcançar o projeto especificado, os seguintes objetivos foram estipulados.

- Realizar pesquisas sobre controle analógico;
- Prototipagem e desenvolvimento em termos de *hardware*.
  - Definir materiais a serem utilizados para construção do protótipo para avaliação.
  - Definir componentes eletrônicos a serem utilizados para *hardware* do sistema.
- Desenvolvimento do *software* de controle do sistema.
  - Definir o esquemático eletrônico do hardware para o transmissor de comandos do sistema.
- Dimensionar componentes para confecção levando em consideração o tamanho do mesmo.
- Construir protótipo do circuito eletrônico responsável pela atividade de envio de sinal na comunicação entre o módulo e servomotores.
- Construir protótipo do circuito eletrônico responsável pela atividade de recebimento de sinal na comunicação entre o módulo e servomotores.
- Implementação e validação da plataforma inercial.

3. REFERENCIAL TEÓRICO

Para o desenvolvimento deste trabalho, é necessário entender o que são estabilidades de plataformas inerciais e controladores, e quais suas utilidades quando aplicados. E nesta seção serão apresentados alguns conceitos sobre e alguns tipos de projetos existentes, para compará-los ao escolhido para este trabalho.

3.1. Posicionamento Inercial

O desenvolvimento de plataformas inerciais começou efetivamente com o programa germânico V-2. Após a II Guerra Mundial, nos laboratórios de instrumentação do *Massachussets Institute of Technology* e *Redstone Arsenal* (FREITAS. 1980), iniciaram-se intensos trabalhos de desenvolvimento de plataformas inerciais, motivados pela ideia de um sistema autocontido, capaz de fornecer informações de velocidade e posição de um veículo, através unicamente da medição da aceleração deste. Surgiram plataformas para desempenharem funções em duas e três dimensões.

Quando a elevação, relativamente a uma superfície de referência permanentemente conhecida ou não, é de interesse, resulta conveniente um sistema operacional em duas dimensões. Já os sistemas tridimensionais são ideais para navegação de mísseis, de veículos espaciais.

A maioria dos PI, proporcional integral, em uso atualmente, foram desenvolvidos a partir da plataforma inercial bidimensional com calibração *Schuler, "Litton LN-15"*, destinada à navegação aeronáutica. Tal plataforma foi inicialmente transformada, mediante contrato entre a *"Litton Guidance and Control Systems"* e o *"US Army Engineer Topographic Laboratories"*. Assim, entre 1965 e 1972 foi desenvolvido um sistema tridimensional de medição denominado *"Position and Azimuth Determining System (PADS)"* visando principalmente aplicações à artilharia (FREITAS. 1980). Tal dispositivo, partindo de um ponto de coordenadas conhecidas é capaz de fornecer posições planimétricas com precisão de ±20m, altimétricas com ±10m e azimute com precisão de 1º para missões de até 4 horas. Posteriormente, em 1974, a *Litton* e o *USAETL* modificaram o PADS com a inclusão na programação de técnicas de compensação de erros durante o levantamento, filtragem kalman, e no fim deste, conhecido o erro de fechamento *"smoothing"*. Adicionalmente foi introduzido um acelerômetro

extrassensível no canal vertical, possibilitando ao sistema interpolar, além de coordenadas, anomalias da gravidade e deflexão da vertical.

3.2. Ângulos de Euler

Descrevem a orientação de um corpo rígido em um espaço euclidiano tridimensional, é um espaço vetorial real de dimensão finita dotado de um produto interno. Transformando as coordenadas que relacionam o sistema inercial com o sistema de coordenadas ligado ao corpo. O matemático e físico Leonhard Paul Euler é o nome responsável por representar a forma matricial da Equação 3.2.1 abaixo:

$$X = T_0^3 x \tag{3.2.1}$$

Onde $X = (X, Y, Z)^T$ é o vetor posição no sistema ligado ao corpo e $x = (x, y, z)^T$ é o vetor posição no sistema inercial. A matriz de rotação T_0^3 descreve a orientação relativa dos dois sistemas. A matriz é definida na Equação 3.2.2 pelo produto matricial de três rotações elementares descritas pelos ângulos de Euler $\alpha, \beta \; e \; \gamma$.

$$T_0^3 = T_0^1 T_1^2 T_2^3 = \begin{bmatrix} cos\gamma & sen\gamma & 0 \\ -sen\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\beta & sen\beta \\ 0 & -sen\beta & cos\beta \end{bmatrix} \begin{bmatrix} cos\alpha & sen\alpha & 0 \\ -sen\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2.2}$$

Cada matriz de rotação produz uma rotação em torno do eixo do sistema de coordenadas vigente (FRIEDLAND. 1986).

3.3. Plataformas Inerciais

Apesar de não ser um problema exatamente recente na literatura, plataformas estabilizadas ainda são tema de interesse em diferentes áreas. Se tratando de artigos mais gerais sobre o tema, abordando as principais aplicações, restrições e desafios presentes no projeto destes trabalhos, fornecendo assim um bom ponto de partida para o estudo.

Em uma dissertação de mestrado em 1980 (FREITAS. 1980), o trabalho tinha como objetivo sistematizar os princípios ligados ao posicionamento inercial, assim como analisar a viabilidade dos posicionadores inerciais para a determinação de pontos de apoio à Cartografia. Apresentando exemplos casuais como: um sistema de navegação inercial constituído por acelerômetros, giroscópios e por dispositivos capazes de processar as informações deles obtidas. O sistema apresentado pelo autor efetua medidas inerciais de forças e deslocamentos angulares. Uma inicialização apropriada da navegação inercial, possibilitando o conhecimento contínuo de velocidade, posição e atitude, livres de qualquer interferência externa.

Ele ainda afirma na página 40 de seu documento, que todos os sistemas de navegação inercial se baseiam nas seguintes funções: instrumentação de um referencial; medida de força inercial; conhecimento do campo gravitacional; processamento de dados para obtenção das informações de referência.

Desempenhando a primeira função por giroscópios. Um referencial tri ortogonal pode ser instrumentado por três giroscópios de um grau de liberdade, cada um com seu eixo de rotação instrumentando um dos eixos ou ainda por dois giroscópios com dois graus de liberdade, onde dois eixos são instrumentados pelo eixo de rotação de cada um dos giroscópios e o terceiro pela eliminação do grau de liberdade redundante entre eles. Torques externos segundo ele são sensoriados pelos giroscópios, sendo tais informações utilizadas para manter a orientação espacial desejável para o referencial. Na página 40 de seu documento, ele descreve a função em uma frase: “Se o conjunto de dois ou três giroscópios é montado sobre uma plataforma cujos movimentos são dirigidos para manter a orientação desejável dos seus eixos de rotação, tal plataforma é denominada plataforma inercial. ”

A segunda função é desempenhada por acelerômetros montados sobre a plataforma inercial e com sensitivos paralelos aos eixos instrumentados pelos giroscópios.

De acordo com o Princípio da Equivalência de Albert Einstein, o autor se baseia e deduz, na página 41 de seu documento, que: “Forças inerciais e gravitacionais são manifestação de um mesmo fenômeno, não sendo possível sua distinção. Portanto, a navegação inercial, dentro de um campo gravitacional, exige o conhecimento deste, para a detecção de componentes de forças inerciais na sua direção”. Apresentando assim a terceira função mencionada.

Partindo do pressuposto que os acelerômetros fornecem como saídas sinais proporcionais à aceleração, os quais, se adequadamente processados, podem fornecer velocidade e posição da plataforma. O autor conclui que em sistemas de navegação inercial modernos, existe um computador associado, o qual efetua tais processamentos, além de efetuar todos os controles da plataforma, sendo esta a última das funções referidas por ele.

Tanto no giroscópio quanto na plataforma inercial, devem ser providos dispositivos capazes de compensar eventuais desvios de predeterminada altitude. A carcaça de um giroscópio serve de elemento de referência para desvios do rotor. Cada vez que este sofre um desvio em relação a carcaça, um dispositivo capaz de aplicar um torque sobre os balancins que o suportam, é acionado, possibilitando assim a recuperação de sua posição inicial em relação à carcaça. Analogamente, na plataforma do autor, sempre que ocorre um desvio de determinada atitude de referência, atuam dispositivos compensadores existentes na estrutura de balancins que a suportam, podendo-se desta forma, comandar está para uma atitude desejada em determinado referencial.

Durante os testes realizados pelo mesmo, obtiveram valores compensados somente para latitude, longitude e altitude. Não efetuaram observações de anomalia da gravidade e deflexão da vertical, uma vez que não existiam, por ocasião dos testes, valores conhecidos destas para os pontos de controle inicial e final de cada uma das linhas.

A aplicação do controlador PI, segundo o autor, sem dúvida, são dispositivos versáteis e altamente produtivos, capazes de fornecer coordenadas de grande precisão, além de outras informações de interesse. Mesmo a sua utilização apresentando ainda certos problemas relacionados principalmente com: distribuição dos pontos; geometria das linhas; operação. Com os resultados esperados e apresentados pelo autor, o mesmo, considerou os testes satisfatórios.

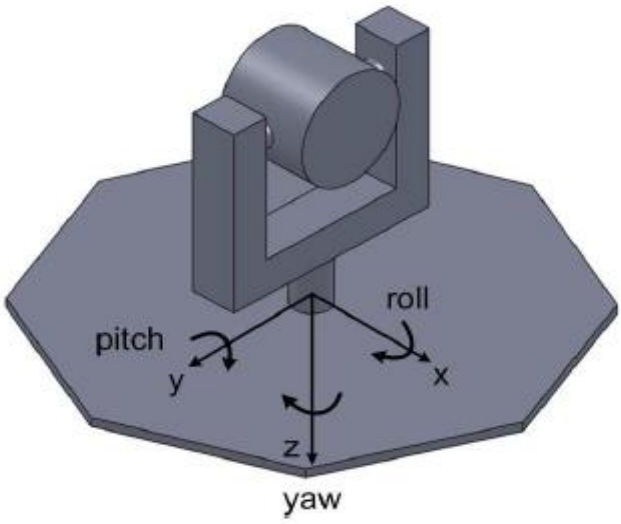
Em uma outra dissertação de mestrado (BATTISTEL. 2011), o problema de estabilização é apresentado no contexto onde se deseja estabilizar um rastreador visual. Para a aplicação no problema aqui considerado, o interesse é por estratégias mais simples

e de baixo custo computacional. O objetivo do autor é seguir um determinado objeto com uma câmera posicionada por um sistema de *gimbals*, utilizando para isso informações da imagem, de modo que o algoritmo de interesse deve ser pouco custoso computacionalmente.

Ele trata o problema de estabilizar inercialmente uma plataforma como, relativamente simples do ponto de vista de cálculo, uma vez que, a orientação desta plataforma seja conhecida - neste caso, é necessário calcular apenas os ângulos de correção que comandarão a posição do *gimbal*. Ele ainda retrata que quando se utiliza uma unidade inercial com conjunto completo de medidas, o cálculo destes ângulos é, por conseguinte, de relativa simplicidade. Para isto, desenvolveu uma série de algoritmos capazes de estabilizar uma plataforma utilizando apenas duas medidas de sensores giroscópios, partindo do pressuposto que a movimentação externa à qual o veículo está sujeito contém algumas restrições.

A representação da orientação utilizada pelo autor neste trabalho é a de ângulos de Euler XYZ, comumente adotada em sistemas de navegação como *Roll-Pitch-Yaw*, cujos ângulos são representados por ψ, Θ e Φ respectivamente. A representação espacial desta escolha é de acordo com a Figura 1. Os ângulos de correção do *gimbal* são referidos como azimute[1] e elevação representados por α e γ, o segundo pode ser visto na Figura 2.

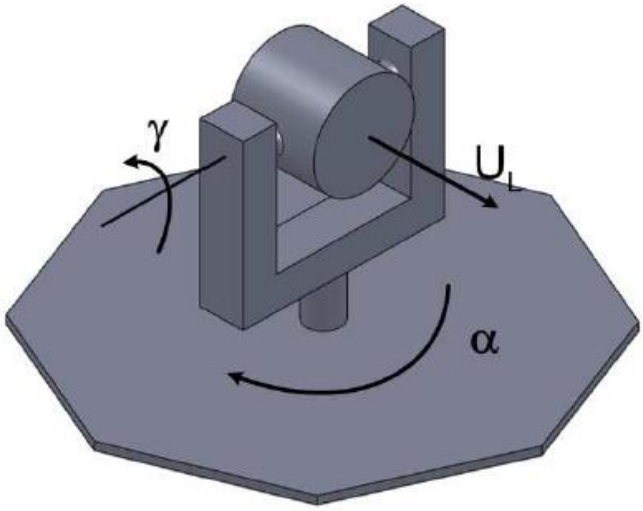
Figura 1 - Representação de ângulos de Euler XYZ, ou *roll, pitch e yaw*.



Fonte: Andrei Giordano Holanda Battistel (2011).

[1] É uma medida de abertura angular do sistema de coordenadas horizontal. Seu valor varia de 0º a 360º contando a partir do norte geográfico até a projeção de um alvo com o horizonte.

Figura 2 - Ângulos de compensação e direção da linha de visada.



Fonte: Andrei Giordano Holanda Battistel (2011).

São estudadas duas possibilidades de alocação dos sensores, conforme estes se situem no eixo externo ou no eixo interno do *gimbal*, casos estes também referidos como estabilização direta e indireta.

Como apenas dois sensores giroscópios são disponíveis, isto é, o movimento em um dos graus de liberdade da plataforma não pode ser medido; algumas hipóteses adicionais foram levadas em consideração pelo autor no desenvolvimento do controle. São elas:

Hipótese 1: "Os movimentos translacionais do veículo não são compensados, apenas os rotacionais";

Hipótese 2: "O *roll* e o *pitch* da plataforma devem ser conhecidos";

Hipótese 3: "O veículo não deve apresentar velocidade angular em algum dos três eixos, isto é, deve ter *roll*, *pitch* ou *yaw* constante".

A dissertação trata as hipóteses da seguinte maneira:

A primeira hipótese deve-se ao fato de os sensores mediram apenas velocidades angulares, de forma que com apenas esta informação não se pode obter o deslocamento linear do veículo e, portanto, não se pode compensá-lo. Para tal faz-se necessário o emprego de acelerômetros, e ainda com certa dificuldade, uma vez que os sinais precisam ser integrados duas vezes, gerando erros de deriva. Embora possa parecer que esta consideração por si só já insira erros no sistema, ele levanta um detalhe sutil que deve ser considerado: no problema de estabilização, ao invés de manter o sistema apontando para

um alvo específico, busca-se mantê-lo apontando para uma direção fixa. Isto é, procura-se manter um vetor constante em respeito ao sistema inercial apesar dos movimentos do veículo. Desta forma, uma câmera posicionada na extremidade deste sistema não terá uma imagem estática devido ao movimento de arco da própria estrutura. Uma vez que o *gimbal* tem uma dimensão qualquer de altura, um movimento em sua base gera um movimento translacional em sua extremidade. Este movimento não é compensado, o vetor da linha de visada é mantido constante, mas não fixo a um ponto.

A segunda hipótese é melhor compreendida no contexto do algoritmo propriamente dito e será discutida. Basicamente, deve-se ao fato de haver uma integração relacionada à obtenção da posição angular a partir de uma informação de velocidade. Uma condição inicial diferente do real naturalmente insere erros no cálculo. O artigo trata esta hipótese como não tão restritiva, já que tanto o *roll* como o *pitch* podem ser facilmente determinados, mesmo que de forma não muito precisa, por um conjunto de acelerômetros. Revelando esta informação, mesmo que aproximada, é suficiente para um desempenho razoável do sistema.

O mesmo trata a terceira hipótese como a mais importante das três estabelecidas. Como são disponíveis medidas ao redor de dois eixos apenas, uma velocidade angular não pode ser medida e alguma consideração adicional deve ser feita sobre o movimento do veículo. Por exemplo, uma plataforma colocada sobre um veículo cujo rumo não varia pode ser estabilizada medindo-se apenas as velocidades nos eixos x e y. Intuitivamente, pode não ser tão óbvio que uma terceira medida possa ser descartada, uma vez que um sensor colocado sobre o eixo z do veículo captaria alguma velocidade quando o sistema fosse submetido a algum *pitch* ou *roll*. No entanto ele afirma que, matematicamente é possível notar que a medida deste sensor hipotético é combinação linear das outras duas.

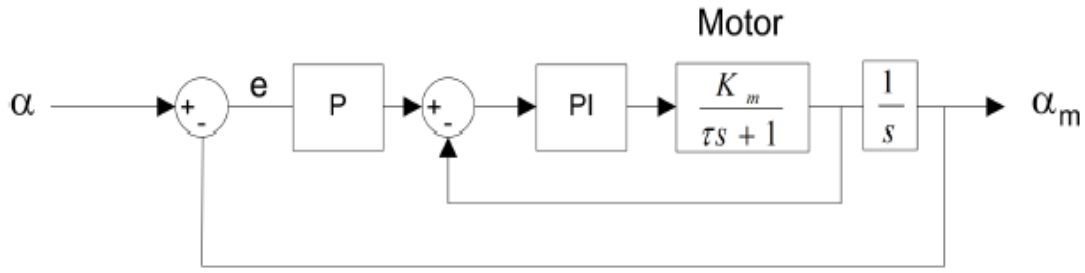
Consideramos aqui os casos onde o veículo não apresenta variações de *yaw*, ou rumo, e onde não apresenta variação de *roll*. Para considerações são citados exemplos de aplicações como uma plataforma sobre um navio de grande porte, cujo rumo varia pouco e muito lentamente. O caso de *roll* constante contempla uma viatura de grande porte, como por exemplo um blindado com esteiras, uma vez que se trata de uma estrutura rígida com pouco balanço.

Para a compensação dos movimentos translacionais o autor utiliza-se de sensores adicionais, como por exemplo acelerômetros, uma vez que os giroscópios captam apenas velocidades angulares. Neste caso, as acelerações devem ser integradas duas vezes a fim de obter o deslocamento linear.

Até então, os algoritmos apresentados na dissertação preocupam-se apenas com a compensação dos movimentos rotacionais do veículo onde a plataforma será utilizada. Para que se possa compensar também a translação do navio, o autor considera necessário a disponibilidade de medidas de acelerômetros, a fim de detectar a magnitude dos deslocamentos.

Para as simulações apresentadas, o autor aplica um controle do tipo Proporcional-Proporcional-Integral, ou P-PI. Este controle corresponde a uma malha interna de velocidade, do tipo PI, com uma malha externa de posição com controle proporcional, conforme mostra a Figura 3.

Figura 3 - Ângulos de compensação e direção da linha de visada.



Fonte: Andrei Giordano Holanda Battistel (2011).

Uma maneira de sintonizar este controlador encontrada pelo autor é, ajustar a sintonia do controle PI de velocidade para obter uma resposta rápida e em seguida ajustar proporcional de posição para reduzir o erro de seguimento. Esta estratégia é interessante uma vez que há uma restrição para o erro máximo na estabilização, erro este que depende diretamente do erro de controle.

Outra alternativa levantada pelo mesmo é, utilizar o PI de velocidade para cancelar o polo do motor. Dado um motor com ganho km e constante de tempo, ele apresenta a função de transferência de posição para velocidade conforme mostrado na Equação 3.3.1:

$$G(s) = \frac{k_m}{\tau s+1} \tag{3.3.1}$$

Sendo o PI apresentado na Equação 3.3.2:

$$G(s) = k_p * \frac{k_i}{s} \tag{3.3.2}$$

O autor escolhe $k_i = k_p$, chegou-se à seguinte Equação 3.3.3 para associação em série:

$$G_{OL}(s) = \frac{k_m}{\tau s+1} * \left(k_p + \frac{k_p}{s}\right) = \frac{k_m+k_p}{\tau s} \tag{3.3.3}$$

Resultando em uma malha fechada de um sistema de primeira ordem, apresentada pelo autor com a Equação 3.1.4:

$$H(s) = \frac{k_m+k_p}{\tau s+ k_m*k_p} \tag{3.3.4}$$

O autor apresenta resultados experimentais para a estabilização de *roll* e *pitch*, basicamente dos ângulos de compensação que permitem calcular um erro aproximado de estabilização, também exibido na dissertação do mesmo. Feito esses testes, o autor verifica alguns dos aspectos já discutidos na simulação e compara o resultado obtido em laboratório com aqueles esperados.

A dissertação é concluída mediante a consideração de restrições na movimentação do veículo onde a plataforma é colocada, ele afirma ser possível obter a estabilização mesmo com duas medidas. Para isto, supôs os cenários onde têm se *yaw* ou *roll* constantes, para duas condições distintas de alocação dos sensores, caracterizando estabilização direta e indireta. Dentro deste quadro, o autor desenvolveu diferentes algoritmos através dos quais se pôde obter a orientação da plataforma calculando-se a velocidade angular desta com apenas duas medidas. De posse da orientação, o mesmo pôde calcular os ângulos de correção. Além do algoritmo, obtiveram expressões para calcular os erros nessas velocidades, no caso onde as hipóteses de *yaw* ou *roll* constantes não são satisfeitas.

Já em uma monografia como parte dos requisitos para obtenção do Grau de Engenheiro de Controle e Automação da Universidade Federal de Ouro Preto (LAGE. 2016), o problema novamente é apresentado com a montagem de uma plataforma servo-

controlada, com dois graus de liberdade, e um sistema de controle a fim de manter equilibrados sobre essa, objetos de pequeno peso.

O autor apresenta o modelo matemático do mecanismo, através de testes experimentais por meio do sensor MPU-6050 que, através dos seus sensores acelerômetros e giroscópios, possibilita o cálculo da posição angular X e Y da plataforma. O autor ainda desenvolveu um controlador do tipo PID com o objetivo de manter a plataforma na posição horizontal, independentemente de movimentos externos. Auxiliado pelos programas *Matlab, Excel e Minitab*. O arduino foi a plataforma escolhida, pelo mesmo, tanto para o processo de aquisição de dados quanto de controle. São abordados diferentes metodologias para a sintonia do controlador como Ziegler-Nichols, *pidtool* do *Matlab* e o método de aproximações sucessivas ou tentativa e erro, que forneceu segundo ele em seu resumo, o resultado mais satisfatório.

Mais adiante no texto ele relata as aquisições de todos os dados, feitas através da comunicação USB entre o arduino e um microcomputador. Exibindo-os através do monitor serial do arduino. Para cada experimento, os dados foram copiados do monitor serial manualmente e inseridos em uma planilha do *Excel* em branco.

Descartando o modo DMP, e tendo em vista que os resultados dos filtros complementar e kalman foram bem próximos, o autor optou por escolher o complementar, que segundo ele a sua implementação era mais fácil, uma vez que não teria a necessidade de usar uma biblioteca externa economizando assim processamento e memória do microcontrolador.

A modelagem matemática da plataforma, foi feita segundo ele aplicando um degrau, ou seja, partindo-se de um valor inicial, normalmente zero, faz-se a plataforma mudar de orientação para um valor conhecido o mais rápido possível, processo feito através dos servos. Verificando a resposta, para ambos os eixos X e Y. Dessa forma foi possível avaliar se o processo corresponde a um sistema de primeira ordem ou segunda ordem.

Novamente em uma outra dissertação de mestrado (OLIVEIRA, Jr. 2016), uma plataforma inercial autônoma com três graus de liberdade foi construída para a aplicação em estabilização de sensores, por exemplo, gravimétricos estacionários e embarcados. O sistema apresentado é desenvolvido utilizando um sensor micro eletromecânico, MEMS, e um microcontrolador para aquisição, processamento e envio dos dados ao sistema de controle e aquisição de dados. Sistema esse bem parecido com o que vamos desenvolver

e apresentar aqui. Para o controle dos ângulos de inclinação e orientação da plataforma, o autor implementou um controlador PID digital utilizando microcontrolador.

Para monitoramento da plataforma o autor desenvolveu um programa para aquisição de dados em tempo real em ambiente *Matlab*, por meio onde ele foi capaz de visualizar e gravar os sinais da IMU, Unidade de Medida Inercial - *Inertial Measure Unit*, os ângulos de inclinação e a velocidade angular.

4. MATERIAIS E MÉTODOS

Neste capítulo será descrito de forma detalhada sobre o caminho percorrido para alcançar os objetivos deste trabalho, descrevendo sobre os componentes utilizados e os cálculos para dimensionamentos dos mesmos, os esquemáticos eletrônicos montados juntamente com suas simulações.

4.1. Metodologia

Este projeto é desenvolvido considerando uma alimentação elétrica de 5V.

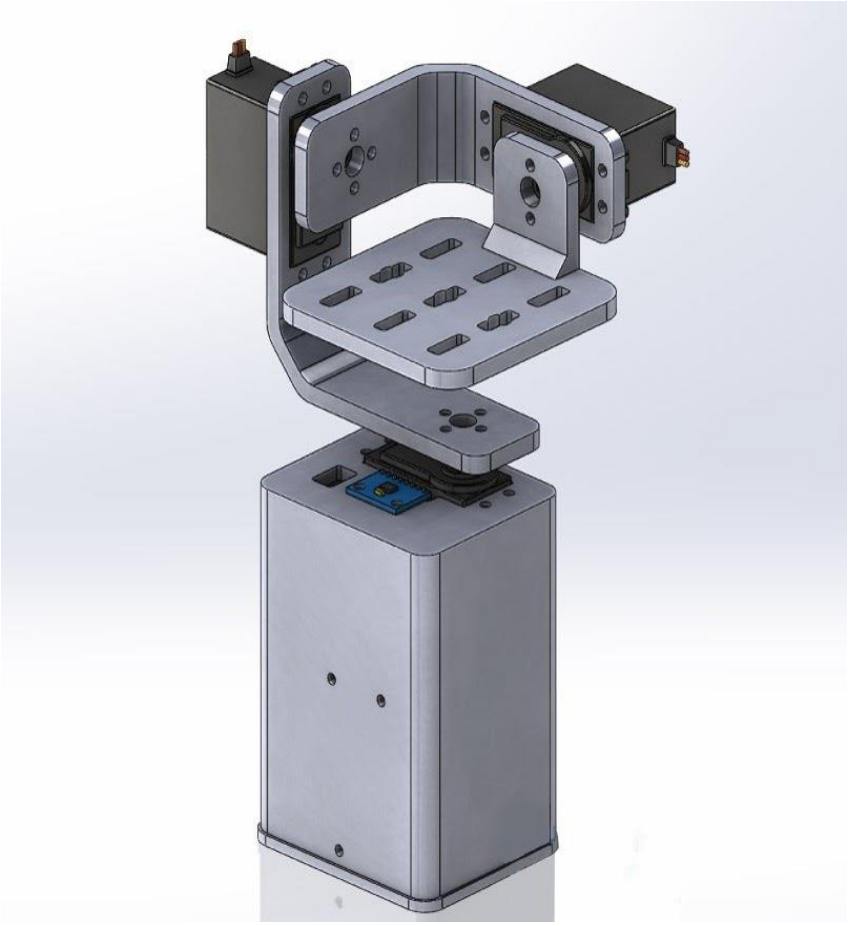
Para implementar e desenvolver de uma forma mais adequada o projeto, as seguintes etapas foram executadas:

- Projeto da plataforma inercial em Software para modelagem 3D para impressão das peças;
- Montagem dos circuitos eletrônicos e componentes nas peças impressas;
- Elaboração do *software*;
- Modelagem do Sistema;
- Aquisição e tratamento dos dados aferidos pelos sensores e atuadores.

4.2. Construção da Plataforma

A plataforma inercial foi desenvolvida em software de modelagem 3D e impressa na impressora 3D do laboratório de instalações elétricas da Universidade Federal de Lavras. Seu projeto deu-se de modo a possibilitar a futura utilização da plataforma para fixação em um manipulador a ser utilizado para trabalhar com aquisição de imagens. A Figura 4 representa o modelo.

Figura 4 - Modelo 3D do manipulador.



Fonte: How to Mechatronics (2020).

4.3. Componentes do Sistema

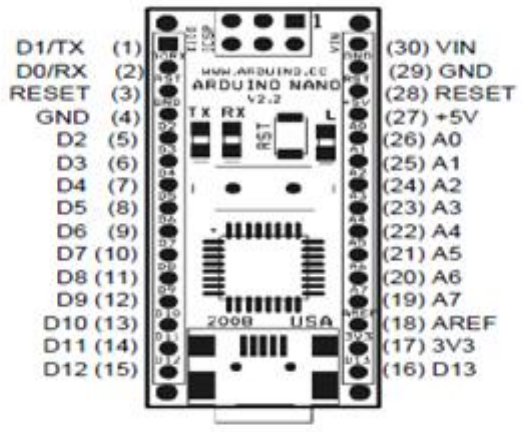
Primeiramente, para desenvolvimento do *hardware*, composto por um arduino nano, três servomotores, um MPU-6050 acelerômetro, é necessário que se tenha alguns conceitos em eletrônica e conhecimento de alguns componentes que foram utilizados no projeto, tais como:

4.3.1 Microcontrolador

Arduino são placas com entradas e saídas analógicas e digitais, interface USB para comunicação e um microcontrolador, e de fácil implementação com uma linguagem própria, muito semelhante a linguagem C, e com bibliotecas úteis para diversos tipos diferentes de aplicações. Existem vários tipos e modelos, como o arduino uno, arduino mega, arduino nano, entre outros. Se diferenciam pelos números de portas, número de PWM's disponíveis, memória, velocidade e entre outras (ARDUINO. 2020).

Para esse trabalho foi escolhido o arduino nano, por atender todos os requisitos, tais como, 3 saídas PWM para os servos, comunicação I2C para o sensor MPU-6050 e comunicação USB. A Figura 5 abaixo mostra o *layout* das pinagens do arduino nano.

Figura 5 - Pinagens do arduino nano.



Fonte: Retirada do *datasheet* (2019).

4.3.2 Servomotores

Servomotores são dispositivos eletromecânico, de malha fechada, ou seja, recebem sinal de referência e, de acordo com sua posição, é calculado o erro. De acordo com esse erro, o controlador atua no motor, enviando o eixo do servo para a posição desejada, com velocidade monitorada.

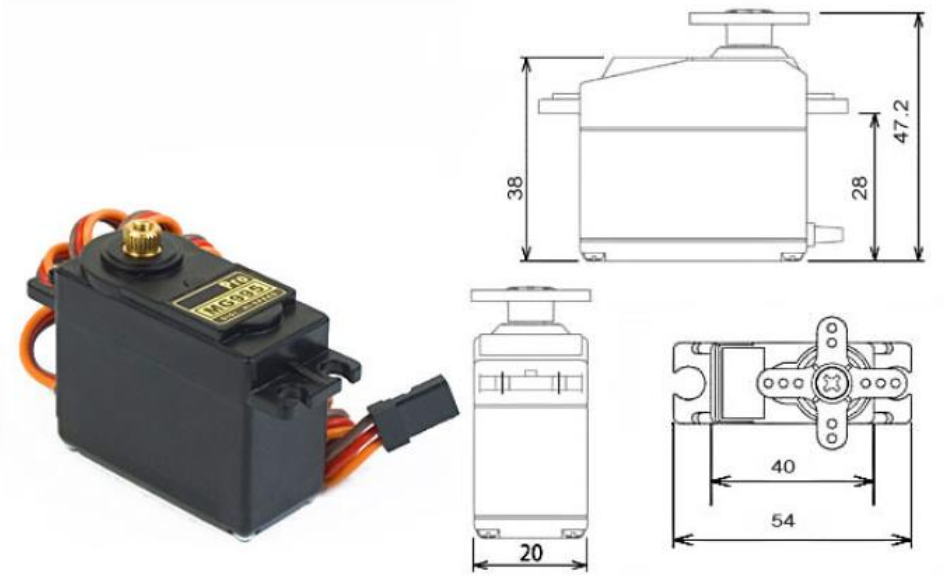
Os servos possuem três fios de interface, sendo um para o gnd de coloração marrom, uma para o vcc de coloração vermelha, e um para o sinal de controle de coloração laranja. O sinal de controle é enviado pelo microcontrolador para informar a posição que se deseja colocar o eixo do servomotor. O sinal de controle é um sinal chamado de PWM - *Pulse Width Modulation* (ME102B. 2015).

O PWM é a forma mais utilizada em sistemas digitais para reguladores de tensão e transferência de potência. Os servomotores utilizam esse tipo de sinal como sinal de referência, ou seja, ele transporta a informação contendo a posição angular desejada no eixo do servo.

A modulação por largura de pulso, conhecida como PWM, é um sinal que contém frequência e período fixos, sendo possível variar o tempo em que o sinal se encontra em nível alto - *duty cycle* - para transportar uma informação sobre um canal de comunicação ou para controlar o valor da alimentação entregue à carga. Em geral, a frequência utilizada para a comunicação com os servomotores é de 50Hz, e o *duty cycle*, largura de pulso do sinal de controle em nível alto, varia de 1ms a 2ms, ou seja, 1ms corresponde a 0 graus e 2ms corresponde a 180 graus. Logo, quando a largura do pulso varia dentro deste intervalo, é possível posicionar o servomotor na posição desejada (LAGE. 2016).

O circuito de controle interno dos servos é formado por componentes eletrônicos, normalmente formando um controlador PID, que recebe o sinal do sensor, potenciômetro, com a posição angular do eixo e o sinal de controle, com a posição desejada. Dessa forma, o circuito aciona o motor no sentido correto para posicionar o eixo na posição desejada. A Figura 6 abaixo mostra um exemplo do servomotor utilizado no trabalho.

Figura 6 - Imagem ilustrativa do servomotor.

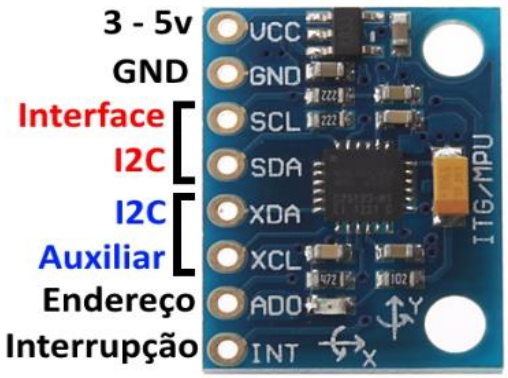


Fonte: Retirada do *datasheet* (2020).

4.3.3 MPU-6050

O MPU-6050 é um sensor integrado, que reúne três sensores giroscópio, um para cada eixo X, Y e Z, três sensores acelerômetro um para cada eixo também e um processador. Ele utiliza a comunicação I2C para a aquisição dos dados, possui três conversores analógico/digital de 16 bits internamente para a discretização dos valores dos acelerômetros e dos giroscópios internos. O giroscópio mede a velocidade rotacional ou a taxa de alteração da posição angular ao longo do tempo. Ele utiliza se da tecnologia MEMS e o efeito coriolis para medir, as saídas do giroscópio são em graus por segundo; portanto, para obter a posição angular, precisamos apenas integrar a velocidade angular (LAGE. 2016). A Figura 7 abaixo mostra um exemplo do MPU-6050 utilizado no trabalho.

Figura 7 - Sensor MPU-6050.



Fonte: Filipeflop (2019).

Resumidamente, ele pode medir a aceleração gravitacional ao longo dos 3 eixos e, usando alguma matemática da trigonometria, pode se calcular o ângulo em que o sensor está posicionado. Portanto, combinando os dados do acelerômetro e do giroscópio, pode se obter informações muito precisas sobre a orientação do sensor (PROJECTS. 2019).

4.3.4. MEMS

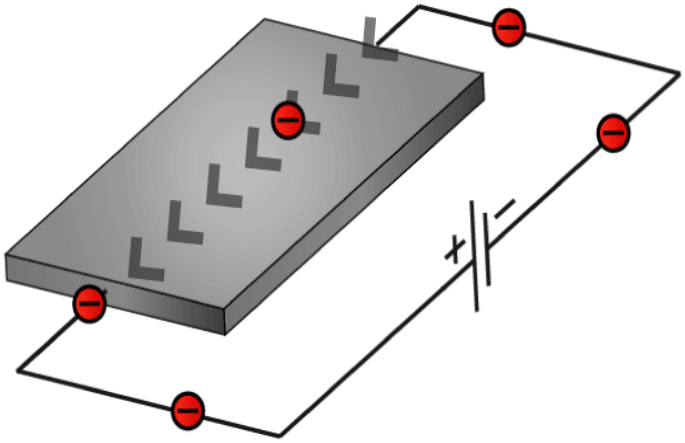
MEMS são sistemas ou dispositivos muito pequenos, compostos por microcomponentes que variam de 0,001 mm a 0,1 mm de tamanho. Esses componentes são feitos de silicone, polímeros, metais e / ou cerâmica, e geralmente são combinados com uma CPU para completar o sistema.

Ele mede a aceleração medindo a mudança na capacitância. Tem uma massa presa a uma mola que é confinada para se mover ao longo de uma direção e fixar placas externas. Portanto, quando uma aceleração na direção específica será aplicada, a massa se moverá e a capacitância entre as placas e a massa mudará. Essa mudança na capacitância será medida, processada e corresponderá a um valor de aceleração específico (MECHATRONICS. 2020).

Magnetômetro MEMS, mede o campo magnético da terra usando o efeito hall[2] ou o efeito resistente ao magneto. Na verdade, quase 90% dos sensores no mercado usam o efeito hall. A Figura 8 abaixo mostra como ele funciona.

[2] É como um transdutor analógico que retorna diretamente um valor de tensão. Com um campo magnético conhecido, pode-se determinar a sua distância até a placa hall.

Figura 8 - Imagem ilustrativa do funcionamento do magnetômetro.



Fonte: How to Mechatronics (2020).

Ao definirmos a corrente para fluir através na placa do magnetômetro, os elétrons fluirão diretamente de um lado para o outro da placa. Se trouxermos algum campo magnético para perto da placa, perturbaríamos o fluxo direto e os elétrons se desviariam para um lado da placa e os polos positivos para o outro lado da placa. Obteremos assim alguma tensão que depende da força do campo magnético e de sua direção.

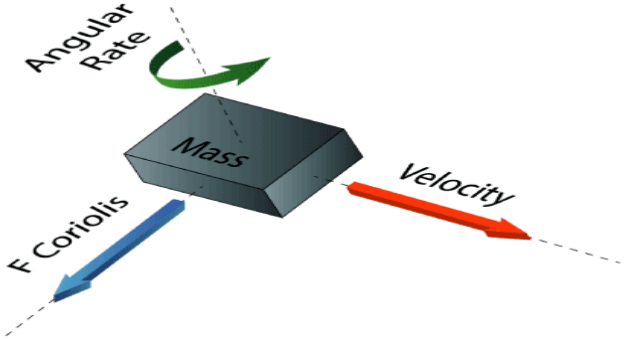
4.3.5. Giroscópio

O giroscópio mede a taxa angular usando o efeito coriolis[3]. Quando uma massa está se movendo em uma direção específica com uma velocidade específica e quando uma taxa angular externa será aplicada como mostra a seta verde, ocorrerá uma força, como mostra a seta azul, o que causará deslocamento perpendicular da massa, como mostra a seta vermelha.

Tão semelhante ao acelerômetro, esse deslocamento causará alterações na capacitância que serão medidas, processadas e corresponderão a uma taxa angular específica (MECHATRONICS. 2020). A Figura 9 abaixo mostra como ele se comporta.

[3] É a tendência que qualquer corpo em movimento sobre a superfície terrestre tem em mudar seu curso devido à direção rotacional e da velocidade da Terra.

Figura 9 - Imagem ilustrativa do comportamento do giroscópio.



Fonte: How to Mechatronics (2020).

Para calcular um ângulo através do giroscópio, é necessário integrar os valores lidos em pequenos intervalos de tempo. Isso gera um custo computacional maior e o fator mais negativo é que o erro tende a se incrementar junto com as integrações.

O MPU-6050 disponibiliza valores adimensionais para os giroscópios. Para se chegar a valores expressos por graus/segundo, deve-se dividir os valores entregues por 131, considerando uma precisão de ±250 graus por segundo. Se for de interesse aumentar essa precisão, o divisor muda. O valor calculado, multiplicado por um intervalo de tempo entre duas leituras, resulta na variação angular do eixo em questão (SENSE. 2013).

O giroscópio sofre um efeito de desvio do valor real, que deveria ser medido ao longo do tempo. Quando ele está em posição estável, sem movimentação, seu valor cresce ou decresce devido à integração. Essa taxa de desvio é conhecida como bias.

4.3.6. Acelerômetro

Considerando uma rotação ou inclinação nos 3 eixos, utiliza-se então das relações trigonométricas apresentadas nas Equações 4.3.1, 4.3.2 e 4.3.3 a seguir, levando em conta a força “g” medida em cada eixo por cada acelerômetro Ax, Ay e Az (TUCK. 2007).

$$\rho = \arctan\left(\frac{A_x}{\sqrt{{A_y}^2+{A_z}^2}}\right) \tag{4.3.1}$$

$$\varphi = \arctan\left(\frac{A_y}{\sqrt{{A_x}^2+{A_z}^2}}\right) \quad (4.3.2)$$

$$\theta = \arctan\left(\frac{\sqrt{{A_x}^2+{A_z}^2}}{A_z}\right) \quad (4.3.3)$$

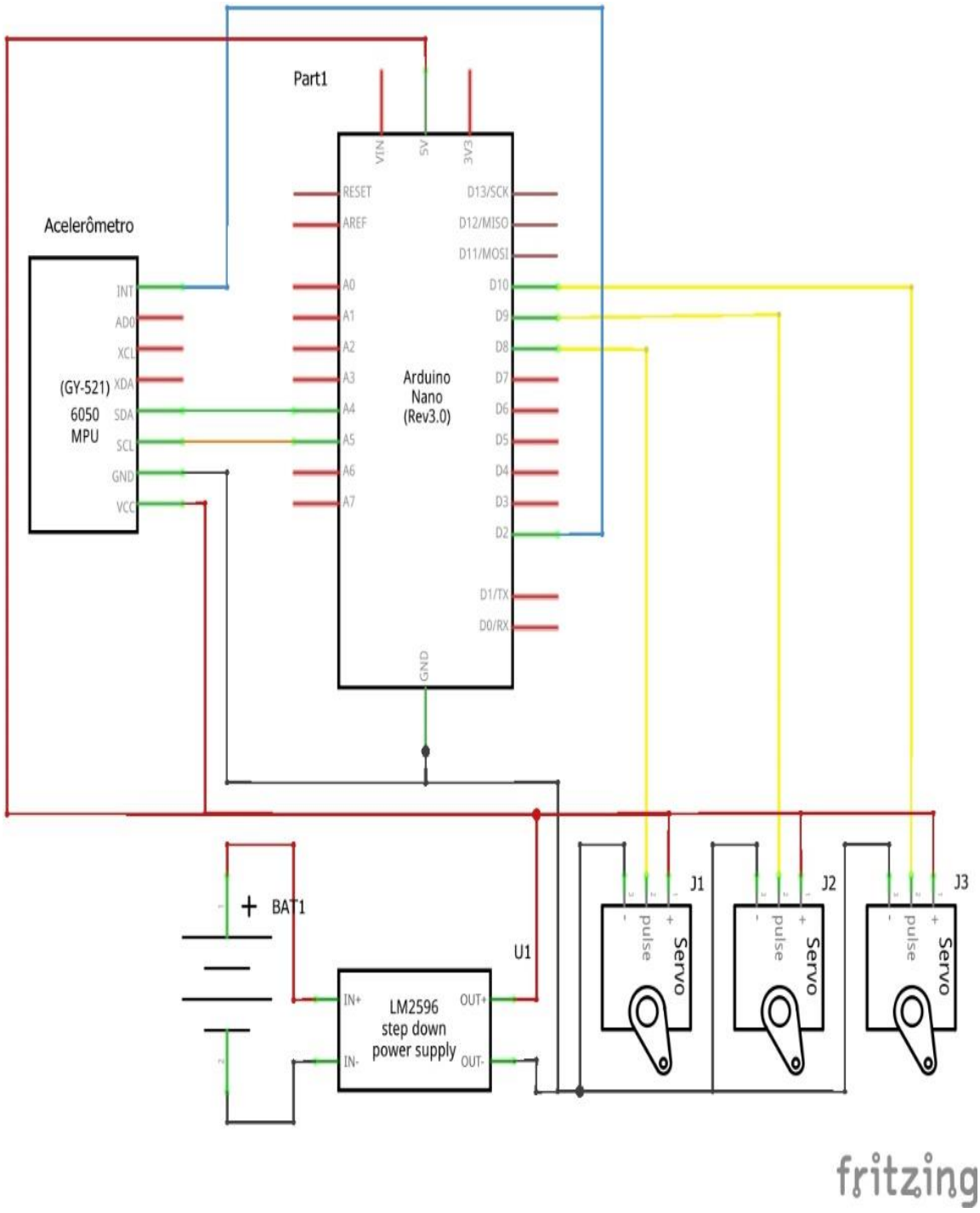
Onde os eixos X, Y e Z correspondem respectivamente a *roll*, *pitch* e *yaw* ou $\varphi, \rho \; e \; \theta$.

O MPU-6050 nos fornece valores adimensionais, chamados *raw values*, que devem ser convertidos para m/s2, mais precisamente, para múltiplos de forças “g” 9,81m/s2. Deve-se dividir o valor aferido por 16.384 para uma sensibilidade de +/- 2g de leitura. Logo, se o sensor retorna um valor de 16.384 no eixo X, significa que existe 1g naquele eixo. Essa divisão pode ser feita antes ou depois de se calcular os ângulos (SENSE. 2013).

4.3.7. Circuito Eletrônico

Com todos os componentes acima citados, foi montado um esquemático para o circuito embarcado, utilizando o *software Fritzing.* A Figura 10 abaixo mostra o circuito.

Figura 10 - Esquemático do circuito.



Fonte: Do autor (2019).

O circuito constitui de uma bateria como fonte de alimentação, uma fonte abaixadora de tensão onde mantém a tensão de alimentação constante em 5 volts para os demais componentes. O vcc está na cor vermelho alimentando os três servomotores na porta vcc do mesmo, alimentando também o arduino nano na porta 5V e por último alimentando o MPU-6050 na porta vcc. O gnd está na cor preta alimentando também os três servomotores na porta gnd, alimentando o arduino e o MPU-6050 pela porta gnd de ambos. Já o sinal de controle pwm está representado na cor amarela interligando os servos ao arduino em portas pré-selecionadas.

A porta SCL, serial *clock*, do MPU-6050 está ligado a porta analógica 5 do arduino, a porta SDA, serial data, do MPU-6050 está ligado a porta analógica 4 do arduino. A porta INT, interrupção, do MPU-6050 está ligado a porta digital 2 do arduino.

4.3.8. Comunicação

A comunicação USB, realizada entre o arduino e um microcomputador, é muito importante para a aquisição dos dados durante a fase de testes, calibragem dos sensores e sintonia do controlador.

O arduino utiliza a comunicação serial UART e um conversor FTDI que converte do protocolo RS-232 para USB. Um driver é instalado no microcomputador e a comunicação é estabelecida. No *software* que acompanha o arduino, existe um monitor serial, por onde é possível acompanhar pelo computador as informações trocadas entre estes (LAGE. 2016).

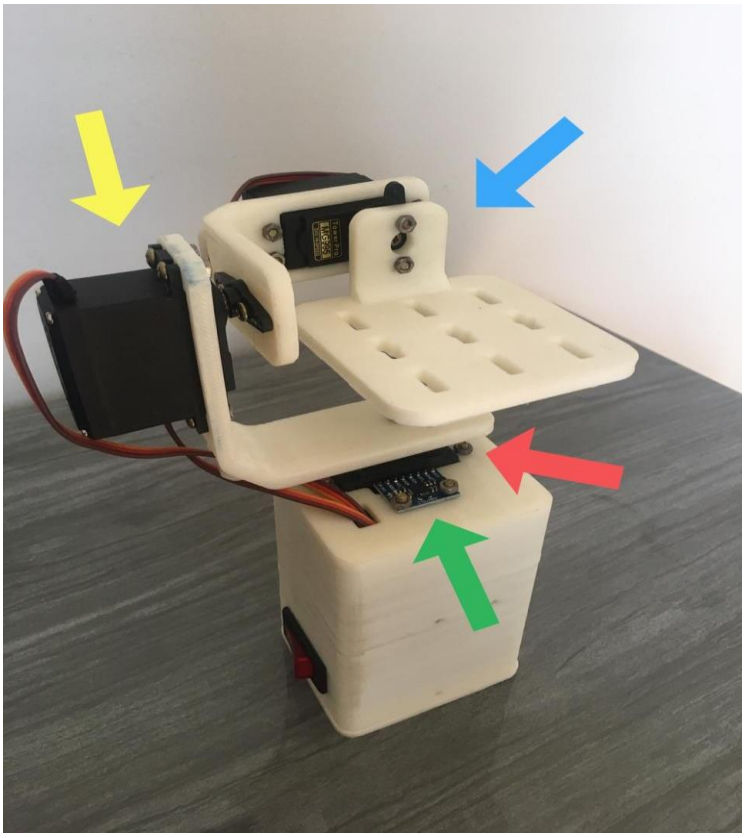
4.4. Montagem dos Componentes

Após impressão das peças, os servomotores foram devidamente fixados em suas posições de modo a controlar o movimento nos três eixos *roll, pitch e yaw*.

Começando com a instalação do servo *yaw*. Usando parafusos e porcas M3, prendi-a na base conforme a seta em vermelho. Em seguida, usando o mesmo método, o servo *roll* foi parafusado conforme a seta amarela. As peças são projetadas especificamente para caber facilmente nos servos MG995. Repetindo esse processo para montar o restante dos componentes, o servo *pitch* e a plataforma superior foram montados conforme a seta azul.

O módulo MPU-6050 foi fixado na parte superior da base conforme a seta verde, para facilitar as leituras e medições aferidas pelo acelerômetro e giroscópio. Todos os fios e demais componentes foram devidamente fixados no interior da base, para facilitar a locomoção da plataforma e transporte do protótipo. A Figura 11 apresenta o manipulador todo montado.

Figura 11 - Manipulador.



Fonte: Do autor (2020).

Para alimentar o projeto, uma bateria recarregável foi implementada ao circuito. Bateria essa que alimenta o circuito por inteiro como: os servos, arduino e o MPU-6050, interligada ao circuito por um conector. A bateria fornece uma tensão de 9V, porém é preciso de 5V para alimentar o arduino e os servos. É por este motivo que um conversor *buck* foi utilizado para converter 9V para 5V.

4.5. Software

Como foi empregado acima, o *software* utilizado foi o arduino, possuindo uma comunidade muito ativa por todo o mundo que desenvolvem bibliotecas úteis para diversos tipos diferentes de aplicações. Dessa forma, o trabalho de desenvolvimento é facilitado e otimizado, sem a necessidade de desenvolvimento do *hardware*, principalmente.

O mesmo é responsável por fazer aquisições de dados e dar movimento ao manipulador, controlando e corrigindo os movimentos.

A comunicação serial via USB, realizada entre o arduino e um microcomputador, é muito importante, ela realiza a aquisição dos dados durante a fase de testes, calibragem dos sensores e sintonia do controlador. No *software* que acompanha o arduino, existe um monitor serial, por onde é possível acompanhar pelo computador as informações trocadas entre eles.

4.6. Calibração do Módulo MPU-6050

Para a calibração, primeiro foi necessário incluir a biblioteca *"Wire.h"* que é usada para a comunicação I2C e definir algumas variáveis necessárias para armazenar os dados. Além disso, podemos selecionar a faixa de escala completa para o acelerômetro e o giroscópio usando seus registros de configuração. Neste trabalho, foi utilizado o intervalo padrão de + - 2g para o acelerômetro e o intervalo de 250 graus/s para o giroscópio. A Figura 12 representa esse trecho do código comentado.

Figura 12 - Definição das variáveis.

```
void setup() {
  Serial.begin(19200);
  Wire.begin();                      // Inicializar comunicação
  Wire.beginTransmission(MPU);       // Iniciar a comunicação com MPU6050 // MPU=0x68
  Wire.write(0x6B);                  // Fale com o registro 6B
  Wire.write(0x00);                  // Make reset - coloque um 0 no registro 6B
  Wire.endTransmission(true);        // Encerrar a transmissão
  /*
  // Configurar a sensibilidade do acelerômetro - faixa de escala completa (padrão +/- 2g)
  // Wire.beginTransmission (MPU);
  // Wire.write (0x1C); // Fale com o registro ACCEL_CONFIG (1C hex)
  // Wire.write (0x10); // Defina os bits de registro como 00010000 (+/- 8g de escala completa)
  // Wire.endTransmission (true);
  // Configurar a sensibilidade do giroscópio - faixa de escala completa (padrão +/- 250graus/s)
  // Wire.beginTransmission (MPU);
  // Wire.write (0x1B); // Fale com o registro GYRO_CONFIG (1B hex)
  // Wire.write (0x10); // Defina os bits de registro como 00010000 (1000graus/s em escala completa)
  // Wire.endTransmission (true);
  // delay (20);
  */

  // Chame essa função se precisar obter os valores de erro IMU para o módulo
  calculate_IMU_error();
  delay(20);

}
```

Fonte: Do autor (2020).

Na seção de loop, é feita a leitura dos dados do acelerômetro. Os dados para cada eixo são armazenados em dois bytes ou registros e podendo ser observado os endereços desses registros no *datasheet* do sensor. Para ler todos eles, foi usado a função *"requiestFrom () "*, onde foi solicitado a leitura dos seis registros para os eixos X, Y e Z.

Em seguida, foi feita a leitura dos dados de cada registrador e, como as saídas são dois complementos, foram combinados adequadamente para obter os valores corretos. A Figura 13 representa esse trecho do código.

Figura 13 - Leitura dos dados do acelerômetro.

```
// === Ler dados do acelerômetro === //
Wire.beginTransmission(MPU);
Wire.write(0x3B); // Comece com o registro 0x3B (ACCEL_XOUT_H)
Wire.endTransmission(false);
Wire.requestFrom(MPU, 6, true); // Ler 6 registros no total, cada valor de eixo é armazenado em 2 registros
// Para um intervalo de + -2g, precisamos dividir os valores brutos por 16384, de acordo com o datasheet
AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // valor do eixo X
AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // valor do eixo Y
AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // valor do eixo Z
```

Fonte: Do autor (2020).

Assim, para obter valores de saída de -1g a + 1g, adequados para o cálculo dos ângulos, dividimos a saída pela a sensibilidade selecionada. A Tabela 1 mostra a *sensitivity* utilizada na divisão.

Tabela 1 - Sensibilidade do Acelerômetro.

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

Fonte: Retirada do *datasheet* (2020).

Utilizando as fórmulas 4.3.1, 4.3.2 e 4.3.3, calculamos os ângulos de rotação e inclinação a partir dos dados do acelerômetro. Usando o mesmo método, foi obtido os dados do giroscópio. Foi feita a leitura dos seis registros do giroscópio, combinando os dados adequadamente. A Figura 14 representa esse trecho do código.

Figura 14 - Leitura dos dados do giroscópio.

```
// === Ler dados do giroscópio === //
previousTime = currentTime;        // O horário anterior é armazanado antes do horário real lido
currentTime = millis();            // Hora atual, hora atual, leitura
elapsedTime = (currentTime - previousTime) / 1000; // Divide por 1000 para obter segundos
Wire.beginTransmission(MPU);
Wire.write(0x43); // Primeiro endereço de registro de dados giroscópicos 0x43
Wire.endTransmission(false);
Wire.requestFrom(MPU, 6, true); // Ler 4 registros no total, cada valor de eixo é armazenado em 2 registro
GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // Para uma faixa de 250graus/s, primeiro dividimos o valor bruto por 131,0, de acordo com o datasheet
GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;
```

Fonte: Do autor (2020).

Os valores obtidos na leitura dos dados foram divididos pela *sensitivity* previamente selecionada, a fim de obter a saída em graus por segundo. A Tabela 2 mostra a *sensitivity* utilizada na divisão.

Tabela 2 - Sensibilidade do Giroscópio.

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

Fonte: Retirada do *datasheet* (2020).

No código, foi corrigido os valores de saída com alguns pequenos valores de erro calculados. Para calcular esses erros, é chamado a função customizada *"calculate_IMU_error()"* enquanto o sensor estiver na posição fixa, os valores de saída esperados devem ser 0. Portanto, com este cálculo, é obtido o erro médio do sensor.

Como as saídas são em graus por segundo, agora foi preciso multiplicá-las com o tempo para obter apenas graus. O valor do tempo é capturado antes de cada iteração de leitura usando a função *"millis () "*. A Figura 15 representa esse trecho do código.

Figura 15 - Correção dos valores.

```
// Corrija as saídas com os valores de erro calculados
GyroX = GyroX + 0.56; // GyroErrorX ~(-0.56)
GyroY = GyroY - 2; // GyroErrorY ~(2)
GyroZ = GyroZ + 0.79; // GyroErrorZ ~ (-0.8)

// Atualmente, os valores brutos estão em graus por segundo, graus/s, portanto, é preciso multiplicar por segundos para obter o ângulo em graus
gyroAngleX = gyroAngleX + GyroX * elapsedTime; // graus/s * s = graus
gyroAngleY = gyroAngleY + GyroY * elapsedTime;
yaw =  yaw + GyroZ * elapsedTime;
```

Fonte: Do autor (2020).

A variável *"elapsedTime"* é a diferença entre o horário atual e o horário anterior lidos das medias, sendo em segundos. Como os valores estão em graus por segundos é preciso multiplicar por segundos para obter somente o ângulo em graus.

Finalmente, foi interligado os dados do acelerômetro e do giroscópio usando um filtro complementar que normalmente é utilizado no meio acadêmico como referência a qualquer algoritmo que tenha como objetivo a fusão de dados redundantes ou similares de diferentes sensores para alcançar uma estimativa ótima de uma determinada variável. Este filtro opera no domínio da frequência e pode ser definido pelo uso de duas ou mais funções de transferência as quais complementam umas às outras.

Num sistema típico de duas entradas de dados, como este, uma delas proverá informação com ruído de alta frequência, que será filtrada através de um filtro passa-baixas. A outra produzirá dados com ruído de baixa frequência, sendo, portanto, filtrados através de um filtro passa-altas. Se estes filtros são matematicamente complementares, então o sinal filtrado será a reconstrução completa da variável sendo monitorada, eliminando os ruídos (BUENO[1]. 2014).

Se trata de uma combinação matemática baseada na definição de pesos para cada variável de entrada, no caso da plataforma, estas variáveis de entradas são os valores de ângulos aferidos pelos acelerômetros e giroscópios. A soma dos pesos deve ser igual a 1, as Equações 4.6.1, 4.6.2 e 4.6.3 mostram utilizadas nos cálculos do filtro encontram-se abaixo.

$$\theta = \alpha * Gy + (1 - \alpha) * Acc \tag{4.6.1}$$

$$\alpha = \frac{\tau}{\tau + dt} \tag{4.6.2}$$

$$Gy = \theta + \omega * dt \quad (4.6.3)$$

Onde:

$dt =$ Tempo de amostragem;
$\tau =$ Constante de tempo maior que a escala do ruído do acelerômetro;
$\omega =$ Velocidade angular atual aferida pelo giroscópio;
$Acc =$ Ângulo atual fornecido pelo acelerômetro.

Foi utilizado um tempo de amostragem de 0,04 segundos e constante de tempo igual a 1. Portanto, o peso alpha obtido foi de 0,96. Ou seja, 96% de peso para o giroscópio e 4% de peso para o acelerômetro. Deste modo, o valor final obtido para os ângulos é bem menos ruidoso e apresenta menores erros e desvios.

4.7. Aquisição e Tratamento dos Dados

Após a montagem da plataforma, foi iniciado a aquisição dos dados referentes aos acelerômetros e giroscópios. Logo após, foi realizado o cálculo dos ângulos e implementação do filtro complementar para diminuição do ruído do sinal.

As aquisições de dados, foram realizadas através da comunicação USB entre o arduino e um computador. Os dados foram exibidos no monitor serial do *software* arduino.

Ao obter os valores, primeiro foi feita a conversão de radianos em graus posteriormente foram executadas 300 leituras, pois o sensor antes disso fica em processo de calibração. Além disso, foi capturado o valor de *yaw*, que no início não é 0 como os valores de *pitch* e *roll*, é sempre algum valor aleatório.

Após as 300 leituras, foi definido o *yaw* como 0. Em seguida, mapeamos os valores de *roll*, *pitch e yaw*, de -90 a +90, em valores de 0 a 180, que corresponde ao envelope de trabalho dos servos. E finalmente, usando a função *"write"*, enviamos esses valores para os servos como sinais de controle. A Figura 16 representa essa parte do código.

Figura 16 - Mapeamento dos valores de *roll*, *pitch e yaw*.

```
// Após 300 leituras
else {
  ypr[0] = ypr[0] - correct; // Defina o Yaw como 0 graus - subtraia o último valor aleatório do Yaw do valor atual para fazer o Yaw 0 graus
  // Mapeie os valores do sensor MPU6050 de -90 a 90 para valores aceitáveis para o servocontrole de 0 a 180 (envelope de trabalho)
  int servo0Value = map(ypr[0], -90, 90, 0, 180);
  int servo1Value = map(ypr[1], -90, 90, 0, 180);
  int servo2Value = map(ypr[2], -90, 90, 180, 0);

  // Controle os servos de acordo com a orientação MPU6050
  servo0.write(servo0Value);
  servo1.write(servo1Value);
  servo2.write(servo2Value);
}
```

Fonte: Do autor (2020).

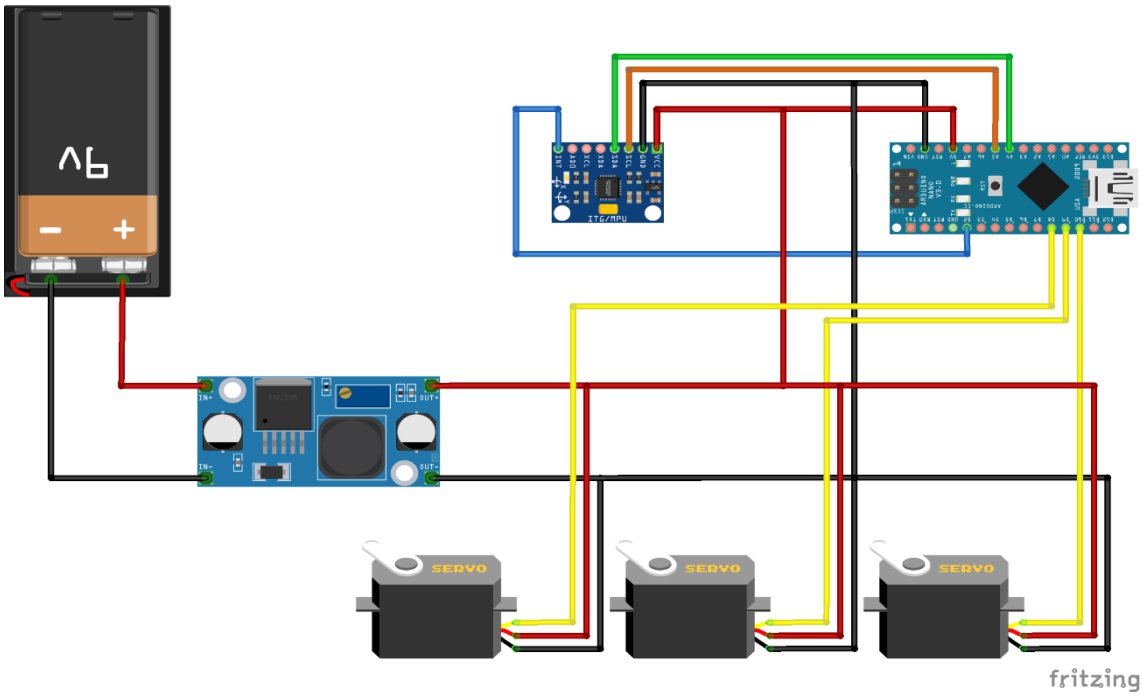
5. RESULTADOS

Neste capítulo serão relatados os resultados de cada teste nas principais partes do trabalho, funcionamento no circuito, implementação do *software*, a resposta do módulo MPU-6050 e o comportamento dos servomotores**.**

5.1. Analise do Funcionamento do Circuito

O circuito eletrônico implementado no trabalho, mostrou ser bem efetivo no que foi confeccionado e respondendo positivamente, a Figura 17 representa o mesmo com seus componentes meramente ilustrado desenhado no *software Fritzing.*

Figura 17 - Circuito eletrônico.



Fonte: Do autor (2019).

O circuito constitui de uma bateria de 9 volts como fonte de alimentação, uma fonte buck abaixadora de tensão onde mantém a tensão de alimentação constante em 5 volts para os demais componentes. O vcc está na cor vermelho alimentando os três servomotores na porta vcc do mesmo, alimentando também o arduino nano na porta 5V e por último alimentando o MPU-6050 na porta vcc. O gnd está na cor preta alimentando

também os três servomotores na porta gnd, alimentando o arduino e o MPU-6050 pela porta gnd de ambos. Já o sinal de controle pwm está representado na cor amarela interligando os servos ao arduino em portas pré-selecionadas.

A porta SCL, serial *clock*, do MPU-6050 está ligado a porta analógica 5 do arduino, a porta SDA, serial data, do MPU-6050 está ligado a porta analógica 4 do arduino. A porta INT, interrupção, do MPU-6050 está ligado a porta digital 2 do arduino.

Conforme mencionado no capitulo 4, tópico 3.7 o circuito segue as mesmas especificações. Os servomotores estão ligados nos pinos 10, 9 e 8 do arduino respectivamente comandam os eixos *yaw, pitch e roll,* na imagem estão apresentados da direita para a esquerda.

O comportamento dos servomotores, inicialmente apresentado no capitulo 4, tópico 3.2. Servomotores são dispositivos eletromecânico, de malha fechada, ou seja, recebem sinal de referência e, de acordo com sua posição, é calculado o erro. De acordo com esse erro, o controlador atua no motor, enviando o eixo do servo para a posição desejada, com velocidade monitorada.

O comportamento dos servomotores estão dentro do desejado, a implementação da correção do erro foi bastante satisfatória. Sendo feita a implementação do filtro complementar somente em dois eixos, *roll* e *pitch*. O comportamento do *yaw* é um pouco mais brusco, nada que interfira no manuseio do manipulador.

Sobre a implementação do *software*, como foi especificado no capitulo 4, tópico 5 o mesmo é responsável por fazer aquisições de dados e dar movimento ao manipulador, controlando e corrigindo os movimentos. Apresentando um comportamento dentro do esperado para o trabalho. Claramente é possível perceber a correção do servo para manter a base sempre na posição horizontal.

A resposta do modulo e a leitura do arduino para comandar os servomotores, apresenta um *delay,* não muito significativo. *Delay* esse esperado pela limitação dos componentes.

Em resposta ao módulo MPU-6050. A inclusão da biblioteca *"Wire.h"* para a comunicação I2C e definição de algumas variáveis teve um resultado satisfatório. O módulo responde aos comandos e perturbações externas, tanto o acelerômetro e o giroscópio estão perfeitamente calibrados e operantes.

As equações matemáticas utilizadas na calibração foram de suma importância e a implementação das mesmas no código gerou resultados satisfatórios. Os tratamentos de dados junto com o equacionamento combinaram na resposta efetiva no manipulador.

6. CONCLUSÃO

Este trabalho apresenta de forma bem detalhada o desenvolvimento de um sistema de controle de estabilização horizontal de uma plataforma inercial com três graus de liberdade, para aplicação prática de controle de dispositivos como os *gimbals*. E por meio dos resultados, os quais foram satisfatórios e da validação pelo manipulador montado, pode ser estabelecida como válido a aplicação deste sistema no âmbito de robótica.

Foram explorados conceitos de eletrônica, instrumentação, filtragem e tratamento de dados, desenvolvimento de *software*, utilização de ferramentas computacionais, redes de comunicação, teoria de controle e alguns conceitos de mecânica.

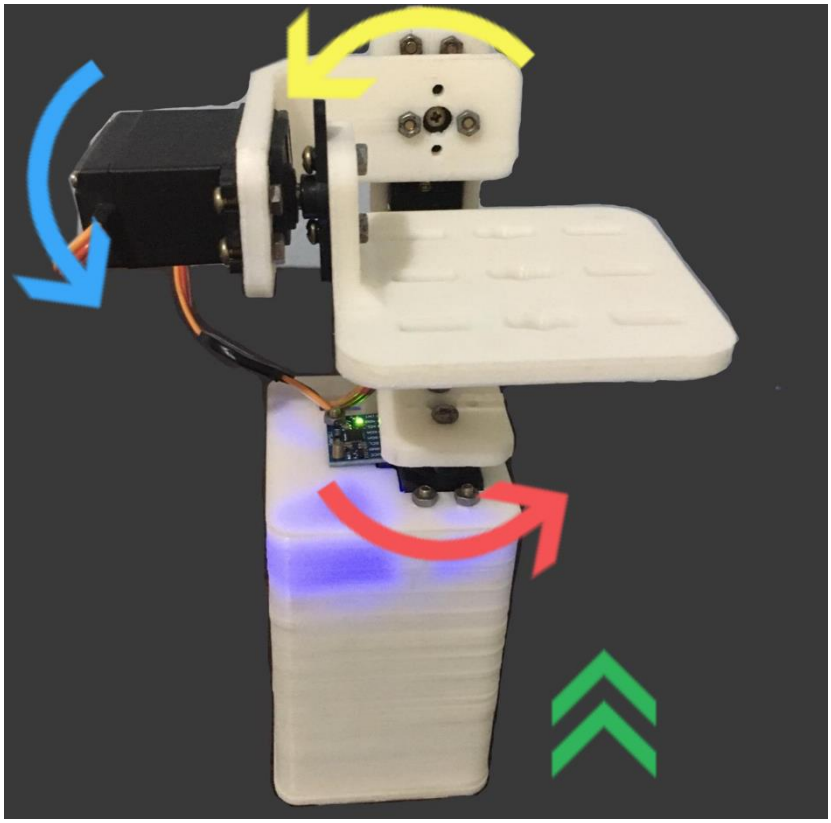
Com base nesses conceitos foram realizados simulações e testes práticos de cada etapa separadamente do sistema. Desde a comunicação entre o arduino e um microcomputador, criação do sinal de correção dos ângulos de inclinação dos servomotores, filtragem e amplificação de sinal, e feito o dimensionamento de cada componente eletrônico utilizado.

Foram feitas as correções adequadas nos erros e incertezas da medição por parte do módulo MPU-6050 em aferir os ângulos com precisão, além de corrigir a angulação adequadamente necessária dos servomotores. Não podemos desconsiderar a limitação dos componentes, como a dos servomotores, em que na sua atuação, seus movimentos de rotação são analógicos feitos por engrenagens, provocando tremor a cada grau de operação, gerando ruídos nas leituras.

A aquisição dos dados, bem como a filtragem e o tratamento dos mesmos se mostrou de suma importância na pratica. Os ruídos dos servomotores, bem como o sinal de alta frequência do PWM bem próximos as linhas de comunicação do I2C, tornando a leitura dificultosa. Este problema realmente acontece na pratica, por exemplo, no chão de fábrica, onde se encontram diversos tipos de atuadores que geram ruídos e que podem interferir nas diversas linhas de comunicação existentes.

Em suma, o *software* projetado é uma proposta viável para controle de estabilidade de plataformas inerciais, necessitando apenas de ajustes pontuais no sistema físico, uma vez que, o material utilizado para confeccionar o mesmo é frágil, não suportando objetos muito pesado na sua plataforma. A Figura 18 abaixo sintetiza o projeto e seu funcionamento.

Figura 18 - Funcionamento do manipulador.



Fonte: Do autor (2020).

A seta em vermelho ilustra o movimento que o eixo *yaw* realiza, a seta em amarelo ilustra o movimento do eixo *roll* em azul ilustra o movimento do eixo *pitch*. A seta em verde ilustra a posição que o manipulador deve ser iniciado em relação ao corpo do operador. O manipulador é mostrado na posição inicial de operação, para corrigir os servomotores, os mesmos não poderiam iniciar do 0, uma vez que, o envelope de trabalho deles são de 0-180º. O servomotor estando em 0º, no limite inferior de operação, não estabilizaria a plataforma.

A Figura 19 é uma correção feita no *software* da Figura 16 mostrada no capitulo 4, tópico 7.

Figura 19 - Correção do envelope de trabalho.

```
// Após 300 leituras
else {
  ypr[0] = ypr[0] - correct; // Defina o Yaw como 0 graus - subtraia o último valor aleatório do Yaw do valor atual para fazer o Yaw 0 graus
  // Mapeie os valores do sensor MPU6050 de -90 a 90 para valores aceitáveis para o servocontrole de 0 a 180 (envelope de trabalho)
  int servo0Value = map(ypr[0], -90, 90, 0, 180);
  int servo1Value = map(ypr[1], -90, 90, -7.15, 180);
  int servo2Value = map(ypr[2], -90, 90, 20, 180);
```

Fonte: Do autor (2020).

Nas linhas de código escrito: *int servo1Value = map(ypr[1], -90, 90, -7.15, 180)* e *int servo2Value = map(ypr[2], -90, 90, 20, 180),* podemos notar a correção no envelope de trabalho na variável nomeada no código de ypr[1] correspondente ao eixo *roll* do manipulador, onde varia de -7,15 a 180º. O mesmo acontece na variável nomeada no código de ypr[2] correspondente ao eixo *pitch* do manipulador variando de 20 a 180º. Para corrigir o ponto inicial da plataforma foram feitas essas alterações.

Mesmo corrigindo o momento inicial do manipulador por meio do *software*, não foi possível corrigir totalmente a posição inicial do mesmo. As engrenagens com seus dentes, obrigatoriamente nos impede de encaixar o elo entre as juntas de qualquer maneira. Os passos em que o servomotor gira também são empecilhos para cálculos exatos de operação. As dificuldades levantadas, foram superadas e implementadas maneiras de suprir necessidades físicas. O manipulador opera dentro do esperado, o que nos deixa aliviado. Validado o trabalho feito.

Como proposta para trabalhos futuros, sugere-se a criação de um *software* supervisório para mudança rápida das constantes do controlador bem como a visualização gráfica instantânea dos dados relevantes da plataforma como posição atual, sinais de controle e etc. Pode-se implementar mais um sensor MPU-6050 no manipulador, de forma a identificar exatamente qual a posição da plataforma em qualquer movimentação. Podendo implementar outros tipos de filtro para estabilidade, como o filtro de kalman, e fazendo uma comparação precisa entre o modelo matemático e o processo real. A troca dos servomotores por outros mais precisos, que apresentam uma resposta mais rápida, uma precisão de centímetros de grau, porém com baixo torque.

REFERÊNCIAS

ARDUINO. **Nano Family**. 2020. Disponível em: <arduino.cc>. Acesso em: março de 2020.

BATTISTEL, A. G. **Rastreamento Visual e Estabilização de Plataformas Inerciais Usando Apenas Duas Medidas de Sensores Giroscópios.** 2011. Dissertação de Mestrado – Universidade Federal do Rio de Janeiro, Rio de Janeiro.

BUENO[1], A. G., & ROMANO, R. A. **Filtro complementar aplicado a medida de inclinação de plataformas móveis.** 2014. Escola de Engenharia Mauá, São Paulo.

FARIA, G. A. **Galileu Galilei.** Disponível em: <http://www.fem.unicamp.br/~em313/paginas/person/galileu.htm>. Acesso em: novembro de 2019.

FREITAS, S. R. **Posicionadores Inerciais.** 1980. Dissertação de Pós-Graduação – Universidade Federal do Paraná, Curitiba.

FRIEDLAND, B. **Control System Design: An Introduction To State-Space Methods.** Col: Dover Books on Electrical Engineering Series. [S.l.]: Dover Publications, Incorporated. 1986. p. 35.

OLIVEIRA Jr, J. B. **Controle de uma Plataforma Inercial Estabilizada com Três Graus de Liberdade.** 2016. Dissertação de Mestrado – Universidade Federal de São Paulo, São Paulo.

LAGE, V. N. **Desenvolvimento de Um Controlador PID para Estabilização De Uma Plataforma Com Dois Graus de Liberdade.** 2016. Monografia – Universidade Federal de Ouro Preto, Ouro Preto.

ME102B, MECHATRONICS. **ME102 Lab 4: RC Servo.** 2015. Disponível em: <http://courses.me.berkeley.edu/ME102B/lab4.html> . Acesso em: dezembro de 2019.

MECÂNICA, HISTÓRIA. **A Mecânica de Newton.** Disponível em: <https://www.passeiweb.com/estudos/sala_de_aula/fisica/mecanica_hist_7_newton>. Acesso em: novembro de 2019.

MECHATRONICS, HOW. **Diy-Arduino gimbal self-stabilizing platform.** Disponível em: <https://howtomechatronics.com/projects/diy-arduino-gimbal-self-stabilizing-platform/> . Acesso em: novembro de 2019.

PROJECTS, G. M. **Gyroscopes and Accelerometers on a Chip.** 2013. Disponível em: <http://www.geekmomprojects.com/gyroscopes-and-accelerometers-on-a-chip/>. Acesso em: dezembro de 2019.

SENSE, INVEN. **MPU6000 and MPU6050 Register Map and Descriptions.** 2013. Revision 4.2.

TUCK, K. **Tilt Sensing Using a Three-Axis Accelerometer.** 2007. Freescale semiconductor application note AN3461.

ANEXO A – Código do arduino

```
// I2Cdev e MPU6050 devem ser instalados como bibliotecas, ou os arquivos .cpp / .h
// Para ambas as classes estarem no caminho de inclusão do seu projeto
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // não é necessário se o arquivo de inclusão MotionApps for usado

// A biblioteca Arduino Wire é necessária se a implementação I2Cdev I2CDEV_ARDUINO_WIRE
// é usado no I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif
#include <Servo.h>
// o endereço I2C padrão da classe é 0x68
// endereços I2C específicos podem ser passados como parâmetro aqui
// AD0 low = 0x68 (padrão para breakout do SparkFun e placa de avaliação do InvenSense)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- usar para AD0 high

// Definindo os três servomotores
Servo servo0;
Servo servo1;
Servo servo2;
float correct;
int j = 0;

#define OUTPUT_READABLE_YAWPITCHROLL

#define INTERRUPT_PIN 2  // definindo o pino 2
bool blinkState = false;

// Controle MPU control/variável de estado
bool dmpReady = false;  // definindo como verdadeiro se DMP init for bem-sucedido
uint8_t mpuIntStatus;   // detém o byte de status de interrupção real do MPU
uint8_t devStatus;      // retorna o status após cada operação do dispositivo (0 = success, !0 = error)
uint16_t packetSize;    // tamanho esperado do pacote (o padrão é 42 bytes)
uint16_t fifoCount;     // contagem de todos os bytes atualmente no FIFO
uint8_t fifoBuffer[64]; // Buffer de armazenamento FIFO

// Variáveis de orientação/movimento
Quaternion q;           // [w, x, y, z]         contêiner  quaternion
VectorInt16 aa;         // [x, y, z]            acelerar as medições do sensor
```

```
VectorInt16 aaReal;     // [x, y, z]           medições de sensor de aceleração sem gravidade
VectorInt16 aaWorld;    // [x, y, z]           medições de sensor de aceleração de quadro
mundial
VectorFloat gravity;    // [x, y, z]           vetor de gravidade
float euler[3];         // [psi, theta, phi]    contêiner  de ângulo de Euler
float ypr[3];           // [yaw, pitch, roll]   contêiner de yaw/pitch/roll e vetor de gravidade

// estrutura de pacotes para a demonstração do bule do InvenSense
uint8_t teapotPacket[14] = { '$', 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0x00, 0x00, '\r', '\n' };



//
============================================================
==
// ===          ROTINA DE DETECÇÃO DE INTERRUPÇÃO            ===
//
============================================================
==

volatile bool mpuInterrupt = false;     // indica se o pino de interrupção da MPU está alto
void dmpDataReady() {
  mpuInterrupt = true;
}

//
============================================================
==
// ===                CONFIGURAÇÃO INICIAL                ===
//
============================================================
==

void setup() {
  // ingressar no barramento I2C (a biblioteca I2Cdev não faz isso automaticamente)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
  Wire.begin();
  Wire.setClock(400000); // Relógio I2C de 400kHz. Comente esta linha se a compilação
der problema
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
  Fastwire::setup(400, true);
#endif

  // inicializar a comunicação serial
  Serial.begin(38400);
  while (!Serial);

  // inicializar dispositivo
  //Serial.println(F("Inicialização de dispositivos I2C ... "));
  mpu.initialize();
```

```
  pinMode(INTERRUPT_PIN, INPUT);
  devStatus = mpu.dmpInitialize();
  //compensações de giroscópio
  mpu.setXGyroOffset(17);
  mpu.setYGyroOffset(-69);
  mpu.setZGyroOffset(27);
  mpu.setZAccelOffset(1551); // Padrão de fábrica 1688 para o meu chip de teste

  // verificação se funcionou (retorna 0)
  if (devStatus == 0) {
    // ligue o DMP
    // Serial.println (F ("Ativando DMP ..."));
    mpu.setDMPEnabled(true);

    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

    // Definição do sinalizador DMP Ready para que a função loop () principal saiba que
não há problema em usá-lo
    //Serial.println(F("DMP pronto! Aguardando a primeira interrupção ... "));
    dmpReady = true;

    // obtém o tamanho esperado do pacote DMP para comparação posterior
    packetSize = mpu.dmpGetFIFOPacketSize();
  } else {
    // ERRO!
     // 1 = falha no carregamento inicial da memória
     // 2 = atualizações de configuração do DMP falharam
     // (se estiver quebrando, normalmente o código será 1)
     // Serial.print (F ("Falha na inicialização do DMP (código")));
     //Serial.print(devStatus);
     // Serial.println (F (")"));
  }

  // Definição dos pinos aos quais os 3 servomotores estão conectados
  servo0.attach(10);
  servo1.attach(9);
  servo2.attach(8);
}
//
==========================================================
==
// ===              LOOP PRINCIPAL DO PROGRAMA              ===
//
==========================================================
==

void loop() {

  if (!dmpReady) return;
```

```
  // Aguardando a interrupção da MPU ou pacotes extras disponíveis
  while (!mpuInterrupt && fifoCount < packetSize) {
    if (mpuInterrupt && fifoCount < packetSize) {
      // Tentar sair do loop infinito
      fifoCount = mpu.getFIFOCount();
    }
  }

  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();

  // Obter a contagem FIFO atual
  fifoCount = mpu.getFIFOCount();

  // Verifique se há excesso (por precaução)
  if ((mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) || fifoCount
>= 1024) {
    // Redefinir
    mpu.resetFIFO();
    fifoCount = mpu.getFIFOCount();
    Serial.println(F("Excesso de FIFO!"));

    // caso contrário, verifique se há interrupção pronta dos dados do DMP (isso deve
 ocorrer com freqüência)
  } else if (mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)) {
    // Aguardar o comprimento correto dos dados disponíveis
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // Leia um pacote do FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // Rastreamento sa contagem FIFO aqui, caso haja >1 pacote disponível
    // Permite ler imediatamente mais sem esperar uma interrupção
    fifoCount -= packetSize;

    // Obter valores Yaw, Pitch e Roll
#ifdef OUTPUT_READABLE_YAWPITCHROLL
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

    // Transformando valores de Yaw, Pitch, Roll de radianos para graus
    ypr[0] = ypr[0] * 180 / M_PI;
    ypr[1] = ypr[1] * 180 / M_PI;
    ypr[2] = ypr[2] * 180 / M_PI;

    // Pula 300 leituras (processo de auto-calibração)
    if (j <= 300) {
```

```
      correct = ypr[0]; // Yaw começa com um valor aleatório, então capturamos o último valor após 300 leituras
      j++;
    }
    // Após 300 leituras
    else {
      ypr[0] = ypr[0] - correct; // Defina o Yaw como 0 graus - subtraia o último valor aleatório do Yaw do valor atual para fazer o Yaw 0 graus
      // Mapeie os valores do sensor MPU6050 de -90 a 90 para valores aceitáveis para o servocontrole de 0 a 180 (envelope de trabalho)
      int servo0Value = map(ypr[0], -90, 90, 0, 180);
      int servo1Value = map(ypr[1], -90, 90, -7.15, 180);
      int servo2Value = map(ypr[2], -90, 90, 20, 180);

      // Controle os servos de acordo com a orientação MPU6050
      servo0.write(servo0Value);
      servo1.write(servo1Value);
      servo2.write(servo2Value);
    }
#endif
  }
}
```

ANEXO B – Código do modulo MPU-6050

```
#include <Wire.h>

const int MPU = 0x68; // Endereço I2C do MPU6050
float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;
float roll, pitch, yaw;
float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
float elapsedTime, currentTime, previousTime;
int c = 0;

void setup() {
  Serial.begin(19200);
  Wire.begin();                 // Inicializar comunicação
  Wire.beginTransmission(MPU);          // Iniciar a comunicação com MPU6050 // MPU=0x68
  Wire.write(0x6B);               // Fale com o registro 6B
  Wire.write(0x00);              // Make reset - coloque um 0 no registro 6B
  Wire.endTransmission(true);       // Encerrar a transmissão
  /*
  // Configurar a sensibilidade do acelerômetro - faixa de escala completa (padrão +/- 2g)
  // Wire.beginTransmission (MPU);
  // Wire.write (0x1C); // Fale com o registro ACCEL_CONFIG (1C hex)
  // Wire.write (0x10); // Defina os bits de registro como 00010000 (+/- 8g de escala completa)
  // Wire.endTransmission (true);
  // Configurar a sensibilidade do giroscópio - faixa de escala completa (padrão +/- 250graus/s)
  // Wire.beginTransmission (MPU);
  // Wire.write (0x1B); // Fale com o registro GYRO_CONFIG (1B hex)
  // Wire.write (0x10); // Defina os bits de registro como 00010000 (1000graus/s em escala completa)
  // Wire.endTransmission (true);
  // delay (20);
  */

  // Chame essa função se precisar obter os valores de erro IMU para o módulo
  calculate_IMU_error();
  delay(20);

}

void loop() {
  // === Ler dados do acelerômetro === //
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // Comece com o registro 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
```

```
  Wire.requestFrom(MPU, 6, true); // Ler 6 registros no total, cada valor de eixo é armazenado em 2 registros
  // Para um intervalo de + -2g, precisamos dividir os valores brutos por 16384, de acordo com o datasheet
  AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // valor do eixo X
  AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // valor do eixo Y
  AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // valor do eixo Z

  // Cálculo de rotação e inclinação a partir dos dados do acelerômetro
  accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) - 0.58; // AccErrorX ~(0.58) Consulte a função customizada calculate_IMU_error()
  accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) + 1.58; // AccErrorY ~(-1.58)

  // === Ler dados do giroscópio === //
  previousTime = currentTime;        // O horário anterior é armazanado antes do horário real lido
  currentTime = millis();            // Hora atual, hora atual, leitura
  elapsedTime = (currentTime - previousTime) / 1000; // Divide por 1000 para obter segundos
  Wire.beginTransmission(MPU);
  Wire.write(0x43); // Primeiro endereço de registro de dados giroscópicos 0x43
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Ler 4 registros no total, cada valor de eixo é armazenado em 2 registro
  GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // Para uma faixa de 250graus/s, primeiro dividimos o valor bruto por 131,0, de acordo com o datasheet
  GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
  GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;

  // Corrija as saídas com os valores de erro calculados
  GyroX = GyroX + 0.56; // GyroErrorX ~(-0.56)
  GyroY = GyroY - 2; // GyroErrorY ~(2)
  GyroZ = GyroZ + 0.79; // GyroErrorZ ~ (-0.8)

  // Atualmente, os valores brutos estão em graus por segundo, graus/s, portanto, é preciso multiplicar por segundos para obter o ângulo em graus
  gyroAngleX = gyroAngleX + GyroX * elapsedTime; // graus/s * s = graus
  gyroAngleY = gyroAngleY + GyroY * elapsedTime;
  yaw =  yaw + GyroZ * elapsedTime;

  // Filtro complementar - combine valores de acelerômetro e ângulo do giroscópio
  roll = 0.96 * gyroAngleX + 0.04 * accAngleX;
  pitch = 0.96 * gyroAngleY + 0.04 * accAngleY;

  // Imprima os valores no monitor serial
  Serial.print(roll);
  Serial.print("/");
  Serial.print(pitch);
  Serial.print("/");
```

```
  Serial.println(yaw);
}


void calculate_IMU_error() {
  // Pode ser chamado essa função na seção de configuração para calcular o acelerômetro e o erro de dados do giroscópio. A partir daqui, obteremos os valores de erro usados nas equações acima, impressos no Serial Monitor.
  // Devemos colocar a IMU nivelada para obter os valores adequados, para que possamos então os valores corretos
  // Leia os valores do acelerômetro 200 vezes
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x3B);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    AccX = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    AccY = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    // Soma de todas as leituras
    AccErrorX = AccErrorX + ((atan((AccY) / sqrt(pow((AccX), 2) + pow((AccZ), 2))) * 180 / PI));
    AccErrorY = AccErrorY + ((atan(-1 * (AccX) / sqrt(pow((AccY), 2) + pow((AccZ), 2))) * 180 / PI));
    c++;
  }
  // Dividindo a soma por 200 para obter o valor do erro
  AccErrorX = AccErrorX / 200;
  AccErrorY = AccErrorY / 200;
  c = 0;
  // Leia os valores do giroscópio 200 vezes
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x43);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    GyroX = Wire.read() << 8 | Wire.read();
    GyroY = Wire.read() << 8 | Wire.read();
    GyroZ = Wire.read() << 8 | Wire.read();
    // Soma de todas as leituras
    GyroErrorX = GyroErrorX + (GyroX / 131.0);
    GyroErrorY = GyroErrorY + (GyroY / 131.0);
    GyroErrorZ = GyroErrorZ + (GyroZ / 131.0);
    c++;
  }
  //Dividindo a soma por 200 para obter o valor do erro
  GyroErrorX = GyroErrorX / 200;
  GyroErrorY = GyroErrorY / 200;
  GyroErrorZ = GyroErrorZ / 200;
  // Imprime os valores de erro no Serial Monitor
```

```
  Serial.print("AccErrorX: ");
  Serial.println(AccErrorX);
  Serial.print("AccErrorY: ");
  Serial.println(AccErrorY);
  Serial.print("GyroErrorX: ");
  Serial.println(GyroErrorX);
  Serial.print("GyroErrorY: ");
  Serial.println(GyroErrorY);
  Serial.print("GyroErrorZ: ");
  Serial.println(GyroErrorZ);
}
```