**UNIVERSIDADE FEDERAL DE OURO PRETO – UFOP**
**ESCOLA DE MINAS – EM**
**COLEGIADO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - CECAU**

# DESENVOLVIMENTO DE UM CONTROLADOR PID PARA ESTABILIZAÇÃO DE UMA PLATAFORMA COM DOIS GRAUS DE LIBERDADE

**VINÍCIUS NUNES LAGE**

**Ouro Preto, 2016**

**VINÍCIUS NUNES LAGE**

# DESENVOLVIMENTO DE UM CONTROLADOR PID PARA ESTABILIZAÇÃO DE UMA PLATAFORMA COM DOIS GRAUS DE LIBERDADE

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Alan Kardek Rêgo Segundo

Ouro Preto
Escola de Minas – UFOP
01/2016

Monografia defendida e aprovada, em 11 de março de 2016, pela comissão avaliadora constituída pelos professores:

Prof. Dr. Alan Kardek Rêgo Segundo - Orientador

Prof. Dr. Paulo Marcos de Barros Monteiro – Professor Convidado

Prof. Dr. Paulo Raimundo Pinto – Professor Convidado

L174d Lage, Vinícius Nunes.
Desenvolvimento de um controlador PID para estabilização de uma plataforma com dois graus de liberdade [manuscrito] / Vinícius Nunes Lage. – 2016.
94f. : il., color., graf., tab.

Orientador: Prof. Dr. Alan Kardek Rêgo Segundo.

Monografia (Graduação) – Universidade Federal de Ouro Preto. Escola de Minas. Colegiado do Curso de Engenharia de Controle e Automação e Técnicas fundamentais.
Área de concentração: Engenharia de Controle e Automação.

1.Automação industrial. 2. Estabilidade estrutural. 3. Controladores PID. 4. Acelerômetros. I. Universidade Federal de Ouro Preto. II. Título.

CDU: 681.5

Fonte de catalogação: bibem@sisbin.ufop.br

## AGRADECIMENTOS

A busca pela sabedoria é eterna, e este trabalho só foi possível graças a algumas pessoas especiais que, de formas diferentes, contribuíram para tal.

Primeiramente agradeço à minha família que é o meu suporte e meu motivo de vida e à minha namorada Cris, pela paciência e companhia alegre de todos os dias.

Em segundo lugar ao Rafael Martins (*germin*), amigo gênio que muito me ensinou com paciência e muitas horas de conversas, e de onde partiu a inspiração para este trabalho.

Ao Alan Kardek, meu orientador e mentor, pela paciência e atenção entregue sempre.

Aos professores e educadores Paulo Monteiro, Rispoli, Cocota, Joca, Sávio e Agnaldo do DECAT, aos Luiz Merschmann e Marcone Jamilson do DECOM, aos Ricardo Tavares e Fábio do DEMAT, ao Carlos Felipe do DEFIS, aos Paulinho, Lula, José Eduardo, Sílvia, Cristiano e Ronaldinho do CEFET, esses, que dedicam suas vidas a ensinar com dignidade e principalmente prazer, não só os ensinamentos teóricos, mas ensinamentos de vida.

Agradeço imensamente à toda turma 09.1, eternos amigos e futuros desse país e à Adriana da Seção de Ensino da Escola de Minas pela atenção e dedicação de forma alegre sempre.

Agradeço também à UFOP e à Fundação Gorceix por tornarem realidade os sonhos de milhares de estudantes que se tornam pessoas que ajudam a combater a pobreza e a diferença social em nosso país.

“Um homem sem objetivo não sabe aonde vai”.

## RESUMO

Neste trabalho realizou-se 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 modelo matemático do mecanismo foi identificado 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. Estudou-se métodos de aquisição de dados e a implementação dos filtros Complementar, Kalman e o modo DMP (Digital Motion Processor) do sensor, a fim de melhorar o resultado das aferições. Em seguida, foi desenvolvido um controlador do tipo PID com o objetivo de manter a plataforma na posição horizontal, independentemente de movimentos externos. Todos os experimentos foram auxiliados pelos programas Matlab, Excel e Minitab. O Arduino foi a plataforma escolhida 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 o resultado mais satisfatório.



**Palavras chave:** controle de estabilidade, plataforma equilibrada, controlador PID, acelerômetro, giroscópio, sensor MPU-6050, filtro Complementar, filtro de Kalman, Ziegler-Nichols, sintonia PID.

## ABSTRACT

This work was carried out mounting a servo-controlled platform with two degrees of freedom, and a control system in order to maintain balanced on it, objects of small weight. The mathematical model of the mechanism has been identified by experiments using the MPU-6050 sensor, and through its accelerometers and gyro sensors, enabled the calculation of the angular position X and platform Y. Data acquisition methods have been studied well as implementation of complementary filters, Kalman and the DMP (Digital Motion Processor) mode of sensor to improve the results of the measurements. Then, a PID controller was developed, with the objective of maintaining the platform in a horizontal position regardless of external movement. All experiments were aided by Matlab, Excel and Minitab software. The Arduino has been the platform of choice for both the acquisition of data and control. Different methods of tunning the controller will be study, as Ziegler-Nichols, *pidtool* of Matlab and method of successive approximations or trial and error, which provided the most satisfactory result.



**Key words:** stability control, balanced platform, PID controller, accelerometer, gyroscope, MPU6050 sensor, Complementary filter, Kalman filter, Ziegler-Nichols, PID tunning.

## LISTA DE FIGURAS

# LISTA DE TABELAS

**SUMÁRIO**

# 1. INTRODUÇÃO

Sistemas de controle de posição e estabilidade são facilmente encontrados. Existem desde veículos modernos, que utilizam suspensões inteligentes para manter os passageiros sempre em sua posição horizontal, como também aeronaves não tripuladas (*drones*), 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. Também é possível encontrar esses sistemas em mesas de sinucas instaladas em navios, para mantém as bolas sempre estáticas, independentemente do movimento da embarcação. O Google, recentemente, adquiriu a startup Levante Lab, que criou o Liftware, uma colher que usa uma série de algoritmos e conta com uma tecnologia que estabiliza o acessório quando a mão da pessoa está tremendo, muito útil para pessoas com Parkinson. Segundo o Google, o uso do dispositivo reduziu em 76% a queda de alimentos em testes próprios (EXAME, 2015).

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 (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.

## 1.1. Objetivos Gerais

O objetivo geral deste trabalho é desenvolver uma plataforma, de baixo custo, com dois graus de liberdade, juntamente com um controlador PID, e sua respectiva sintonia, para manter esta plataforma estabilizada na posição horizontal, com o mínimo de vibrações possíveis.

## 1.2. Objetivos Específicos e Etapas do Desenvolvimento

- Montagem mecânica da plataforma com dois graus de liberdade;
- Montagem dos servos e dos sensores na plataforma;
- Aquisição e tratamento dos dados aferidos pelos sensores;
- Modelagem matemática através de aferições e testes com a plataforma;
- Implementação do controlador PID;
- Sintonia do controlador;
- Comparação e discussão entre os resultados reais com os teóricos;

## 1.3. Metodologia Proposta

Com o intuito de tornar possível a reprodução deste trabalho, foi adotada uma metodologia simples, objetiva e clara de discussão de todas as etapas desenvolvidas.

A montagem mecânica foi feita com materiais de baixo custo e facilmente encontrados no mercado nacional, sendo realizada de forma rápida e com baixa complexidade.

Para a medição dos ângulos instantâneos da plataforma, foi utilizado o sensor MPU-6050. A comunicação com o sensor utiliza o protocolo de comunicação I2C, por onde será feita a aquisição dos valores dos acelerômetros e dos giroscópios. Após esse processo é possível fazer o cálculo do ângulo. Para melhorar a precisão das medições, foram utilizados dois filtros para realizarem rejeição de distúrbios e ruídos na aquisição de dados: (i) o filtro Complementar, que se trata de um filtro mais simples, porém com ótimos resultados; e (ii) o filtro de Kalman, inteligente e muito robusto, largamente utilizado em sistemas de aquisição de dados na área da computação e robótica.

A posição angular da plataforma é controlada através de dois servomotores, um para cada grau de liberdade. Esses são acionados através de sinais PWM, que são gerados pelo Arduino Nano.

Para a modelagem matemática da plataforma, bem como o projeto do controlador, foi utilizado o software Matlab. A aquisição dos dados do sensor para a modelagem foi feita por meio de uma comunicação USB entre o Arduino e um microcomputador. Todos os comandos utilizados no software Matlab são descritos em cada etapa.

## 1.4. Estrutura do Trabalho

Este trabalho foi dividido em cinco partes distintas, apresentadas nas subseções apresentadas a seguir:

### 1.4.1. Base Teórica

Foram discutidas todas as informações teóricas a respeito dos dispositivos utilizados, bem como seus respectivos funcionamentos.

### 1.4.2. Montagem Mecânica

Discute como foi feita a montagem mecânica da plataforma e os materiais utilizados.

### 1.4.3. Montagem Eletrônica

Discute informações sobre os dispositivos eletrônicos utilizados bem como o esquema elétrico completo utilizado para o perfeito funcionamento da plataforma.

### 1.4.4. Aquisição de Dados, Implementação e Comparação entre os Filtros Complementar, Kalman e o modo DMP do sensor MPU-6050

Após o Arduino se comunicar corretamente com o sensor MPU-6050, com os servomotores e com o microcomputador, foram feitas algumas aferições. Foi implementado o filtro Complementar e de Kalman. Logo após foi implementado o modo DMP (*Digital Motion Proccess*) do MPU-6050 e feito uma comparação entre os três métodos e estabelecido qual o melhor filtro para o processo de aquisição precisa do ângulo por meio do sensor. Todos os processos foram auxiliados pelos softwares Matlab, Monitor Serial do Arduino e Excel.

### 1.4.5. Modelagem Matemática da Plataforma

Após executados os testes, escolhido o melhor filtro, e com auxílio do software Matlab, foi feita a modelagem matemática aproximada do sistema.

### 1.4.6. Projeto do Controlador PID e Sintonia

Com a plataforma funcionando corretamente e o modelo matemático, foi feito o projeto do controlador PID. O controlador foi testado teoricamente no Matlab, discretizado e implementado no Arduino. A sintonia inicialmente foi testada através do método Ziegler-Nichols. Porém este método não se aplicou ao processo da plataforma, então foram utilizados o Matlab, através da sua ferramenta de *autotunning* e também o método das aproximações sucessivas ou tentativa e erro, que forneceram em conjunto o melhor resultado.

### 1.4.7. Discussão dos Resultados e Conclusões

Depois de implementado o controle, os resultados foram comparado com os resultados teóricos obtidos anteriormente e discutidos.

# 2. BASE TEÓRICA

Nesta sessão serão abordados os temas necessários para o entendimento do funcionamento dos dispositivos utilizados, bem como as técnicas utilizadas para a aquisição de dados e para o acionamento dos atuadores. Alguns fatores foram essenciais para o funcionamento correto do sistema e outros utilizados como forma de melhoria dos resultados.

## 2.1. Arduino

O Arduino é uma plataforma de prototipagem de código aberto baseado em hardware e software de fácil utilização (ARDUINO, 2015). Os Arduinos são placas com entradas e saídas analógicas e digitais, interface USB para comunicação com um microcomputador, e de fácil implementação de códigos com uma linguagem própria, muito parecida com a linguagem C. Os Arduinos utilizam os microcontroladores ATMEGA, e as placas já possuem todos os componentes necessários para o seu funcionamento, como resistores, capacitores, cristal e circuito regulador de tensão para alimentação. O Arduino possui 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 Arduino possui diversos tipos e modelos diferentes, como o Arduino Mega, Arduino Micro, Arduino Nano, entre outros. Os modelos se diferem quanto ao número de portas (entradas e saídas), número de PWM’s disponíveis, memória, velocidade de clock, entre outras características.

Para este trabalho foi escolhido o Arduino Nano, devido ao seu tamanho reduzido e por atender a todos os seguintes requisitos: 2 saídas PWM para os servos, comunicação I2C para o sensor MPU-6050 e comunicação USB para comunicar com o microcomputador. A Figura 2.1 mostra um Arduino Nano.

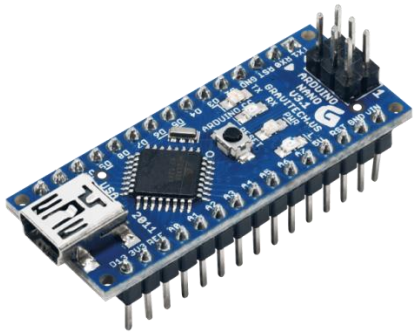


Figura 2-1 - Arduino Nano v3.0
Fonte: Ral Technology, 2016

O Arduino trabalha com alimentação de 3,3 ou 5 volts. Ele pode ser alimentado por uma fonte externa como baterias ou diretamente pela USB de um microcomputador. Para os testes iniciais, será utilizado um computador para alimentá-lo, porém após todas as implementações, serão utilizados baterias externas, para que a plataforma seja portátil.

## 2.2. Comunicação USB

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.

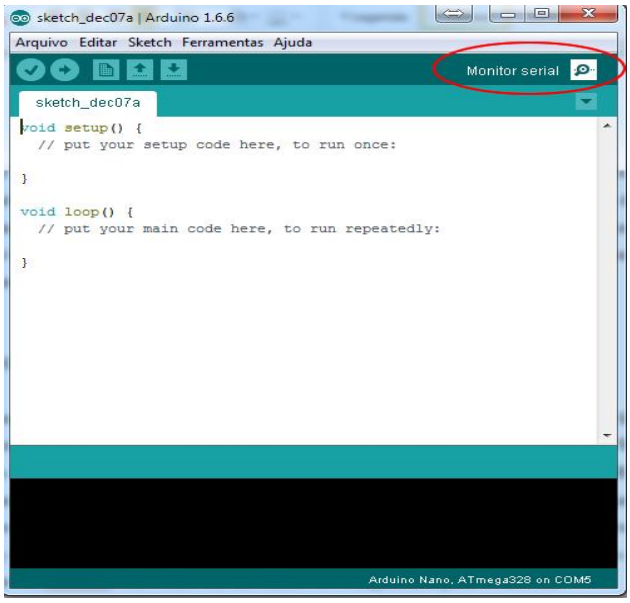



Figura 2-2 - Como Iniciar o Monitor Serial do Arduino

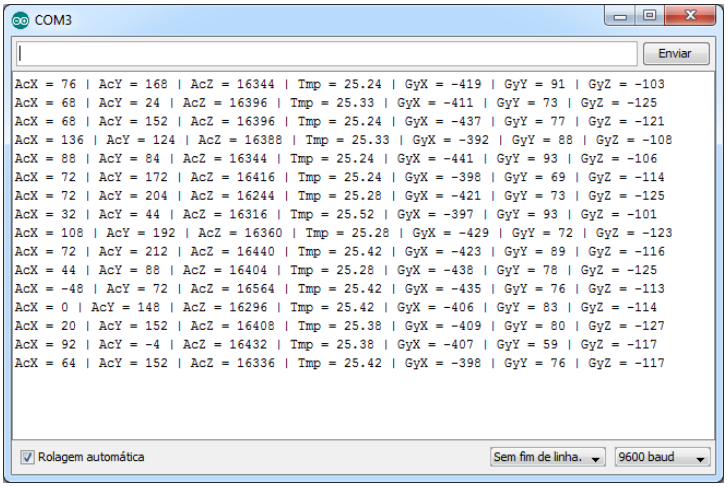



Figura 2-3 - Monitor Serial do Arduino

Para a comunicação, o Arduino já possui nativamente funções para envio e recepção dos dados. A velocidade de comunicação pode ser configurada entre 9.600 bits por segundo até 115.200 bits por segundo, que são as velocidades utilizadas no protocolo RS-232.

Toda a programação é discutida posteriormente neste trabalho.

## 2.3. Servomotores

Servomotor é um dispositivo eletromecânico muito aplicado na robótica e na indústria. São dispositivos de malha fechada, ou seja, recebem um sinal de referência e, de acordo com a posição atual, é calculado o erro. De acordo com este erro, o controlador atua no motor, enviando o eixo do servomotor para a posição desejada, com velocidade monitorada. Neste trabalho foram utilizados dois servomotores realimentados através de um potenciômetro, que é o mais comum para este tipo de dispositivo não industrial.

Os servomotores normalmente giram apenas 180 graus, e não 360 como é o caso de motores, porém, são precisos quanto a sua posição.

O potenciômetro do servomotor está diretamente ligado ao eixo, realimentando o circuito de controle interno. Através da variação da resistência do potenciômetro, o circuito de controle identifica a posição angular atual do servo. A qualidade desse potenciômetro influencia diretamente na precisão do controle, na estabilidade e na vida útil do servo.

O sistema atuador do servo utilizado é constituído por um motor elétrico de corrente contínua e uma caixa de engrenagens de plástico, que faz a redução da velocidade do motor, porém aumenta consideravelmente o torque. Neste trabalho foram utilizados os servomotores como atuadores justamente por já possuírem essa caixa de redução. Dessa forma podem trabalhar na plataforma objetos com pesos maiores que se fossem utilizados motores diretamente como atuadores.

O servomotor utilizado possui torque de 3kg*cm, ou seja, ele tem torque suficiente para mover um peso de 3kg diretamente em seu eixo. Porém, para cada centímetro de distância afastando do eixo, o torque diminui proporcionalmente. Por exemplo, a 10cm do eixo, o toque máximo suportado por este servo seria de 300g.

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.

Os servos possuem três fios de interface, sendo dois para a alimentação (*gnd* e *vcc*) e um para o sinal de controle. 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*), que será discutido posteriormente na seção 2.2.1.

A Figura 2.4 mostra uma imagem de um servomotor, suas engrenagens e seu sistema de controle interno.

Figura 2-4 - Visão da Caixa de Redução e do Circuito de Controle de um Servomotor
Fonte: ENGLEANDROALVES, 2015

### 2.3.1. *Pulse Witdh Modulation* (PWM) ou Modulação por Largura de Pulso

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 entrega à carga (FAMBRINI, [S.d.]). 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.

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.

A Figura 2.5 mostra alguns exemplos de posicionamento de um servomotor de acordo com o sinal de PWM enviado.

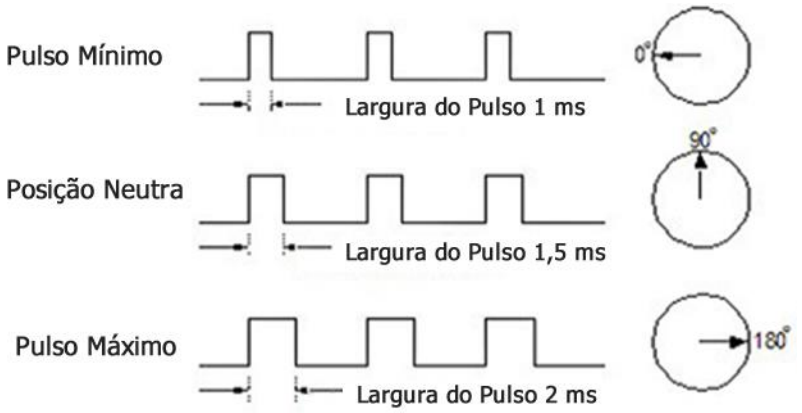



Figura 2-5 - Visão de um sinal de PWM e a posição respectiva de um servomotor
Fonte: Mechatronics ME102B Fall, 2015

Para o posicionamento dos servomotores foi utilizado a biblioteca “Servo.h” do Arduino, que já gera o sinal de PWM corretamente de acordo com o ângulo desejado.

## 2.4. Sensor MPU-6050

O MPU-6050 é um sensor integrado fabricado pela empresa IvenSense Inc., que reúne 3 sensores giroscópio (1 para cada eixo X, Y e Z), 3 sensores acelerômetros (também um para cada eixo) e um Processador Digital de Movimentos (*Digital Motion Processor* ou DMP) em um único chip (DATASHEET MPU-6050, 2015). Ele utiliza a comunicação I2C para a aquisição dos dados, possui 3 conversores analógico/digital de 16 bits internamente para a discretização dos valores dos acelerômetros e dos giroscópios internos. O sensor possui precisão para movimentos rápidos ou lentos. O giroscópio possui escalas de ±250, ±500, ±1000 ou ±2000 graus por segundo (muitas vezes chamados de dps - *degrees per second*). Já os acelerômetros, possuem intervalos de ±2g, ±4g, ±8g ou ±16g, sendo 1g a aceleração da gravidade, que corresponde a aproximadamente 9,81 $m/s^2$.

O sensor MPU-6050 é largamente utilizado em aplicações portáteis (*tablets*, *smartphones*, entre outros) e apresenta baixo consumo de energia.

A figura 2.6 mostra o sensor MPU-6050 instalado em uma placa pronta para ser utilizada com um Arduino.


Figura 2-6 - Sensor MPU-6050

Fonte: Robu.in, 2015

### 2.4.1. Acelerômetro

Acelerômetro é um equipamento que mede a aceleração própria de um corpo (TEXAS INSTRUMENTS, 2015). Eles são utilizados principalmente em sistemas de posicionamento, sensores de inclinação, bem como sensores de vibração. Um exemplo prático e bastante conhecido de acelerômetros são as telas dos aparelhos celulares: elas se ajustam de acordo com o ângulo que fazem em relação à aceleração da gravidade.

Um exemplo do princípio de funcionamento dos acelerômetros é, de acordo com a segunda lei de Newton, colocar um copo com água até a metade sobre uma mesa e empurrá-lo, horizontalmente de forma acelerada. Nota-se que a água se desloca em relação ao copo. O acelerômetro mede essa força da aceleração em comparação com a aceleração "g" da gravidade. Portanto, se o acelerômetro fornecer 1g em um eixo, significa que existe naquela direção e sentido, uma aceleração equivalente a 9,81 $m/s^2$.

O MPU-6050 possui 3 acelerômetros, um para cada eixo X, Y e Z. Dessa forma, é possível adquirir a aceleração em cada um dos eixos e o posicionamento angular do sensor. Na maioria dos casos, a aceleração é tratada como um vetor que pode ser usado para detectar a orientação do dispositivo, mais precisamente *pitch* (rotação em graus no eixo Y), *roll* (rotação em graus no eixo X) e *yaw* (rotação em graus no eixo Z) (PAULA, F. O, 2015). Porém, qualquer movimento acelerado em qualquer direção ou sentido, afeta esses valores. Daí a utilização dos giroscópios em conjunto com os acelerômetros para uma aferição precisa dos ângulos independentemente do sensor estar em movimento acelerado ou não.

Como a plataforma irá fazer movimentos acelerados para se posicionar corretamente, esses movimentos afetam diretamente nos valores dos acelerômetros, podendo desestabilizar complemente o sistema.

Diversos tipos de acelerômetros são encontrados no mercado como os piezoelétricos, piezoresistivos, capacitivos, entre outros. Este trabalho não tem a intenção de aprofundar nos detalhes de cada sensor, mas sim compreender o seu princípio de funcionamento e aplicá-lo.

### 2.4.2. Cálculo do Ângulo Utilizando Apenas os Acelerômetros

Quando o acelerômetro é colocado na sua posição horizontal, em uma mesa, por exemplo, ele irá medir uma força de 1g em um dos seus eixos, provavelmente no eixo Z. Porém, quando o sensor está inclinado, a força g é distribuída em dois eixos. Logo, é possível medir os ângulos através de uma trigonometria simples.

A Figura 2.7 mostra um acelerômetro com uma inclinação θ no eixo X e Z.

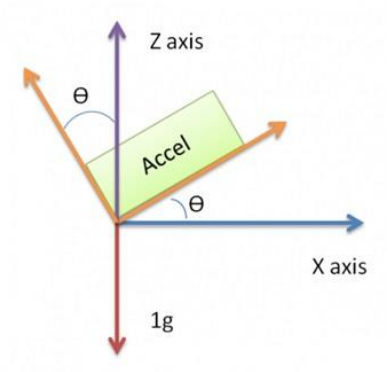



Figura 2-7 – Inclinação no plano XZ

Fonte: PAULA, F. O, 2015

Pode-se calcular o ângulo θ através da Equação 2.1 e 2.2:

$$tan\theta = \frac{x}{z} \tag{2.1}$$

$$\theta = \arctan({}^{x}/_{Z}) \tag{2.2}$$

sendo “x” o valor entregue pelo acelerômetro no eixo X e “z” o valor entregue neste eixo, ambos na grandeza proporcional à aceleração da gravidade “g”.

Considerando uma rotação ou inclinação nos 3 eixos, que é o que acontece na realidade (raramente um eixo fica alinhado perfeitamente), utiliza-se então das relações trigonométricas de acordo com as equações 2.3, 2.4 e 2.5 a seguir, levando em conta a força “g” medida em cada eixo por cada acelerômetro (Ax, Ay e Az).

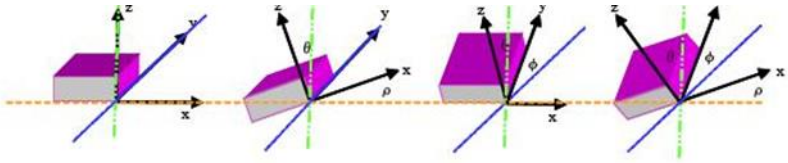


Figura 2-8 - Ângulos através dos acelerômetros para 3 eixos

Fonte: PAULA, F. O, 2015

$$\rho = arctan\left(\frac{Ax}{\sqrt{Ay^2 + Az^2}}\right) \tag{2.3}$$

$$\phi = arctan\left(\frac{Ay}{\sqrt{Ax^2 + Az^2}}\right) \tag{2.4}$$

$$\theta = arctan\left(\frac{\sqrt{Ax^2 + Ay^2}}{Az}\right) \tag{2.5}$$

Dessa forma, chega-se aos 3 ângulos desejados. Para a plataforma serão utilizados apenas dois eixos, o X e o Y ou o *roll* e o *pitch* (**ρ** e ϕ).

O MPU-6050 nos fornece valores adimensionais, chamados *raw values*, que devem ser convertidos para m/s$^2$, mais precisamente, para múltiplos de forças “g” (9,81m/s$^2$). O *datasheet* informa que 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.

Lembrando que este valor não é real quando se tem um movimento acelerado, pois o movimento gera novas forças além da gravitacional nos acelerômetros. Utilizam-se como auxílio os giroscópios.

### 2.4.3. Giroscópio

Os giroscópios são utilizados para manter ou para medir orientação. Um giroscópio mecânico consiste de um disco rotativo onde os eixos ligados a ele são capazes de se deslocar livremente em qualquer direção (PAULA, F. O, 2015).

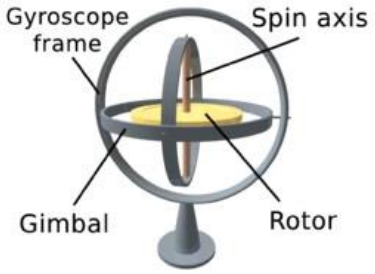



Figura 2-9 – Giroscópio
Fonte: Wikipédia, 2015

Um giroscópio microeletrônico (*MEMS*) é muito semelhante, mas em vez de um disco giratório, ele consiste em um tipo de ressonador vibrando. A ideia é a mesma. Um objeto com vibração tende a continuar vibrando no mesmo plano que as suas bases de apoio.

A Figura 2.10 mostra uma foto de um giroscópio eletrônico.

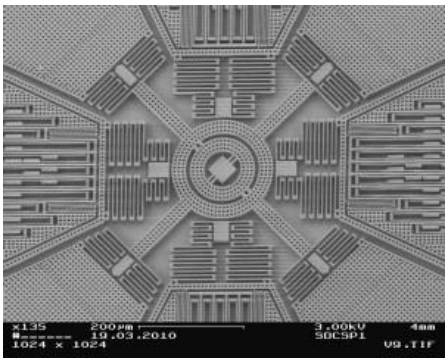



Figura 2-10 - Giroscópio eletrônico

Fonte: SERRANO, D. E, 2013

Um giroscópio microeletrônico mede a velocidade angular (graus por segundo) a partir do qual se pode calcular o ângulo e o deslocamento.

Para processar a orientação, é preciso inicializar o sensor com uma posição conhecida (utilizando o acelerômetro, por exemplo) e então medir a velocidade angular nos eixos X, Y e Z por meio dos giroscópios, dentro de um intervalo de tempo conhecido (GEEK MOM PROJECTS, 2015). O valor do ângulo será o valor medido, multiplicado pelo intervalo de tempo, somado ao valor inicial fornecido pelo acelerômetro.

O problema neste caso é que 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, o *datasheet* informa que 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.

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*.

A Figura 2.11 mostra o valor aferido por um giroscópio, inicialmente em repouso, e então é feito um movimento para um sentido e para outro e volta à sua posição inicial.

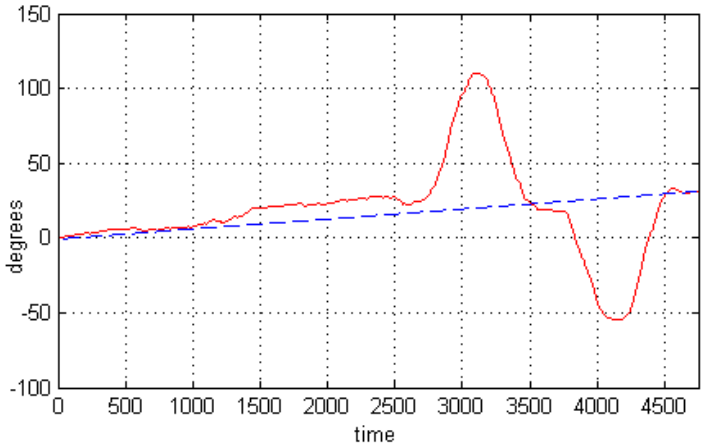



Figura 2-11 - Desvio do giroscópio: "bias"

O valor inicial do giroscópio, sem movimentos em posição horizontal é zero (pode ser necessário uma normalização caso a posição inicial do giroscópio seja diferente de zero). Porém, após alguns movimentos, é possível ver que, mesmo voltando o sensor para a posição inicial, o valor aferido pelo giroscópio se desviou do inicial. A linha tracejada mostra claramente este desvio que aumenta com o passar do tempo. Após 4.500 amostras (aproximadamente 12 segundos nos testes), o valor inicial se desviou em aproximadamente 30 graus. Essa taxa de variação precisará ser utilizada nos cálculos.

### 2.4.4. Cálculo do Ângulo Final

Como mencionado, o ângulo em cada um dos eixos X, Y e Z pode ser calculado através das relações trigonométricas com os acelerômetros ou a partir de uma integração ao longo do tempo com os giroscópios, considerando pequenos intervalos de tempo “dt” entre as aferições. O acelerômetro possui um valor bem preciso a longo prazo, após a estabilização do sensor, porém possui muito ruído nas leituras instantâneas e quando está sofrendo movimentações. Já o giroscópio possui um valor preciso a curto prazo, bem como durante as mudanças de orientação. Porém, devido à integração para o cálculo do ângulo final, ele acaba gerando um erro que tende a aumentar a longo prazo.

A solução para reduzir os erros e chegar a um valor preciso em curto prazo, e sem erros se propagando em um longo prazo, é utilizar o acelerômetro e o giroscópio em conjunto. Para isso, utilizaram-se dois filtros: Filtro de Kalman e Filtro Complementar.

### 2.4.5. Filtro Complementar (*Complementary Filter*)

O Filtro Complementar ou *Complementary Filter* é um combinação linear matemática simples na qual se define um peso para cada variável de entrada, neste caso uma para o acelerômetro e outra para o giroscópio, sendo que a soma desses pesos deve ser igual a 1.

As equações 2.6, 2.7 e 2.8 expressam as relações matemáticas utilizadas por este filtro.

$$\theta = \alpha\, Gy + (1 - \alpha) Acc \tag{2.6}$$

Sendo:

$$\alpha = \frac{\tau}{\tau + dt} \tag{2.7}$$

$$Gy = \theta + \omega\ \mathrm{dt} \tag{2.8}$$

Tendo que:

- dt: tempo de amostragem, ou dt;
- τ: constante de tempo maior do que a escala de tempo típica do ruído do acelerômetro;
- ω: velocidade angular atual aferida diretamente do giroscópio;
- Acc: ângulo atual fornecido pelo acelerômetro;

Como foi utilizado um tempo de amostragem de dt = 0,04s e uma constante de tempo τ = 1, obtém-se um valor de $\alpha \approx 0.96$. Este valor será utilizado durante todo este trabalho.

Utilizando este valor para α, chega-se a uma medição mais precisa e menos ruidosa. Outra abordagem mais robusta, porém mais complexa, é a utilização do Filtro de Kalman.

### 2.4.6. Filtro de Kalman

O filtro de Kalman é um método matemático criado por Rudolf Kalman, e apareceu na literatura em 1960 (GREWAL, M. S., ANDREWS, A. P., 2008). Segundo os mesmos autores, o filtro de Kalman é um conjunto de equações matemáticas em forma de um algoritmo computacional iterativo, com a finalidade de realizar previsões futuras e estimar variâncias de modelos para séries temporais.

O algoritmo do filtro de Kalman é aplicado em séries temporais que podem ser escritas em formas de espaços de estados (HARVEY, A. C., 2001). Quase todos os modelos convencionais de séries temporais podem ser representados dessa forma (GREWAL, M. S., ANDREWS, A. P., 2008).

O filtro apresenta diversas aplicações para tecnologias espaciais, militares e da engenharia de controle. O estudo do filtro de Kalman é complexo e muito extenso.

Basicamente, o filtro de Kalman é um algoritmo iterativo que estima valores futuros para uma dada grandeza (neste caso o ângulo atual do sensor) e, após fazer a leitura real desta medição que foi estimada, faz-se uma média ponderada entre elas. É um sistema recursivo e que requer apenas a última estimativa, e não o histórico completo e corrige os pesos com o objetivo de manter a estimativa sempre o mais próximo possível da realidade. Os pesos são calculados através da covariância, uma medida da incerteza estimada da predição do estado do sistema. Ele leva em consideração incertezas de medições, dando peso maior aos valores com grau de incerteza menor. As estimativas geradas pelo método tendem a estar mais próximas dos valores reais que as medidas originais, pois a média ponderada apresenta uma melhor estimativa de incerteza que ambos os valores utilizados no seu cálculo.

O filtro de Kalman só deve ser aplicado em sistemas em que as variáveis de interesse apresentem distribuição normal.

Os cálculos para a estimativa do estado e as covariâncias são representados por matrizes, já que se trabalha com mais de uma dimensão em um único cálculo.

Encontram-se diversas dissertações e estudos que abordam exclusivamente o assunto. O interesse deste trabalho não é aprofundar no estudo desse filtro e suas equações, mas fazer uma implementação especificamente para o problema da estimação do ângulo através dos

acelerômetros e giroscópios e então, fazer uma comparação entre os filtros Complementar, Kalman e o modo DMP.

LAUSZUS, K. (2016) desenvolveu uma biblioteca de código aberto para Arduino para a implementação do filtro de Kalman que foi utilizada neste trabalho.

### 2.4.7. Digital Motion Processor (DMP)

O sensor MPU-6050 possui internamente um pré-processamento para o cálculo do ângulo atual em tempo real. Esse processamento utiliza informações do giroscópio e do acelerômetro para o cálculo do ângulo. Utiliza também algoritmos de filtros para excluir os efeitos de ruídos. O modo de funcionamento DMP gera uma interrupção externa em um dos pinos quando o valor está disponível para leitura (DATASHEET MPU6050, 2015).

Utilizando o modo DMP, não é necessário mais utilizar os filtros Complementar e de Kalman, nem mesmo calcular os ângulos utilizando os acelerômetros e os giroscópios como mencionado anteriormente. Dessa forma economiza-se muito processamento no Arduino, já que todos estes cálculos são feitos internamente no sensor MPU-6050. Porém, o objetivo deste trabalho é, além do controle da plataforma, entender o funcionamento dos sensores e fazer uma comparação entre esses três métodos de leitura e filtro. O modo DMP executa internamente os cálculos que foram mencionados anteriormente.

O *datasheet* do MPU-6050 informa que o processo dos algoritmos internos, utilizando o modo DMP, acontecem a uma frequência de 200Hz, o que nos possibilita fazer a leitura dos ângulos com um tempo mínimo de amostragem de 5ms. Esse tempo é mais que suficiente para o sistema deste trabalho.

Porém, como será abordado posteriormente, o *datasheet* não informa como trabalhar com o modo DMP. Este modo também possui uma auto calibração que atrasa o *startup* e o início do processo de controle.

### 2.4.8. Comunicação I2C – Inter-Integrated Circuit

A comunicação com o sensor MPU-6050 para a medição dos valores dos 3 acelerômetros e dos 3 giroscópios é feita por meio de uma comunicação I2C.

O I2C (*Inter-Integrated Circuit*) é um barramento serial multimestre desenvolvido pela Philips usado para conectar dispositivos. (NXP SEMICONDUCTORS, 2016).

O I2C é utilizado em memórias EEPROMs, SDRAM, DDR e DDR2, SDRAM, em telas OLED/LCD, entre diversos outros dispositivos.

O I2C utiliza apenas duas linhas bidirecionais para a comunicação, sendo uma linha de dados seriais (SDA) e outra linha de *clock* (SCL). Neste projeto, é utilizado para a comunicação um mestre (Arduino) e um escravo (sensor MPU-6050).

São usados dois resistores de *pull-up* nas linhas de dados e *clock* para mantê-las em nível alto quando não estiverem em funcionamento, evitando que ruídos atrapalhem a comunicação. Tipicamente é usado +5 volts como tensão padrão ou +3.3 volts. Cada dispositivo tem um endereço numérico, tipicamente de 7 bits e a velocidade padrão da comunicação I2C é de 100kbits/s. A distância entre os dispositivos não deve ultrapassar alguns metros, e a capacitância do barramento deve ser de 400pF.

O dispositivo mestre é o responsável por gerar os pulsos de *clock* na linha SCL e inicia a comunicação com os escravos. Já os dispositivos escravos, recebem o *clock* e respondem quando solicitado pelo dispositivo mestre. O I2C permite multimestres, porém não é o caso deste trabalho.

Para iniciar uma comunicação o mestre passa a linha de dados SDA de nível alto para nível baixo, mantendo o nível da linha de *clock* (SCL) em nível alto. Os próximos bits completam a informação desejada, contendo o endereço do escravo, que irá receber ou enviar dados, e a informação em si. O *bit* que finaliza a transmissão é sinalizado por uma transição de nível baixo para alto em SDA quando SCL está em nível alto.

O funcionamento em nível lógico é mostrado pela Figura 2.12.

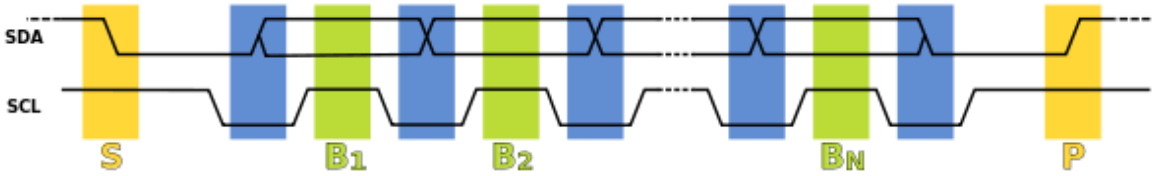



Figura 2-12 - Comunicação I2C

Fonte: Wikipédia, 2015

Funcionamento mostrado na Figura 2.12:

- S: início da comunicação. SDA vai de nível alto para nível baixo e SCL se mantém em nível alto;
- Primeira transição de dados (azul): o bit de dados em SDA deve ser configurado ou *setado* sempre quando SCL está em nível baixo;
- $B_1$: o primeiro dado (bit em SDA) é recebido quando SCL passa de nível baixo para nível alto;
- $B_2$: o processo de envio de dados se repete para envio dos dados até $B_n$, sendo que os dados são sempre recebidos quando a linha SCL faz a transição de nível baixo para nível alto (neste momento, o valor de SDA é o bit do dado em questão).
- P: sinaliza o término da transmissão, quando SDA passa de nível baixo para nível alto enquanto SCL está em nível alto;

O endereço do sensor para a comunicação I2C é mencionado pelo *datasheet* do MPU-6050, e é informado em binário, sendo “1101000”, correspondente a 104 em decimal ou 64 em hexadecimal (0x64).

O Arduino possui uma biblioteca nativa para a comunicação I2C chamada “Wire.h” que já faz todo o processo mencionado anteriormente com as linhas SDA e SCL. É necessário informar o endereço do dispositivo que se deseja comunicar e os dados que deseja transmitir e a biblioteca se encarrega de fazer todo o trabalho em baixo nível.

Também será utilizada uma biblioteca desenvolvida por Jeff Rowberg que, além da biblioteca para comunicação I2C, contém uma biblioteca específica para comunicação com o sensor MPU-6050 utilizando Arduino para trabalhar no modo DMP (ROWBERG, J., 2016).

# 3. MONTAGEM MECÂNICA DA PLATAFORMA

A montagem da plataforma foi feita de forma simples e com material facilmente encontrado e de baixo custo. As listas dos materiais utilizados bem como as etapas da montagem seguem nas seções seguintes:

## 3.1. Materiais utilizados

Foram utilizados os seguintes materiais:

- Placa de carbonato alveolar, com dimensões 20cm x 10cm;
- Um tubo PVC de 5cm de diâmetro e 1m de comprimento;
- Uma cantoneira de alumínio, com 70cm de comprimento;
- Uma barra de alumínio com 70cm de comprimento de 1cm de largura;
- Dois servos-motores de 3kg*cm de torque;
- Um sensor MPU-6050, com 3 giroscópios e 3 acelerômetros;
- Um Arduino Nano versão 3.0;
- Um microcomputador para aquisição dos dados e modelagem do sistema;

A Figura 3.1 mostra uma foto de alguns dos materiais utilizados antes da montagem.

Figura 3-1 - Alguns Materiais Utilizados na Plataforma

### 3.2. Montagem da Plataforma

Com o material em mãos, a cantoneira de alumínio foi dobrada e fixada na placa de carbonato. Logo após foi dobrada a barra de alumínio e fixados os servomotores. Um dos servos foi fixado ao tubo PVC, apenas por pressão. Foi colocado então o sensor MPU-6050 na parte inferior da placa de carbonato e os fios passados internamente pelo tubo PVC.

A plataforma final pode ser vista na Figura 3.2:

Figura 3-2 - Plataforma Montada com os Servos e o Sensor MPU-6050

A Figura 3.2 mostra claramente o sensor e os dois graus de liberdade da plataforma, que pode se mover no eixo X e no eixo Y através dos dois servos. Ao lado esquerdo é possível ver um pequeno contrapeso de metal com o objetivo de equilibrar a plataforma, contrabalanceando o peso do servo do lado direito.

Os fios que ligam os servos e o sensor foram montados de forma que não atrapalhem os movimentos da plataforma.

O sensor MPU-6050, que fornece o ângulo da plataforma, foi instalado com uma fita dupla-face na parte inferior da placa de carbonato.

O tubo de PVC, que servirá como manopla, onde é possível segurar a plataforma, tem um comprimento de 1 metro, de forma que a plataforma possa ficar em uma boa altura.

# 4. MONTAGEM ELETRÔNICA DA PLATAFORMA

A montagem eletrônica foi feita em um *protoboard* para execução da fase de testes. Foi utilizado um microcomputador e uma fonte de tensão externa para alimentação dos sensores e servomotores. Após os testes, o Arduino foi embutido dentro do tubo PVC juntamente com uma bateria externa.

## 4.1. Dispositivos Utilizados na Montagem Eletrônica

Foram utilizados 1 Arduino Nano v3.0, 1 capacitor cerâmico de 100nF (visualmente 104 no encapsulamento), 2 resistores de 2,2 kΩ, 1 *ferrite bead* (proteção contra interferências, discutido posteriormente) e 1 *protoboard*.

## 4.2. Ligações Eletrônicas

O sensor MPU-6050 utiliza 5 ligações, sendo 2 da alimentação (*vcc* e *gnd*), duas para a comunicação I2C, como mencionado anteriormente (SDA e SCL), e uma outra para a interrupção externa, que sinaliza quando os dados estão prontos para serem aferidos. Os servos utilizam 3 ligações, sendo 2 para a alimentação e uma para o PWM. Os dois resistores foram utilizados como resistores de *pull-up* (um resistor para cada uma das duas linhas de comunicação conectando cada linha à fonte de alimentação de +5v) na comunicação I2C entre o sensor e o Arduino e o capacitor foi inserido entre a alimentação e o terra (*gnd*) do circuito com o objetivo de filtrar qualquer tipo de ruído ou pequenas variações da fonte de tensão para alimentação do Arduino e do sensor.

As conexões entre o Arduino e os dispositivos utilizados foram:

- Pino D5: PWM do servo no eixo Y;
- Pino D6: PWM do servo no eixo X;

- Pino A4: SDA do sensor MPU-6050;
- Pino A5: SCL do sensor MPU-6050;
- Pino D2: interrupção externa do sensor MPU-6050;
- VIN: alimentação do sensor MPU-6050;

A Figura 4.1 mostra uma imagem com o esquema elétrico utilizado.

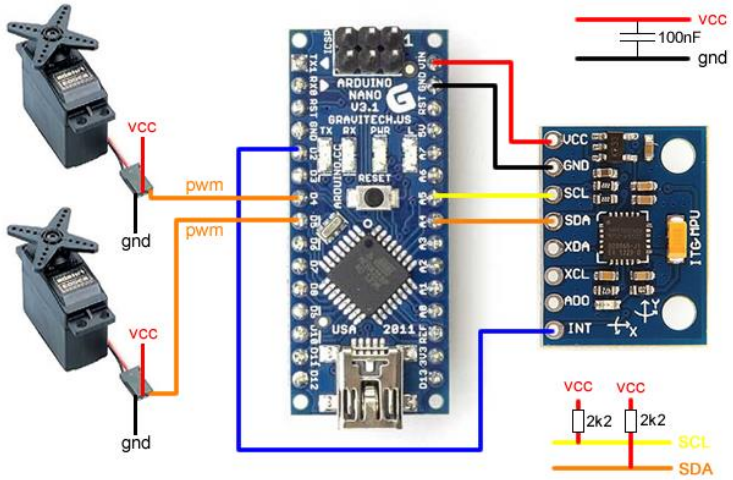



Figura 4-1 - Esquema Elétrico da Plataforma

Uma informação muito importante é que o sensor MPU-6050 foi instalado de cabeça para baixo na parte inferior da placa de carbonato. A orientação X e Y escolhida para a plataforma, de acordo com a colocação do sensor MPU-6050, foi (olhando a plataforma pela parte de trás), X positivo para frente, e Y positivo para o lado direito. Logo a orientação dos movimentos, que deverá ser levada em consideração durante todo o processo, deve ser a seguinte:

- Rotação no eixo X da plataforma: é a rotação efetuada pelo servomotor instalado na parte direita da plataforma, sendo positiva para frente e negativa para trás;
- Rotação no eixo Y da plataforma: é a rotação efetuada pelo servomotor instalado na parta de trás da plataforma, sendo a rotação positiva (para o acelerômetro), para esquerda e negativa para a direita. Para o giroscópio, foi verificado nos testes que é o inverso, ou

seja, o giroscópio Y tem a orientação invertida do acelerômetro em Y, ou seja, positiva para a direita e negativa para esquerda;

Os sinais foram corrigidos em software para ficarem mais intuitivos. A orientação final segue o que é mostrado na Figura 4.2, tanto para os acelerômetros quanto dos giroscópios:

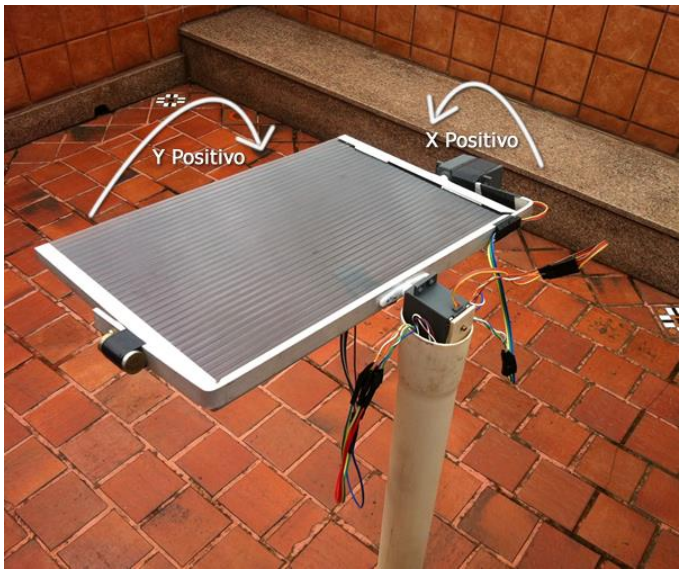



Figura 4-2 - Direções e convenções de sinais da plataforma

Partindo desse princípio de orientação, iniciaram-se os testes para a aquisição de dados.

## 5. AQUISIÇÃO DE DADOS E IMPLEMENTAÇÃO DOS FILTROS

Após a montagem mecânica da plataforma bem como a montagem eletrônica, e tomando a orientação citada anteriormente, iniciou-se a aquisição dos dados dos acelerômetros e giroscópio. Logo após é feito os cálculos dos ângulos e então os filtros foram implementados.

## 5.1. Metodologia Utilizada Para a Aquisição e a Análise dos Dados

As aquisições de todos os dados foram feitas através da comunicação USB entre o Arduino e um micro computador. Todos os dados foram exibidos 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 (todas as planilhas com os dados estão disponíveis no *web-site* www.vinicius.info). Após a inserção dos dados no *Excel*, foram gerados gráficos, com os dados desejados, utilizando o *Matlab* através do comando "*plot()*" e com legendas inseridas através do comando "*legend()*". Através dos gráficos, foram feitas todas as análises necessárias.

É importante ressaltar que a maioria dos gráficos possui o número da amostra como a unidade do eixo das abcissas (adimensional). Como todas as amostras foram feitas em um mesmo intervalo de tempo, que foi calculado para cada experimento realizado e está presente em todas as planilhas de dados, não houve a necessidade de uma conversão para a unidade de tempo, deixando o número das amostras como a unidade do eixo das abscissas, mantendo fiel os gráficos gerados. Em média o tempo entre cada amostragem foi calculado em 30ms. Logo, o eixo das abscissas pode ser convertido para a unidade de tempo (segundos) multiplicando-a por 0,03 caso seja de interesse do leitor.

## 5.2. Aquisição Pura dos Valores dos Acelerômetros e Giroscópios

A leitura dos dados do sensor foram feitas através da comunicação I2C, utilizando a biblioteca "Wire.h" nativa do Arduino. O código completo pode ser visto no **ANEXO I.**

Como foi mencionado anteriormente, os valores lidos pelos acelerômetros devem ser divididos por 16.384 e dos giroscópios por 131 (de acordo com o *datasheet*).

A uma temperatura constante de 26,51 graus (também fornecida pelo sensor MPU-6050), foram feitas as aquisições dos dados dos acelerômetros e dos giroscópios através do Monitor Serial do Arduino, fazendo um movimento de +90 graus e depois -90 graus. Com auxílio do Matlab, foi plotado o gráfico com os valores dos acelerômetros e dos giroscópios dos eixos X e Y. Vale lembrar que a unidade do acelerômetro é forças "g" (múltiplos de 9,81m/s$^2$) e dos

giroscópios é DPS (*degrees per second* ou graus por segundo). Foi gerado um gráfico para os valores dos acelerômetros (Figura 5.1) e outro com os dados dos giroscópios (Figura 5.2):

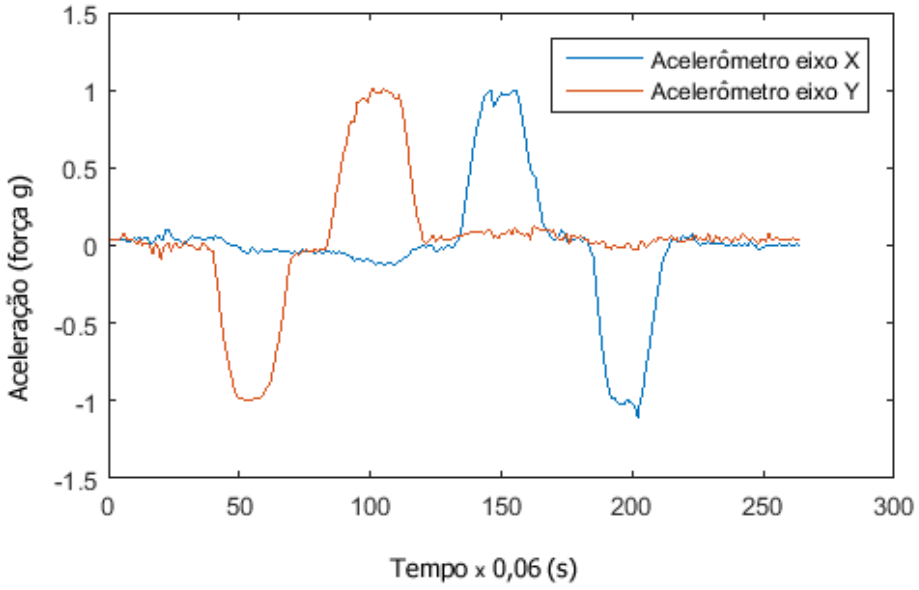
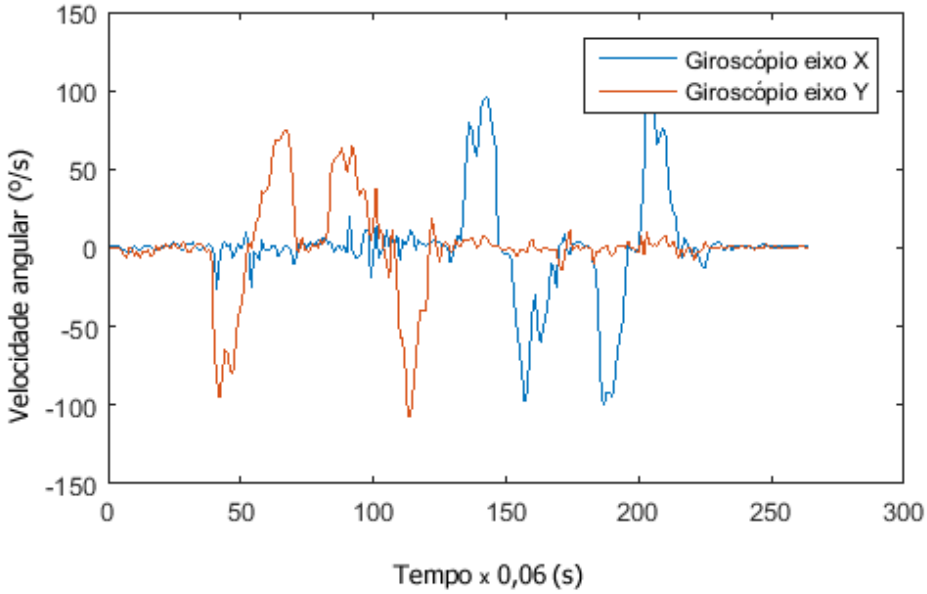



Figura 5-1 - Gráfico dos Acelerômetros X e Y em forças g's




**zm**

Figura 5-2 - Gráfico dos Giroscópios X e Y em graus por segundo

É visível que o giroscópio é mais ruidoso do que o acelerômetro. Seus valores variaram em uma faixa maior que os acelerômetros (em média cerca de -100 a 100 graus por segundo enquanto os acelerômetros variam de -1 a 1 força g). O sinal negativo indica apenas o sentido, de acordo com a orientação adotada neste trabalho. Foi desconsiderado o eixo Z nesse estudo, pois para a plataforma só é necessário levar em consideração os eixos X e Y.

## 5.3. Cálculo dos Ângulos

Aplicaram-se as fórmulas discutidas anteriormente e foram feios os calculos dos ângulos primeiramente apenas com os valores dos acelerômetros, depois com os valores dos giroscópios. Foram avaliados os resultados, e então foram aplicados os filtros.

### 5.3.1. Utilizando os Acelerômetros

Através da trigonometria já apresentada anteriormente, é possível chegar ao valor do ângulo de cada eixo através dos acelerômetros. No Arduino foi utilizado a função "*atan2( num, den )*" que devolve o arco tangente entre o numerador "*num*" e o denominador "*den*". Esse valor é dado em radianos, logo foi preciso converter para graus multiplicando por 180/π. O código pode ser visualizado no **ANEXO II.**

Novamente foi feito um movimento de -90 graus e depois +90 graus. Os resultados são mostrados, na Figura 5.3:

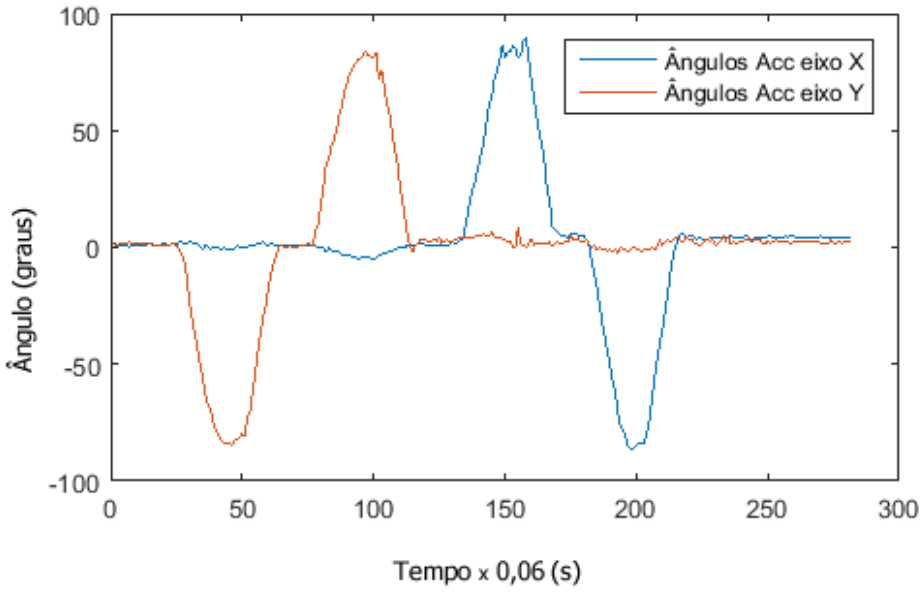



Figura 5-3 - Gráfico dos Ângulos X e Y gerados pelos Acelerômetros

Como o movimento foi feito manualmente, foram observado algumas variações no gráfico. Porém, o resultado mostra que os ângulos variaram de aproximadamente -90 a +90 graus corretamente, confirmando que os cálculos e a conversão retratam o movimento real da plataforma.

Um sério problema que existe com os acelerômetros, é o distúrbio quando acontecem vibrações ou movimentos acelerados externos (de um lado para o outro, por exemplo), mesmo sem modificar a sua orientação angular.

Foi feito um teste, movimentando a plataforma para um lado e para o outro, tentando manter a plataforma na posição inicial horizontal, sem modificar a sua posição angular, e o seguinte resultado foi obtido:

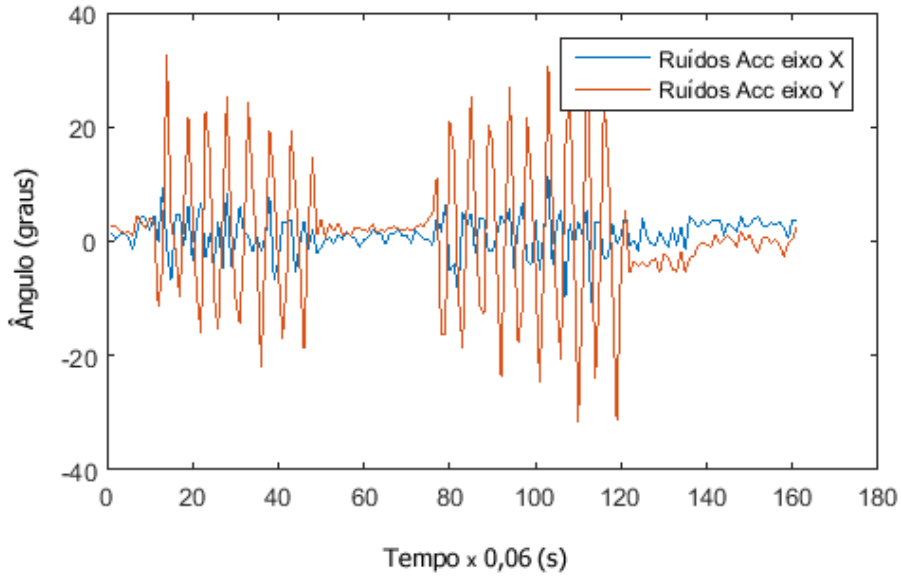



Figura 5-4 – Ruído nos Acelerômetros em Movimento Horizontal

Mesmo sem mudar a orientação dos acelerômetros (mantendo os ângulos da plataforma estáveis e horizontais), apenas os movimentando para um lado e para o outro, como existe uma aceleração nos movimentos, aparecem forças "g" que são sentidas pelos acelerômetros. Devido a este fato, se fosse levado em conta apenas os acelerômetros para se calcular ângulos no controle da plataforma, este seria impossível.

### 5.3.2. Cálculo do ângulo utilizando os giroscópios

Através da integração simples, como mencionado anteriormente, é possível calcular o ângulo atual através dos giroscópios. No Arduino utiliza-se a função "*micros()*" que fornece o tempo, em microssegundos, passados desde que o programa foi iniciado. A documentação do Arduino informa que a função "*micros()*" tem um limite de 70 minutos de contagem. Passando este tempo, ele é zerado automaticamente. Essa função é utilizada para o cálculo do "*dt*" para a integração, ou seja, o tempo de amostragem. Tendo o "*dt*", pode-se integrar a velocidade no tempo, e chegar ao ângulo percorrido. A integração, no tempo discreto, que é o caso da programação no Arduino, é um simples somatório.

### 5.3.3. Cálculo do Desvio dos Giroscópios - BIAS

Os valores iniciais dos giroscópios normalmente são diferentes de zero. Para zerar estes, calculou-se a média do valor adimensional de cada giroscópio (sem fazer a divisão por 131), chegando aos seguintes valores:

- Eixo X: 42,02;
- Eixo Y: -137,88;
- Eixo Z: -72,80;

Subtraindo estes valores em cada eixo, são zerados os valores dos giroscópios quando em posição estável.

É feito então a conversão do valor do giroscópio para graus por segundo (dividindo por 131) e então feita a integração, de acordo com a Equação 5.1:

$$Gy = \int \omega \, dt \tag{5.1}$$

sendo *Gy* o valor do ângulo em graus, percorrido durante o intervalo de tempo *dt*, e $\omega$ a velocidade angular instantânea fornecida pelo giroscópio.

Como o giroscópio fornece a velocidade angular atual, partindo de uma orientação estável, cada giroscópio fornece valores nulos, não tendo a informação do ângulo atual. A partir da integração sabe-se qual foi o deslocamento angular, mas sem saber o ângulo inicial, no momento em que se começa a integração, o valor do ângulo final será apenas o deslocamento, e não a posição angular real. Para saber o valor do ângulo inicial, é utilizado o acelerômetro.

A Figura 5.5 mostra o valor dos ângulos para os eixos X e Y, utilizando apenas os giroscópios, e partindo da posição inicial nula.

Foi efetuado um movimento de -90 graus até +90 graus, primeiramente no eixo Y e então no eixo X.

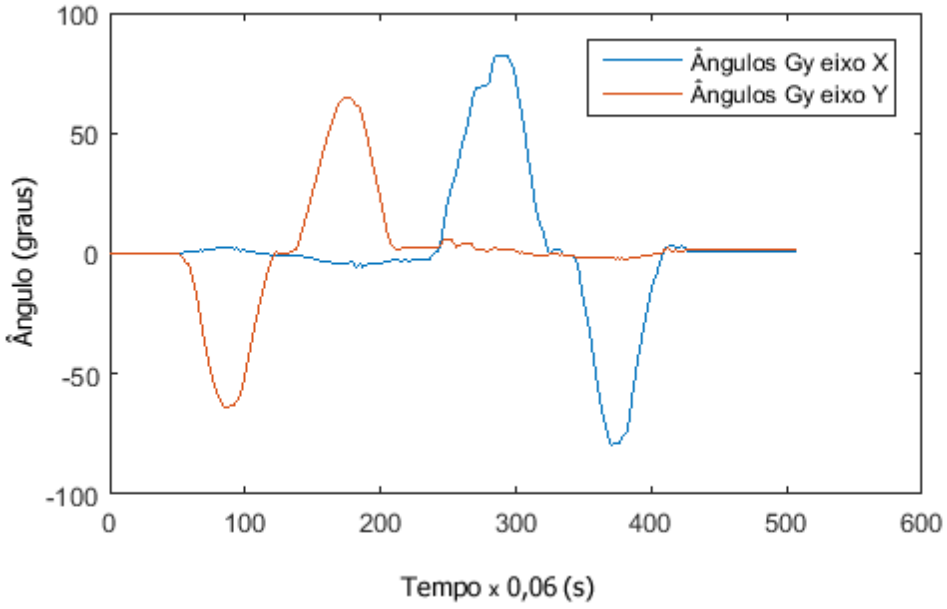



Figura 5-5 - Ângulos Fornecidos pelos Giroscópios

Fazendo uma comparação entre os valores obtidos pelos giroscópios e os dos acelerômetros, é visto que os resultado são muito próximos, apesar dos acelerômetros realmente possuírem mais ruídos.

Foi feito novamente um movimento com a plataforma, aferidos os valores dos ângulos apenas com os acelerômetros e os giroscópios e comparado os resultados graficamente. A Figura 5.6 mostra os resultados obtidos.

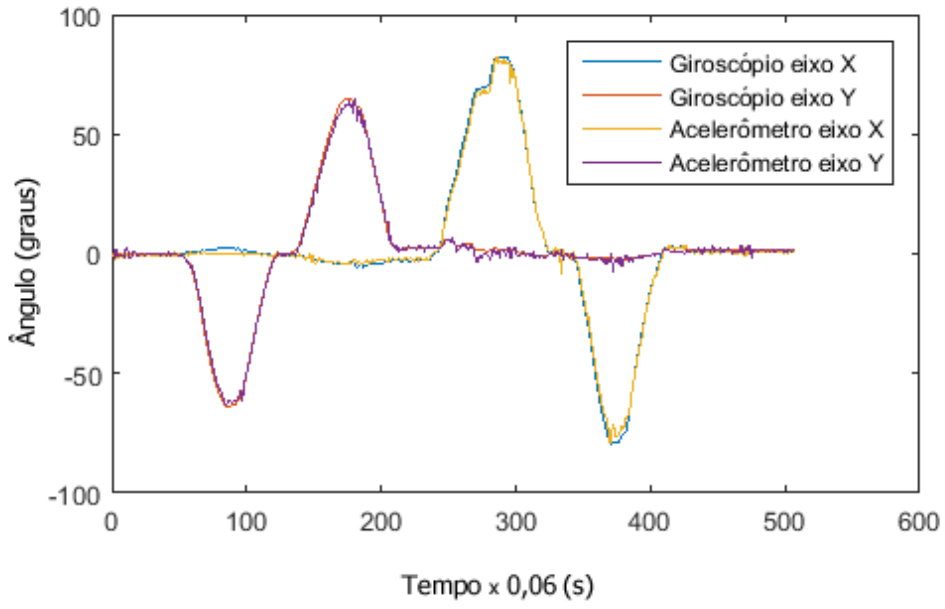



Figura 5-6 - Comparação entre os ângulos calculados pelos Acelerômetros e Giroscópios

O tempo de amostragem "*dt*", utilizado para a integração dos giroscópios foi de 60,1ms. Como foi utilizado um *delay* de 50ms no loop (como pode ser visto no código no **ANEXO III**), significa que pode-se melhorar ainda mais a integração, diminuindo o atraso manualmente através do comando "*delay()*" e, consequentemente, diminuir o "*dt*".

Como foi mencionado anteriormente, existe um desvio no valor calculado pelos giroscópios devido à integração. Ao longo do tempo o erro vai aumentando. Deixando a plataforma em uma posição estável por cerca de 5 minutos, esse desvio é observado pelos giroscópios.

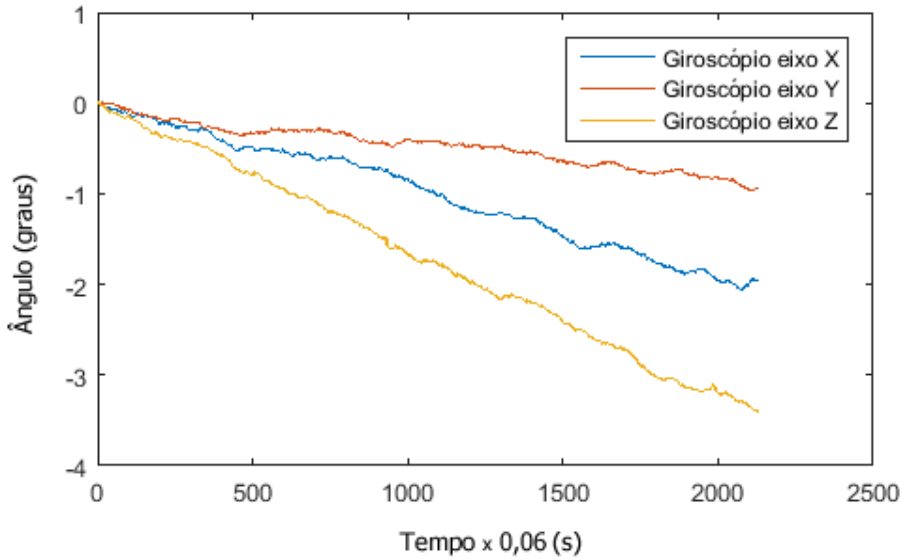



Figura 5-7 - Desvio dos Giroscópios ao longo do tempo: *bias* com a plataforma estática por 5 min

Para o eixo X (azul), o valor variou de 0 a -2 graus em 123 segundos, ou seja, um desvio de - 0,0162 graus por segundo.

Para o eixo Y (verde), o valor variou de 0 a -0,78 graus em 114 segundos, ou seja, um desvio de 0,008 graus por segundo.

Para o eixo Z (vermelho), o valor do desvio foi de 0,035 graus por segundo.

O desvio (*bias*) deve ser subtraído do valor instantâneo fornecido pelo giroscópio, e então é feito a integração. A Equação 5.2 mostra o novo cálculo.

$$Gy = \int (\omega - bias)\, dt \tag{5.2}$$

Aplicando os valores mencionados dos desvios, os desvios continuaram existindo, porém menores. Mesmo após várias tentativas de novos valores para o "*bias*", sempre existe um desvio ao longo do tempo, para o valor dos ângulos calculados apenas através dos giroscópios, porém isso não acontece com os acelerômetros. Outro fato importante, é que após desligar e ligar o sensor, os valores dos desvios se alteram. A Figura 5.8 mostra que, mesmo após várias tentativas de correção dos desvios, ele ainda existe.

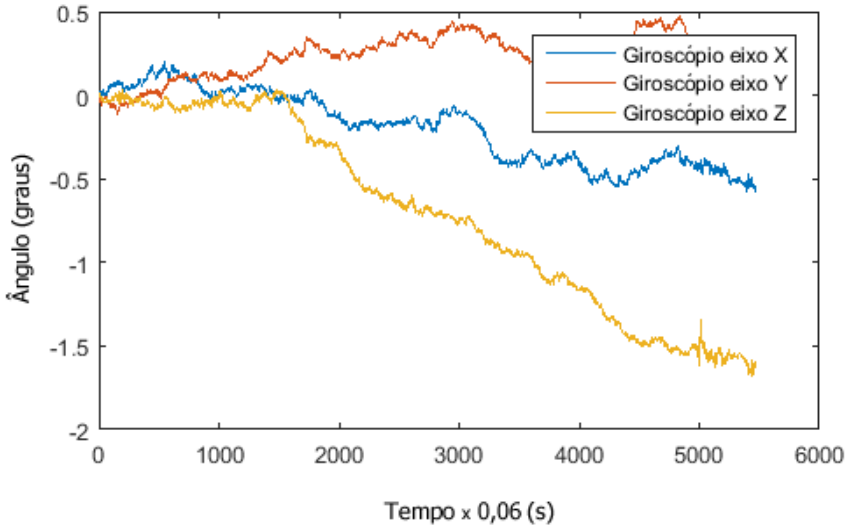



Figura 5-8 - Tentativa de compensação do desvio dos giroscópios causados pela integração

Para essas mesmas medidas, foi gerado o valor dos ângulos através dos acelerômetros, confirmando que, apesar de existir uma variação no valor lido, o desvio se mantém dentro de uma média, não existindo desvio ao longo do tempo, como acontece com os giroscópios. Nos valores mostrados pela Figura 5.9, o desvio padrão foi de 0,22 graus para o eixo X, 0,18 graus para o eixo Y e 2,09 graus para o eixo Z. Logo, os desvios para os eixos que nos interessam foram bem pequenos e realmente não há desvio ao longo do tempo.

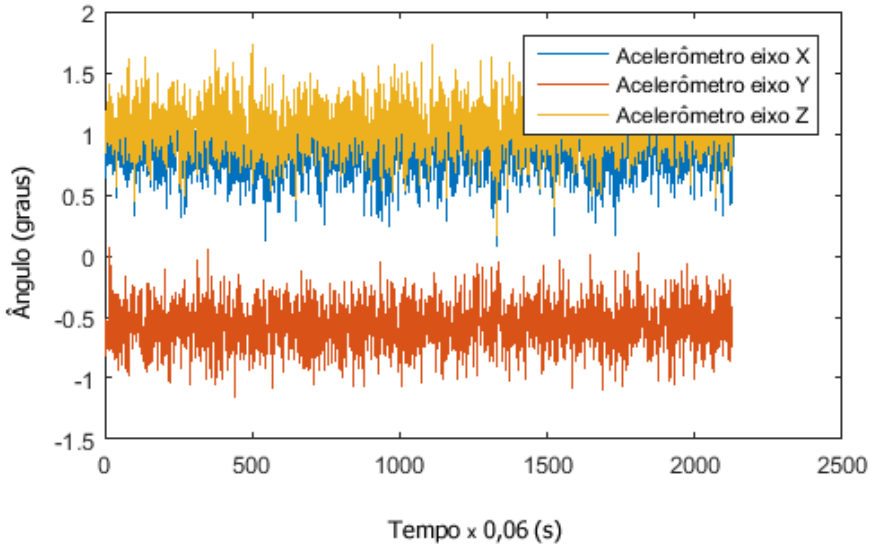



Figura 5-9 – Ângulos fornecidos pelos acelerômetros com a plataforma estática

Os ruídos devido à movimentação lateral, que causam um caos na leitura dos acelerômetros, não foram verificados nos giroscópios. Foi efetuado um movimento horizontal da plataforma, sem alterar muito a orientação angular dos eixos X, Y e Z, e o resultado obtido é mostrado na Figura 5.10.

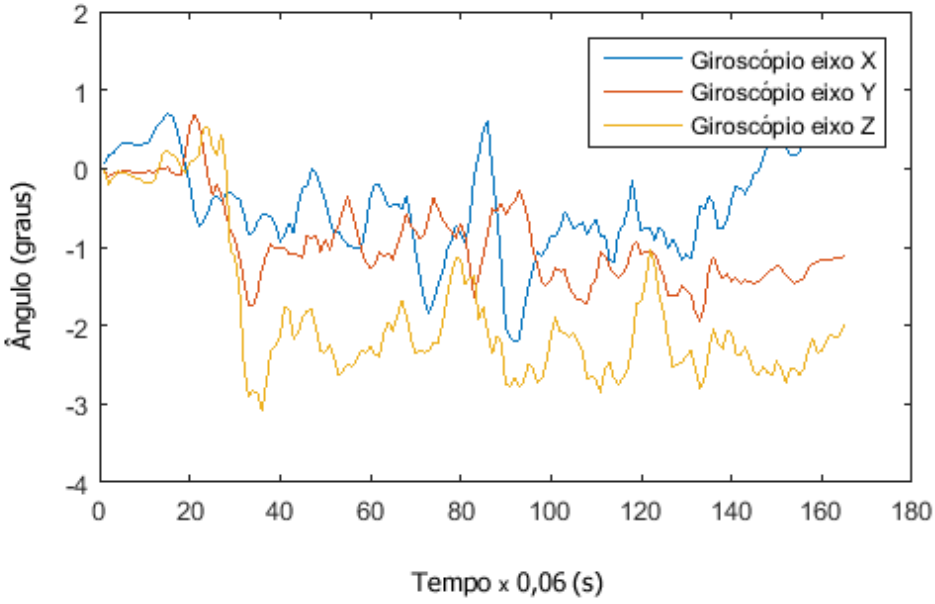



Figura 5-10 - Ruído no giroscópio devido ao movimento horizontal, considerado nulo

A primeira vista parecer ter havido um ruído grande nos valores dos ângulos fornecidos pelos giroscópios devido à movimentação horizontal da plataforma, porém ao analisar melhor, é visto que a variação máxima foi de 3 graus. Isso aconteceu devido à movimentação ter sido feita manualmente, e essa variação na orientação do ângulo deve ter ocorrido realmente, ou seja, os giroscópios não sofrem variação na sua leitura devido à movimentações ou acelerações externas que não alterem sua posição angular.

Concluído o entendimento e os testes com os acelerômetros e giroscópios, faz-se necessário os filtros, utilizando o giroscópio para valores instantâneos e o acelerômetro para valores a longo prazo.

## 5.4. Implementação dos Filtros para o Cálculo dos Ângulos

### 5.4.1. Filtro Complementar

Mesclando os valores dos acelerômetros e dos giroscópios, atribuindo um peso maior ao giroscópio que fornece valores menos ruidosos e mais precisos instantaneamente, chega-se a um valor preciso e com menos ruídos para o ângulo atual, além de eliminar o desvio ao longo do tempo ocorrido devido à integração.

Através da Equação 2.6, utilizando $\tau$ = 1s e um tempo de amostragem *dt* = 40ms, chega-se a um $\alpha$=0,96 ou 96% de peso para o giroscópio e 4% para o acelerômetro. Vale ressaltar que se utiliza o valor instantâneo dos giroscópios (velocidade angular instantânea), e não mais o ângulo calculado pela integração, excluindo definitivamente o efeito do *bias*.

A Figura 5.11 mostra uma comparação entre o valor do ângulo fornecido pelo acelerômetro (em verde e roxo) e pelo filtro complementar (em vermelho e azul), fazendo-se uma mudança manualmente e aleatória de orientação da plataforma. Ao final é introduzido uma movimentação horizontal, sem muita mudança na orientação, para avaliar os ruídos.

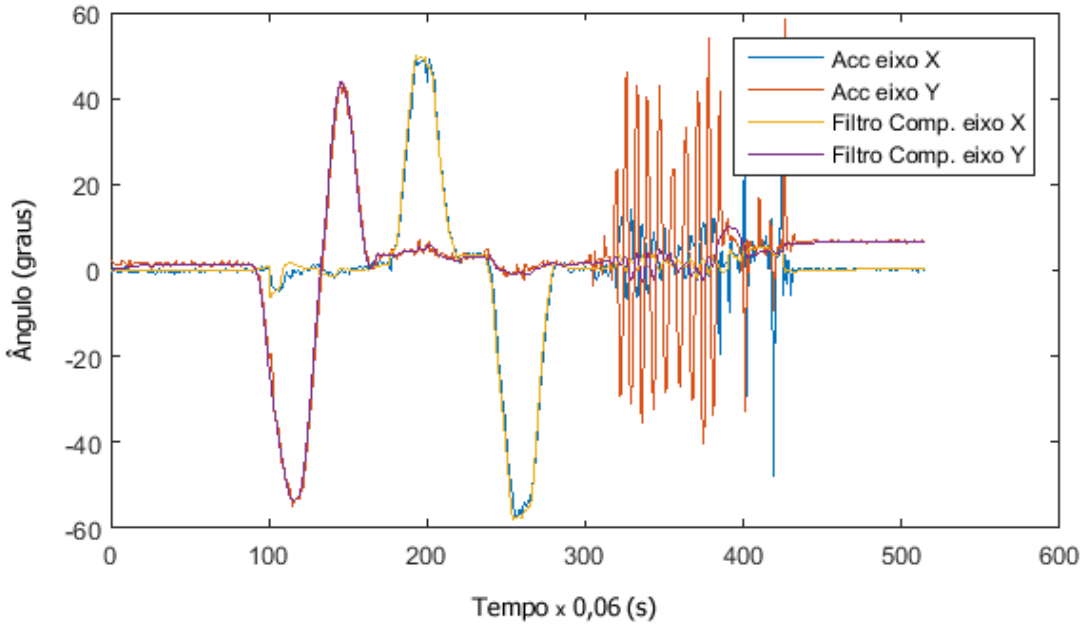



Figura 5-11 – Comparação entre os valores fornecidos pelos acelerômetros e o Filtro Complementar

O valor do ângulo gerado pelo filtro é bem menos ruidoso que o ângulo calculado pelos acelerômetros. Também é visto que os acelerômetros sofrem a ação do movimento horizontal que o filtro consegue descartar (pois os giroscópios não o sofrem). As pequenas variações que o filtro apresenta durante este período de ruídos sofridos pelos acelerômetros ocorreram realmente, já que o experimento foi feito manualmente e realmente deve ter havido uma pequena mudança na orientação angular da plataforma.

Existe um atraso no filtro para convergir para o valor inicial. Para comprovar tal efeito, a plataforma foi colocada em uma posição inicial de aproximadamente 22 graus no eixo Y, e mantida estática. Era esperado que o filtro demorasse um pouco até convergir para o valor correto, o que realmente ocorreu e é mostrado na Figura 5.12.

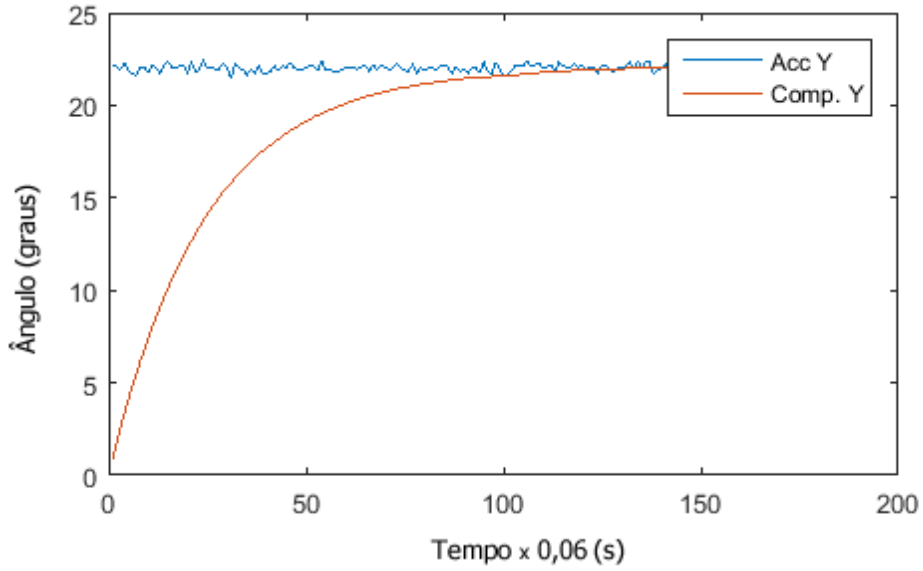



Figura 5-12 - Atraso do Filtro Complementar para convergir

Para corrigir este problema de convergência inicial do filtro Complementar, basta iniciar o valor do ângulo do filtro com o valor inicial do acelerômetro e não mais com zero. A convergência instantânea é vista na Figura 5.13.

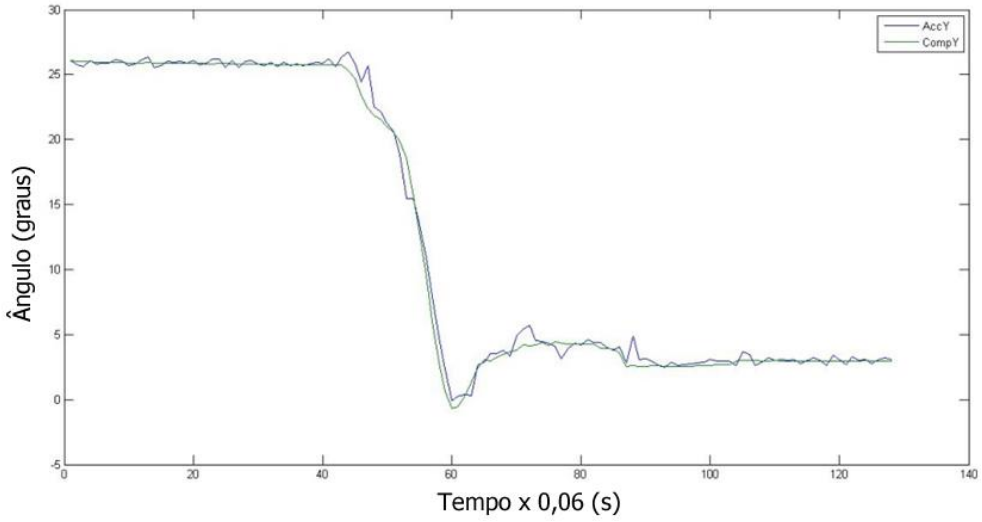



Figura 5-13 - Convergência imediata do Filtro Complementar

Mesmo que a plataforma se inicie em uma posição diferente da horizontal, o filtro já se inicia com o valor correto, sem perda de tempo na convergência.

### 5.4.2. Filtro de Kalman

O outro filtro sugerido nas bibliografias e muito utilizado é o Filtro de Kalman. Foi feito a sua implementação e comparado o resultado com o valor do acelerômetro puro, com o giroscópio e comparado também ao filtro complementar.

A Figura 5.14 mostra essa comparação. Os ruídos gerados pelos acelerômetros (azul) quase escondem o valor fornecidos pelos filtros (vermelho e verde). A Figura 5.15 mostra os resultados com uma aproximação da imagem.

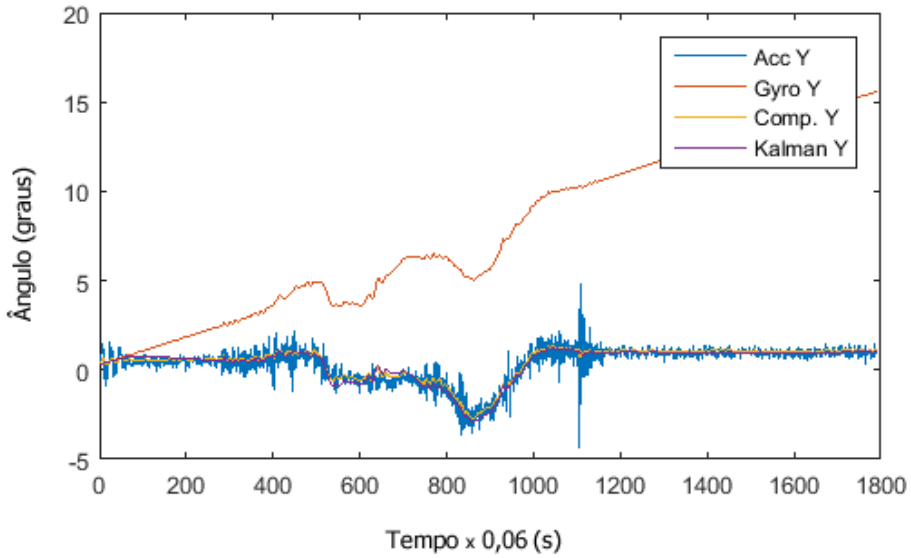



Figura 5-14 – Giroscópio, acelerômetro, filtro Complementar e Filtro de Kalman

Ao visualizar os resultados, vê-se claramente o desvio natural da integração do giroscópio (bias), os ruídos do acelerômetro e a filtragem através do filtro de Kalman e Complementar. Uma visualização mais aproximada mostra que ambos os filtros trabalham próximos e muito bem:

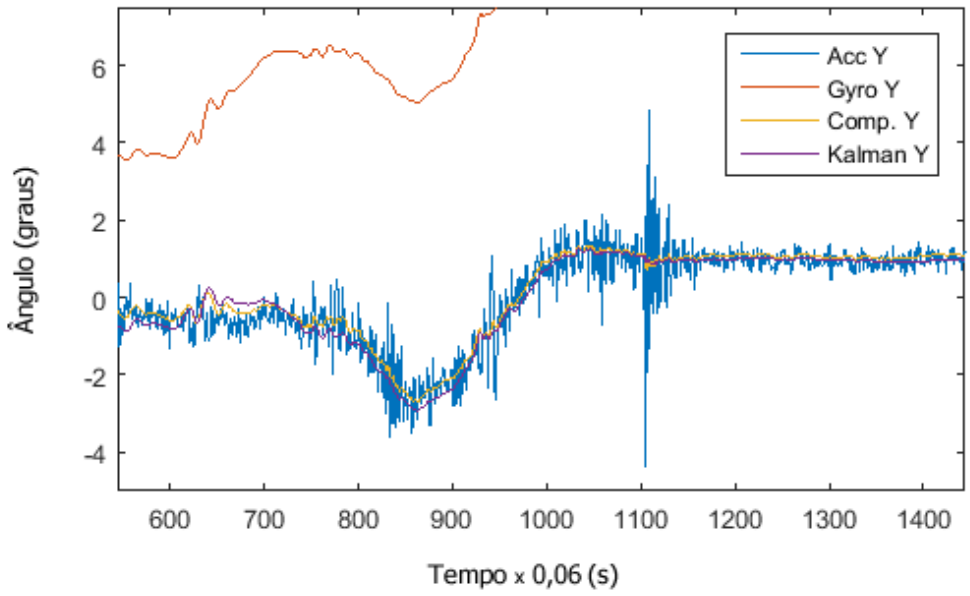



Figura 5-15 - Comparação entre Filtro de Kalman e Complementar

### 5.4.3. Modo DMP – Digital Motion Processor

A implementação do modo DMP foi dificultada, pois o próprio *datasheet* não informa os procedimentos para configurar o sensor MPU-6050 para trabalhar neste modo. Por isso, foi utilizada a biblioteca escrita por *Jeff Rowberg*, que através de engenharias reversas, conseguiu configurar e trabalhar no modo DMP.

Os testes mostraram que realmente o resultado é muito bem filtrado, praticamente sem ruídos, e acompanham a mudança de orientação da plataforma muito bem, apesar de um leve atraso em tempo de execução. Porém existe um grande tempo inicial de convergência para o valor inicial do ângulo.

O *datasheet* não informa qual algoritmo é utilizado no processamento interno, ou seja, qual tipo de filtro utiliza. Não é possível modificar os parâmetros do modo DMP, seja o alpha (se ele utilizar o filtro complementar), os pesos da matriz de covariância (se ele utilizar o filtro de Kalman), ou qualquer outro parâmetro, de forma que se consiga uma convergência mais rápida. Fica bem claro pelos dados amostrados, que o modo DMP faz uma auto calibração antes de começar o processo, e provavelmente é este o motivo da demora da convergência do valor do ângulo inicial.

A Figura 5.16 mostra a curva do valor do ângulo no eixo Y entregue pelo sensor em modo DMP, mantendo-se a plataforma o mais horizontal possível inicialmente, e então feito um movimento manualmente neste eixo:

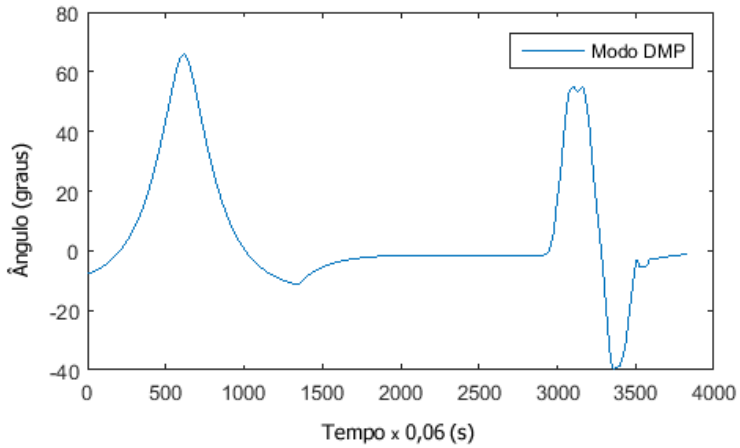



Figura 5-16 – Sensor em Modo DMP

A curva inicial não foi gerada por nenhuma movimentação, o que leva a crer que se trata de uma auto calibração até que se chegue ao valor do ângulo inicial correto, que neste caso é aproximadamente zero (plataforma na horizontal). Logo após é visto o movimento feito no eixo Y. O movimento foi de +90 graus a -50 graus, aproximadamente. O valor é entregue praticamente sem variação e sem ruído nenhum e se mostrou bem preciso. Porém o tempo inicial para a convergência (cerca de 2.000 leituras, que equivale a aproximadamente 2 minutos já que cada amostragem para este experimento foi de 0,06s ou 60ms) é muito alto. Sempre que o circuito é desligado e ligado novamente, existe esta auto calibração, logo o modo DMP foi descartado para uso neste trabalho. Fica a sugestão para trabalhos futuros uma investigação mais aprofundada deste modo, de forma a tentar fazer a convergência acontecer mais rapidamente.

### 5.5. Comparação e Escolha Entre os Filtros

Descartando o modo DMP, e tendo em vista que os resultados dos filtros Complementar e Kalman foram bem próximos, a escolha do Complementar se justifica devido a sua facilidade de implementação (não há a necessidade de usar uma biblioteca externa, como é o caso do filtro de Kalman), economizando processamento e memória do microcontrolador.

## 6. PROJETO DE CONTROLE DA PLATAFORMA

Com a aquisição do ângulo realizada com sucesso, foi feito o controle para manter a plataforma sempre na posição horizontal, ou seja, manter os ângulos dos eixos X e Y nulos. Os servomotores serão os responsáveis por isso, ou seja, sempre que for verificado qualquer valor diferente de zero no ângulo do eixo X ou do eixo Y, o servo correto entrará em ação, atuando diretamente no processo para corrigir este erro.

A grande vantagem em usar os servos, é que como eles já possuem um controlador interno de posição, basta informá-los qual a posição que se deseja. Dessa forma não será necessário projetar um controlador de posição para os motores dos servos. Um controlador tipo

seguidor de referência será projetado, com a referência sendo o valor nulo para os ângulos no eixo X e Y para manter a plataforma sempre na posição horizontal, independentemente de distúrbios ou movimentos externos. O valor de saída do controlador será o ângulo em graus que deve se informar a cada servo. Alguns parâmetros importantes desejados no controlador são:

- Anular o erro no menor tempo possível (tempo de acomodação Ts menor possível);
- Minimizar ao máximo o *overshoot*;

O primeiro passo para projetar o controlador é conhecer o processo. Como não existe um modelo matemático teórico que represente a plataforma, este será encontrado a partir de alguns testes.

## 6.1. Modelagem Matemática da Plataforma

Para a modelagem da plataforma, primeiramente foi aplicado 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). Verificou-se 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.

Mantendo a plataforma estabilizada horizontalmente, fez-se um movimento de 45 graus, primeiramente no eixo X depois no eixo Y, e avaliou a resposta através dos ângulos fornecidos pelo sensor. As Figuras 6.1, 6.2, 6.3 e 6.4 mostram o resultado deste movimento para cada eixo.

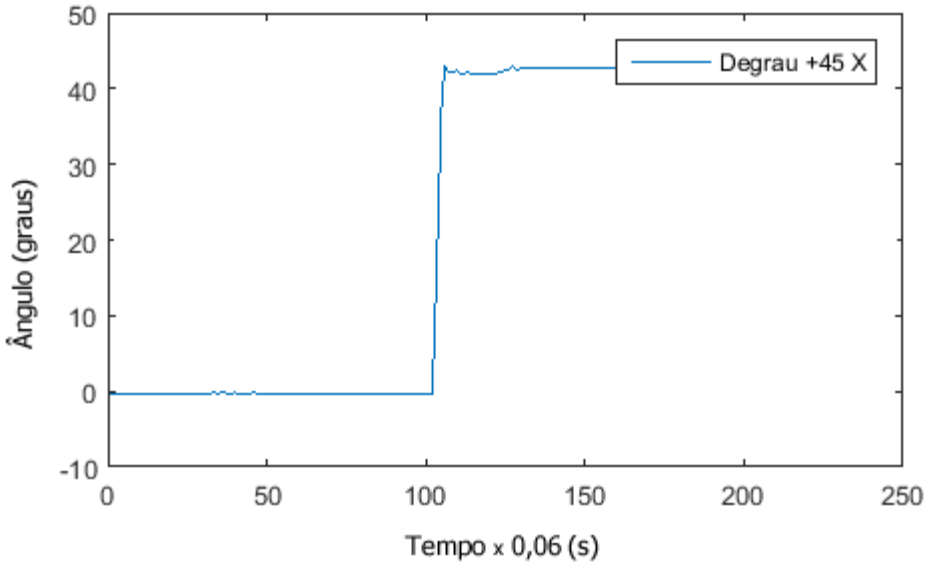



Figura 6-1 - Degrau +45 graus no eixo X

A convergência para o ângulo de 45 graus foi bem rápida (cerca de 280ms) e o *overshoot* visualizado foi bem pequeno.

O *overshoot* provavelmente ocorreu devido à inércia da plataforma e ao próprio controle interno do servomotor.

Foi feito o mesmo teste, agora para -45 graus ainda no eixo X, como mostra a Figura 6.2:

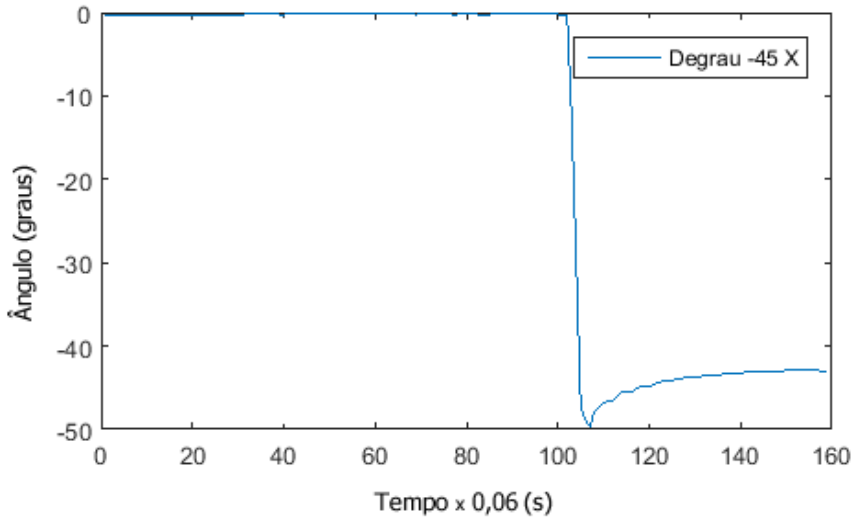



Figura 6-2 - Degrau -45 graus no eixo X

Neste caso, o *overshoot* observado foi maior, provavelmente devido à uma maior inércia da plataforma no sentido deste movimento, mas ainda assim também foi pequeno e aceitável.

Também foram aplicados os degraus ao eixo Y, de acordo com as Figuras 6.3 e 6.4.

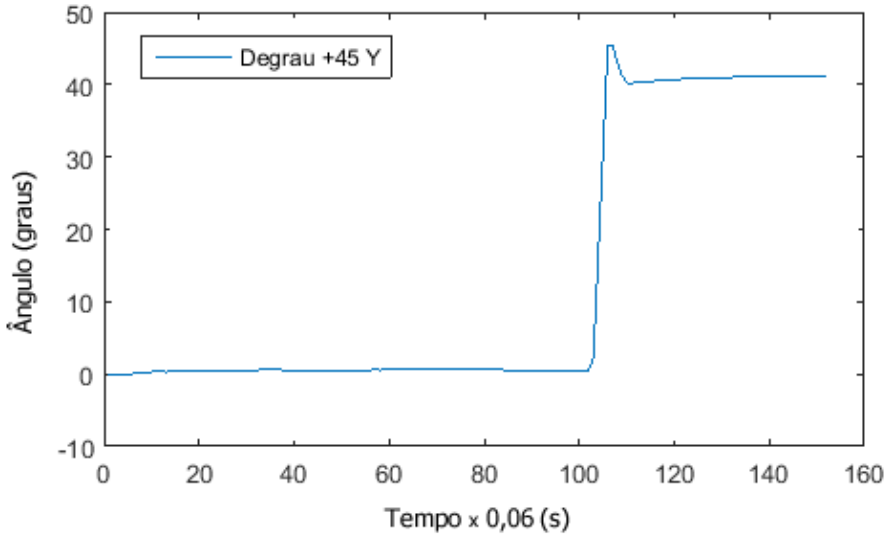



Figura 6-3 - Degrau de +45 graus no eixo Y

É nítido que o *overshoot* para este movimento foi bem maior, e é compreensível visto que a plataforma possui a sua largura maior do que o seu comprimento, tendo uma inércia maior no eixo Y. Porém, rapidamente o controle do servo corrigiu o efeito e a plataforma se estabilizou. O mesmo é visto para o movimento de -45 graus neste mesmo eixo.

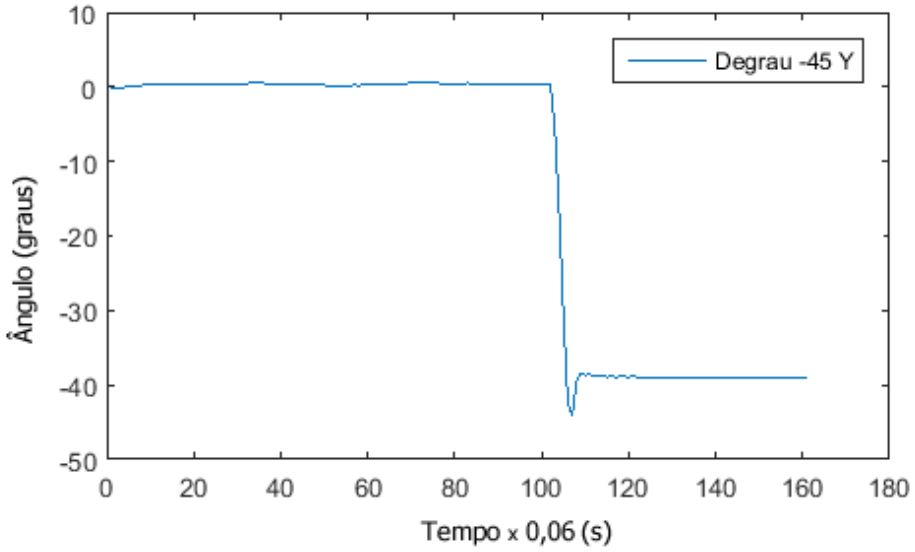



Figura 6-4 - Degrau de -45 graus no eixo Y

Uma informação importante é que o servo que controla o eixo Y possui uma folga grande em suas engrenagens, que são de plástico e desgastam naturalmente (foi verificado cerca de 3 graus de folga). Por isso também o *overshoot* deste eixo é maior. A folga influencia diretamente no *overshoot*, que foi de cerca de 4 graus (8% de 45 graus) para o eixo Y.

Fica claro que o comportamento da plataforma é levemente diferente para os eixos X e Y, porém, para que não haja a necessidade de se fazer um controlador distinto para cada eixo, considera-se que o comportamento é igual. Foi feito um controlador e uma sintonia para ambos os eixos.

Finalmente, analisando os gráficos, pode-se considerar que o sistema da plataforma segue um modelo de segunda ordem, já que foi verificado um *overshoot* em todos as movimentações.

Com objetivo de revisão, será feito dois ajustes para o modelo. Um considerando que a plataforma é um sistema de primeira ordem (desconsiderando os *overshoots*) e outro a considerando como um sistema de segunda ordem (mais próximo do real).

Um sistema de primeira ordem se comporta de acordo com a Equação 6.1:

$$T(s) = \frac{1}{\tau s + 1} \tag{6.1}$$

Sendo $\tau$ chamado de constante de tempo (63,2% do tempo de subida).

Considerando o degrau de +45 graus efetuado no eixo Y, como referência para a modelagem (τ = 0.167 segundos), chega-se ao seguinte modelo matemático da plataforma:

$$T(s) = \frac{1}{0{,}167s + 1} \tag{6.2}$$

O modelo é criado no Matlab, considerando o ângulo final de 41,2 graus (ou seja, será aplicado um degrau no software com este valor, e não unitário para fins de comparação). É aplicado então um degrau, com atraso de 7 segundos, apenas com a finalidade de comparação entre o resultado experimental e o modelo calculado, já que na prática, o degrau foi aplicado também após 7 segundos.

Vale ressaltar que incluir um atraso, é equivalente a multiplicar a função de transferência no domínio do tempo por $e^{(-ts)}$. Por meio do Matlab foi gerado a resposta ao degrau do modelo, como é visto na Figura 6.5.

```
G = tf( 41.2, [0.167 1], 'InputDelay', 7);
step(G)
```

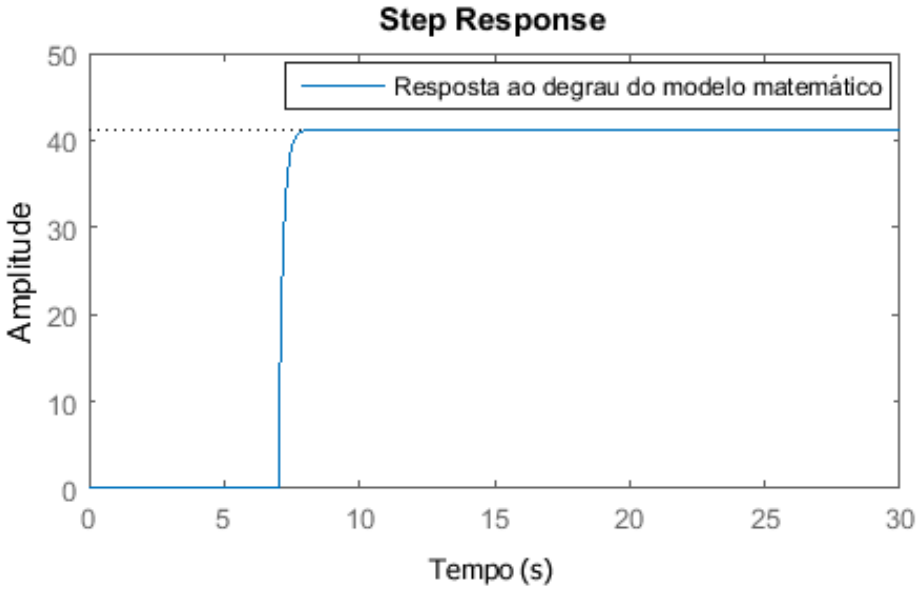



Figura 6-5 - Resposta a um degrau do modelo matemático de primeira ordem

O resultado ficou muito próximo da resposta real, validando o modelo de primeira ordem. Como dito anteriormente, este modelo será considerado como uma boa aproximação para o modelo de ambos os eixos X e Y e como forma de revisão dos modelos de primeira ordem.

Se o sistema for considerado como um sistema de segunda ordem, que é realmente mais correto, este sistema deve ser representado pelas equações que se seguem.

O modelo matemático de segunda ordem tem a forma genérica de acordo com a Equação 6.3.

$$T(s) = \frac{Wn^2}{s^2 + 2\,\zeta\,Wn\,s + Wn^2} \tag{6.3}$$

Para os cálculos é necessário especificarmos o tempo de acomodação "*Ts*", o *overshoot* "*Os*" e o tempo de pico "*Tp*", e então calcula-se os demais parâmetros de acordo com as equações 6.4, 6.5, 6.6 e 6.7.

$$Ts = \frac{4}{\zeta\,Wn} \tag{6.4}$$

$$\%Os = 100\, e^{\left(\frac{-\zeta\,\pi}{\sqrt{1-\zeta^2}}\right)} \tag{6.5}$$

$$Tp = \frac{\pi}{Wn\sqrt{1-\zeta^2}} \tag{6.6}$$

$$\zeta = \frac{-\ln(\,{}^{\%Os}/_{100}\,)}{\sqrt{\pi^2 + \ln\left({}^{\%Os}/_{100}\right)^2}} \tag{6.7}$$

Sendo ζ (*dzeta*) o fator de amortecimento do sistema e *Wn* a frequência natural.

Considerando o degrau efetuado de +45 graus no eixo Y observado, chega-se aos seguintes valores:

- Tempo de acomodação Ts: 1,61s;
- *Overshoote* Os: 4,43 graus ou 10,7%;
- Tempo de pico Tp: 0,35s;

Calculando os valores para ζ (*dzeta*) e *Wn* (frequência natural do sistema), chega-se aos seguintes resultados (estes valores, bem como os calculados para o modelo de primeira ordem, podem ser calculados através da planilha em Excel criada durante este trabalho, que está disponível juntamente com todos os outros dados no website como mencionado anteriormente):

- ζ = 0,59 – o valor faz todo sentido, já que foi observado poucas oscilações nos dados experimentais, ou seja, o sistema é subamortecido (0 < ζ < 1);
- Wn = 4,20;

Logo, o modelo de segunda ordem é dado pela Equação 6.8.

$$T(s) = \frac{17{,}66}{s^2 + 4{,}97s + 17{,}66} \tag{6.8}$$

Mais uma vez, incluindo um atraso de 7 segundos antes de aplicar o degrau unitário a fim de comparar o resultado com o experimento realizado na prática, no Matlab foi feito os seguintes procedimentos:

*G2=tf(17.66, [1 4.97 17.66], 'InputDelay', 7);*

*step(G2);*

*legend('Degrau unitário para o modelo de segunda ordem')*

A resposta ao degrau fornecida pelo Matlab é mostrada na Figura 6.6.

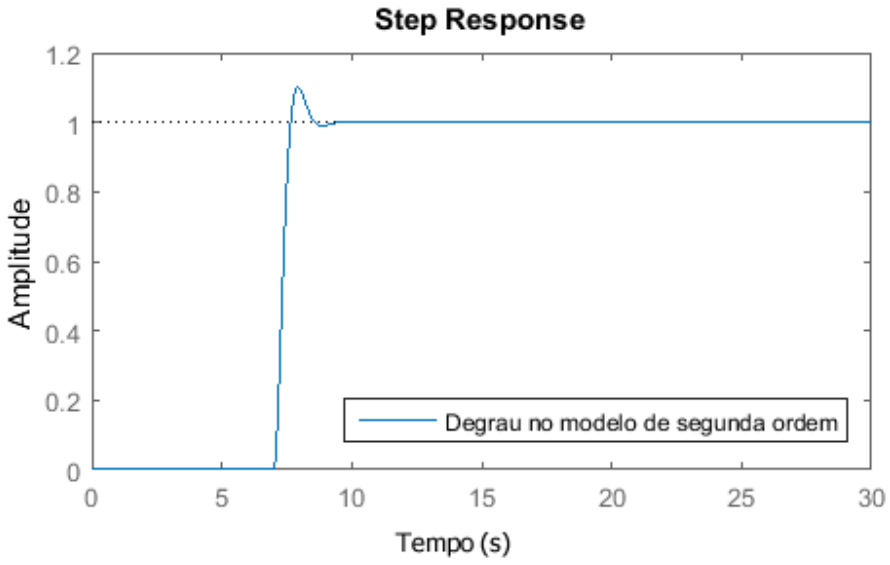



Figura 6-6 - Resposta ao degrau do modelo matemático de segunda ordem

Mais uma vez o resultado foi muito satisfatório. O resultado do modelo teórico ficou praticamente idêntico ao experimental. O *overshoot* ficou dentro do especificado, o tempo de acomodação bem como o tempo de atraso para a aplicação do degrau também ficaram dentro do especificado. O subamortecimento também ficou muito próximo ao observado

experimentalmente. Logo, é possível dizer que este modelo matemático de segunda ordem representa bem a plataforma.

A modelagem matemática a partir de experimentos e análises práticas mostrou-se ser um método eficaz e aplicável em processos reais. Normalmente os modelos matemáticos de sistemas dinâmicos são calculados a partir de parâmetros nominais como momento de inércia, peso, entre outros, que são variáveis muitas vezes difíceis de mensurar com precisão. Essas medidas ficam ainda mais difíceis de serem aferidas quando um sistema é composto por diversos mecanismos e elementos distintos.

O modelo matemático encontrado neste trabalho mostrou-se muito próximo da dinâmica real do sistema da plataforma, mesmo sem termos informações precisas dos parâmetros que seriam necessários para se calcular matematicamente tal modelo.

Os dois modelos encontrados, de primeira e de segunda ordem, podem ser considerados para a plataforma, visto a proximidade de ambos em relação à dinâmica real da plataforma.

## 6.2. O Controlador PID

O objetivo deste trabalho é desenvolver um controlador para manter a plataforma na posição horizontal, logo, o objetivo real é desenvolver um controlador tipo seguidor de referência, sendo a referência, o ângulo igual a zero para os eixos X e Y. Para isso é necessário realimentar a malha, ou seja, comparar o valor atual do ângulo de cada eixo com o valor de referência (neste caso zero). Se não existir diferença, significa que a plataforma está na posição horizontal (erro nulo) e os atuadores não precisam atuar. Caso contrário, o controlador deve atuar no processo a fim de anular este erro.

O controlador é um modelo matemático que em conjunto com o processo, torna-o controlado de acordo com as especificações desejadas, (mínimo *overshoot*, menor tempo de subido e de acomodação, etc).

A Figura 6.7 mostra uma malha de controle:

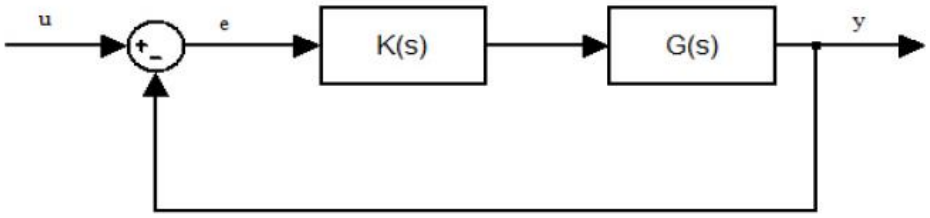



Figura 6-7 - Malha de Controle Básica
Fonte: OGATA, 2003

Sengo G(s) o modelo do processo e K(s) o modelo do controlador, a função de transferência da malha de controle é dada por Equação 6.9.

$$FTMF = \frac{G(s)\,K(s)}{1 + G(s)\,K(s)} \tag{6.9}$$

O controlador PID (proporcional, derivativo e integrativo) é um tipo de controlador muito usado. Muitas vezes, quando se utiliza métodos de projetos de controladores como por exemplo IMC (*Internal Modelo Control*) ou lugar das raízes, o modelo final fica próximo das equações 6.10 e 6.11:

$$K(s) = Kp + \frac{Ki}{s} + Kd\,s \tag{6.10}$$

$$K(s) = \frac{Kd\,s^2 + Kp\,s + Ki}{s} \tag{6.11}$$

Este é um modelo genérico do controlador PID. Todo modelo de controlador que tiver esse padrão matemático, é considerado um controlador PID, sendo que *Kp*, *Ki* e *Kd* são constantes que podem aparecer ou não no modelo.

Basicamente o controlador proporcional minimiza o erro com rapidez, o integrativo zera este erro em regime permanente e o derivativo age antecipando os prováveis erros (OGATA, 2003).

Algumas informações interessantes são mostradas na tabela 6.1 sobre os parâmetros do controlador PID:

Tabela 6.1 - Influência das constantes de um controlador PID

| **Constante** | **Tempo de Subida** | **Overshoot** | **Acomodação** | **Erro estacionário** |
|---|---|---|---|---|
| Kp | Diminui | Aumenta | Peq. Mudança | Diminui |
| Ki | Diminui | Aumenta | Aumenta | Elimina |
| Kd | Peq. Modificação | Diminui | Diminui | Não modifica |

Fonte: OGATA, 2003

Todos os experimentos foram feitos em malha aberta, ou seja, sem realimentação. Para o controlador, a malha será realimentada, como dito anteriormente.

O Matlab fornece ferramentas como a função "*pid( kp, ki, kd )*" para gerarmos uma função de transferência para um controlador PID, bem como a função "*feedback()*" que faz a realimentação da nossa malha. Também foi utilizado o *Simulink*, do Matlab, para montar a malha visualmente, bem como utilizar o controlador PID e estudar os resultados para diferentes sintonias. A ferramenta "*pidtool( sys )*" também foi utilizada para a sintonia e entregando excelentes resultados.

A Figura 6.8 mostra a malha de controle implementada no *Simulink.*

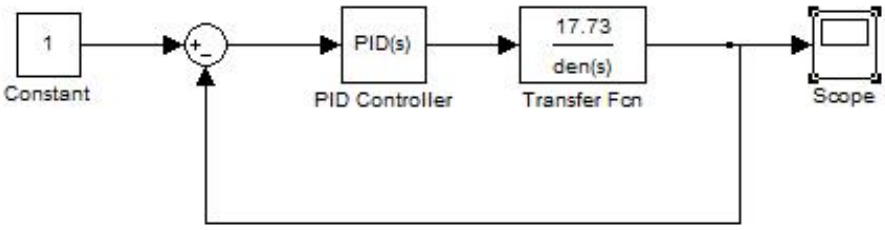



Figura 6-8 - Malha do sistema com controlador no Simulink

O primeiro teste, utilizando Kp=3, Ki=6, Kd=0, gerou o seguinte resultado:

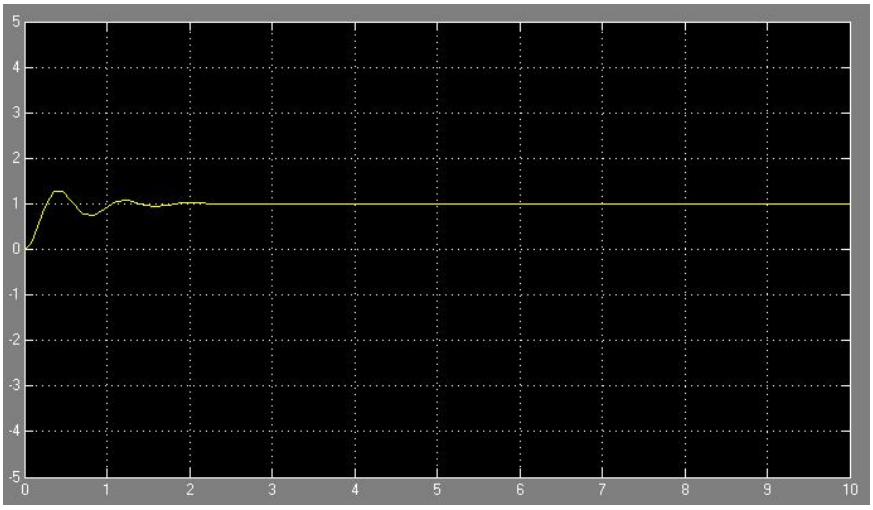



Figura 6-9 - primeiro teste do controlador

Fazendo várias modificações nas constantes do controlador PID, chega-se a diversos resultados, às vezes melhores, às vezes piores. O trabalho de descobrir qual a melhor combinação das constantes é tópico de diversos estudos. Uma sintonia bem minuciosa e fina é algo realmente complexo.

A seção seguinte discute diferentes formas de sintonia para os controladores do tipo PID.

## 6.3. Métodos de Sintonia para o Controlador PID

Com o modelo matemático da plataforma, é possível fazer a sintonia utilizando um dos diferentes métodos existentes ou utilizando o Matlab, através do "*Automatic PID Tunning*", disponível a partir da versão 2010b com o comando "*pidtool( sys )*". Essa ferramenta testa diversas combinações distintas para as constantes do controlador PID, e disponibiliza o melhor resultado de acordo com os parâmetros informados, como *overshoot* e tempo de resposta.

O algoritmo utilizado pelo Matlab por padrão calcula os melhores parâmetros *Kp, Ki* e *Kd* de acordo com a frequência de corte em malha fechada do sistema para uma margem de fase de 60 graus (MATLAB DOCUMENTATION, 2015). Porém é possível alterar essas margens através de dois *slides* disponíveis na ferramenta, e então o software faz novos cálculos para disponibilizar ganhos possíveis para o controlador.

Através do modelo matemático de segunda ordem, *G2*, as constantes do controlador PID foram encontradas inicialmente através do Matlab com o comando "*pidtool( G2 );"*, fixando os parâmetros anteriormente escolhidos (*overshoot* máximo 10% e tempo de acomodação de 1,6s). Os valores calculados pelo software foram: Kp=8,82; Ki=20,51 e Kd=0,95.

Além do uso do Matlab, um outro método de sintonia muito utilizado é o Ziegler-Nichols, pela sua simplicidade e bons resultados. Este método foi desenvolvido por J.G. Ziegler e N. B. Nichols, que se trata de uma metodologia para a sintonia de controladores PID, que determina os valores de Kp, Ti e Td, através da resposta temporal de um processo mediante a um degrau de entrada (OGATA, 2003).

Aplica-se um degrau em malha aberta no processo, e verifica se este não possui sobressalto. Foi considerado para este caso, o modelo matemático encontrado de primeira ordem, por não possuir *overshoot* e se aproximar bem da resposta do processo real. A condição necessária para aplicação deste método é a resposta ao degrau ser do tipo sobreamortecido, apresentando o aspecto de um S, de acordo com a Figura abaixo (OGATA, 2003):

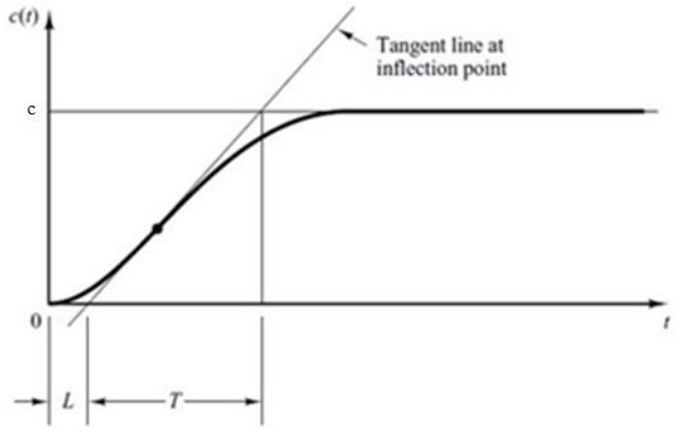



Figura 6-10 - Curva esperada da resposta ao degrau
Fonte: OGATA, 2003

A curva apresenta os parâmetros atraso L e a constante de tempo T, que são definidas pela tangente traçada a partir do ponto de inflexão da curva, onde se determina a intersecção da linha tangente com a reta c(t) = c, onde o sistema estabiliza, e o eixo do tempo t (OGATA, 2003).

A partir dos parâmetros *L*, *T* e *K*, Ziegler e Nichols sugeriram escolher os valores de *Kp*, *Ti* e *Td*, de acordo com a Tabela 6.2, bem como a sua função de transferência, representada pela Equação 6.13.

O parâmetro *K* é definido de acordo com a Equação 2.12 (IBRAHIM, 2002):

$$K = \frac{Y_f - Y_i}{X_f - X_i} \tag{6.12}$$

$$G_c(s) = K_p \left(1 + \frac{1}{T_i s} + T_d s\right) \tag{6.13}$$

Onde $Y_f$ e $Y_i$ são, respectivamente, a saída final e a saída inicial da malha de controle (no caso deste trabalho, este valor é dado em graus), após estabilização do sistema e $X_f$ e $X_i$ representam, respectivamente, a entrada final e a entrada inicial da malha de controle (também dado em graus), aplicadas no atuador para obtenção da curva de resposta do sistema.

Tomando o modelo de primeira ordem, e com auxílio do Matlab e das funções "*grid*" e "*ginput()*", além da ferramenta para traçar uma reta no gráfico, chega-se aos seguintes valores: Yf

= 41,2; Yi = 0; Xf = 45; Xi = 0; L = 0 (o tempo de resposta é praticamente instantâneo); T = 0,26. Calcula-se então facilmente o K = 0,915.

A partir da Tabela 6.2, calcula-se os demais parâmetros do controlador PID:

Tabela 6.2 - Parâmetros de Ziegler-Nichols baseada na resposta ao degrau da planta

| Tipo de controlador | $K_p$ | $T_i$ | $T_d$ |
|---|---|---|---|
| P | $\frac{T}{KL}$ | $\infty$ | 0 |
| PI | $0{,}9\frac{T}{KL}$ | $3{,}3L$ | 0 |
| PID | $1{,}2\frac{T}{KL}$ | $2L$ | $0{,}5L$ |

Fonte: IBRAHIM, 2006.

Porém, como a curva de resposta ao degrau da plataforma não tem a forma um S bem definido, já que o tempo de atraso L é praticamente nulo, fica impossível fazer a sintonia utilizando o método Ziegler-Nichols.

Tomando como base os valores das constantes fornecidos pelo Matlab, foi utilizado então o método de aproximações sucessivas ou tentativa e erro para melhorar ainda mais a resposta do processo real. Este método consiste em alterar as constantes do controlador continuamente avaliando as respostas reais do processo até se chegar a uma dinâmica satisfatória.

Ogata (2013) sugere o seguinte procedimento para este método:

- Eliminar a ação integrativa (Ki=0) e derivativa (Kd=0);
- Aumentar Kp até que o sistema entre em oscilação constante;
- Diminuir Kp pela metade e aumentar Ki até o sistema começar a oscilar constantemente;
- Diminuir Ki em 1/3 e aumentar Kd até o sistema oscilar constantemente;

É importante que a saída do controlador não sature durante estes procedimentos para que não prejudique nenhum equipamento.

Durante os testes, verificou-se que o tempo de amostragem interfere diretamente na resposta do sistema. Quanto maior o tempo de amostragem, maior devem ser as constantes, porém o sistema responde mais lentamente. Quanto menor o tempo de amostragem, menor devem ser as constantes, e mais rápido o sistema responde. Isso se deve à discretização do modelo contínuo do controlador PID, para a implementação no Arduino. Este tópico será discutido posteriormente na seção 6.4.

O tempo de amostragem inicial foi de 300ms. Ao diminuir este tempo pela metade, as constantes também foram diminuídas pela metade. Assim a resposta do sistema ficou mais rápida.

O limite mínimo para o tempo de amostragem está diretamente ligado ao tempo de resposta e de acomodação dos sensores e dos atuadores. Este tempo limite foi encontrado na prática, sendo de 30ms. Logo, como é 10 vezes menor que o tempo de amostragem inicial, as constantes também foram divididas por 10, e então realizados novos testes. Chegou-se às seguintes constantes para o controlador, deixando o sistema bem controlado e com uma resposta bem rápida (utilizando-se 30ms como tempo de amostragem):

- Kp=0,2;
- Ki=0,05;
- Kd=0,01;

## 6.4. Discretização e Implementação do Controlador PID no Arduino

Para a implementação do controlador no Arduino, é necessário discretizar a função de transferência do controlador e trabalhar com a equação à diferença, para que possa ser escrita em forma de um programa na linguagem C. O controlador PID, por ser muito utilizado, tem sua discretização já conhecida, de acordo com a Equação 6.15e sua Equação no domínio do tempo, de acordo com a Equação 6.14 (DORF; BISHOP, 2005):

$$u(t) = K_p \left( e(t) + \frac{1}{T_i} \int_0^t e(t)\, dt + T_d \frac{d\, e(t)}{dt} \right) \tag{6.14}$$

As constantes *Kp*, *Ki* e *Kd* são conhecidas, o erro atual, *e(t)*, também é conhecido (diferença entre o *setpoint* e o valor atual do sensor). A operação de integração é basicamente um somatório, como mencionado anteriormente para os giroscópios, e a operação de derivação é, basicamente, o cálculo da tangente do ângulo formado entre o valor do último erro e do atual observado, ou seja, é a taxa da variação do erro, que é calculado com uma simples divisão, utilizando o tempo de amostragem *dt*. Logo, o modelo discreto de um controlador PID, em equações a diferença, é dado por:

$$K(n) = Kp\, e(n) + Kd\, \frac{[e(n) - e(n-1)]}{dt} + Ki \sum e(n)\, dt \tag{6.15}$$

Com esse modelo discreto, fica simples a implementação em qualquer linguagem de programação. Neste trabalho a função foi implementada no Arduino.

É importante que se implemente algumas saturações, como por exemplo não enviar sinais para os servomotores se posicionarem em posições maiores que 180 graus ou menores que zero graus. Também foi considerada uma tolerância de 2,5 graus como erro, devido à folga existente nos servomotores, ou seja, só há atuação do controlador quando houver um erro maior que 2,5 graus em algum dos eixos.

Para validação do modelo discreto implementado no Arduino, bem como uma melhor compreensão de cada termo do controlador PID, foram feitos alguns movimentos aleatórios na plataforma e analisados os termos proporcional, derivativos e integrativos individualmente.

As Figuras 6.11, 6.12 e 6.13 mostram os resultados práticos de cada controlador.

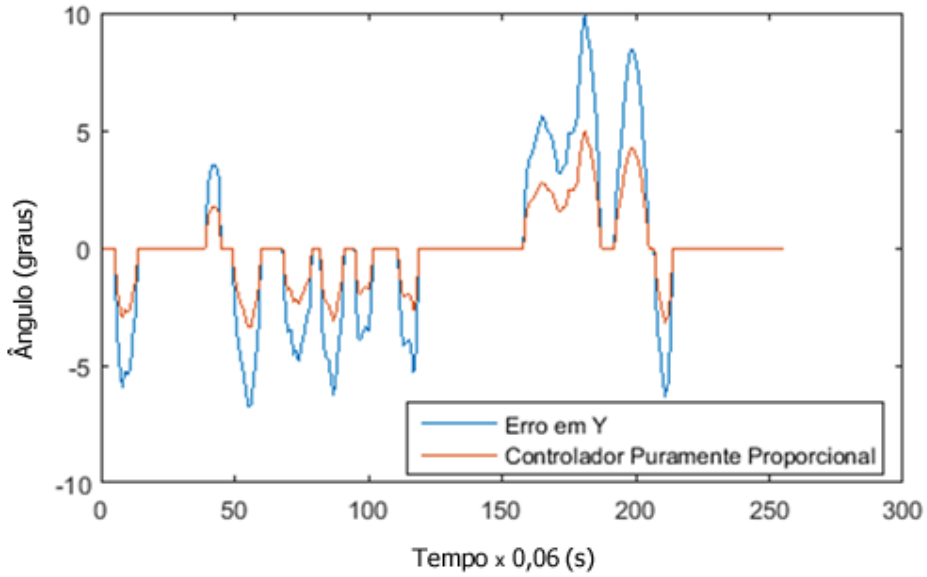



Figura 6-11 - Controlador Puramente Proporcional

Com o *Kp* = 0,5 o sinal de saída do controlador, puramente proporcional, é simplesmente o sinal de erro multiplicado por essa constante, neste caso, a metade do sinal de erro, o que foi mostrado claramente no gráfico, validando a implementação desse controlador.

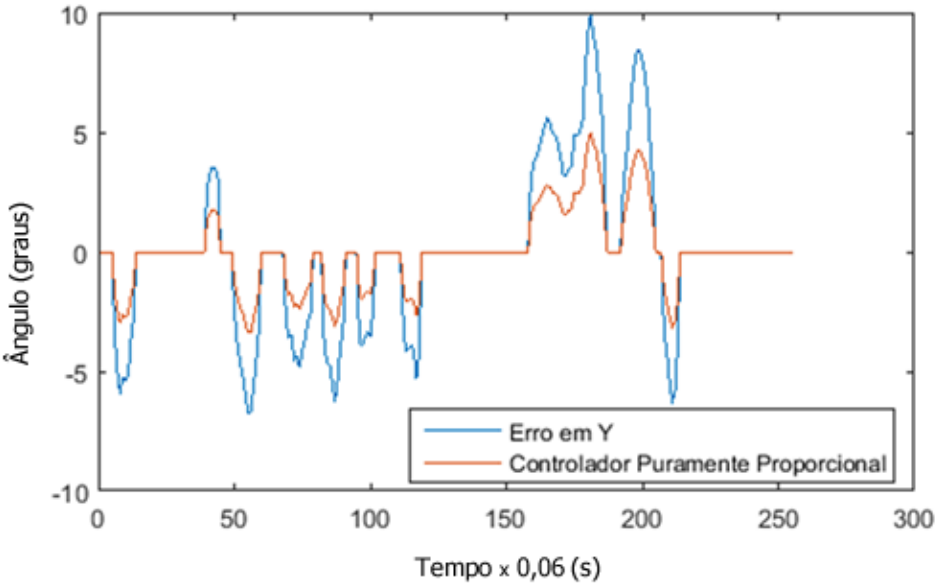



Figura 6-12 – Controlador Puramente Integrativo

A Figura 6.12 mostra o sinal de erro e a saída do integrador. O termo integrador do controlador PID faz um somatório do erro enquanto ele existir (um histórico de como o erro está se comportando), multiplicado pela constante *Ki* (neste caso 0,05). Logo é visto que o integrador foi bem implementado, pois o seu valor aumenta rapidamente quanto o erro aumenta, e aumenta mais lentamente quando o erro diminui. O sinal de saída do controlador puramente integrativo é zerado sempre que o erro também é zero, de forma que não sature o atuador bem como não envie sinal quando não há erro.

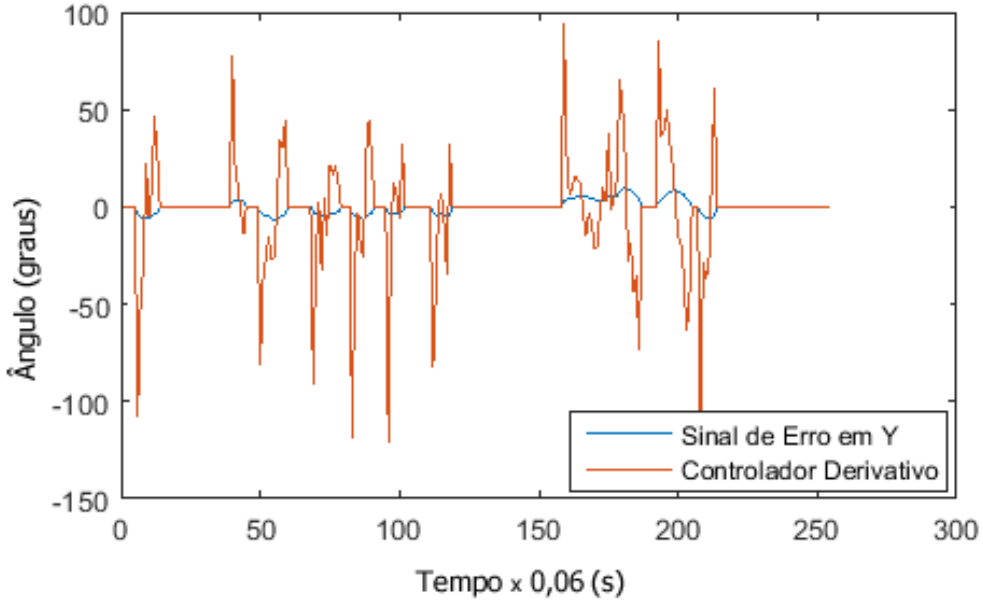



Figura 6-13 - Controlador Puramente Derivativo

O sinal de saída do controlador puramente derivativo, visto na Figura 6.13 também validou o modelo discreto implementado no Arduino. O termo derivativo trabalha em relação à taxa de variação do erro, ou seja, se ele aumenta rapidamente, o termo derivativo age com mais intensidade, e vice-versa. Também foi multiplicado por uma constante (*Kd* neste caso foi de 0,01). É importante verificar que a derivada é negativa quando o erro está diminuído, positiva quando está aumentando e nula quando o erro é zero, o que é visto no gráfico. A soma de todos os controladores independentes gera o sinal de saída final do controlador PID.

Depois de implementado o controle, com as constantes do controlador mencionadas anteriormente, e tomando um tempo de amostragem limite de (30ms), foram feitos vários movimentos manuais e aleatórios na plataforma, segurando na manopla, algumas vezes lentamente, outras rapidamente, e o resultado obtido foi o mostrado nas Figuras 6.14 e 6.15.

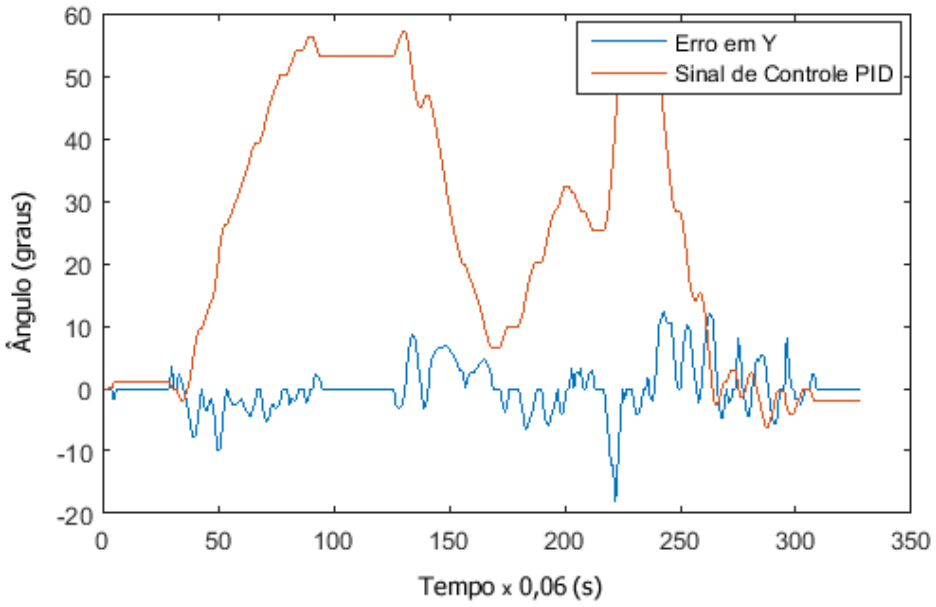



Figura 6-14 – Sinal de controle e de erro no eixo Y

Houve erros grandes, maiores que 10 graus. Um grave problema mecânico na plataforma, como já foi citado, é a folga que existe no eixo Y de mais de 3 graus, o que, infelizmente, dificulta ou torna impossível o controle com erros menores que este, ou seja, qualquer erro menor que 3 graus, está diretamente ligado à folga existente no servo. Porém, apesar de alguns erros grandes que ocorreram, devido a alguma mudança brusca de orientação com a plataforma, este foi minimizado rapidamente. Outro fator interessante, que ficou claro na análise dos dados, é que o controlador estabiliza o seu valor de saída quando o erro é nulo, ou seja, como não há erro, o controlador não atua, o que é visível diversas vezes no gráfico.

O gráfico do sinal de referência é, neste caso, claramente uma aproximação do movimento externo que a plataforma sofreu, porém invertido, ou seja, o controlador tenta prever o movimento que a plataforma está sofrendo para mantê-la na posição horizontal.

A Figura 6.15 mostra o que aconteceu com o eixo X, simultaneamente ao movimento já analisado com o eixo Y:

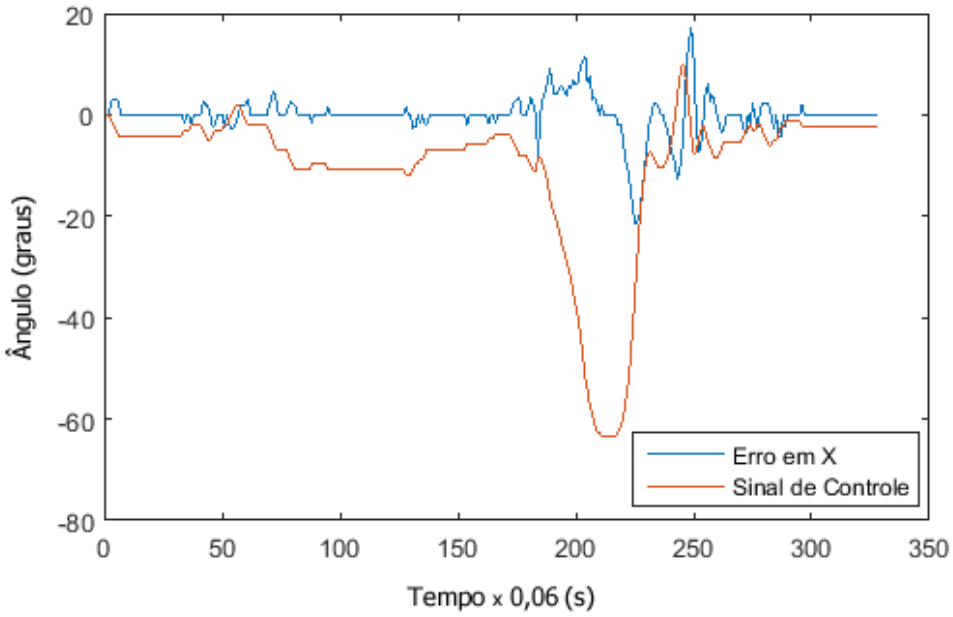



Figura 6-15 - Sinal de controle e de erro no eixo X

É visto que o sinal de referência tende a ser simétrico ao sinal de erro, porém, na análise do que ocorreu no eixo X, existe um momento em que essa simetria foge completamente, o que mostra que houve um movimento externo muito brusco e o controlador atuou imediatamente e com intensidade, aumentando o sinal de saída para minimizar ao máximo o erro na plataforma durante essa dinâmica, o que realmente aconteceu.

Não foi possível fazer uma comparação fiel entre a resposta do controlador gerada pelo modelo matemático e a resposta real, pois o movimento aplicado à plataforma foi feito manualmente, não sendo um movimento conhecido nem medido. Seria necessário aplicar o mesmo movimento ao modelo para fazer uma comparação fiel, e para isso, o movimento aplicado deveria ser padrão (utilizando, por exemplo, um robô externo ou uma plataforma com movimento conhecido). Pode-se utilizar outro sensor instalado fixamente na manopla, e medir essa movimentação externa aplicada.

Outra movimentação aleatória foi aplicada para reforçar os resultados. As Figuras 6.16 e 6.17 mostram os resultados.

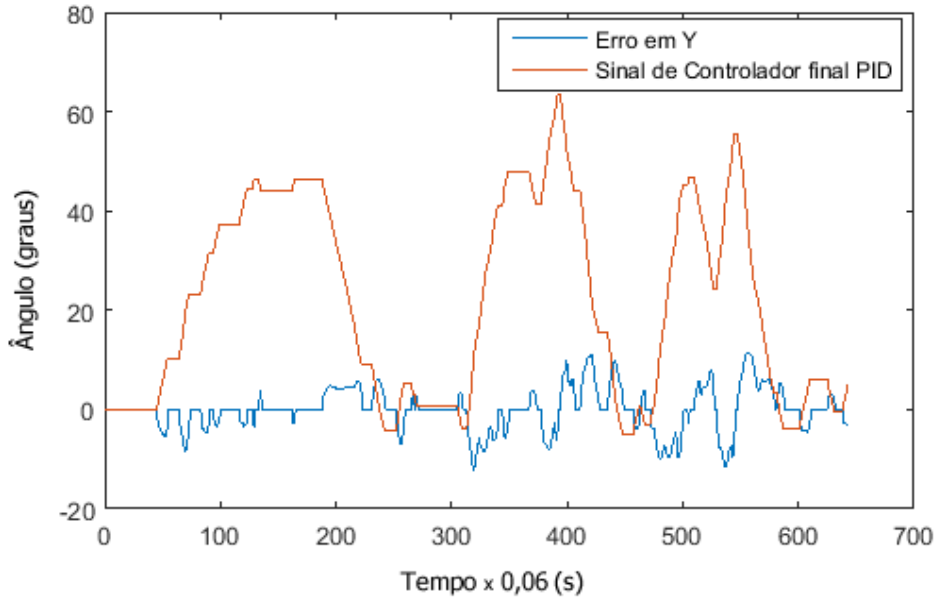



Figura 6-16 - Controlador PID Final, eixo Y

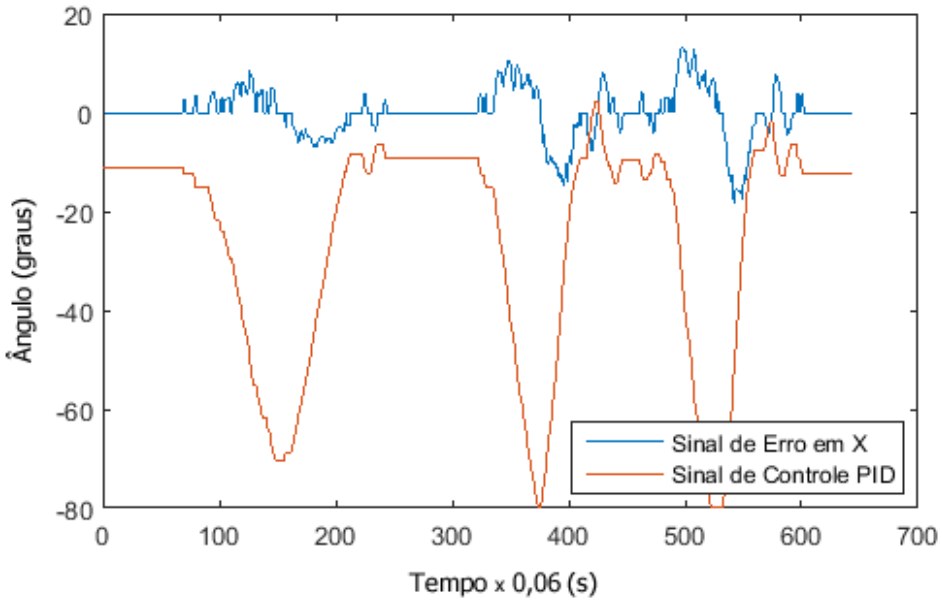



Figura 6-17 - Controlador PID Final, eixo X

Os resultados foram bem satisfatórios. A resposta ficou rápida, os erros dentro do aceitável (principalmente levando-se em consideração a folga de 3 graus), tornando os ganhos *Kp=0,2; Ki=0,05; Kd=0,01* os definitivos para o controlador deste trabalho.

# 7. CONCLUSÕES

Este trabalho foi uma busca incessante para tentar, de forma sintetizada e prática, estudar e aplicar conceitos mais relevantes envolvidas no curso de Engenharia de Controle e Automação, em um só trabalho. 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 até mesmo alguns conceitos da mecânica. A ideia inicial era, através dessa síntese, elaborar um material completo sobre os conceitos e as práticas aplicadas, para futuras consultas e revisões, o que acredito ter sido alcançado.

A aquisição dos dados, bem como a filtragem e o tratamento dos mesmos se mostrou importantíssimo na prática. A comunicação entre o sensor MPU-6050 através do protocolo I2C em conjunto com os servomotores, que são atuadores que trabalham através de motores e geram muito ruído, só foi possível através de correções físicas feitas na montagem eletrônica, como a inserção de um *ferrite beat*, resistores de polarização (*pull-up)* e o capacitor entre a alimentação e o terra (*gnd*). Antes disso, a comunicação parava sem nenhuma explicação, ou chegavam valores absurdos. Os ruídos dos motores, bem como o sinal de alta frequência do PWM bem próximos às linhas de comunicação do I2C, claramente “embaralhavam” os dados dessa comunicação, o que a tornava incompreensível. Esse problema realmente acontece na prática, por exemplo, no chão de fábrica, onde se encontram diversos tipos atuadores que geram ruídos e que podem interferir nas diversas linhas de comunicação existentes. Por isso, realmente devem-se usar cabos blindados ou, dependendo da intensidade dos ruídos, outros meios de comunicação como comunicação sem fio ou cabos de *fibra optica*, que não sofrem interferências desse tipo e são muito usados em fábricas. Este tipo de problema é de difícil identificação e pode exigir dias para serem detectados, sendo que no mercado, pode custar muito caro. Logo, antes do estudo dos modelos de um processo e da aplicação de um sistema de controle, é necessária uma intensa dedicação à instrumentação do processo de aquisição dos dados. Essa tarefa é mais importante que o próprio controle, pois se ela não disponibilizar boas medições, precisas e exatas como desejado, o controle nunca funcionará.

Os conceitos e as diversas aplicações utilizando os acelerômetros e os giroscópios foi desafiador. Entender o funcionamento de cada um, as diferentes aplicações individuais deles bem como trabalhar com os dois em conjunto, através da utilização de filtros, ora bem complexos como o filtro de Kalman, ora bem simples como o filtro Complementar, e alcançar um resultado

importante (o valor da posição angular atual) com precisão e pouco ruído, foi um complemento intelectual muito gratificante e valioso a tudo que já havia sido estudado durante o curso.

Outro conceito muito importante que foi aprofundado e colocado em prática, foi a de estudar e modelar um processo real, através de ferramentas como o Matlab, sem conhecimento das mais diversas variáveis físicas do processo, e comparar com a realidade e comprovar que se aproximam muito. A partir daí, foi possível criar um modelo de controle, aplicá-lo e entender melhor como cada parâmetro do controlador, neste caso as constantes do controlador PID, afetam na prática o processo, além de trabalhar na prática com diferentes métodos de sintonia. Ver na prática o comportamento da plataforma para cada alteração feita nos parâmetros do controlador tornou claro o entendimento dos mesmos.

No controle discreto, aplicado utilizando o Arduino, o tempo de amostragem dos dados bem como o tempo de espera para que os servomotores finalizassem o seu movimento (tempo de atuação dos atuadores), foram cruciais para uma melhoria dos resultados e interferem diretamente também nas constantes do controlador discreto. Foi também visto a importância da saturação dos atuadores bem como a reinicialização do fator integrador quando o erro se torna nulo.

Como proposta para continuidade deste projeto, sugere-se que sejam trocados os servos por outros mais precisos e preferencialmente com engrenagens de metal. Os servos podem ser substituídos ainda por motores *brushless*, que apresentam uma resposta muito rápida, uma precisão de centésimos de grau, porém baixo torque. Sugere-se também 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 etc. Pode-se também implementar mais um sensor MPU-6050 na manopla, de forma a identificar exatamente qual a posição desta durante qualquer movimentação, podendo melhorar ainda mais o controle e fazer uma comparação precisa entre o modelo matemático e o processo real.

## 8. REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO. ***What is Arduino.*** Disponível em <HTTPS://www.arduino.cc/en/Guide/Introduction>. Acesso em 17 Dez. 2015.

INVENSENSE INC, DATASHEET MPU6050, Revision 3.4, 2013. Disponível em <HTTP://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf>. Acesso em 22 Dez. 2015.

DORF, R.C.; BISHOP, R. H. ***Modern Control Systems*****.** 10ª Ed, 2005. New York: Prentice Hall, 2005. 881 p.

ENGLEANDROALVES. Controlando Servo Motores com PIC 16F877A. Disponível em <HTTPS://engleandroalves.wordpress.com/2012/07/15/controlando-servo-motores-com-pic16f877a/>. Acesso em 22 Dez. 2015.

EXAME. Google inventa colher para ajudar pessoas com Parkinson. Disponível em <HTTP://exame.abril.com.br/tecnologia/noticias/google-inventa-colher-para-ajudar-pessoas-com-parkinson>. Acesso em: 27 de Dez. 2015

FAMBRINI, P. F. ***Solução por Software para Implementar PWM em qualquer Microcontrolador PIC. [S.d.]***. Universidade Anhanguera, [S.d.].

GEEK MOM PROJECTS. Gyroscopes and Accelerometers on a Chip. Disponível em <HTTP://www.geekmomprojects.com/gyroscopes-and-accelerometers-on-a-chip/>. Acesso em 13 Dez. 2015.

GREWAL, M. S., ANDREWS, A. P. ***Kalman Filtering: theory and practice using matlab.*** 3. Ed.John Wiley & Sons, 2008. 575 p.

HARVEY, A. C. ***Forecast, structural time series models and Kalman filter.*** Cambridge UniversityPress. 2001, 554 p.

IBRAHIM, D. **Microcontroller Based Applied Digital Control.** Chinchester: John Wiley & Sons, Ltd, 2006. 313p.

IBRAHIM, D. **Microcontroller Based Temperature Monitoring and Control**. 1ª Ed. Newnes, 2002.

LAUSZUS, K. Kalman filter library for any microcontroller that supports float math. Disponível em <HTTPS://github.com/TKJElectronics/KalmanFilter>. Acesso em 02 Jan. 2016.

MESSNER, B., TILBURY, D. Control Tutorials for Matlab & Simulink. Disponível em <HTTP://ctms.engin.umich.edu/CTMS/index.php?aux=Home>. Acesso em 19 Nov. 2015.

MATLAB DOCUMENTATION. PidTunner. Disponível em <HTTP://www.mathworks.com/help/control/ref/pidtuner.html>. Acesso em 02 Mar. 2015.

MECHATTRONICS ME102B FALL 2015. ME102 Lab 4: RC Servo. Disponível em <HTTP://courses.me.berkeley.edu/ME102B/lab4.html>. Acesso em 25 Dez. 2015.

NXP SEMICONDUCTORS. I2C-bus specification and user manual. Rev. 6, abr. 2014. Disponível em <HTTP://www.nxp.com/documents/user_manual/UM10204.pdf>. Acesso em: 03 Jan. 2016.

OGATA, K. **Engenharia de Controle Moderno.** 5ª Ed. São Paulo: Prentice Hall, 2003.

PAULA, F. O. Sensores IMU – Uma Abordagem Completa. Disponível em <HTTP://www.decom.ufop.br/imobilis/sensores-imu-uma-abordagem-completa-parte-1/>. Acesso em: 23 Dez. 2015.

RAL TECHNOLOGY. Arduino Nano. Disponível em <HTTP://www.raltech.com.br/?product=arduino-nano>. Acesso em 22 Nov. 2015.

ROWBERG, J**.** I2Cdev Library Collection. Disponível em <HTTPS://github.com/jrowberg/i2cdevlib>. Acesso em 15 Out. 2016.

ROBU.IN. MPU-6050 Gyro Sensor 2 + Accelerometer. Disponível em <HTTP://robu.in/shop/mpu-6050-gyro-sensor-2-accelerometer/>. Acesso em 28 Dez. 2015.

SERRANO, D. E., *Design and Analysis of MEMS Gyroscope***.** Georgia Institute of Tecnology. **IEEE Sensors**. 2013.

TEXAS INSTRUMENTS. Accelerometers and How They Works. Disponível em <HTTP://www2.usfirst.org/2005comp/Manuals/Acceler1.pdf>. Acesso em: 30 Dez. 2015.

# 9. ANEXOS

## 9.1. Aquisição de Dados Puros, RAW VALUES

```
//Carrega a biblioteca Wire
#include <Wire.h>

//Endereco I2C do MPU6050
const int MPU=0x68;

//Variaveis para armazenar valores dos sensores
float AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;

// sensibilidade do acelerômetro de acordo com o datasheet
// para converter em força g as leituras do acelerômetro
float Acc = 16384;

// sensibilidade do giroscópio de acordo com o datasheet
// para converter em DPS (graus por segundo) leituras do giroscópio
float Gy = 131;

void setup()
{
  Serial.begin(9600);

  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);

  //Inicializa o MPU-6050
  Wire.write(0);
  Wire.endTransmission(true);

  // informacoes
  Serial.println("[ACC] forca g \t [GY]: graus por segundo (DPS) \t [TEMP] graus");
  Serial.println();

  //Envia valor X do acelerometro para a serial e o LCD
  Serial.println(" AcX \t AcY \t AcZ \t\t GyX \t GyY \t GyZ \t\t TEMP");

}

void loop()
{
```

```
Wire.beginTransmission(MPU);
Wire.write(0x3B);            // starting with register 0x3B (ACCEL_XOUT_H)
Wire.endTransmission(false);

//Solicita os dados do sensor
Wire.requestFrom(MPU,14,true);

//Armazena o valor dos sensores nas variaveis correspondentes
AcX = Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
AcY = Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
AcZ = Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
Tmp = Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H)   & 0x42 (TEMP_OUT_L)
// por algum motivo estava trocado o GyX com GyY
GyY = Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H)  & 0x44 (GYRO_XOUT_L)
GyX = Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H)  & 0x46 (GYRO_YOUT_L)

GyZ = Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H)  & 0x48 (GYRO_ZOUT_L)

AcX = AcX/Acc;    GyX = GyX/Gy;
AcY = AcY/Acc;    GyY = GyY/Gy;
AcZ = AcZ/Acc;    GyZ = GyZ/Gy;


// para a plataforma, como o sensor foi colocado de cabeça pra baixo
// alterei os sinais de algumas variáveis para considerar a seguinte orientação
// X positivo: pra frente
// Y positivo: para a direita
// Z positivo: para a direita
// Para ver os valores sem essa correção, basta comentar abaixo

//*//////////////////////
AcX = -AcX;   GyX = -GyX;
AcY = -AcY;
AcZ = -AcZ;
//*//////////////////////


Serial.print(AcX); Serial.print("\t");
Serial.print(AcY); Serial.print("\t");
Serial.print(AcZ); Serial.print("\t\t");

Serial.print(GyX); Serial.print("\t");
Serial.print(GyY); Serial.print("\t");
Serial.print(GyZ); Serial.print("\t\t");
Serial.print(Tmp/340.00+36.53); Serial.print("\t");
Serial.println();
```

```
  //Aguarda 300 ms e reinicia o processo
  delay(100);
}
```

### 9.2. Cálculo dos Ângulos Através dos Acelerômetros e Giroscópios

```
//Carrega a biblioteca Wire
#include <Wire.h>

//Endereco I2C do MPU6050
const int MPU=0x68;

//Variaveis para armazenar valores dos sensores
float AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;

// variáveis para guardar os ângulos
float AngX, AngY, AngZ;

// sensibilidade do acelerômetro de acordo com o datasheet
// para converter em força g as leituras do acelerômetro
float Acc = 16384;

// sensibilidade do giroscópio de acordo com o datasheet
// para converter em DPS (graus por segundo) leituras do giroscópio
float Gy = 131;

// constante para conversão dos angulos dados em radianos
// pela função atan2() para graus
float const_convert = 180/M_PI;

void setup()
{
  Serial.begin(9600);

  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);

  //Inicializa o MPU-6050
  Wire.write(0);
  Wire.endTransmission(true);
```

```
  // informacoes
  Serial.println("[ACC] forca g \t [GY]: graus por segundo (DPS) \t [TEMP] graus");
  Serial.println();

  //Envia valor X do acelerometro para a serial e o LCD
  Serial.println(" AcX \t AcY \t AcZ \t\t GyX \t GyY \t GyZ \t\t TEMP");

}

void loop()
{

  Wire.beginTransmission(MPU);
  Wire.write(0x3B);            // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);

  //Solicita os dados do sensor
  Wire.requestFrom(MPU,14,true);

  //Armazena o valor dos sensores nas variaveis correspondentes
  AcX = Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY = Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ = Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Tmp = Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H)   & 0x42 (TEMP_OUT_L)
  // por algum motivo estava trocado o GyX com GyY
  GyY = Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H)  & 0x44 (GYRO_XOUT_L)
  GyX = Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H)  & 0x46 (GYRO_YOUT_L)

  GyZ = Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H)  & 0x48 (GYRO_ZOUT_L)

  AcX = AcX/Acc;    GyX = GyX/Gy;
  AcY = AcY/Acc;    GyY = GyY/Gy;
  AcZ = AcZ/Acc;    GyZ = GyZ/Gy;


  // para a plataforma, como o sensor foi colocado de cabeça pra baixo
  // alterei os sinais de algumas variáveis para considerar a seguinte orientação
  // X positivo: pra frente
  // Y positivo: para a direita
  // Z positivo: para a direita
  // Para ver os valores sem essa correção, basta comentar abaixo

  //*////////////////////////
  AcX = -AcX;   GyX = -GyX;
  AcY = -AcY;
  AcZ = -AcZ;
  //*////////////////////////
```

```
  AngX = atan2( AcX, sqrt( AcY*AcY + AcZ*AcZ ) ) * const_convert;
  AngY = atan2( AcY, sqrt( AcX*AcX + AcZ*AcZ ) ) * const_convert;
  AngZ = atan2( sqrt( AcX*AcX + AcY*AcY ), AcZ ) * const_convert;

  Serial.print(AngX); Serial.print("\t");
  Serial.print(AngY); Serial.print("\t");
  Serial.print(AngZ); Serial.print("\t\t");

  Serial.print(GyX); Serial.print("\t");
  Serial.print(GyY); Serial.print("\t");
  Serial.print(GyZ); Serial.print("\t\t");
  Serial.print(Tmp/340.00+36.53); Serial.print("\t");
  Serial.println();

  //Aguarda 300 ms e reinicia o processo
  delay(100);
}
```

### 9.3. Firmware Final do Trabalho com Controlador

```
//////////////////////////////////////////
//
// Trabalho de Conclusão de Curso - TCC
// Aluno Vinícius Nunes Lage
// Universidade Federal de Ouro Preto
// Engenharia de Controle e Automação
//
//////////////////////////////////////////


// Carrega a biblioteca Wire
#include <Wire.h>


// Carrega bibliotea para os Servos
#include <Servo.h>
Servo servoX, servoY;


// Endereco I2C do MPU6050
const int MPU=0x68;

// Variaveis para armazenar valores dos sensores
float AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;

// colocando a plataforma na horizontal
// observa-se o desvio do gyroscópio ao longo do tempo
float gyBiasX=-0.018, gyBiasY=-0.0075, gyBiasZ=-0.04;

// valores iniciais dos giroscópios
// méidias calculadas
float gyIniX=42.02, gyIniY=-137.88, gyIniZ=-72.80;

// variáveis para guardar os ângulos
float AngX, AngY, AngZ;
float AnGyX=0, AnGyY=0, AnGyZ=0;
float AngCompX=0, AngCompY=0, AngCompZ=0;
float dt=0, dt_ant=0;

// variáveis para guardar o deslocamento dos gyros
float DeslGyX=0, DeslGyY=0, DeslGyZ=0;

// sensibilidade do acelerômetro de acordo com o datasheet
```

```
// para converter em força g as leituras do acelerômetro
float Acc = 16384;

// sensibilidade do giroscópio de acordo com o datasheet
// para converter em DPS (graus por segundo) leituras do giroscópio
float Gy = 131;

// constante para conversão dos angulos dados em radianos
// pela função atan2() para graus
float const_convert = 180/M_PI;

// parâmtreo alpha para o filtro Complementary
float alpha = 0.93;

// flag para iniciar o filtro complementar
// com o valor dos angulos dos acelerometros
// só na primeira vez, quando liga o arduino
bool flagComp=0;

// flag para o degrau com os servos
unsigned long int contServo=0;
bool flagServo=0;


////////////////////////// CONTROLE PID

float X,Y, X_last=0, Y_last=0;
float erroX, erroY, erroX_last, erroY_last;
float SPX = 0;
float SPY = 0;
float PVX = 90;
float PVY = 90;

float kiX=0, kiY=0;
float dtCont=0, dtCont_ant=0;

unsigned long int tempControl=0;

//////////////////////


void setup()
{
  Serial.begin(115200);

  Wire.begin();
  Wire.beginTransmission(MPU);
```

```
  Wire.write(0x6B);

  //Inicializa o MPU-6050
  Wire.write(0);
  Wire.endTransmission(true);

  // inicia os servos
  // o servo Y, devido à montagem mecânica
  // ficou cou um erro de 6 graus, que foi compensado
  // no sofwtare, por isso o valor de 96 graus
  servoX.attach(5);
  servoY.attach(6);

  servoX.write(90);
  servoY.write(96);

  /*/ informacoes para análise
  Serial.println("[ACC] forca g \t [GY]: graus por segundo (DPS) \t [TEMP] graus");
  Serial.println();
  */

  // informacoes para controle
  Serial.println("X\tY\tErrX\tErrY\tOutX\tOutY\tIntX\tDerX\tIntY\tDerY\tdt");

  //Envia valor X do acelerometro para a serial e o LCD
  //Serial.println(" AcX \t AcY \t AcZ \t\t GyX \t GyY \t GyZ \t\t CompX \t CompY \t CompZ \t\t
TEMP");

}

void loop()
{

  Wire.beginTransmission(MPU);
  Wire.write(0x3B);
  Wire.endTransmission(false);

  //Solicita os dados do sensor
  Wire.requestFrom(MPU,14,true);

  //Armazena o valor dos sensores nas variaveis correspondentes
  AcX = Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY = Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ = Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Tmp = Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H)   & 0x42 (TEMP_OUT_L)

  // por algum motivo estava trocado o GyX com GyY
```

```
GyY = Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H)  & 0x44 (GYRO_XOUT_L)
GyX = Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H)  & 0x46 (GYRO_YOUT_L)

GyZ = Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H)  & 0x48 (GYRO_ZOUT_L)


// zera giroscópios
GyX -= gyIniY;
GyY -= gyIniX;
GyZ -= gyIniZ;

// passa para acelerometro para força g
// w giroscpoio para dps (graus por segundo)
AcX = AcX/Acc;    GyX = GyX/Gy;
AcY = AcY/Acc;    GyY = GyY/Gy;
AcZ = AcZ/Acc;    GyZ = GyZ/Gy;


// para a plataforma, como o sensor foi colocado de cabeça pra baixo
// alterei os sinais de algumas variáveis para considerar a seguinte orientação
// X positivo: pra frente
// Y positivo: para a direita
// Z positivo: para a direita
// Para ver os valores sem essa correção, basta comentar abaixo

//*////////////////////////
AcX = -AcX;   GyX = -GyX;
AcY = -AcY;
AcZ = -AcZ;
//*////////////////////////


// cálculo do ângulo a partir dos acelerômetros
// utilizando trigonometria simples

AngX = atan2( AcX, sqrt( AcY*AcY + AcZ*AcZ ) ) * const_convert;
AngY = atan2( AcY, sqrt( AcX*AcX + AcZ*AcZ ) ) * const_convert;
AngZ = atan2( sqrt( AcX*AcX + AcY*AcY ), AcZ ) * const_convert;


// calcula dt, dado em microsegundos e passa para segunndos
// já que o giroscopio entrega em graus por segundo
dt     = (micros() - dt_ant)/1000000;
dt_ant = micros();

// cálculo do ângulo utilizando os giroscópio
// faz-se uma integração
```

```
// retirando o desvio observado ao longo do tempo
// guarda o deslocamento para usar no filtro complementary

DeslGyX = (GyX-gyBiasX)*dt;   AnGyX += DeslGyX;
DeslGyY = (GyY-gyBiasY)*dt;   AnGyY += DeslGyY;
DeslGyZ = (GyZ-gyBiasZ)*dt;   AnGyZ += DeslGyZ;


//// Filtro Complementary
// utiliza 93% (alpha) do valor do gyro (apenas deslocamento)
// calculado a partir do último valor filtrado
// e 7% do valor do acelerômetro
// o angulo inicial é o do acelerômetro

if(!flagComp)
{
  AngCompX = AngX;
  AngCompY = AngY;
  AngCompZ = AngZ;
  flagComp = 1;
}

AngCompX = alpha*( AngCompX+DeslGyX ) + (1-alpha)*(AngX);
AngCompY = alpha*( AngCompY+DeslGyY ) + (1-alpha)*(AngY);
AngCompZ = alpha*( AngCompZ+DeslGyZ ) + (1-alpha)*(AngZ);



/****** EXIBE OS DADOS
 *  quando não há controle


Serial.print(AngX); Serial.print("\t");
Serial.print(AngY); Serial.print("\t");
Serial.print(AngZ); Serial.print("\t\t");

Serial.print(DeslGyX); Serial.print("\t");
Serial.print(DeslGyY); Serial.print("\t");
Serial.print(DeslGyZ); Serial.print("\t\t");

Serial.print(AngCompX); Serial.print("\t");
Serial.print(AngCompY); Serial.print("\t");
Serial.print(AngCompZ); Serial.print("\t\t");

Serial.print(Tmp/340.00+36.53); Serial.print("\t");
Serial.print(dt,4); Serial.print("\t");
Serial.println();
```

```
  */


  /*****
  // degrau para análise do comportamento
  // da plataforma

  if(!flagServo && contServo++ < 100)
    delay(1);

  else if(!flagServo)
  {
    flagServo=1;
    servoY.write(51);
    //servoY.write(96);
  }

  */


  // o controlador deve atuar a cada 30ms
  // que é o tempo médio de atuação dos servos

  if( (millis()-tempControl) > 30)
  {
    tempControl = millis();
    X = AngCompX;  Y = AngCompY;
    pid();
  }

}



void pid()
{

  dtCont     = (micros() - dtCont_ant)/1000000;
  dtCont_ant = micros();

  float kdX=0, kdY=0;

  float kp = 0.2;
  float ki = 0.05;
  float kd = 0.01;

  // erro minimo para atuar
```

```
float sens = 2.5;

float SX=0, SY=0;


////////////////////////////// ERRO COM REALIMENTAÇÃO NEGATIVA - FEEDBACK
erroX_last = erroX;
erroY_last = erroY;

erroX = SPX + X;
erroY = SPY + Y;


// sensibilidade ou margem de erro aceitável: 2 graus
if     ( erroX > 0 && erroX <= sens )  erroX = 0;
else if( erroX < 0 && erroX >= -sens ) erroX = 0;

if     ( erroY > 0 && erroY <= sens )  erroY = 0;
else if( erroY < 0 && erroY >= -sens ) erroY = 0;



////////////////////////////// TERMO INTEGRATIVO
// reinicialização do integrador
// após um certo período do tempo com erro nulo
// caso exista erro, o integrador deve entrar em ação

if( erroX == 0  )  kiX  = 0;
else               kiX += erroX*dtCont;

if( erroY == 0  )  kiY  = 0;
else               kiY += erroY*dtCont;



////////////////////////////// TERMO DERIVATIVO
if(erroX != 0) kdX = (erroX-erroX_last)/dtCont;
if(erroY != 0) kdY = (erroY-erroY_last)/dtCont;


////////////////////////////// SAÍDA DO CONTROLADOR PID
SX = kp*erroX + ki*kiX + kd*kdX;
SY = kp*erroY + ki*kiY + kd*kdY;


////////////////////////////////////////// SATURAÇÃO MÍNIMA
// os servos se movimentam no mínimo 1 grau
```

```
  // é a precisão dos atuadores

  if     ( erroX > 0 && SX < 1 )  SX = 1;
  else if( erroX < 0 && SX > -1 ) SX = -1;

  if     ( erroY > 0 && SY < 1 )  SY = 1;
  else if( erroY < 0 && SY > -1 ) SY = -1;


  ////////////////////////// CONTROLE
  PVX = PVX - SX;
  PVY = PVY - SY;


  // saturação do atuador
  // não deixa passar destes valores
  if( PVX > 170 ) PVX = 170;
  if( PVX < 10 )  PVX = 10;

  if( PVY > 170 ) PVY = 170;
  if( PVY < 10 )  PVY = 10;


  ////////////////////////////////////// PRINT ERROS
  Serial.println();
  Serial.print(X);     Serial.print("\t");
  Serial.print(Y);     Serial.print("\t");

  Serial.print(erroX); Serial.print("\t");
  Serial.print(erroY); Serial.print("\t");

  Serial.print(PVX-90);   Serial.print("\t");
  Serial.print(PVY-90);   Serial.print("\t");

  Serial.print(kiX);   Serial.print("\t");
  Serial.print(kdX);   Serial.print("\t");

  Serial.print(kiY);   Serial.print("\t");
  Serial.print(kdY);   Serial.print("\t");

  Serial.print(dtCont,4);  Serial.print("\t");

  servoX.write( PVX );
  servoY.write( PVY+6 );

}
```