Universidade Federal
do Rio de Janeiro
Escola Politécnica

DESENVOLVIMENTO DE UM CONTROLADOR PARA APLICAÇÃO EM UM RASTREADOR SOLAR

Saullo Cardoso Esterque Rodrigues

Projeto de Graduação apresentado ao Curso de Engenharia de Controle e Automação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientadores: Marcelo Martins Werneck
Alexandre Silva Allil

Rio de Janeiro
Dezembro de 2019

DESENVOLVIMENTO DE UM CONTROLADOR PARA APLICAÇÃO EM UM RASTREADOR SOLAR

Saullo Cardoso Esterque Rodrigues

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE AUTOMAÇÃO.

Examinado por:

Prof. Marcelo Martins Werneck, Ph.D.

Eng. Alexandre Silva Allil, B.Sc.

Prof. Claudio Miceli de Farias, D.Sc.

Prof. Gustavo da Silva Viana, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2019

Cardoso Esterque Rodrigues, Saullo

Desenvolvimento de um controlador para aplicação em um Rastreador Solar/Saullo Cardoso Esterque Rodrigues. – Rio de Janeiro: UFRJ/ Escola Politécnica, 2019.

XIV, 117 p.: il.; 29,7cm.

Orientadores: Marcelo Martins Werneck
Alexandre Silva Allil

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia de Controle e Automação, 2019.

Referências Bibliográficas: p. 115 – 117.

1. Energia Solar. 2. Rastreamento Solar. 3. Controlador PID. I. Martins Werneck, Marcelo *et al.* II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Título.

Gostaria de dedicar este projeto aos meus pais Ruda e Sônia pelo carinho, dedicação e sacrifícios feitos pela nossa família para que eu pudesse chegar a este momento. E por me ensinarem que mesmo nos tempos difíceis nunca devemos desistir. Este projeto e o que ela representa é o legado que eles construíram e deixam no mundo através de mim.

Agradecimentos

Existem muitas pessoas que cruzam nossos caminhos, que vão, vem, deixam marcas e memórias. A essas pessoas cuja amizade, amor e carinho levo comigo para os tempos de derrota e vitória deixo minhas considerações, afinal, direta ou indiretamente vocês riram, choraram, brigaram, desistiram, persistiram e lutaram comigo nessa odisseia chamada vida.

Aos meus pais Ruda e Sônia e meus avós (pais) Antônio Ferreira e Irene, dedico este projeto. Que vocês se lembrem que todos os sacrifícios que vocês fizeram, serviram para deixar um legado de boas contribuições ao nosso país.

Deixo aos meus irmãos Rudá, Katia, Janaina e Rafael Matos, Rebecca, Sarah, Pedro e Sophia meu carinho, por todos os tempos bons e ruins que fazem parte de nossa jornada, nessa família doida e bagunçada, que por mais distante possa estar, nunca deixa de estar unida quando precisa se defender.

Também existem aquelas famílias que não tem nosso sangue, mas tem nosso coração. Então aos meus pais adotivos Gerson e Laura Torres e meus irmãos adotivos Erika Herdy, Lucas Maurício e Victor Torres deixo meus eternos agradecimentos, pois com vocês, desde 2007, aprendo todo dia que existem pais e irmãos que estão apenas em nosso coração. Obrigado por não me deixarem sozinho, por terem sonhado, sofrido ao meu lado, estaremos juntos nessa e em outras jornadas.

Existem irmãos de sangue, de vida e os de luta. Algumas vezes não entendemos por que a vida nos leva para certos caminhos, foi o que me ocorreu quando fui fazer Bacharelado em Física em 2009 na UFRJ, ainda não sei por que a vida me levou para lá, apenas sei que mesmo não entendendo não me arrependo, pois conheci muitos bons amigos mas em especial Lucas Campos e Danielle Tostes, vocês estavam lá comigo nos dias de maior desespero, no laboratório ou na salinha da pós-graduação, contando piadas, rindo, estudando ou tomando choque no laboratório, pelos lanchinhos da Dani e desesperos com o Lucas nas provas, foram tempos de incertezas e de boas lembranças mas principalmente de boas pessoas.

Mesmo sem saber o que o futuro me guardava nesse furacão chamado UFRJ, muitas coisas boas aconteceram, passei por empresa Júnior onde conheci o incrível Eduardo Gouveia (a quem devo muito carinho por ter estado comigo em lutas que só nós dois sabemos o quão difícil foram, vitórias e derrotas que nos transformaram e nos

ensinaram), também passei pelo LIF onde conheci o professor Marcelo Werneck e o engenheiro Alexandre Allil, a quem agradeço pela oportunidade e tempo dedicado que resultou nesse projeto. Muitas coisas aconteceram e muitas ainda estão por acontecer, apenas sei que nenhuma delas foi em vão e todas serviram e servirão para me construir...

Todas essas pessoas passaram e passam em minhas vidas, de algumas sobram as memórias, de outras o carinho que preocupação, distancia e tempo nenhum acaba. Mas nessa mesma vida de idas e vindas, algumas pessoas ficam ao nosso lado, seja por que escolhemos ou por que nos escolheram, o que importa é que quando mais achei que as coisas caminhariam a seus passos constantes e retos você Isabella Garcia aparece em minha vida, do mais improvável lugar, no mais improvável momento você se torna a luz que ilumina e bagunça. De todos aqueles que passam por nós, você a quem tenho a sorte de chamar de esposa, me escolheu para jamais ficar sozinho. Esse projeto é um grande passo em minha vida, o término de uma etapa mas o começo dos projetos que construiremos juntos, nesse tempo que vivemos juntos você se tornou meu alicerce, você me segurou, disse para eu não desistir, me abraçou e me amou. Eu poderia ficar escrevendo um capítulo inteiro sobre nós dois, não irei pois além de não convir acredito que o que importa ainda estar por vir, esse projeto de vida que decidimos construir juntos jamais terá fim.

A todos vocês que me aturaram e me acompanharam um muito obrigado e a você Isabella Garcia o meu projeto termina, mas o nosso começa.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Automação.

DESENVOLVIMENTO DE UM CONTROLADOR PARA APLICAÇÃO EM UM RASTREADOR SOLAR

Saullo Cardoso Esterque Rodrigues

Dezembro/2019

Orientadores: Marcelo Martins Werneck
Alexandre Silva Allil

Curso: Engenharia de Controle e Automação

O objetivo deste trabalho é desenvolver um controlador de ação proporcional e integral (PI) para um motor de corrente contínua (CC), utilizando uma eletrônica apropriada através de microcontroladores, além de sensores de aceleração e amplificadores para o posicionamento de um sistema mecânico que deverá rastrear o Sol de forma autônoma. Dessa forma pretende-se obter a máxima captação de luz solar no foco de uma lente de Fresnel que contêm um feixe de fibras ópticas plásticas (POF) para guiar a luz solar até determinados ambientes fechados. Este projeto poderá ser utilizado para aumentar a eficiência de painéis solares, para iluminação local, e também na otimização da produção de plantas ou tanques químicos que necessitem de luz para a realização de processos químicos, economizando energia elétrica.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer.

DEVELOPING AN CONTROLER FOR APLICATION IN SOLAR TRACKERS

Saullo Cardoso Esterque Rodrigues

December/2019

Advisors: Marcelo Martins Werneck
Alexandre Silva Allil

Course: Automation and Control Engineering

The objective of this work is to develop a proportional and integral controller (PI) for a DC motor, using microcontrollers, besides accelerometer sensors and amplifiers, in order to position a mechanical system which will track the sun automatically. It is intended to focus the solar light on a plastic optical fiber (POF) bundle with a Fresnel lens. Theses fibers will guide the sunlight into different ambiences. The project may be used to increase the efficiency of solar panels, for local lighting and the production of plants on chemical tanks that require light, to save electricity carrying efficiency these process.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nesta sessão está apresentada a motivação e objetivos deste projeto e as justificativas que levam a viabilidade do mesmo. Para isso será apresentado empresas com projetos equivalentes e demonstrações financeiras que justificam o investimento neste tipo de pesquisa.

1.1 Motivação

Devido a sua grande extensão territorial e localização privilegiada no globo terrestre o Brasil é um dos países mais privilegiados em incidência solar do mundo. Apesar desse potencial, ainda temos muito que investir para podermos efetivamente aproveitá-lo.

Segundo [12], o Brasil tem na sua matriz energética apenas 1,27% de energia solar, que deverá ser aumentado pois estão em construções alguns projetos na área, principalmente no Nordeste. Atualmente temos cerca de 7439 empreendimentos em operação, sendo que para os próximos anos outros 200 estão em construção, o que aumentará nossa capacidade em mais $20GW$, e outros 391 previstos para terem suas construções iniciadas. Com isso, a energia solar deixará de contribuir com apenas 1,27% passando para 17,7% da energia gerada no Brasil quando todos esses empreendimentos estiverem concluídos..

A energia solar tem chamado a atenção, pelo fato de ser gratuita, não poluente e abundante no mundo todo. Isso não apenas demonstra a sua capacidade, mas o futuro para o qual a humanidade, e consequentemente o Brasil, está caminhando. Observando isso, desde 2015 o governo federal vem criando programas de incentivo para a compra e instalação de painéis solares por empresas e pessoas físicas. O Programa de Desenvolvimento da Geração Distribuída de Energia Elétrica, o ProGD, criado pelo então ministro de Minas e Energia, Eduardo Braga, em 2015, tinha como objetivo a estimulação da geração de energia a partir de placas solares dentro de unidades públicas e privadas, que possa ser compartilhada com o sistema

das distribuidoras de energia. O governo previu um potencial de investimentos de $R\$100$ bilhões e que 2,7 milhões de unidades públicas e privadas poderiam aderir ao programa até 2030. Para se ter uma ideia do custo-benefício, o preço médio da implementação e uso de placas solares pode ser visto na figura 1.1.

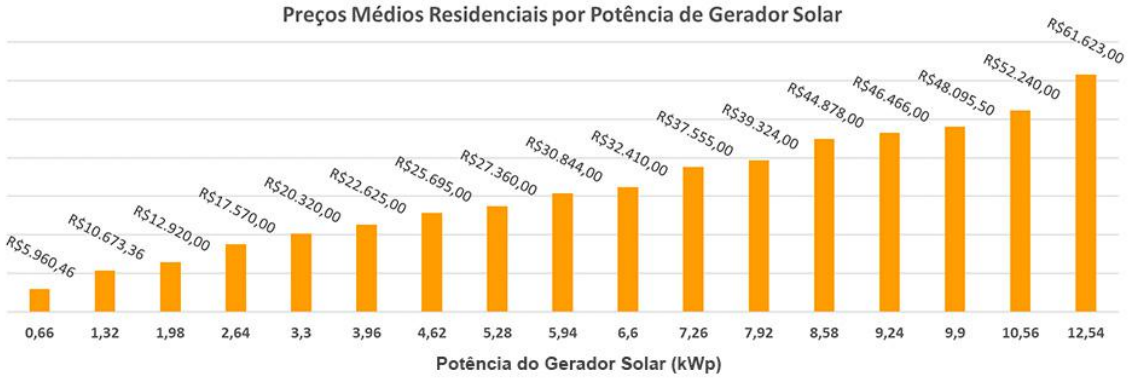


Figura 1.1: Custo-benefício da energia elétrica residencial [[1], [2]].

O custo para os diferentes tipos de residência pode ser encontrado na tabela 1.1 onde podemos comparar residências de diferentes portes.

Tabela 1.1: Tabela média de consumo energético. Tabela retirada de [1].

Tipo de Casa	Número pessoas	Produção (KWp)	Preço médio ($R\$$)
Pequena	2	1,32	$10mil$
Média	3 a 4	2,64	$17mil$
Média	4	3,3	$20mil$
Grande	4 a 5	4,62	$25mil$
Grande	5	6,6	$32mil$
Mansão	5	10,56	$52mil$

Dessa forma podemos fazer os seguintes cálculos para um sistema de energia solar fotovoltaica de $3.3kWp$:

- Supondo um Investimento médio de $R\$20.000$;
- a manutenção em 25 anos custa hoje em média $R\$5.000$.

Temos com isso um custo total de $R\$25.000$ com uma energia gerada em 25 anos equivalente a $100.000kWh$. Dessa forma o custo total (investimento e manutenção) será dado pela equação 1.1 :

$$\frac{25.000}{100.000} = R\$0{,}25/kWh \tag{1.1}$$

Se levarmos em consideração o estado do Rio de Janeiro a energia residencial que se compra da rede está custando hoje $R\$0{,}82/kWh$, na bandeira branca, ou seja,

a energia solar é mais barata que a energia da rede elétrica, tendo uma economia média, baseada nos valores atuais, de $R\$0{,}57/kWh$ ao longo de 25 anos.

Considerando os contínuos aumentos tarifários da conta de energia no Brasil, e conforme apresentada na figura 1.2, a energia solar vem cada vez mais diminuindo seu custo, se torna um investimento certo e ambientalmente correto cada vez mais atrativo para residências, comércio e indústria visto que outros meios de produção elétrica como nuclear, hidroelétrica e termoelétrica causam altos impactos ambientais e dependem de grandes investimentos.

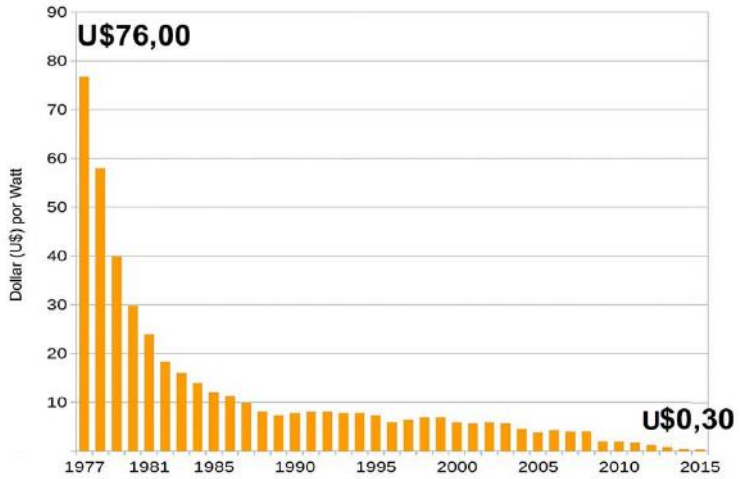


Figura 1.2: Preço médio da energia solar ao longo dos anos [1].

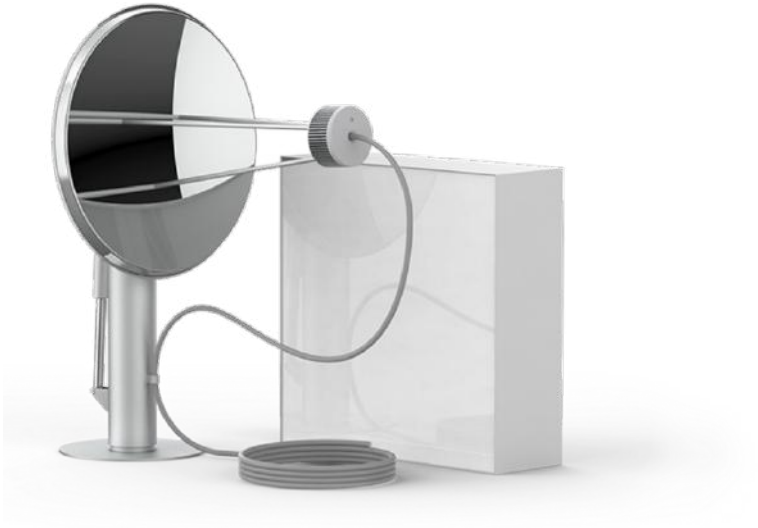
Em função disso diversas universidades e empresas têm investido em novas tecnologias para o reaproveitamento de energia solar não apenas através de painéis solares. Dentre essas tecnologias estão o rastreador solar, que hoje são usados em grandes usinas de energia para manter o painel solar perpendicular ao sol e também em tecnologias que utilizam fibras ópticas, espelhos e lentes para encaminhar o Sol para ambientes fechados para iluminação com baixo custo de energia elétrica como as empresas Solros e Pontual.

As empresas sueca Solros e brasileira Pontual Soluções Sustentáveis, demonstram a capacidade de utilização de diversas tecnologias reaproveitando a luz solar para iluminação e cultivo de plantas em ambientes fechados.

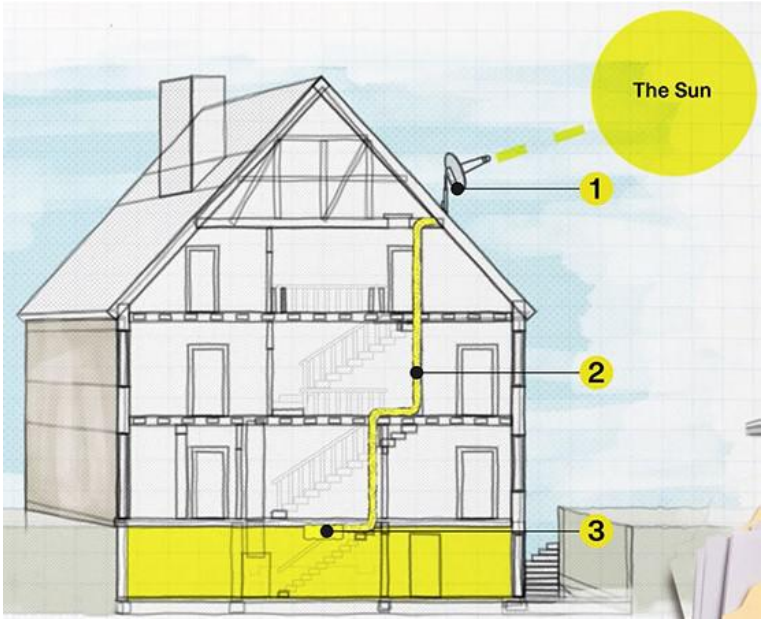
A combinação das diversas tecnologias de reaproveitamento da luz solar pode resultar numa maior economia energética para uma residência ou empresa. O uso da combinação das soluções das empresas Solros, Figura 1.3, e Pontual, Figura 1.4, com painéis solares poderia permitir que uma empresa usasse painéis solares apenas para alimentar tomadas e guardar energia em baterias para o uso noturno, diminuindo a necessidade de implementação de mais painéis solares que podem estar limitados a quantidade de espaço e custo que a empresa dispõe.

No atual cenário de mercado econômico com mercados globais e empresas cada

vez mais competitivas sendo constantemente cobradas pelas suas boas ações ambientais, esse tipo de solução pode trazer múltiplos benefícios reduzindo consumo elétrico. Com contas menores é possível tornar os preços dos produtos fabricados pela empresa mais competitivos ou simplesmente melhorar a saúde financeira da empresa.



(a) Equipamento da empresa sueca Solros.



(b) Exemplo de aplicação, captação solar para iluminação ambiente.

Figura 1.3: Tecnologia desenvolvida pela empresa Solros. Figura retirada do site da empresa Solros em 12/01/2019.

Conforme pode ser visto nas figura 1.3a a empresa usa uma antena parabólica espelhada acoplado a um feixe de fibras óticas para guiar a luz para ambientes fechados. Na figura 1.3b estão representado os andares da casa onde o nível 1 a parabólica capta a luz para um feixe de fibras ópticas, que atravessa o nível 2 até o

nível 3, onde irá iluminar o ambiente mais interno da casa.

Já a empresa Pontual, como pode ser visto nas figuras 1.4, usa tubos revestidos com material reflexivo para poder guiar a luz para ambientes fechados.

(a) Vista em cima telhado.

(b) Vista dentro ambiente.

Figura 1.4: Tecnologia desenvolvida pela empresa brasileira Pontual SS. Figura retirada do site da empresa Pontual SS em 12/01/2019.

Na figura 1.4a vemos o telhado com as lentes por onde irão entrar a luz solar. Já na figura 1.4b temos a saída da luz solar, pode-se observar que parecem lustres normais (lustres circulares) e também se nota que os lustres com lampadas de tubo (lustres retangulares) estão apagadas, em ambientes fechados que precisa de iluminação o dia todo com essa tecnologia pelo menos metade do dia (enquanto houver sol) ficará iluminada sem precisar usar energia elétrica para tal.

Mas não só empresas, repartições públicas como a UFRJ (Universidade Federal

do Rio de Janeiro) através do investimento em energia solar poderia ter uma economia nas suas contas, não tendo que arcar com altos custos de conta de energia ou com custos advocatórios para recorrer em negociações com os fornecedores, além do mais, ter tecnologias solares em suas instalações poderia servir de preparação para o mercado de trabalho dos alunos de engenharia através de atividades supervisionadas de manutenção e instalação[1].

Com a diminuição de custo a universidade poderia usar seus recursos para outras fontes. Imaginando todos os órgãos públicos fazendo o mesmo, o quanto de recursos o Brasil não pouparia? Com um custo operacional menor, poderíamos discutir sobre a redução de impostos, o que tornaria a qualidade de vida da população melhor e o custo para se manter empresas e indústrias menores, alavancando a geração de emprego e a capacidade do Brasil de competir no mercado internacional.

Para tanto, com o objetivo de colaborar com projetos de tecnologia para o setor de energia do Brasil, este projeto foi proposto. Inicialmente o projeto foi oriundo a partir de uma solicitação da estatal Petrobras S.A. que detém alguns projetos de pesquisa na produção de biocombustível através de algas. Segundo [14] essas algas ficam em grandes tanques a céu aberto pois necessitam de energia solar para fazer seus processos químicos, porém, para que isso seja feito esses tanques ficam submetidos aos intemperismos do ambiente e, consequentemente, contaminações devido a sujeiras e outros microrganismos o que torna o processo pouco eficiente e inadequado .

Dessa forma o rastreador solar se adéqua as necessidades, pois pode garantir a incidência de Sol em tanques fechados sem perda de produtividade. Entretanto, não só é prático para essa aplicação, a iluminação de ambientes confinados e a melhoria da eficiência de painéis solares também são aplicações úteis, pois ao manter os painéis perpendiculares ao Sol evitamos o aparecimento de sombra, garantindo total incidência solar com máxima produção.

1.2 Objetivo

Este projeto tem como objetivo desenvolver um sistema de controle para o rastreador solar 2. O rastreador solar consiste de um dispositivo munido de uma lente de Fresnel, acoplado a uma estrutura mecânica que gira, através de um motor CC, paralelamente ao sol com o objetivo de captar luz solar para um feixe de fibras ópticas. A luz guiada pelo feixe de fibras irá iluminar ambientes fechados .

Para implementar o motor CC é exigido a construção de um sistema de controle,

[1]Pensando na economia da conta de luz o MEC (Ministério da Educação) criou um programa em 2019 de isentivo a instalação de energia solar em universidades públicas conforme pode ser visto em [13].

a complexidade inserida pelo sistema de controle é compensada pela melhoria da precisão do rastreador no espaço. Dessa forma ao utilizar o motor CC passamos a ter uma precisão maior na incidência da luz solar na fibra óptica melhorando a captação de luz solar.

Para este projeto foi implementado um microcontrolador Arduino, programado em linguagem $C++$. O microcontrolador recebe as informações de hora/data fornecidas por um medidor RTC, através de um algoritmo junto da latitude e longitude pré-programado; calcula-se a posição do Sol no espaço, e através de um controlador PI, usando como referência o sinal de um sensor de posição (que fornecerá a posição angular θ) posiciona-se, através do motor CC, o rastreador perpendicular a posição do Sol. A luz solar irá incidir em uma lente de Fresnel, essa lente irá focalizar a luz em uma área, de forma que ela passe por um filtro de infra-vermelho (esse filtro evita que a luz a altas temperaturas danifique as fibras ópticas) e entre em contato com um conjunto de fibras óptica plásticas (POF) onde a luz será guiada para o uso desejado, como pode ser visto na Figura 1.5.

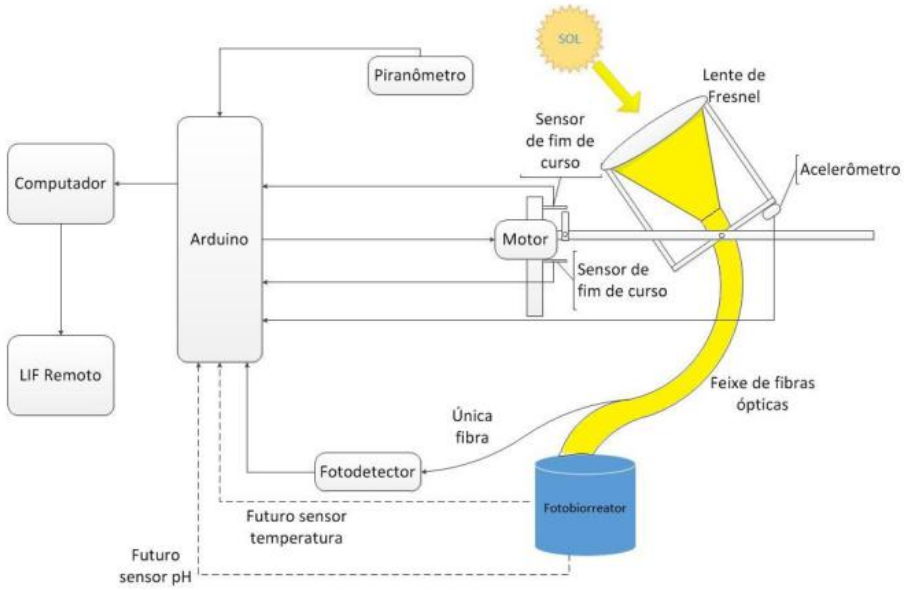


Figura 1.5: Diagrama do rastreador solar [3].

Um sensor de luz piranômetro irá verificar a luz ambiente, como passagem de nuvem ou qualquer coisa que diminua a radiação solar ambiente e um sensor de luz fotodetector irá verificar a incidência de luz nas fibras ópticas. Com esses dois parâmetros a inteligencia do projeto será capaz de saber se o rastreador esta descalibrado (quando o fotodetector indicar diferença de incidência com relação ao piranômetro) ou se é uma nuvem que está diminuindo a capacidade de captação de luz solar por exemplo. Isso tudo é feito para melhorar a eficiência.

O rastreador solar 2 é o aperfeiçoamento da versão 1 do rastreador solar, inicialmente desenvolvido no Laboratório de Instrumentação e Fotônica do Centro de

Tecnologia da UFRJ conforme [10] e [3]. Sua proposta visa buscar um meio mais econômico e eficiente de desenvolver o rastreador solar, além de fazer melhorias e aperfeiçoamentos a partir das dificuldades enfrentadas no primeiro projeto. Assim, o projeto aqui apresentado tem dois objetivos distintos muito importantes: desenvolver a tecnologia necessária para torná-lo viável e garantir uma tecnologia mais barata e eficiente que pode ser aplicada no reaproveitamento da energia solar tendo em mãos um produto viável para fabricação e comercialização. O objetivo final almejado no projeto dos idealizadores é que este, de fato, vire um empreendimento, visto o interesse comercial de algumas empresas no projeto.

Diferente do primeiro rastreador solar, este foi feito com materiais mais leves, além do seu desenho ser mais compacto. Buscou-se utilizar mais peças de alumínio e aço inox, visto que no primeiro o Sol, o vento e a chuva, causaram grandes estragos em pouco tempo de uso devido a oxidação. Adicionalmente, foram projetados compartimentos adequados para o motor e para a eletrônica, além da alteração de alguns sensores aos quais seus modelos e motivos serão apresentados. Entretanto a principal mudança foi a retirada do motor de passo para a aplicação de um motor CC, essa mudança apresenta uma grande diminuição no custo de produção e manutenção do equipamento.

1.3 Organização do Trabalho

Este trabalho está organizado em 6 capítulos dispostos da seguinte forma: no capítulo 1, é descrita a motivação para o projeto, o objetivo do trabalho e a estrutura de organização do mesmo. No capítulo 2, é apresentada a revisão bibliográfica descrevendo os conteúdos teóricos envolvidos no desenvolvimento do projeto. No capítulo 3, é apresentado o rastreador solar, suas características, os sensores e equipamentos utilizados, desenvolvimentos feitos, assim como as dificuldades encontradas e a solução para superá-las.

No capítulo 4 são apresentados os resultados obtidos e as respectivas análises. No capitulo 5 é apresentada a conclusão com o resultado do projeto, além da comparação de consumo energético e custo-benefício relacionado ao gasto com lâmpadas no bloco H do Centro de Tecnologia da UFRJ, o final do capítulo se refere a continuação do projeto como propostas de melhoria e continuidade. No apêndice, capítulo 6, consta os anexos, com as plantas dos diagramas elétrico e eletrônico do projeto, assim como, o método utilizado para realizar os cálculos de consumo elétrico utilizados no capítulo de conclusão, informações sobre o motor e o preço dos componentes e os códigos utilizados para fazer implementar o projeto.

Capítulo 2

Revisão Bibliográfica

Nesta sessão está apresentado os conceitos teóricos necessários para a construção e correto funcionamento do rastreador solar. Dessa forma será abordado a teoria por trás do controlador PID, alguns método de calibração do PID dentre eles o método que usa a função de transferência do motor e os conceitos de zona morta em engrenagens. Para medir o ângulo de posição do rastreador solar é apresentado na sessão os cálculos usados para converter os sinais fornecidos pelos sensores de posição em ângulo e a combinação deles através do filtro complementar. Por fim será apresentado algumas técnicas de rastreamento solar. Esta sessão abre o caminho para entender o que foi proposto e implementado na sessão 3, além do entendimento das dificuldades encontradas.

2.1 Controlador PID

De acordo com [15], o controlador PID é um controlador que utiliza ações proporcionais (P), integrais (I) e derivativas (D) para minimizar e zerar o erro em um sistema de forma a atingir um referêncial. Cada uma das ações tem sua aplicação: a utilização da ação proporcional diminui o *offset* no sistema aproximando o sinal de saída da referência desejada. A ação integral tem como objetivo zerar o erro entre a referência e a saída, visto que nem sempre a ação proporcional é suficiente, mas apesar de zerar o erro a ação integral insere um atraso e pode vir a gerar um *overshoot*. Quando o atraso é demasiado para o correto funcionamento usa-se a ação derivativa que diminui o tempo de assentamento do sistema de forma que ele não leve um tempo grande para levar atingir o referencial.

As ações podem ser combinadas como: P, PI, PD ou PID, cada uma dependendo de sua aplicação e projeto. O PID teórico é representado pela Equação 2.1.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.1}$$

Onde K_p, K_i e K_d são os ganhos proporcional, integral e derivarivo, respectivamente, $e(t)$ o erro na resposta e $u(t)$ a lei de controle, ou sinal de controle.

Na Figura 2.1 podemos ver um exemplo de resposta de um PID simulado no *software* MatLab, supondo $K_p = 1$, $K_i = 10$ e $K_d = 1$

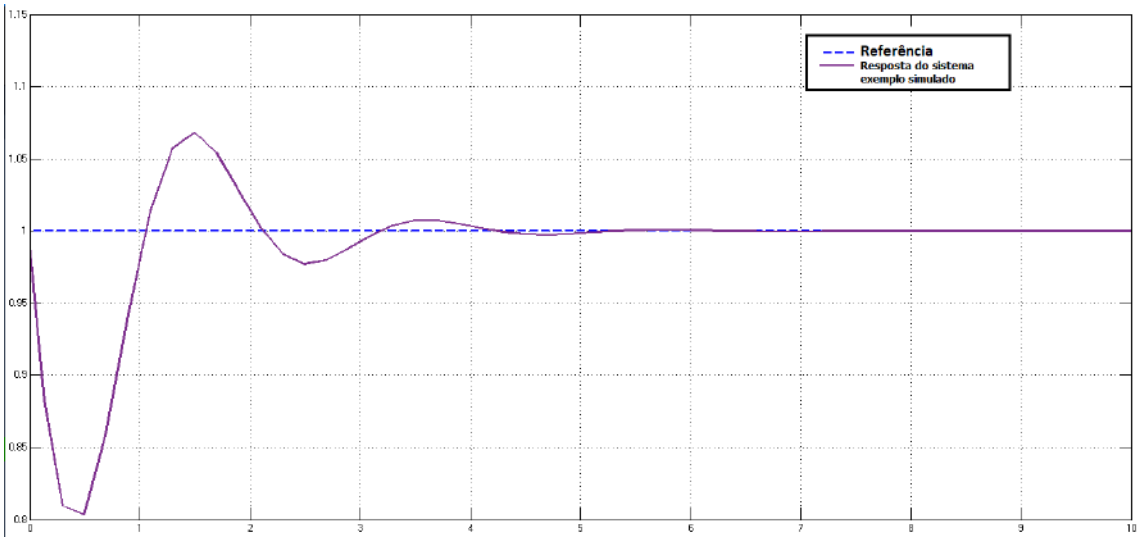


Figura 2.1: Exemplo da resposta ao degrau por um controlador PID.

Para determinar os valores dos ganhos K_p, K_i e K_d existem várias técnicas que podem ser aplicadas, a escolha de cada uma depende das necessidades do projeto, de sua facilidade e requisitos técnicos. Algumas dessas técnicas podem ser encontradas na referência [15].

2.2 Métodos de sintonia do controlador PID

Nesta seção apresentamos alguns métodos utilizados para a sintonia do controlador PID, a aplicação de cada método depende das informações disponíveis e da necessidade de cada projeto. Neste projeto inicialmente foi usado os métodos de Ziegler-Nichols e calibração manual que por motivos que serão apresentados não apresentou o resultado desejado.

- Calibração manual:
  A calibração manual é feita variando os ganhos K_p, K_i e K_d até que seja encontrada, graficamente, uma resposta $u(t)$ adequada às necessidades do projeto. A dificuldade em utilizar este método é devido ao fato dos ganhos poderem apresentar quaisquer valores (de 0 até ∞), sendo o valor ótimo um resultado

meramente empírico que muitas vezes pode depender mais da sorte do que do conhecimento técnico-teórico.

Uma das formas de fazer a calibração manual é zerar o K_i e K_d e variar o K_p até ele atingir a saturação ou o valor da referência. Então, variar o K_i até a resposta $u(t)$ apresentar um valor razoável e se necessário aplicar o K_d.

Esse método pode ser cansativo e pouco prático, mas conhecendo alguma informação do sistema podem-se obter parâmetros para saber por onde investigar os valores corretos.

- Método de Ziegler-Nichols (ZN):
  Este método consiste em pôr o objeto controlado em um regime de estabilidade crítica, ou seja, ele oscila sem estabilizar, conforme referência [15]. Para isso com o sistema em malha aberta faz-se K_d igual a zero e K_i infinito, então é procurado um K_p crítico (K_{cr}) que faça o sistema oscilar, conforme figura 2.2.

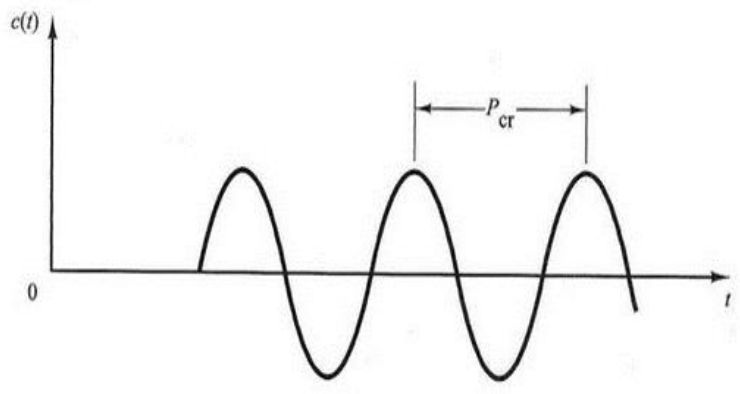


Figura 2.2: Exemplo de estabilidade crítica pelo método de ZN.

Uma vez encontrado o K_{cr}, no qual a estabilidade fique crítica, mede-se o período P_{cr} e usa-se a tabela 2.1 para calcular os parâmetros do controlador P, PI ou PID.

Tabela 2.1: Tabela conversão de Ziegler-Nichols.

Tipo de controlador	K_p	T_i	T_d
P	$0{,}5K_{cr}$	∞	0
PI	$0{,}45K_{cr}$	$\frac{P_{cr}}{1{,}2}$	0
PID	$0{,}6K_{cr}$	$0{,}5P_{cr}$	$1{,}125P_{cr}$

- Outros métodos: Há também, o método conhecido como o lugar das raízes e métodos de sintonia (mais comuns de serem aplicados em plantas químicas) que podem ser adotados; como foge do escopo deste projeto eles não serão mais citados pois não tiveram, a priori, aplicação.

2.3 A função de transferência do motor CC de imã permanente controlado pela armadura

Segundo [4] a função de trânsferência (FT) do motor de corrente contínua (CC) pode ser calculada levando em consideração dois modos: o controlado pela armadura e o controlado pela corrente de campo.

Para esse projeto, o motor é controlado pela armadura. Esse modo é comumente utilizado no acionamento de máquinas operacionais pois a tensão e a corrente no campo são constantes de forma que o fluxo magnético produzido também é constante e com isso, a variação da tensão na armadura faz variar, consequentemente, a rotação da máquina de forma direta, mantendo o torque constante e a potência variável proporcional à velocidade.

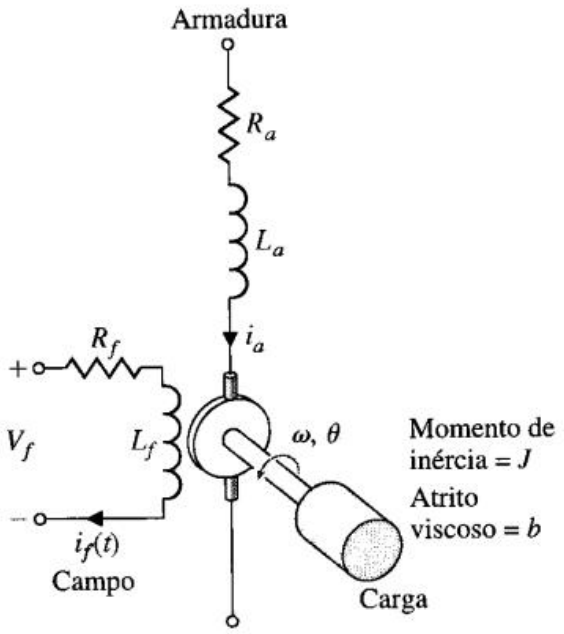


Figura 2.3: Representação do motor [4].

Considerando que o campo não esteja saturado, o fluxo no entreferro do motor (Φ) é proporcional à corrente do campo i_f.

$$\Phi = K_f i_f(t) \tag{2.2}$$

Onde, se considerarmos o torque (T_m) linear à corrente ($i_a(t)$), de forma que teremos:

$$T_m = K_1 K_f i_f(t) i_a(t) \tag{2.3}$$

No caso do controle por armadura a corrente i_a (corrente do campo) é constante, de forma que podemos aplicar a transformada de Laplace exprimindo o torque (T_m) no motor por:

$$T_m = K_m I_a(s) \tag{2.4}$$

$$K_m = K_1 K_f i_f(t) \tag{2.5}$$

Onde K_f, K_1 e K_m são constantes de proporcionalidade próprias do motor. Dessa forma conforme pode ser retirado do circuito da figura 2.3 podemos relacionar a tensão de armadura (V_a) com o torque e a tensão eletro-motriz (V_b), por:

$$V_a = (R_a + L_a s) I_a(s) + V_b(s) \tag{2.6}$$

$$V_b = K_b \omega(s) \tag{2.7}$$

O torque da carga (T_L) no motor é dada por:

$$T_L(s) = Js^2\theta(s) + bs\theta(s) = T_m - T_d \tag{2.8}$$

Onde T_d é o torque da força perturbadora ao motor. Com essas informações temos que a função de transferência do motor controlador pela armadura é dada por:

$$W(s) = \frac{\theta(s)}{V_a(s)} = \frac{K_m}{s[(R_a + L_a s)(Js + b) + K_b K_m]} \tag{2.9}$$

Onde seu diagrama de blocos aparece representado na figura 2.4.

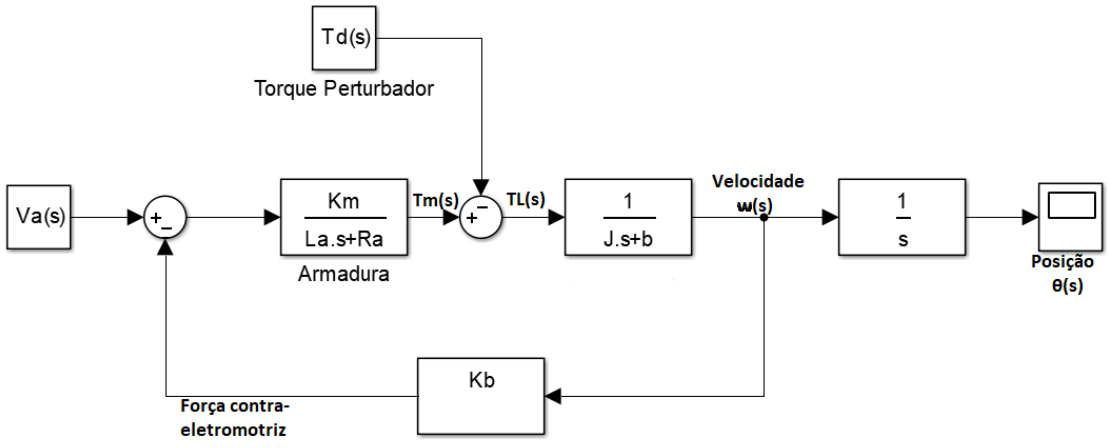


Figura 2.4: Função de transferência do motor [4].

Para aplicarmos no projeto podemos fazer ensaios para identificar os parâmetros que devem, verificar com o fabricante ou buscar em seu manual (*datasheet*) quando disponíveis.

Se considerarmos uma aproximação (que ocorre em alguns casos) na qual $L_a/R_a << 1$, teremos que $W(s)$:

$$W(s) = \frac{K_a}{\tau s + 1} V_a(s) - \frac{K_d}{\tau s + 1} T_d(s) \tag{2.10}$$

Onde,

$$K_a = \frac{K_m}{R_a b + K_b K_m} \tag{2.11}$$

$$K_d = \frac{R_a}{R_a b + K_d K_m} \tag{2.12}$$

$$\tau = \frac{J R_a}{R_a b + K_d K_m} \tag{2.13}$$

Essas simplificações servem para facilitar na hora de fazer ensaios que possibilitem conhecer melhor o motor.

Além disso, podemos considerar situações que não exista perturbação e o torque externo seja nulo, de forma que podemos simplificar a função de transferência para:

$$W(s) = \frac{K_a}{\tau s + 1} V_a(s) \tag{2.14}$$

Essa forma simplificada é importante pois alguns testes feitos no projeto basearam-se nela como forma de aproximação do modelo, para melhor investigar

o sistema disposto. Alguns dos métodos de identificação dos parâmetros do motor podem ser encontrados na referência [16] e serão melhor abordados mais tarde quando for demonstrado como algumas informações sobre o motor CC do projeto foram retiradas.

2.4 Zona morta em engrenagens

Uma folga na engrenagem (chamada de folga de fundo, em mecânica) acontece quando a conexão entre as engrenagens tem diferentes tamanhos, conforme mostrado na figura 2.5.

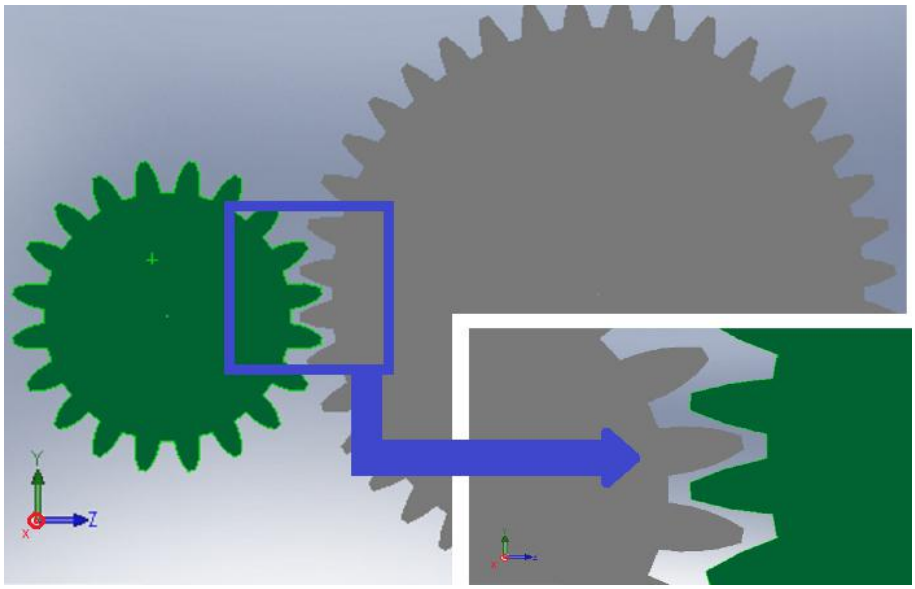


Figura 2.5: Exemplo de folga em engrenagens. As engrenagens não estão tão juntas, havendo um espaçamento entre um dente de uma e o vão dos dentes de outra.

Este problema faz com que o motor ao ser parado em uma posição acabe rotacionando um pouco mais em um deslocamento indefinido mas dentro do tamanho da folga. Como a posição desejada nunca é atingida, isso também faz com que o controle PI fique reajustando a posição causando oscilações indesejadas no sistema. Com isso, o sistema fica oscilando indefinidamente deslocando-se em pontos de equilíbrio instáveis.

Segundo [5] o nome dessa folga em controle é zona morta, que é uma região onde existe um percentual que indica a maior variação em uma variável sem provocar mudança na indicação ou sinal de saída, ou seja, é uma região ao qual não se consegue exercer influência.

Na Figura 2.6 é mostrado um exemplo de zona morta em uma região que compreende um intervalo de $[-a,a]$ correspondente a região "morta", ou seja a região da folga. Para resolver o problema da zona morta algumas técnicas de controle

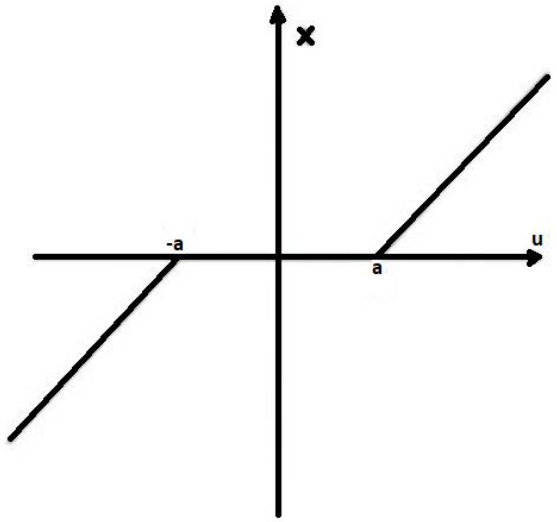


Figura 2.6: Exemplo genérico de zona morta [5], a região compreendida entre $[-a,a]$ corresponde a região onde não se pode exercer influencia.

chamadas de compensação de zona morta como: "método do reajuste do ganho proporcional", "método do uso de controle PI linear", "método do uso de controle proporcional variável com o erro"e o "método do uso de ganho proporcional fixo combinado com linearização vibracional", conforme referência [5], podem ser empregadas.

A dificuldade em implementá-las se deve, não apenas pelo custo computacional mas também pela dificuldade em se obter informações corretas sobre o real espaçamento que compreende o intervalo $[-a,a]$. Como não é o escopo desse projeto entrar em detalhes sobre cada técnica, elas são apresentadas aqui para que o leitor fique ciente de sua existência e que as decisões de projeto tomadas para resolver o problema da zona morta e outros, tiveram como consideração a aplicação destas técnicas.

2.5 Medida angular com sensores acelerômetro e giroscópio

2.5.1 Sensor acelerômetro

Deseja-se calcular valores de ângulo do acelerômetro considerando sua inclinação, como mostra a Figura 2.7 onde está representado o sensor de posição inclinado de um ângulo θ. Para o acelerômetro mede-se a inclinação com relação a aceleração da gravidade.

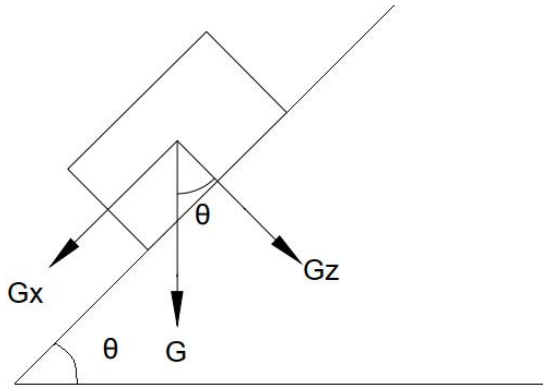


Figura 2.7: Representação de um sensor acelerômetro inclinado no espaço.

A figura 2.7 representa um sensor acelerômetro no inclinado no espaço, onde G é a gravidade e G_x e G_y são suas decomposições no eixo x e y respectivamente. Dessa forma podemos obter os seguintes valores para o ângulo de inclinação de acordo com os eixos do acelerômetro:

$$\theta_x = tg^{-1}(\frac{a_x}{\sqrt{a_y^2 + a_z^2}}) \tag{2.15}$$

$$\theta_y = tg^{-1}(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}) \tag{2.16}$$

$$\theta_z = tg^{-1}(\frac{\sqrt{a_x^2 + a_y^2}}{a_z}) \tag{2.17}$$

em que $\theta_{i=x,y,z}$ é o ângulo com relação aos eixos x, y e z respectivamente e $a_{i=x,y,z}$ é a aceleração com relação aos eixos x, y e z respectivamente.

2.5.2 Sensor giroscópio

Já o giroscópio usa a variação do movimento angular para medir o ângulo da superfície no qual ele foi empregado, conforme figura 2.8.

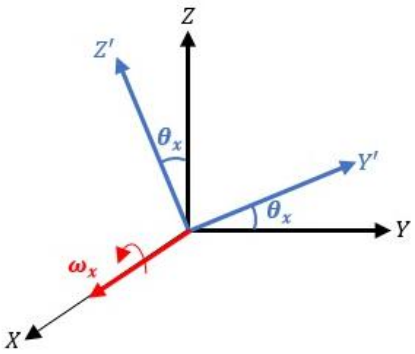


Figura 2.8: Giroscópio rotacionando em torno do eixo x.

Em que θ_x é o ângulo rotacionado no eixo x e ω_x é a velocidade angular do eixo x. Integrando a velocidade angular ($\omega_{i=x,y,z}$) no tempo com relação a medida anterior temos a posição angular para cada eixo. Dessa forma integrando-se a velocidade angular obtemos a posição:

$$\theta_x = \int_t^{t_0} \omega_x dt \tag{2.18}$$

$$\theta_y = \int_t^{t_0} \omega_y dt \tag{2.19}$$

$$\theta_z = \int_t^{t_0} \omega_z dt \tag{2.20}$$

De forma que para um intervalo de tempo infinitesimal δt:

$$\theta_x = \theta_{x_0} + \omega_x \delta t \tag{2.21}$$

$$\theta_y = \theta_{y_0} + \omega_y \delta t \tag{2.22}$$

$$\theta_z = \theta_{z_0} + \omega_z \delta t \tag{2.23}$$

Dessa forma, os sensores podem ser adotados para medir o ângulo desejado. Conforme será explicado, cada sensor tem uma aplicação dentro de suas limitações e a combinação de ambos é empregada para se obter ter um melhor resultado do ângulo aferido.

2.6 Filtro complementar

Segundo [17] um filtro complementar é um filtro que combina a leitura de dois ou mais parâmetros para se adquirir uma saída confiável. Para este projeto o filtro complementar combina a leitura do sinal de um sensor IMU (*Inertial Measurement Unit*) que consiste do sinal de um acelerômetro para dados de baixa frequência e o sinal do giroscópio com dados de alta frequência. O objetivo é utilizar o filtro complementar utilizando mais de um sensor para zerar o erro e aumentar a confiabilidade da medida da variável física desejada.

Basicamente pode-se modelar o filtro complementar do IMU conforme a Figura 2.9, que demonstra de forma simplificada as transformações feitas nos sensores e a combinação entre eles para termos o valor de ângulo final.

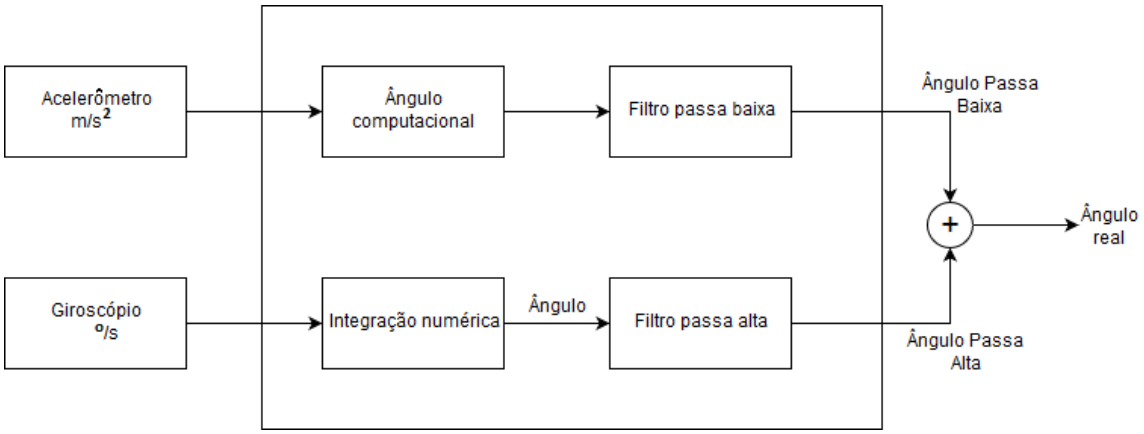


Figura 2.9: Diagrama do filtro complementar para duas entradas do IMU.

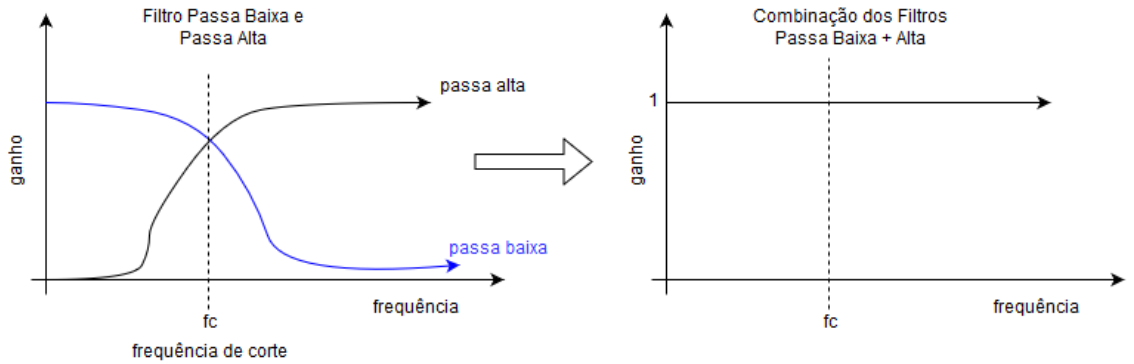
O acelerômetro produz ruído em alta freqûencia, já o giroscópio tem um sinal menos ruidoso, porém conforme equação 2.24 precisamos integrar o seu sinal (ω) para obter o valor do ângulo, mas isso integra, também, o ruído de forma que o sinal fica mais ruidoso. É difícil eliminar esse ruído, visto que, ele é aleatório, mas pode-se atenuá-lo utilizado um filtro.

$$\theta_{gir} = \int (\omega + ruído)dt = drift \tag{2.24}$$

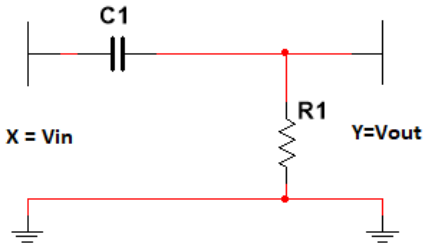
Na equação 2.24 o *drift*[1] é o ruído no sinal.

Para eliminar esses ruídos são aplicados um filtro passa-baixa a fim de eliminar os ruídos de alta freqûencia no acelerômetro e um filtro passa-alta para eliminar os ruídos de baixa freqûencia no giroscópio. Na Figura 2.10 demonstra-se um exemplo da construção e resposta desses filtros, onde a combinação deles resulta em um sinal de magnitude unitária.

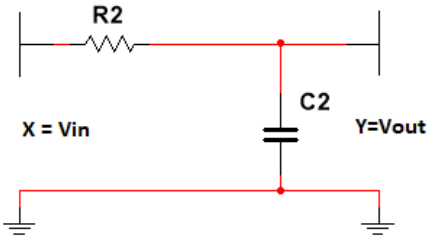
[1]O drift do giroscópio é principalmente devida à integração de dois componentes: uma variável de variação lenta, quase DC, chamada instabilidade de polarização e um ruído variável de alta frequência, denominado passeio aleatório angular (ARW - *"Angular Randon Walk"*). Esses parâmetros são medidos em graus de rotação por unidade de tempo.



(a) Resultado gráfico da combinação dos filtros.



(b) Filtros passa-alta.



(c) Filtro passa-baixa.

Figura 2.10: Exemplo de filtro passa-baixa e passa-alta.

Na Figura 2.10 a frequência de corte (fc ou *cut-ff frequency*, em inglês) será escolhida dependendo da aplicação do sensor. Para os sensores que combinam acelerômetro e giroscópio em geral se quer saber os valores da freqûencia de corte, do *bias*[2] e do ruído aleatório que podem ser encontrados no *datasheet* do sensor.

A freqûencia de corte também pode ser calculada a partir dos circuitos dos filtros conforme a Equação 2.25.

$$f_c = \frac{1}{2\pi RC} \tag{2.25}$$

Onde R e C são, respectivamente, a resistência e a capacitancia do circuito.

O filtro passa-baixa pode ser modelado da seguinte forma:

$$y_i = \alpha_{fpb} x_i + (1 - \alpha_{fpb}) y_{i-1} \tag{2.26}$$

$$\alpha_{fpb} = \frac{\Delta t}{RC + \Delta t} \tag{2.27}$$

E o filtro passa-alta pode ser modelado da seguinte forma:

[2] o *bias* é o erro de fator de escala, este erro é acumulativo devido a integração, ele é medido em m/s^2.

$$y_i = \alpha_{fpa} y_{i-1} + \alpha_{fpa}(x_i - x_{i-1}) \tag{2.28}$$

$$\alpha_{fpa} = \frac{RC}{RC + \Delta t} \tag{2.29}$$

Onde α_{fpb} e α_{fpa} são coeficientes do filtro passa-baixa e do filtro passa-alta, respectivamente, $R(= R_1 = R_2)$ é o valor da resistência do circuito e $C(= C_1 = C_2)$ é a capacitancia do capacitor do circuito.

Para o sensor de posição modelam-se os filtros da seguinte forma, conforme feito acima:

$$FPB_i = \alpha_{fpb} Acc_i + (1 - \alpha_{fpb}) FPB_{i-1} \tag{2.30}$$

$$FPA_i = \alpha_{fpa} FPA_{i-1} + \alpha_{fpa}(AngGir_i - AngGir_{i-1}) \tag{2.31}$$

Onde FPB_i e FPA_i são a função passa-baixa e a função passa-alta, respectivamente e Acc_j é o ângulo medido pelo sensor acelerômetro e $AngGir_j$ é o ângulo medido pelo sensor giroscópio no instante j.

Para o filtro complementar o α_{fpb} e α_{fpa} serão um só e ele será escolhido dependendo do que se almeja. Essa escolha será feita dependendo dos movimentos de interesse e onde a resposta do sensor deverá ter predominância trabalhando na faixa da baixa frequência ou da alta frequência de forma que, α_{fc} será igual à:

- para sinais em baixas frequências $\rightarrow$ acelerômetro;
- para sinais em altas frequências $\rightarrow$ giroscópio.

Ou seja, o valor de α_{fc} será escolhido dependendo do quão rápido ou lento o sistema deverá se deslocar.

Dessa forma o filtro complementar pode ser modelado conforme a Equação 2.32.

$$\theta = \alpha_{fc}('\hat{A}nguloAcelerômetro') + (1 - \alpha_{fc})('\hat{A}nguloGiroscópio') \tag{2.32}$$

Como α_{fc} varia entre [0,1], pode-se observar que quanto mais perto de 0 o valor de ângulo medido no giroscópio será mais dominante e quanto mais próximo de 1 o valor de ângulo medido pelo acelerômetro será mais dominante. Nesse projeto queremos uma resposta mais rápida então o α_{fc} escolhido foi de 0.96, que foi verificado empiricamente e a partir de projetos estudados em algumas referências como [17] e [18]. Boa parte do material para o entendimento sobre o filtro complementar foi retirado das vídeo aulas do professor Dr. Viswanath Talasila, do departamento

de engenharia de telecomunicação do Ramaiah Institute of Technology, na India referência [19].

2.7 Técnicas de rastreamento solar

Existem diversas técnicas de rastreamento solar (e/ou captação de luz) cada uma delas é aplicada conforme a necessidade de projeto e/ou disponibilidade de recursos, alguns estudos sobre a reflexão de luz em diferentes meios podem ser vistas nas referências [20] onde algumas dessas técnicas estão listas abaixo:

- **Rastreamento por amostragem**
  Conforme apresentado em [6] essa técnica consiste em utilizar diversos sensores de fotoluminosidade para decidir em qual direção o Sol está e posicionar o equipamento na direção adequada. Apesar de fornecer bons resultados e ser relativamente simples, essa técnica apresenta grande imprecisão; para uma precisão maior é necessário aumentar o número de sensores, o que aumenta a complexidade do projeto.

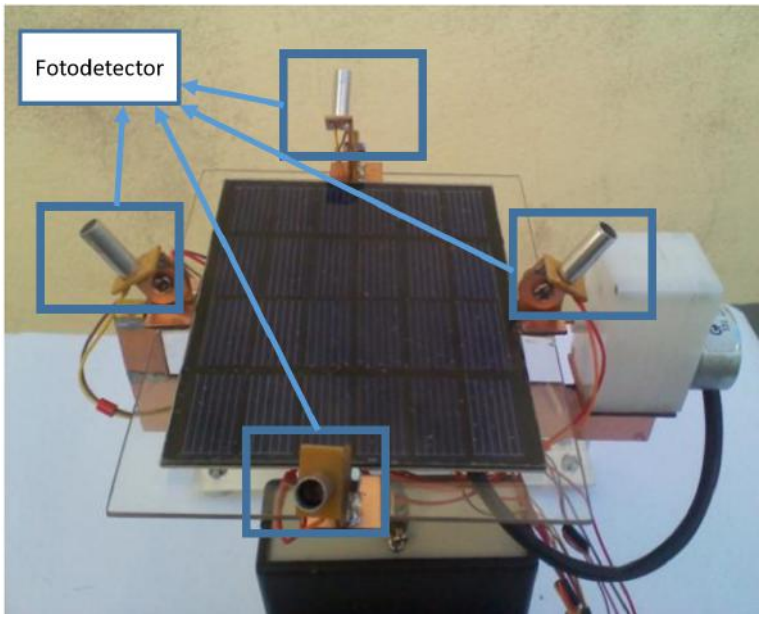


Figura 2.11: Exemplo de Rastreador por amostragem [6]. Onde pode ser visto os fotodetectores nas pontas o que limita a precisão a intensidede em um dos 4 fotodetectores.

- **Tubos reflexivos de vidro**
  A técnica empregada pela empresa Pontual Soluções Sustentáveis consiste em aplicar um tubo reflexivo para guiar a luz; esse tudo reflexivo geralmente consiste de espelhos ou materiais metálicos ou pode ser feito com resinas. A desvantagem de adotar os espelhos é que a dispersão da luz ao entrar e ao sair

é muito grande devido ao índice de refração do espelho e as perdas entre as camadas de vidro e alumínio. A aplicação deste tipo de tecnologia pode ser vista nas estações de ônibus BRT na cidade do Rio de Janeiro, Brasil.

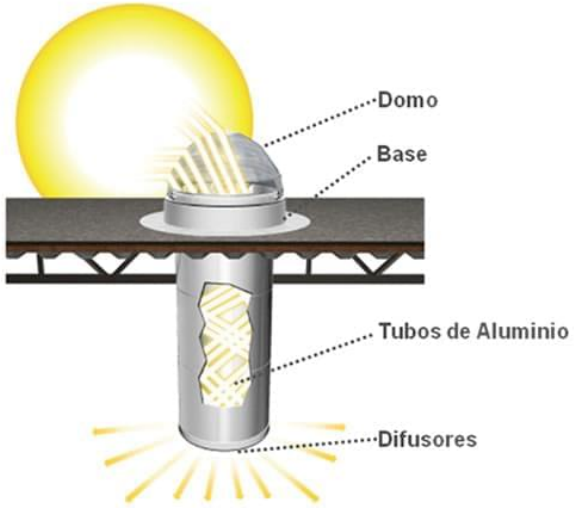


Figura 2.12: Exemplo de tubo reflexivo de luz [7].

- **Tubos reflexivos de resina**
  Uma ideia muito empregada em lugares sem recursos ou que faz reciclagem, por exemplo, é o uso de garrafa PET com uma resina para direcionar a luz em ambientes, como os tubos reflexivos eles dispersam muito a luz de forma que a luminosidade ambiente seja mais baixa do que a que poderia ser obtida com fibra óptica.

Figura 2.13: Exemplo de tubo reflexivo de resina usando garrafa pet e resina [8].

- **Sistema de dois e um eixo**

  Os sistemas de um (uniaxial) e dois (biaxial) eixos são usados para determinar o número de direções no qual um sistema de rastreamento solar gira, no caso do rastreador deste projeto ele é de um eixo (uniaxial), o segundo eixo deve ser ajustado manualmente, a cada 21 dias. A vantagem de aplicar o de dois eixos é a auto-calibração que se pode inserir nele, a desvantagem é o fato de aumentar a lógica de controle, além da necessidade de mais um motor consumindo energia e aumentando os custos do projeto. Um exemplo de sistema de dois eixos é a *smartflower* desensolvido pela empresa *Smartflower* [9], que consiste em uma série de paineis solares cortados em formato de pétala e que gira em 360^o nos dois eixos para rastrear o sol, como pode ser visto na figura 2.14.

Figura 2.14: Exemplo de rastreador biaxial [9].

Capítulo 3

Metodologia

Nesta sessão será apresentada as informações sobre o desenvolvimento do projeto, como os sensores e equipamentos eletrônicos utilizados, suas ligações elétricas e eletrônicas, lógica de programação e diagrama de blocos proposto, além de explicar as dificuldade encontradas no projeto e fazer a introdução para a sessão 4 onde será explicado com mais detalhes as soluções empregas para contornar as dificuldades encontradas.

3.1 Proposta inicial

A primeira proposta do projeto consistia na construção de um controlador PI para o rastreador solar acoplado a um motor CC de forma que o rastreador gire acompanhando paralelamente o movimento do sol. Utilizando o motor CC e um controle PI é possível aumentar a precisão do rastreador acabando com o acumulo de erro acumulativo que o motor de passo insere conforme mostrado em [10]. Com essa ideia o diagrama de blocos do projeto seria apresentado inicialmente como o da Figura 3.1.

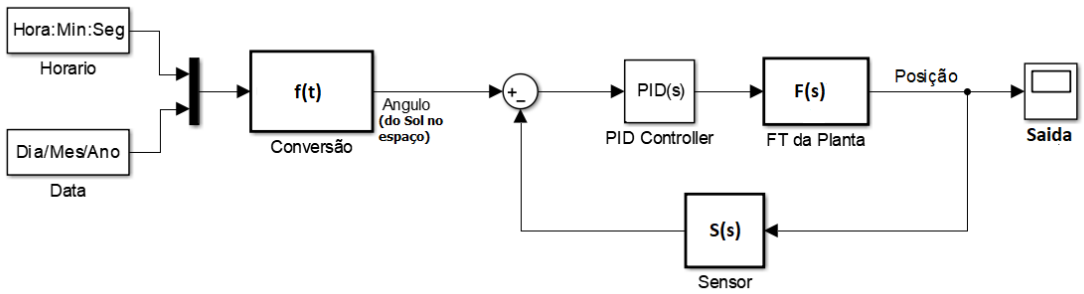


Figura 3.1: Diagrama de blocos proposto.

Na Figura 3.1 o tempo (Hora:Min:Seg) e a data atual (Dia/Mes/Ano) são inserida em uma função $f(x)$ para o cálculo do ângulo que se encontra o Sol no espaço em

um dado instante. Uma vez obtido o ângulo do Sol (o referêncial), a informação é inserida no sistema em malha fechada com um controlador PID (no caso deste projeto PI) que deve posicionar o rastreador perpendicular ao Sol. O diagrama de blocos da Figura 3.1 representa o modelo ideal, onde a partir dos dados da planta pode-se medir a resposta ao degrau e usar o sinal de saída para fazer a calibração do PID. Mas conforme será apresentado nas próximas sessões uma série de problemas foi encontrado, aos quais são debatidos neste presente trabalho seus motivos e soluções. Tais problemas foram responsáveis por não garantir uma simplicidade maior do sistema, fazendo com que o mesmo não tivesse inicialmente o equilíbrio desejado.

3.2 Bancada de testes

Para este projeto foi criada uma bancada de testes como mostra a Figura 3.2.

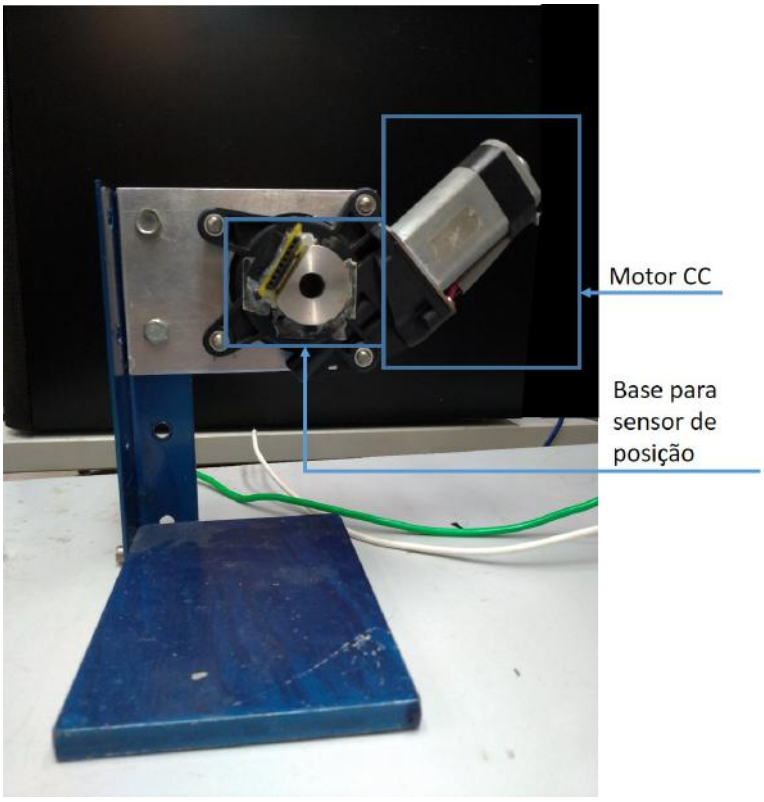


Figura 3.2: Bancada de testes.

Nesta bancada foram implementados os testes iniciais com os sensores e do código, isso permitiu a identificação de erros e evitar danos no equipamento. Uma vez testadas em bancada, os testes e calibrações foram feitas na planta.

3.3 Montagem e especificações técnicas

Para este projeto foram utilizados alguns sensores e dispositivos eletro-mecânicos, cuja suas finalidades serão explicadas abaixo a fim de esclarecer o que são e justificar suas aplicações. Além disso será demonstrado como foi a implementação e conexões desses equipamentos no projeto.

3.3.1 Disposição do rastreador solar

Conforme pode ser visto na Figura 3.3 e 3.4, o Rastreador é composto de diversas partes onde o novo modelo foi feito para ser mais leve e sofrer menos com intemperismo do que o anterior.

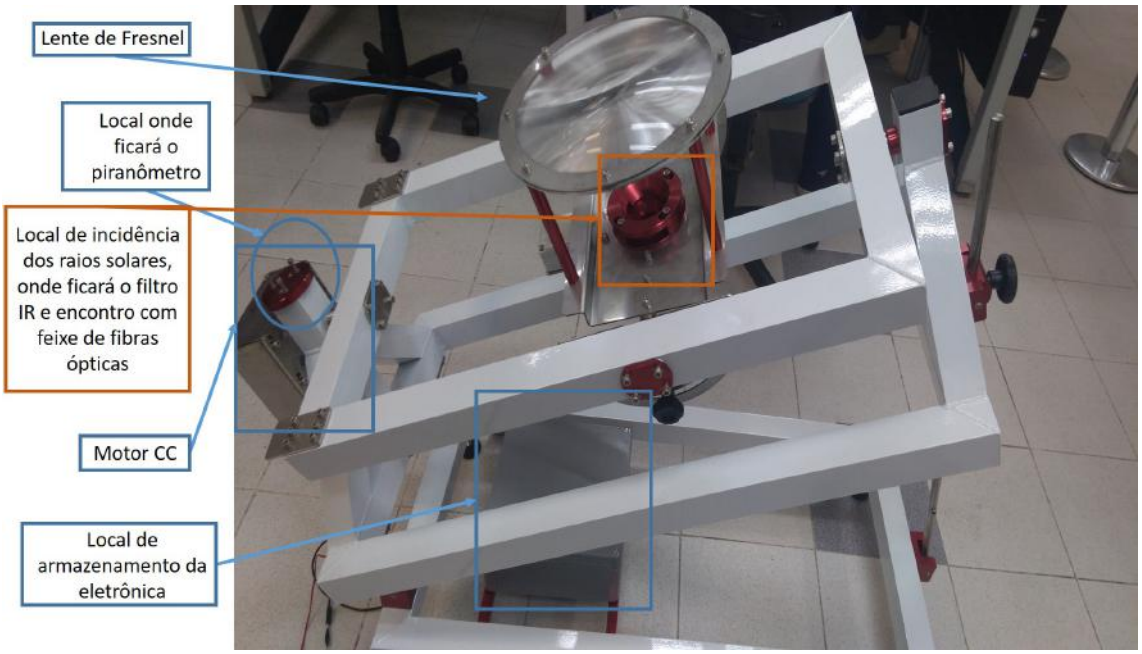


Figura 3.3: Elementos do rastreador solar.

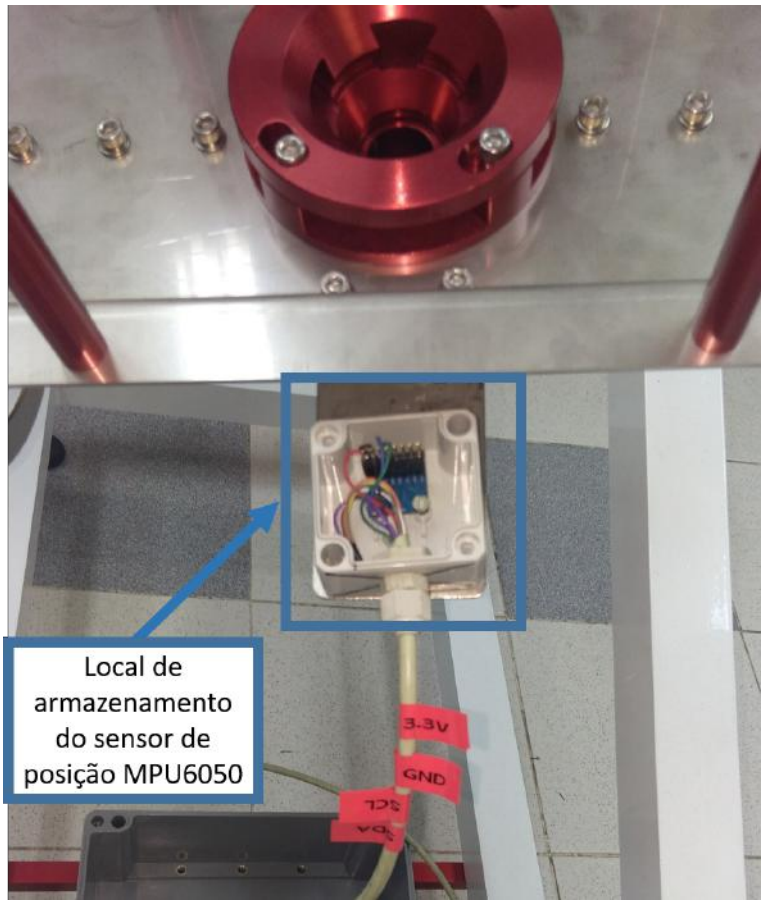
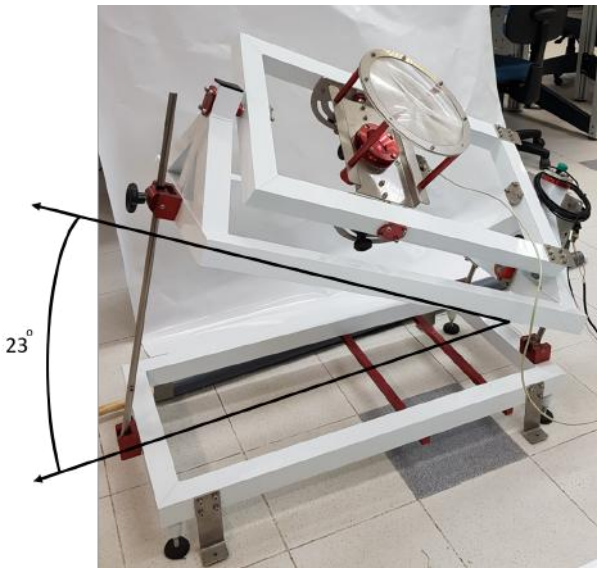


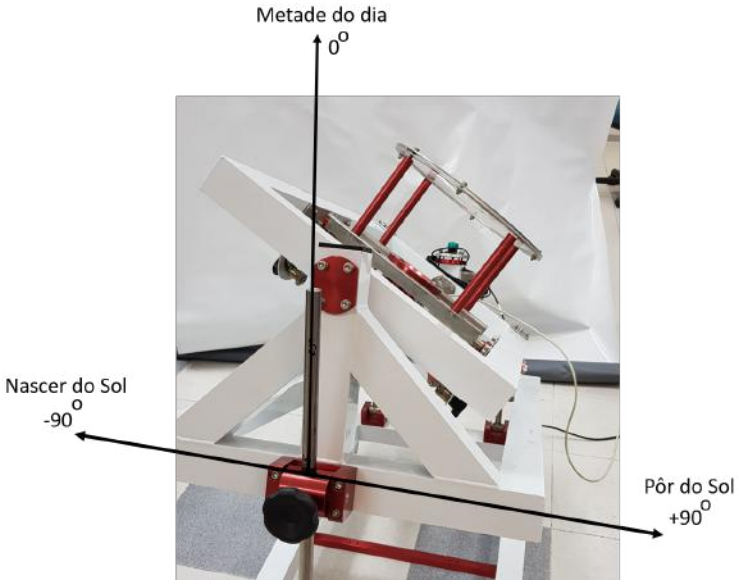
Figura 3.4: Sensor de posição no rastreador solar.

O rastreador é composto pela lente de Fresnel; com ela os raios de sol convergem para um ponto focal que dispõe de um filtro infravermelho e impede danos devido a energia referente a radiação infravermelha no feixe de fibra óptica. O motor fica disposto em um compartimento próprio, assim como o sensor de posição. Os demais equipamentos eletrônicos ficam instalados em um compartimento que é isolado contra penetração de água e partículas de poeira.

Para o seu funcionamento o rastreador gira em $\pm 90^o$ e tem uma inclinação de 23^o conforme pode ser visto na Figura 3.5. O ângulo de 23^o é devido a posição atual de instalação no globo terrestre e deve ser calibrada conforme a latitude do local onde o equipamento será instalado.



(a) Angulação com relação a latitude.



(b) Angulação de giro.

Figura 3.5: Grau de liberdade do Rastreador.

3.3.2 Sensor de posição: acelerômetro e giroscópio

Neste projeto o sensor de posição escolhido foi o $MPU6050$ devido ao seu baixo custo e boa *performance* em projeto de *drones*.

O $MPU6050$ é composto de um giroscópio e de um acelerômetro, a ideia de combinar os dois é porque os acelerômetros medem todas as forças que estão atuando sobre o objeto, o que faz com que ele seja muito mais sensível. Cada pequena força perturba a medida, completamente. Se o trabalho é feito em um sistema atuado (como um *quadrocopter*), então as forças que acionam o sistema também serão sentidas pelo sensor. Os dados do acelerômetro são confiáveis somente a longo

prazo (ou seja, após várias medições), portanto, um filtro passa-baixa tem que ser implementado. Já os giroscópios são fáceis de obter um medição precisa que não seja suscetível a forças externas, o problema ocorre devido à integração ao longo do tempo pois a medição tem a tendência de se desviar, não retornando a zero quando o sistema desloca-se à sua posição original, isso torna os dados do giroscópio confiáveis apenas a curto prazo, uma vez que começa a se desviar ao longo do tempo.

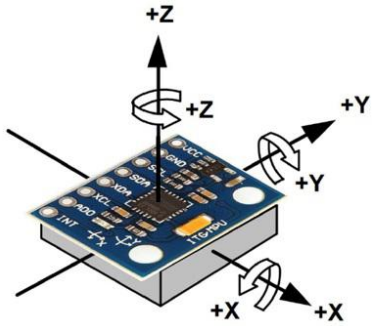


Figura 3.6: Sensor de posição MPU6050 e seus eixos.

Os cálculos desenvolvidos para a conversão dos sinais do giroscópio e do acelerômetro foram demonstrados na seção 2.5 e podem ser vistos na literatura conforme referências [17] e [18], os códigos para implementa-lo pode ser visto no anexo VI no código "$mainFirmware.ino$" linha 325 até 378.

3.3.3 Sensor de fim de curso: sensor Hall

O sensor de fim de curso tem como objetivo, neste projeto, limitar o ângulo com o qual o eixo do motor pode rotacionar impedindo, também, que o motor seja danificado. Conforme [10] na primeira versão do rastreador solar foi usado como sensor de fim de curso um micro *switch*, este dispositivo consiste em uma chave que quando acionada conectará o circuito, emitindo um sinal elétrico. Porém, para funcionar no rastreador ele deveria ficar do lado de fora, sofrendo oxidação rapidamente devido a chuva, sol e poeira que entravam no sensor.

Portanto, decidiu-se usar o sensor Hall, modelo $ACS712T/30A$, como um sensor de fim de curso. O sensor Hall, através de efeitos eletromagnéticos, mede a corrente que passa através dele. Seu funcionamento se faz de forma estatística onde são feitas algumas leituras com o sensor e a média dessas leituras retorna o valor aproximado da corrente. A figura 3.7 mostra o sensor utilizado no projeto, ele mede até $30A$ de corrente.

Figura 3.7: Modelo do sensor Hall utilizado no projeto.

O fato de ter que ser lido diversas vezes acarreta em dois problemas:

- Ele gera um atraso no controlador do motor, se o mesmo microcontrolador que for usado para fazer o controle fizer a leitura do sensor Hall;
- Para o sensor Hall funcionar como um sensor de fim de curso ele deve verificar um excesso de corrente sendo consumida pelo motor. Ou seja, quando o equipamento colide com um obstáculo que o impede de continuar se movendo o motor consome mais corrente para aplicar mais torque (conjugado) e superar essa limitação. Como o motor nunca vai conseguir superar isso ele irá consumir corrente até o seu limite ou até queimar, antes desse momento o sensor Hall deverá verificar que o consumo está excessivo, associando ao fato do equipamento ter chegado no limite e então desligar o motor. O problema deste método é que a menos que se tenha um motor que consuma menos corrente que o seu máximo para exercer torque no equipamento em qualquer situação nominal, em algum momento de situação normal de funcionamento o motor irá consumir máxima corrente e com isso o sensor Hall irá inadvertidamente desligar o motor.

Para resolver este problema, no momento o sensor Hall não foi implementado. Para garantir a segurança da planta foi implementado via *software* uma função que verifica quando o rastreador solar girou para ângulos maiores do que $\pm 70^o$, quando estes limites são atingidos o próprio microcontrolador desliga o motor. Esse limite de $\pm 70^o$ foi escolhido pois foi verificado que para além desses ângulos o sol já não está tão expressivo, ou seja, não há mais tanta incidência de radiação solar, não havendo tanta luz para captar, não havendo tanta perda de produção.

Porém, o sensor Hall será implementado em projetos futuros através de dois microcontroladores: um que faça a segurança e demais funções periféricas (como a aplicação do sensor Hall) e outro que faça o controle da planta através do controlador PI.

3.3.4 Sensor de luminosidade: pirânometro e fotodetector

Existem dois sensores de luminosidade neste projeto: o piranômetro e o fotodetector. A proposta de utilizar os dois é ter uma referência da eficiência do rastreador.

O fotodetector Figura 3.8, que funciona como um diodo LDR (*Light Dependent Resistor*), é usado na fibra óptica. Uma única fibra óptica é cortada, a luz atravessa a lente de Fresnel e o filtro UV, entra na fibra óptica e vai direto para o detector, conforme a Figura 3.17. Dessa forma, se tem informações sobre a calibração e o correto posicionamento do rastreador. Na Figura 3.17 o sensor está sendo calibrado usando uma luz de alta intensidade através de uma fibra óptica, igual a que será na implementada no projeto.

Figura 3.8: Sensor de luz (fotodetector) modelo SFH250V.

O pirânometro é um sensor de luz que será usado para medir radiação ambiente. Através deste sensor será possível obter informações sobre o clima de forma que futuramente o microcontrolador possa comparar a radiação que está sendo obtida com o fotodetector e a radiação obtida com o piranômetro e decidir se está havendo alguma descalibração no rastreador ou se são eventos climáticos que impedem maior captação de luz solar.

Figura 3.9: Sensor de luz (piranômetro) modelo Apogee SP-110.

3.3.5 RTC (*Real-Time Clock*)

O RTC não é bem um sensor, mas um dispositivo eletrônico que usa a contagem de *clocks* em um cristal para saber quantos milissegundos se passaram. Uma vez

calibrada a data (dia, mês e ano) correta e a hora local correta, ele dispõe de uma bateria que impede que seja desligado em caso de perda de energia, dessa forma, caso a energia elétrica acabe e o sistema desligue, quando a energia retornar o controlador será capaz de saber qual a nova posição atual correta independente de quanto tempo tenha passado.

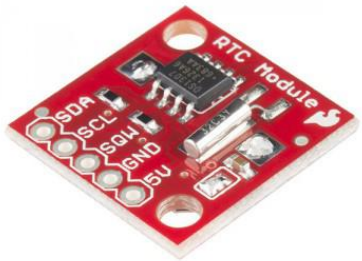


Figura 3.10: RTC.

O modelo do RTC usado é DS1307 da fabricante Sparkfun como pode ser visto na Figura 3.10.

3.3.6 Ponte H

A ponte H é um dispositivo capaz de reverter o sentido de rotação do motor, ele é composto de dois pares de transistor de forma que a combinação de acionamento deles faz com que o motor CC gire em um sentido ou em outro.

Figura 3.11: Ponte H, modelo: BTS7960.

Como o sentido de rotação e a velocidade do motor depende da polaridade da alimentação, usamos a ponte H para reverter esse sentido de rotação sem precisar chavear uma estrutura mecânica, como pode ser visto na Figura 3.12. Essa Figura é da ponte H para o acionamento de um motor de passo, usado na primeira versão do

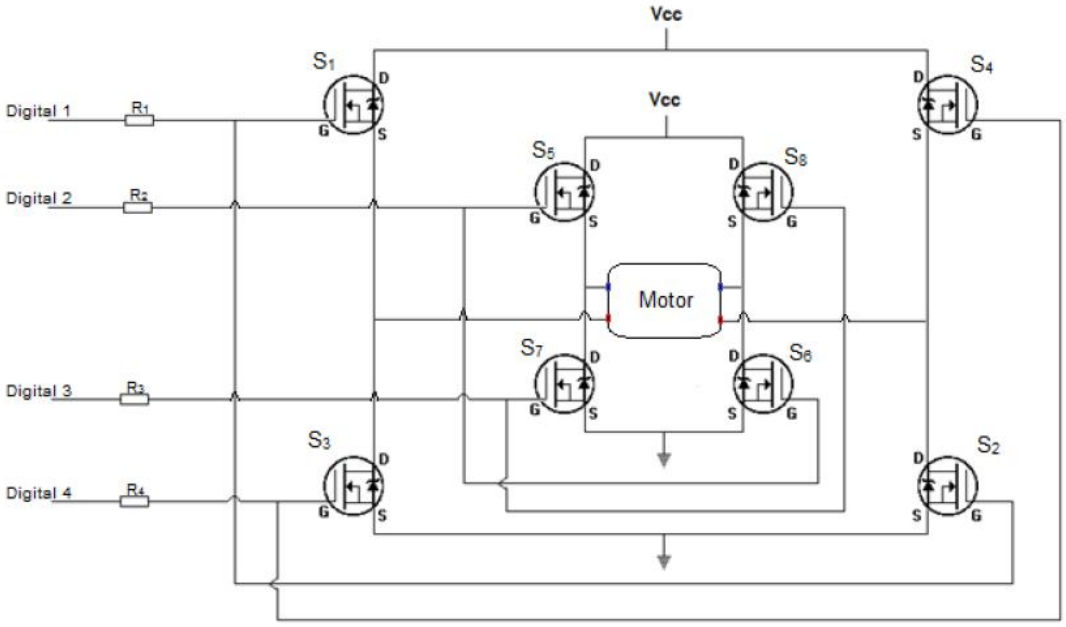


Figura 3.12: Ponte H de esquemático de funcionamento [10].

rastreador, mas seu funcionamento segue o mesmo principio do BTS7960 que pode ser visto na Figura 3.11.

Ao acionar o pino Digital 1, as chaves $S1$ e $S2$ são fechadas, permitindo o fluxo de corrente a partir da fonte de alimentação ao terra. Posteriormente, ao acionar o pino Digital 2, as chaves $S5$ e $S6$ são fechadas e, novamente, o motor se desloca. A seguir ao acionar o pino Digital 3, as chaves $S7$ e $S8$ são fechadas. Por fim, ao acionar o pino Digital 4, as chaves $S3$ e $S4$ são fechadas e o motor rotaciona no outro sentido. O processo então se repete e a velocidade de rotação do motor estará associada ao intervalo de comutação entre as chaves apresentadas. Quanto menor o tempo entre as comutações, mais rápida será a rotação do motor. Caso se deseje uma rotação no sentido oposto, basta realizar a comutação apresentada pelo parágrafo anterior no sentido inverso, isto é, começando a comutação a partir do pino Digital 4 até o pino Digital 1. É importante citar que este circuito apresenta, também, uma possível desvantagem. O projetista deve informar ao usuário deste circuito que os relés, ou chaves, existentes na mesma perna da ponte H, isto é, $S1$ e $S3$ ou $S4$ e $S2$ ou $S5$ e $S7$ ou ainda $S8$ e $S6$, jamais devem ser acionados simultaneamente. Isto provocaria um curto circuito direto na fonte de alimentação do motor, ocasionando, possivelmente, a queima da fonte além de uma abertura nas trilhas do circuito impresso. E mais, caso os relés não apresentem isolamento elevado entre a parte de controle e a parte de potência, é possível que o circuito de controle também seja danificado conforme demonstrado por [10].

Para fazer o controle de velocidade do motor foi usado o conjunto Arduino e ponte H com método PWM (Modulação por Largura de Pulso ou *pulse width modulation*,

em inglês). Podemos pensar nesse método como uma chave que liga e desliga um circuito, quando ligamos a chave estamos fornecendo 100% da tensão e de potencia à carga, já quando a chave esta aberta a tensão é nula e assim a potência é 0. Mas se controlamos o tempo que a chave fica ligada e desligada podemos controlar a potencia média entregue a carga, por exemplo: fazermos a chave ficar 50% do tempo ligada e 50% do tempo desligada, em média temos 50% do tempo com corrente e 50% sem. Portanto, a potência média aplicada na carga é a própria tensão média, ou seja, 50%. Um exemplo pode ser visto na figura 3.13 onde variamos a entrada analógica do microcontrolador Arduino para variar o tempo de forma a variar a tensão fornecida.

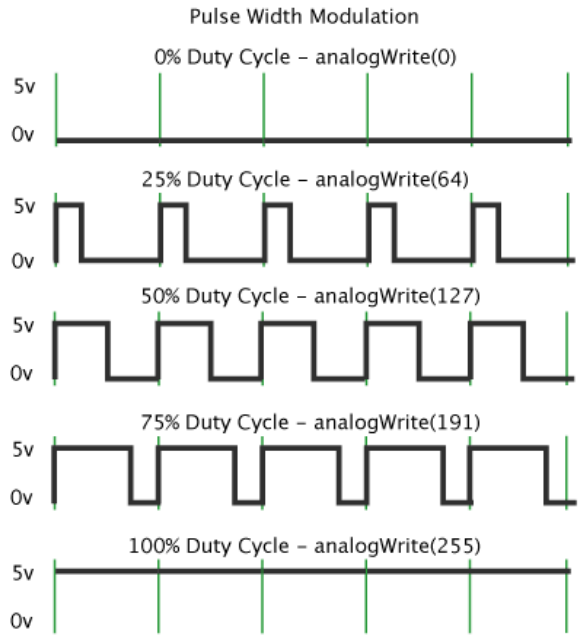


Figura 3.13: PWM via porta analógica no Arduino [11].

3.3.7 Motor CC

O motor CC (corrente contínua, ou DC de *Direct Current*, em inglês) é um dispositivo elétrico-mecânico composto de bobinas capazes de girar um eixo. Apesar de sua grande aplicação na indústria ele não é capaz de parar em um posição específica ou determinar o quanto de movimento fará como o motor de passo, onde cada alimentação girará uma quantidade de graus tão pequenas quando mais precisa forem as engrenagens de sua estrutura interna.

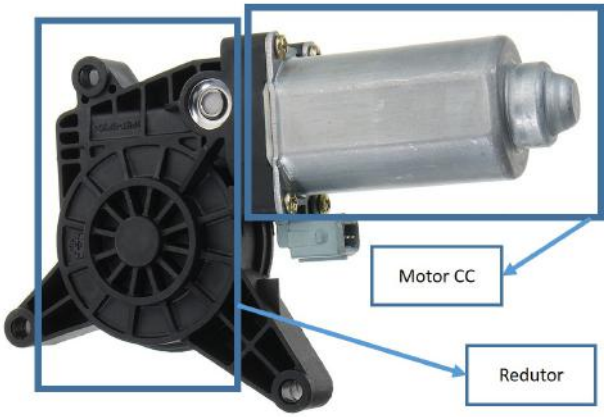


Figura 3.14: Motor CC utilizado no projeto. Informações técnicas no Anexo IV.

O modelo utilizado foi um Bosch que é comumente usado para movimentar vidros laterais de automóveis como o que pode ser visto na Figura 3.14. O motor utilizado é composto de um redutor. O redutor (equivalente ao mostrado na Figura 3.15), através de uma relação de engrenagens, altera o passo de resolução da rotação do motor. O redutor foi usado pois devido a forma de construção do rastreador, para o seu correto funcionamento é necessário girar em 90^o o eixo de rotação do motor como pode ser visto na Figura 3.15.

3.3.8 A zona morta no redutor do motor

Durante o desenvolvimento do projeto foi observado que a aplicação do redutor no motor gerou um efeito de zona morta no rastreador.

Problema: o motor CC quando desenergizado deveria ter zero movimento em seu eixo, conforme foi constatado nos testes feitos em laboratório, o motor com o redutor disponível neste projeto apresenta folga, que foi verificado serem em função das engrenagens que ligam o eixo do motor com o do redutor. Essa folga gera o que chamamos de zona morta no motor e dentre as principais implicações dela no controle é a dificuldade em posicionar corretamente o sistema.

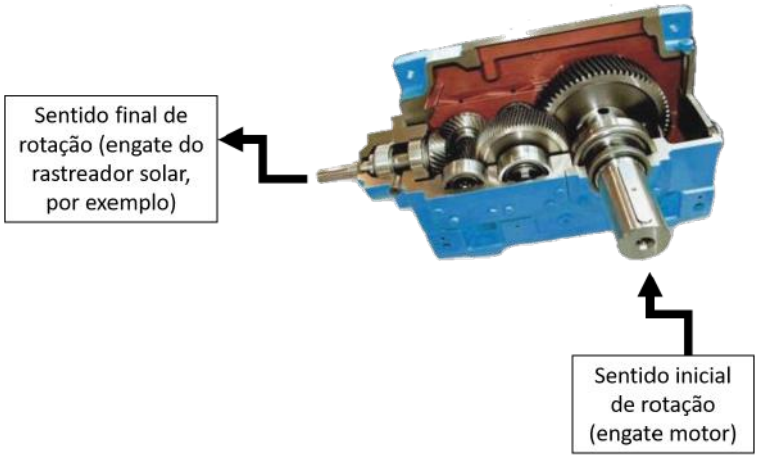


Figura 3.15: Exemplo de redutor. O redutor é um mecanismo que altera o eixo de rotação do motor, no caso deste projeto, o eixo de rotação é rotacionado em 90^o.

O sistema nunca fica posicionado corretamente pois cada vez que o controlador PID envia uma resposta que deveria fazer o motor parar no local correto, quando o motor é desenergizado, ele, por inércia, desloca-se um pouco mais dentro dessa folga. Fazendo com que o controlador PID nunca consiga atingir o ponto ótimo e esteja sempre tentando corrigir o motor e consequentemente, além de gastar energia, nunca alcança a estabilidade conforme pode ser visto na Figura 3.16.

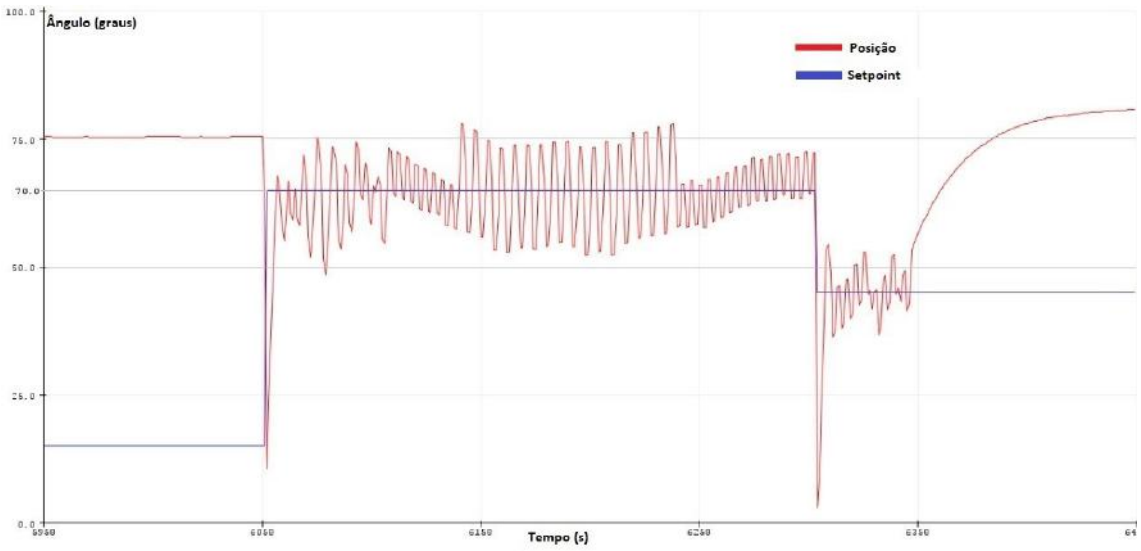


Figura 3.16: Instabilidade devido a zona morta, testes de bancada.

Solução: apesar dos métodos citados para compensação de zona morta, alguns fatores foram levados em consideração na busca da solução para este problema:

- O custo computacional;
- A dificuldade para ter informações precisas sobre o tamanho da zona morta;

- A real necessidade de uma solução mais complexa.

Levando esses pontos em consideração foi verificada qual seria a tolerância aceitável para este projeto de forma que não houvesse perda de intensidade de luz solar no feixe de fibra óptica. Para fazer esses testes, primeiro foi verificado no rastreador, com o auxílio de uma lanterna se ocorreria o aparecimento de sombra no feixe de fibra óptica, observou-se que para pequenas variações não haveria perdas significativas. Então para um estudo mais minucioso foram usado uma fibra óptica, um fotodetector e uma lâmpada de alta intensidade, conforme pode ser visto na Figura 3.17.

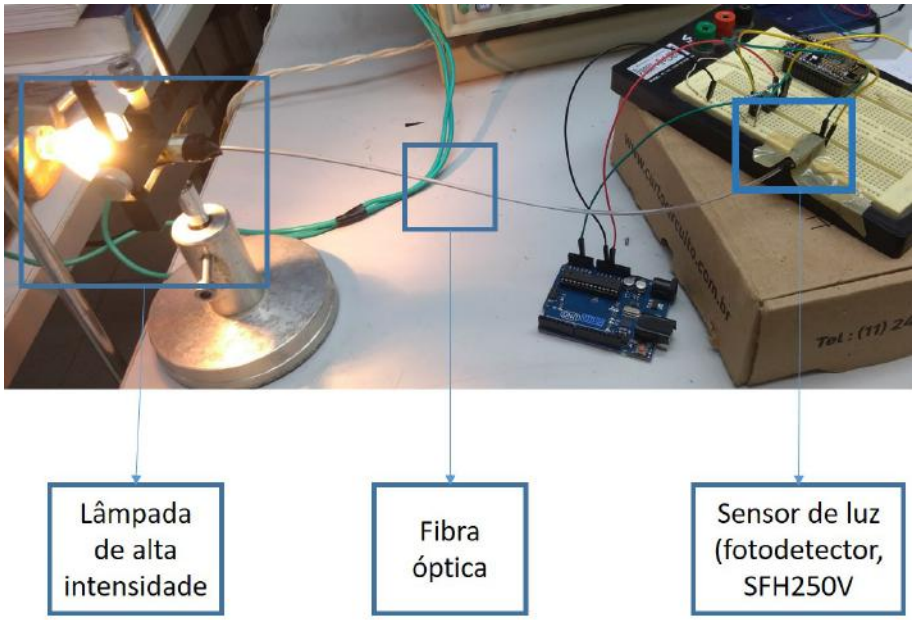


Figura 3.17: Medida da tolerância de erro da perpendicularidade do Sol com o feixe de fibra óptica.

Os testes foram feitos movimentando de forma angular a lâmpada e observando a resposta do fotodetector. Observou-se que para variações próximas de 1^o, não havia perdas significativas da luminosidade no feixe de fibra óptica. Testes feitos posteriormente permitiram usar uma margem de erro de $\pm 0{,}5^o$.

Para tal feito, foi implementado computacionalmente o seguinte código:

```
1 double erro = 0.5;
2 double Posicao_Referencia,posicao;
3 void Pare_o_Motor();
4
5 void loop(){
6    if ( Posicao_Referencia + erro <= posicao){
```

```
7           Pare_o_Motor();
8     }
9     else if ( Posicao_Referencia - erro >= posicao){
10          Pare_o_Motor();
11    }
12 }
```

Listing 3.1: Código exemplo de funcionamento do sistema de margem de erro do rastreador

No código acima 3.1[1] é definido um erro de 0,5^o, esse erro corresponde ao quanto fora da posição de referência o motor pode ficar. As variáveis $'Posicao\_Referencia'$ e $'posicao'$ correspondem ao *setpoint* e posição real do motor, respectivamente. Então o Algorítimo inicializa (ao entrar em *void loop*), no *if* é verificado se a posição do motor é menor ou igual ao *setpoint* mais o erro, caso positivo, o algorítimo irá desligar o motor. No *else if* é verificado se a posição do motor é maior ou igual ao *setpoint* menos o erro, caso positivo, o algorítimo irá desligar o motor. Dessa forma temos uma margem de erro superior e inferior para a o *setpoint* permitindo que mesmo com a folga (zona morta) o motor estabilize em torno da posição desejada.

3.3.9 Microcontrolador Arduino e nodeMCU

O microcontrolador Arduino é conhecido por sua simplicidade de uso e possibilidade de trabalhar com a linguagem C++ para programá-lo. Por ser ser muito difundido ele foi inicialmente preferível pois detém vasta biblioteca disponível gratuitamente para os sensores usados neste projeto. Neste projeto foi utilizado o Arduino UNO apresentado na Figura 3.18.

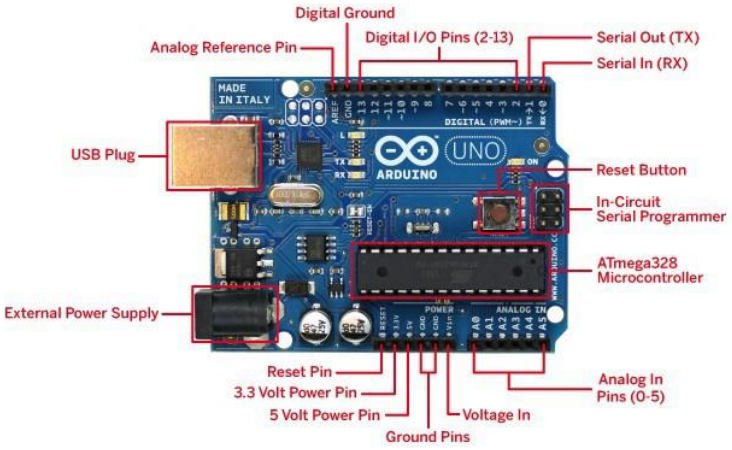


Figura 3.18: Microcontrolador Arduino UNO.

[1]Este algorítimo é um resumo do funcionamento, o código real utilizado pode ser encontrado no anexo VI.

Assim como na primeira versão do rastreador existe o projeto de criar um sistema de monitoramento, para isso já foi planejado a implementação do NodeMCU, Figura 3.19, pois ele dispõe de um *chip* Wi-Fi integrado em sua placa, que facilitaria o envio de informações de forma remota para um computador externo. Pensando no futuro já foi planejado sua utilização e implementada sua construção na placa de circuito impresso que foi usada como placa mãe do projeto como pode ser visto na Figura 3.20. Apesar de ter sido planejada ele apenas foi testando não tendo uma aplicação específica, ainda, para esta fase do projeto.

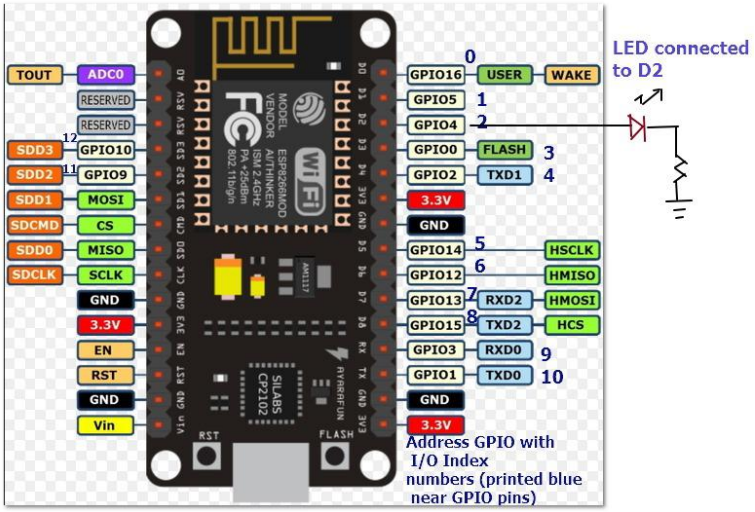
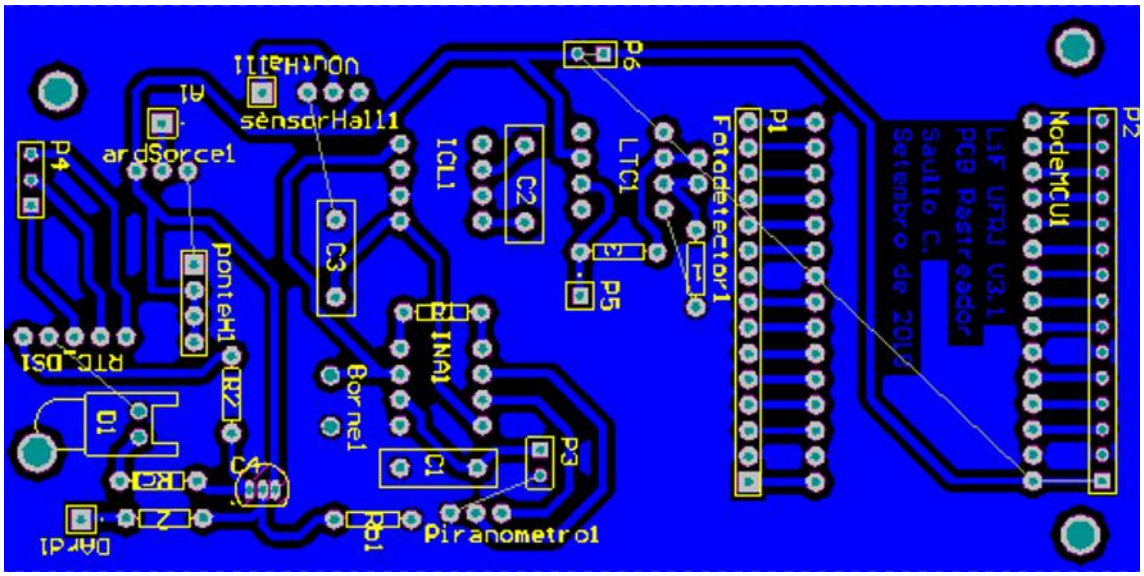


Figura 3.19: Microcontrolador nodeMCU.

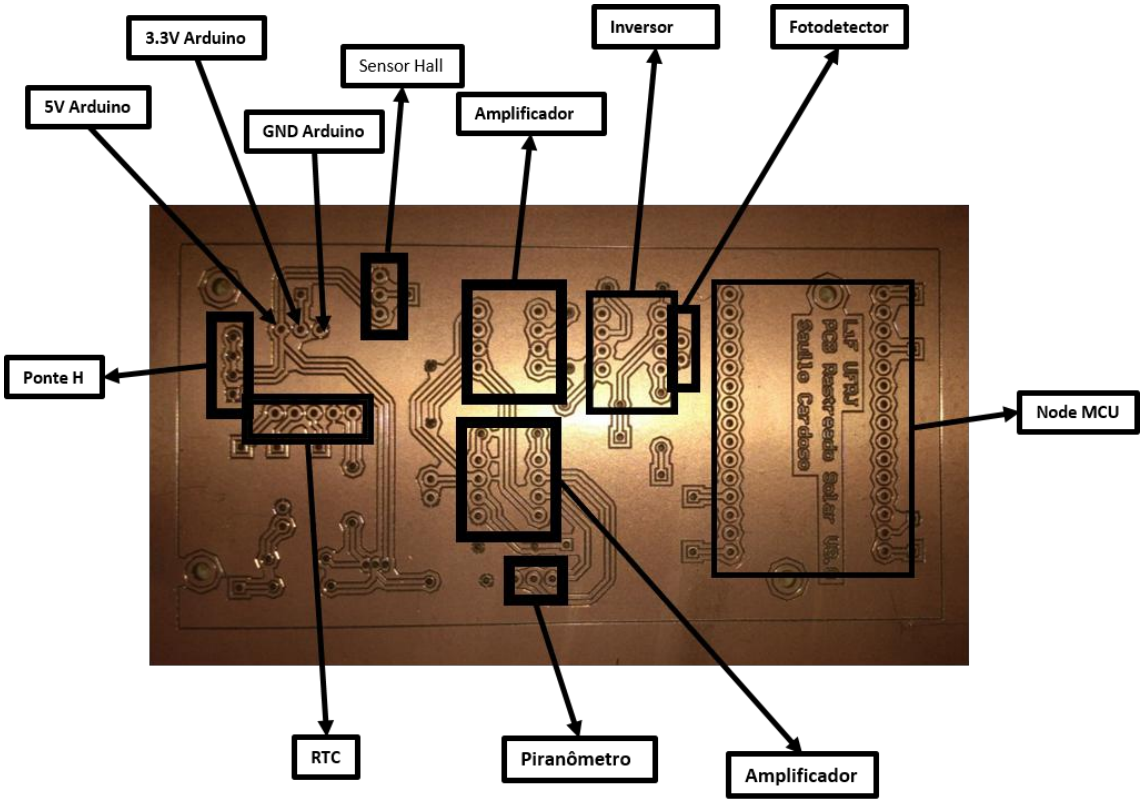
3.3.10 PCB (placa de circuito impresso)

As placas de circuito impresso (PCB) são placas utilizadas para a soldagem de componentes eletrônicos. Para esse projeto foi desenvolvida no *software Altium*®14 a placa que funciona como uma espécie de placa mãe do rastreador solar, ela pode ser observada conforme Figura 3.20a.

As conexões eletrônicas usadas para desenvolver a PCB da Figura 3.20 estão apresentadas na figura do anexo III, alguns dos componentes foram desenhados para este projeto e, portanto, não se encontram na biblioteca padrão do *software Altium*®14. Uma vez projetada a placa foi utilizada uma fresadora do tipo PCB Proto 2, desenvolvida pela fabricante brasileira TTP Indústria Mecânica. A função desta máquina é realizar a confecção da PCB desenvolvida, de forma que se obteve a placa da Figura 3.20b.



(a) Diagrama eletrônico da placa PCB em CAD através do *software Altium*® 14.



(b) PCB após confecção e seus principais elementos.

Figura 3.20: PCB impressa

3.3.11 Diagrama elétrico

O esquemático elétrico de ligação elétrica do rastreador solar pode ser visto na Figura 3.21. Ele mostra como foram feitas as conexões elétricas e os sinais de dados entre os dispositivos.

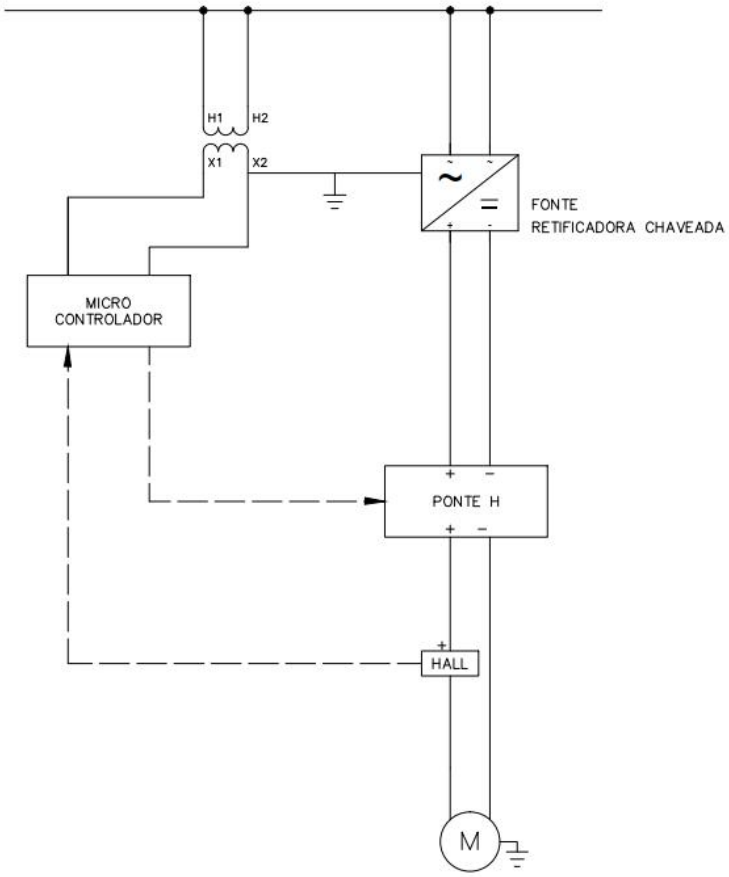


Figura 3.21: Esquemático elétrico. Verificar Anexo II.

NA Figura 3.21 uma fonte CC fornece corrente para a ponte H que controla o sentido de rotação e a velocidade de acordo com o especificado pelo microcontrolador; o sensor Hall mede a corrente que o motor está consumindo e envia a informação para o microcontrolador que pode decidir por parar o motor através da ponte H.

3.3.12 Caixa de componentes do rastreador

Uma vez desenvolvida a PCB, os componentes eletrônicos foram soldados e a placa parafusada junto dos demais dispositivos elétricos em uma placa de acrílico. Essa placa de acrílico foi disposta dentro de uma caixa (IP-65). Alguns prensa-cabo foram usados para conectar os cabos que vêm do ambiente externo com os de dentro da caixa a fim de diminuir tempo e dificuldade de manutenção como problemas nas placas caso algum fio seja puxado e entrada de água e poeira. O resultado final é mostrado na Figura 3.22. Na Figura 3.23 podemos ver a vista frontal da caixa, com os prensa cabos para cada um dos cabos necessários.

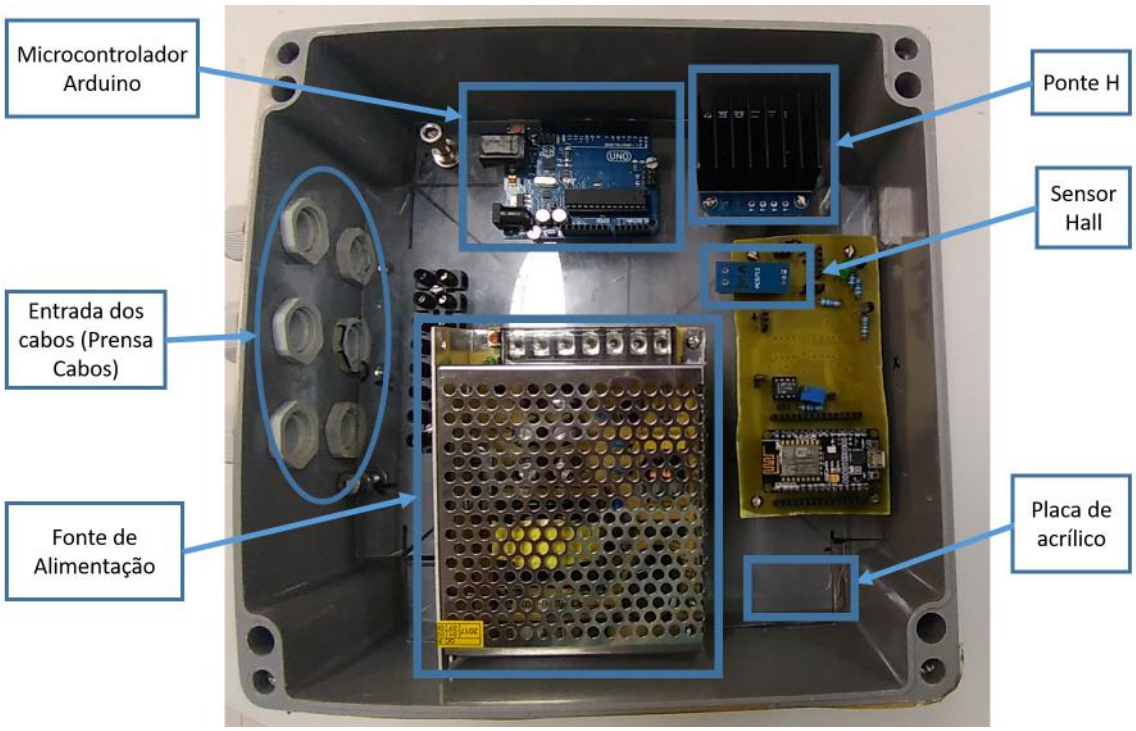


Figura 3.22: Dispositivos instalados no interior da caixa. Vista superior.

Figura 3.23: Vista frontal da caixa de componentes.

3.4 Dificuldades encontradas

Neste capítulo serão abordados os problemas encontrados durante a fase de desenvolvimento do projeto. A importância deste capítulo se dá pelo fato dele demonstrar o raciocínio e a justificativa das implementações feitas no projeto de forma que traga mais clareza sobre o resultado final e como ele foi atingido.

3.4.1 Função de transferência do motor

Problema: conforme foi discutido em sessões anteriores, a FT do motor pode ser usada para calcular a calibração do PID de forma a se ter a resposta ao impulso desejado. Porém, para se ter uma FT apropriada seria necessário acesso aos valores das constantes características do motor, valores esses que não foram encontrados no *datasheet* do motor e apesar das inúmeras tentativas de contato não foram dispostos pela fabricante Bosh.

Solução: para contornar esses problemas algumas abordagens foram feitas para calcular a FT do motor. A primeira delas foi a aproximação da função de transferência do motor, para um sistema de primeira ordem. Apesar de se ter conhecimento de que este não é um sistema de primeira ordem, pretendia-se entender mais acerca do motor de forma que fosse possível investigar melhor como calcular as constantes do controlador PI.

Para tal, foi seguido os procedimentos de caracterização de motor elétrico apresentado nas referências [16], [21] e [22]. Dessa forma foi utilizado um tacômetro MDT-2244B para medir a velocidade de rotação do motor em rotação por minuto (RPM) e um multímetro para medir sua tensão em volts como pode ser visto na Figura 3.24 de forma a obtermos o valor de κ do motor.

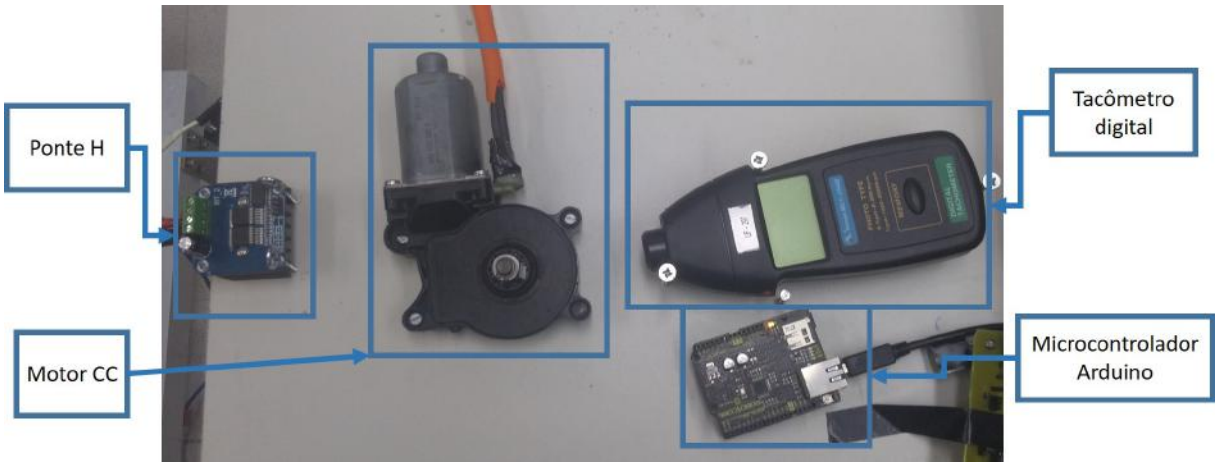
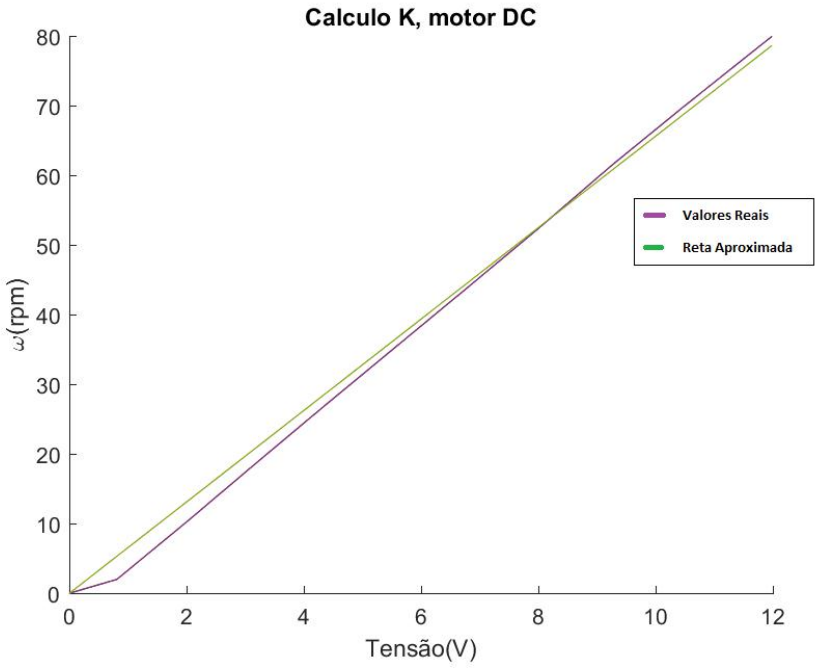
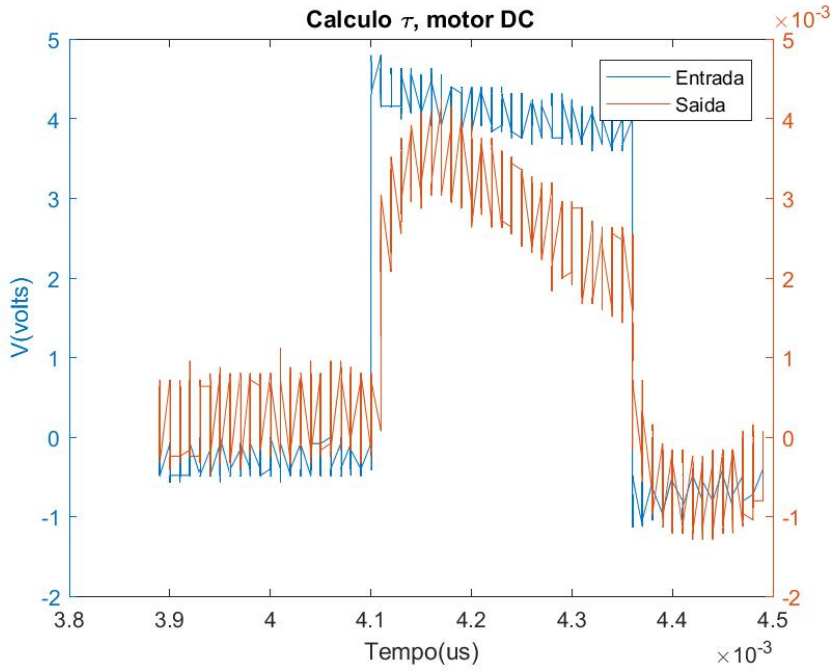


Figura 3.24: Experimento para estimar a FT do motor.

Para o cálculo do τ foi gerado um degrau através do Arduino nos terminais do motor e adquirido os dados, através do osciloscópio cuja as informações e gráficos foram analisadas usando o *software* $MatLab^{®}$. A Figura 3.25 demonstra os resultados obtidos pelas medidas feitas.



(a) Gráfico do cálculo do κ.



(b) Gráfico de cálculo do τ.

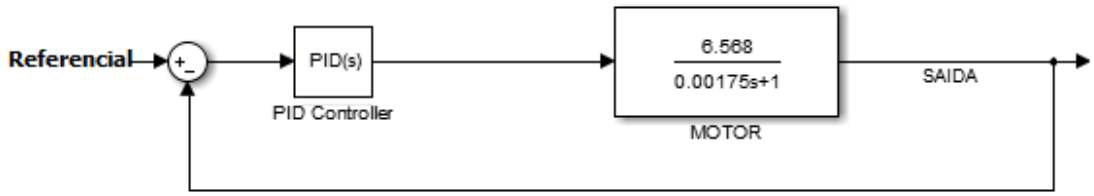
Figura 3.25: Cálculo da FT do motor: cálculo de κ e τ.

Dessa forma, conforme a Tabela 3.1, temos os parâmetros para a FT do motor de primeira ordem.

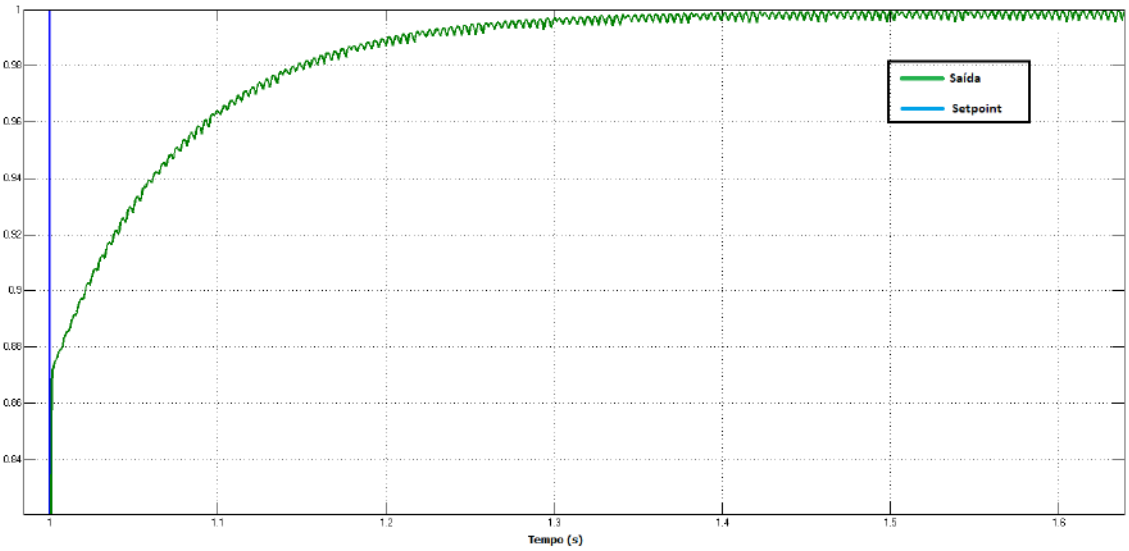
Tabela 3.1: Valores dos parâmetros do motor.

$\tau(ms)$	0,00175
$\kappa(V/V)$	6,568

Na Figura 3.26 foi testados os valores obtidos na Tabela 3.1 através de uma simulação. Observando os parâmetros do motor e usando uma simulação feita no *software Simulink*®, foi possível observar sua resposta ao degrau de forma a investigar valores aproximados para as constantes do controlador PI.



(a) Diagrama de blocos.



(b) Resposta ao degrau.

Figura 3.26: Resposta ao degrau no motor simulado sem ruído.

Na Figura 3.26 temos a resposta ao degrau da função de transferência do motor usando os parâmetros encontrados anteriormente na experimentação. Para a simulação os valores encontrados para os parâmetros do PI podem ser vistos na tabela 3.2. Esses valores foram encontrados zerando inicialmente os valores de K_I e K_D e então variando o valor de K_P até a resposta apresentar um valor aceitável. Uma vez atingido a saturação da resposta ao K_P, foi variado o parâmetro K_I até o sistema fornecer a resposta desejada. Como não foi necessário ter uma resposta mais rápida

K_D foi mantido zero, de forma que foi obtido os valores da Tabela 3.2.

Tabela 3.2: Parâmetros do controlador PID na simulação do motor.

Parâmetros	valores simulação
K_P	1,00
K_I	15,00
K_D	0,00

Na figura 3.26b pode-se ver que a saída do controlador oscila. Isso ocorre pois na tentativa de ajustar a posição do motor de forma a atingir o *setpoint* o controlador PI faz o motor girar no sentido horário e no sentido anti-horário várias vezes.

Os valores da Tabela 3.2 foram usados como uma abordagem inicial para encontrar os valores dos parametros do controlador PI no rastreador solar.

3.4.2 O atraso no sensor de posição

Problema: para evitar danos no equipamento foi implementado, via *software*, uma trava impedindo o motor de girar mais de $\pm 70^o$. Portanto, quando o sensor detectar que o motor girou para ângulos fora dessa margem de tolerância ele deverá parar. Apesar dessa trava percebeu-se que nem sempre o motor parava, principalmente quando a velocidade do motor era alta. Isso fazia com que o motor girasse indefinidamente enquanto o sensor de posição apresentava valores atrasados com relação ao real.

Uma vez observado esse fenômeno, concluiu-se que o tempo de leitura do sensor era inadequado para a velocidade com que o sistema funcionava.

Solução: para resolver esse problema foi inserido um atraso no motor. Na verdade, esse atraso é um chaveamento muito rápido no qual sua lógica de funcionamento é apresentado na Figura 3.27.

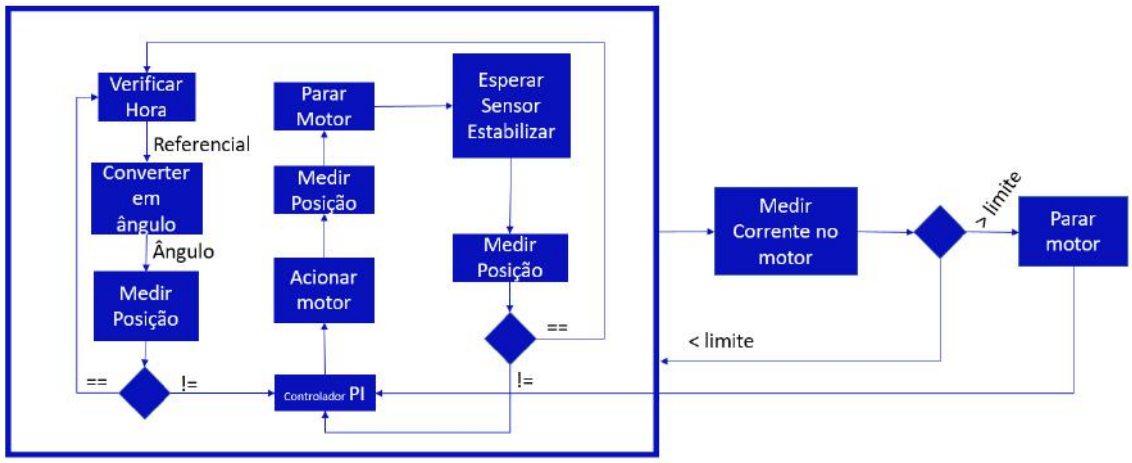


Figura 3.27: Diagrama lógico de funcionamento.

Na Figura 3.27 pode ser visto a lógica de funcionamento do rastreador. Inicialmente ele verifica a hora e converte a hora em ângulo, para saber a posição angular do sol. Então através dos sensores acelerômetro e giroscópio ele compara a posição do rastreador com a do sol, se as posições convergirem ele volta para o inicio e fica verificando a posição do rastreador e a do sol.

Quando a posição do sol e a do rastreador forem diferentes (não estiverem paralelos um ao outro) ele aciona o controlador PI que irá acionar o motor para movimentar o rastreador, como temos um retardo na leitura do sensor com relação a velocidade de rotação do motor, o microcontrolador desliga o motor por um tempo (até que o sensor estabilize) e então medi a posição do rastreador, caso a posição esteja diferente da posição desejada é acionado o controlador PI para novamente mover o rastreador.

Isso garante que o sensor estabilize antes do motor voltar a girar. Para saber quantas leituras eram necessárias foram feitos testes empíricos de forma a se obter um valor ideal que nem atrase muito o sistema e nem faça o sensor apresentar um valor falso. Para este projeto foi identificado o valor de 150 leituras para cada chaveamento do motor.

Esse ciclo se repete infinitamente, na Figura 3.27 existe, também, um limitador que mede a corrente do motor e o para caso ela exceda um valor pré-determinado, pois isso indica que o rastreador atingiu o seu limite de movimento e não deve mais se mover. Para esta etapa do projeto não está sendo medida a corrente pois não foi instalado o sensor Hall devido a questões já apresentados, mas foi implementado, via software um limitador que desliga o motor caso o rastreador atinja gira para ângulos além de $\pm 70^o$.

Capítulo 4

Resultados

Aqui será apresentado os resultados encontrados após a aplicação das soluções mencionadas e comparado o esperado com o obtido de forma a atestar o sucesso do projeto.

4.1 Diagrama de blocos final

Uma vez implementadas as devidas correções, o diagrama de blocos do sistema passou a ser igual ao da Figura 4.1.

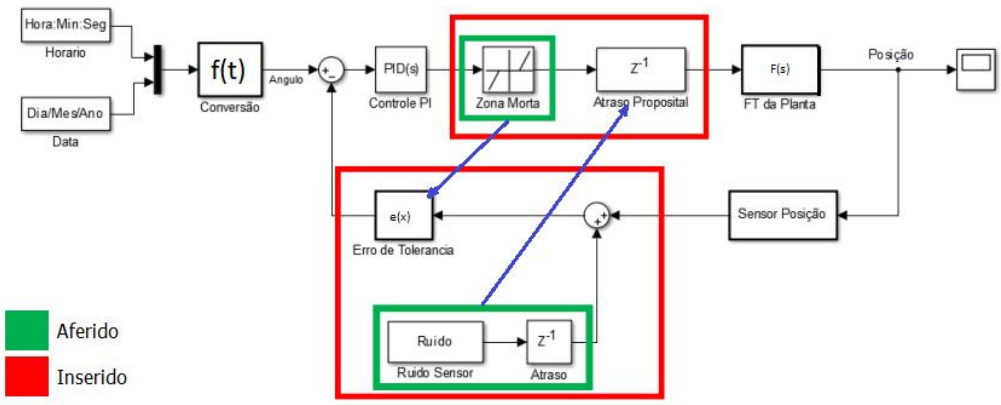


Figura 4.1: Diagrama de blocos final.

Na Figura 4.1 podemos ver as soluções anteriormente citadas aplicadas de forma a entender como a lógica de controle do projeto funciona. Com essa lógica foi atestado os resultados apresentados a seguir.

4.2 Medidas finais

Pela Figura 4.2 devemos observar que as soluções aplicadas foram bem mais simples do que àquelas propostas na literatura como as do livro [5], devemos levar em consideração que segundo o *Project Management Body of Knowledge* (ou PMBOK) uma solução de projeto deve estar dentro da tríade do gerenciamento (custo, qualidade e tempo). Neste projeto a solução de engenharia encontrada não deveria prejudicar a qualidade, então, pensando até mesmo em uma questão de tempo de implementação, observou-se que as soluções atendiam as necessidades do projeto, sem prejuízo.

A implementação de mecanismos mais complexos, implicaria na mudança tanto da parte mecânica quanto da parte eletrônica. Então, buscando atender aos objetivos com o material disposto as soluções encontradas atenderam às expectativas, principalmente com relação a viabilidade do projeto.

Com isso, observa-se na Figura 4.2 os dados tomados ao longo de um dia.

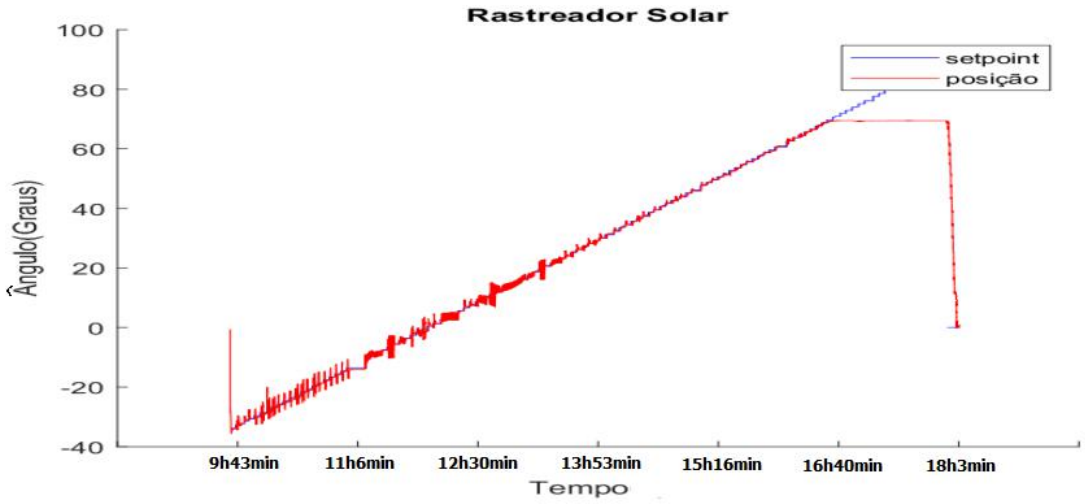


Figura 4.2: Dados adiquiridos ao longo de um dia.

Na Figura 4.2 a partir do ponto $18h3min$ vemos que o sinal do rastreador volta para zero enquanto a referência continua seguindo em frente, isso acontece pois para ângulos muito altos já está de tarde quase noite e a quantidade de luz solar já não se torna expressiva. Então por uma questão de segurança, opta-se por enviar o rastreador para o ponto de partida afim de iniciar um novo ciclo no próximo dia, novamente.

4.3 Análise da leitura do sensor de posição

Conforme pode ser visto na Figura 4.3 foram feitos testes de medidas para o tempo de leitura do sensor, esse tempo de leitura foi medido em número de leituras do sensor pois facilita a programação e o entendimento.

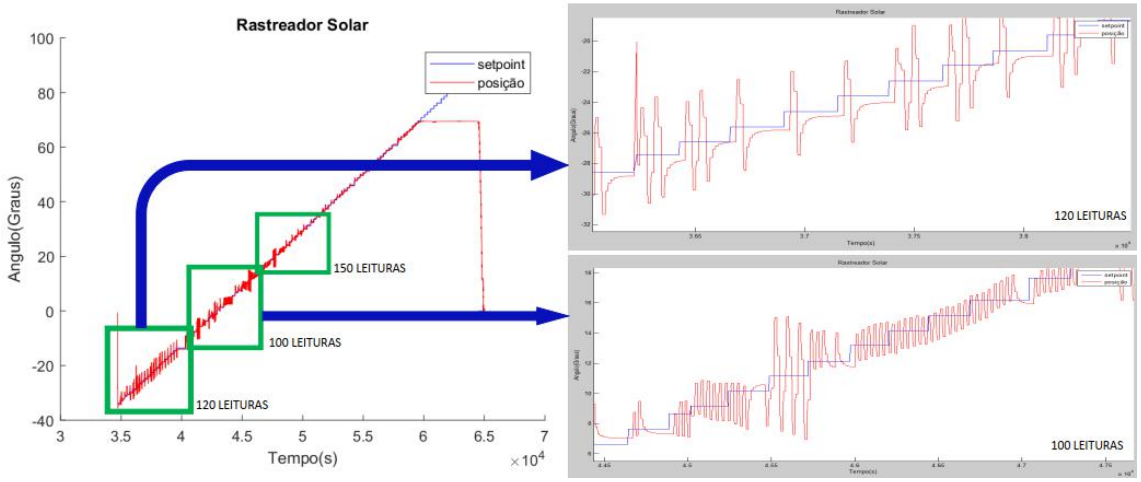


Figura 4.3: Análise do controlador PI para menos de 120 leituras, sistema instável.

Os teste empíricos realizados demonstraram que para números de leituras acima de 120 começava-se a ter um sinal mais estável, estes mesmos testes demonstraram que 150 leituras, conforme pode ser visto na Figura 4.4, atendia as necessidades e que para valores acima de 150 leituras o sistema ficava muito lento, prejudicando o desempenho da planta.

4.4 Análise do controlador PI na planta

Conforme podemos ver na figura 4.4 para 150 leituras a planta apresenta comportamento adequado as necessidades do projeto. Como pode-se observar o comportamento não é uniforme ao longo dos ângulos, isso ocorre pois o sistema mecânico muda sua geometria, e consequentemente seu centro de massa e torque, ao longo do tempo o que em um caso mais extremo precisaria de algum controle adaptativo[1]. Como o sistema proposto é lento frente as necessidades não foi visto uma real necessidade de um estudo mais complexo, a priori.

Os valores utilizados para as constantes do PID estão apresentados na tabela 4.1

Tabela 4.1: Constantes do controlador PID, real e simulado.

Parâmetros	**Real**	**Simulado**
K_P	0,19	1,00
K_I	10,80	15,00
K_D	0,00	0,00

Na tabela 4.1 podemos ver os valores reais utilizados para fazer o rastreador funcionar e obter o resultado demonstrado na Figura 4.2 e comparar com os valo-

[1]Controle adaptativo pode ser definido como uma técnica de controle que possui a capacidade de mudar seu comportamento de acordo com as modificações da dinâmica de uma planta ou por distúrbios que afetam este sistema [23].

res resultantes da simulação. Os valores da simulação e real se mostram próximos demonstrando como o cálculo da função de transferência do motor e da simulação, juntos das soluções empregas, auxiliaram bem na busca dos parâmetros do controlador PI que permitisse o rastreador funcionar conforme desejado.

Na tabela 4.1 o valor de K_D foi escolhido zero, pois não há interesse em aumentar o tempo de resposta, a priori, da planta. O PID foi calibrado variando suas variáveis e verificando a que apresentava melhor resultado usando como ponto de partida os valores encontrados na simulação do motor. Outros métodos de calibração poderiam ter sido empregados mas para este presente momento não verificou-se a necessidade.

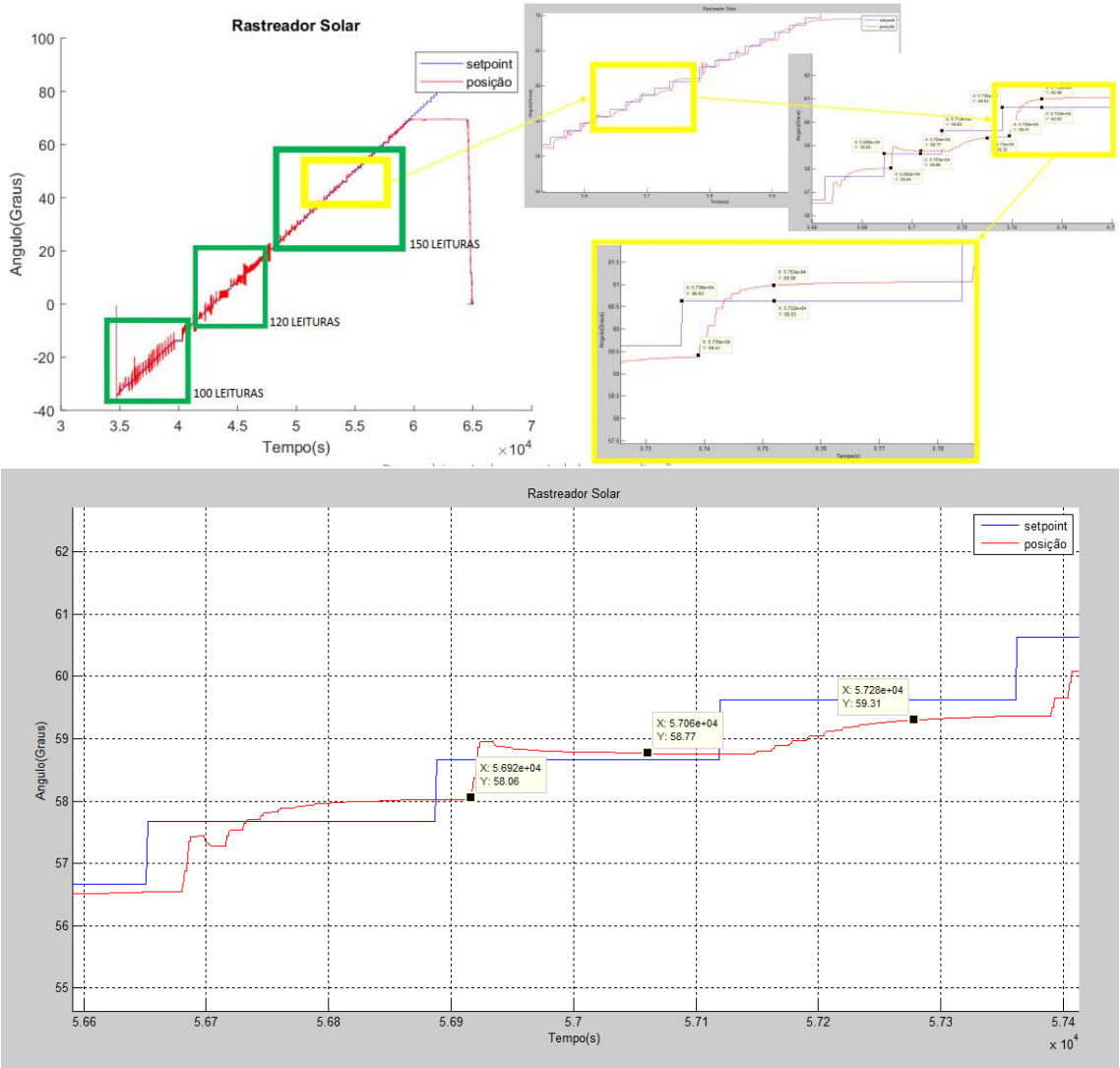


Figura 4.4: Análise do controlador PI para 150 leituras, sistema estável.

4.5 Análise dos resultados

Foram coletados alguns valores da figura 4.4 e conforme pode ser visto na tabela 4.2 os valores médios encontrados após a calibração do PID e a correta calibração

do tempo de leitura do sensor estão de acordo com o que foi especificado para o projeto.

Tabela 4.2: Valores encontrados.

Referencial (Grau)	Posição (Grau)	Erro (Grau)	Tempo Acomodação (Minuto)	*Overshoot* (Grau)
58,66	58,77	0,11	2,30	0,33
59,63	59,32	0,11	2,60	0,00
60,63	60,96	0,36	2,66	0,00
Especificado:	-	$< 0{,}50$	$< 4{,}00$	0,00

Os valores médios encontrados demonstram que o erro na medida estão em conformidade com o tolerado. Os valores de *Overshoot* variam, dependendo do momento angular da planta, não sendo tão uniforme como poderia se esperar do comportamento em outras plantas, para um estudo mais exato deveria ser feito um estudo mecânico do rastreador. Além disso, deve-se levar em consideração que a planta está sujeita a intemperismo e ações externas que vão desde vibrações no solo até vento e isso pode gerar variações no desempenho do PID na planta.

O tempo de acomodação esperado deveria ser entre 3 e 4 minutos, como foi obtido em media 2,5 minutos o pretendido foi atendido, podendo ser aperfeiçoado para projetos futuros. O tempo de acomodação de 4 minutos foi calculado na primeira versão do rastreador [10], esse tempo foi o observado como o necessário para que houvesse diferença de incidência perceptível de intensidade de luz solar na fibra óptica.

Durante os testes com a planta foi provocado, propositalmente, ruídos no rastreador movimentando-o bruscamente. Tirando ele da posição correta, verificou-se que o mesmo se reposicionava corretamente.

Capítulo 5

Conclusões

Com os resultados apresentados observa-se que o método atende as necessidades do projeto. O tempo de acomodação ficou abaixo dos 4 minutos pretendidos, sem grandes *overshoots*. Apesar de em teoria de controle desejarmos ter o mínimo de atraso no controlador, o método aplicado obteve um bom tempo-benefício isso é importante pois através das soluções empregadas não houve a necessidade de trocar os equipamentos dispostos ou partir para uma solução que aumentasse os custo de produção e manutenção do Rastreador Solar.

5.1 Comparação entre as versões do rastreador

Segundo [10] a primeira versão do rastreador solar usava um motor de passo, que apesar de funcionar muito bem fazia com que o sistema acumulasse um erro com o tempo. Esse erro começava com $\approx 2^o$ e no final do dia atingia $\approx 15^o$ ou 20^o, na versão deste projeto esse erro não existe, pois qualquer erro pode ser diminuído, ou até mesmo zerado, melhorando o controlador PI e como foi apresentado o erro é aleatório e mínimo além de não acumular com o tempo.

Na tabela 5.1 temos a comparação da primeira versão do rastreador solar, que pode ser visto na referência [10], com a deste projeto.

Tabela 5.1: Comparação entre as mudanças desenvolvidas nas duas versões do rastreador solar. Os preços e locais de compra podem ser vistos no anexo V.

Equipamento Anterior	**Versão Anterior**	**Preço (*R*$)**
Sensor de posição	ADXL362	60,00
Sensor de fim de curso	dois *Switches*	2,00/cada
Ponte H	Relé CPC 1090J	256,00 (8 unid.)
Tipo motor	Passo	740,00
Controlador	ON/OFF	-
Estrutura	Desing inicial	-
Fibra óptica	Plástica	250,00/m
Equipamento novo	**Versão Nova**	**Preço (*R*$)**
Sensor de posição	MPU6050	20,00
Sensor fim de curso	Sensor Hall	20,00
Ponte H	BTS7960	100,00
Tipo motor	DC (CC)	40,00
Controlador	PI	-
Estrutura	Desing reformulado	-
Fibra óptica	Plástica	250,00/m
	Custo total novo:	*R*$180,00
	Custo total antigo:	*R*$1060,00

Portanto, observamos que o custo caiu 83% com relação ao modelo anterior, tornando o projeto bem mais viável economicamente.

Observações sobre a Tabela 5.1:

- As fontes dos preços dos equipamentos são apresentados no anexo V (Referência da Precificação de componentes) podendo variar do momento da compra para o momento da leitura;

- O custo da fibra óptica não foi levado em consideração no cálculo dos custos da Tabela 5.1, mesmo estando indicado na tabela para efeitos de curiosidade. O seu valor não foi levado em consideração pois o custo depende de quantos metros serão utilizados durante a instalação e não acrescentaria muito na comparação dos custos vistos que nos dois projetos a fibra óptica acrescentaria o mesmo valor montante;

- O custo de desenvolvimento do projeto (mão de obra, por exemplo), também não foi levado em consideração.

5.2 Eficiência Energética e Custo Benefício

Para ter uma noção do custo benefício do projeto foi levado em consideração o seu consumo de energia e quantas lâmpadas podem ser substituídas. Algumas etapas dos cálculos foram omitidas aqui mas presentes no Anexo I (Cálculos de custo benefício).

5.2.1 Consumo do Rastreador Solar

Devido ao fato do microcontrolador consumir menos que $1mW$ de potência, foi considerado que para toda e eletrônica o Rastreador solar consome $1W$ o que pelas ordens de grandeza do projeto é muito maior do que o real, garantindo uma boa margem de erro. Já o motor consome por mês um total de 124,4$Wh/mês$. Ou seja, o Rastreador consome por mês um total de 125,4$Wh/mês$.

5.2.2 Consumo médio de uma lâmpada

Conforme pode ser visto Anexo I, uma lâmpada led pontual de 50W gasta em média durante 1 mês $43kWh/mês$ para funcionar, enquanto duas lâmpadas de tubo de $9W$ gastam em média $13kWh/mês$.

5.2.3 Luminosidade

Segundo [24], o rastreador obteve uma eficiência de $1000lux$ para uma lâmpada pontual, enquanto para um lustre (equivalente à duas lâmpadas de tubo) obteve uma eficiência de $40lux$. Enquanto uma lâmpada pontal de $50W$ produz cerca de $216lux$ e duas de tudo produzem $124lux$.

5.2.4 Comparação entre lâmpada e Rastreador Solar

Com esses dados, vemos que o rastreador substituiu cerca de 5 lâmpadas pontuas[1], enquanto as lâmpadas em tubo não se mostra muito eficiente, isso se deve pela forma como a luz se propaga na fibra. Para o funcionamento como lâmpada pontual, a luz sai concentrada em uma única direção antes de dispersar e para o funcionamento como lâmpada de tubo a luz sai dispersa sendo absorvida pelos materiais refletores e/ou se dispersando no meio.

[1] Lâmpadas pontuais são lâmpadas de bulbo (ou formato pera).

Quanto ao consumo energético, segundo o Portal da Light[2], para uma rede residencial trifásica, no mês de junho/2019, 5 lâmpadas de $50W$ gastariam por mês cerca de R\$174,00 (sem considerar o custo de manutenção), enquanto 1000 rastreadores (pois o consumo mínimo para essas condições, segundo a Light, é de $100kWh$) gastariam R\$1209,00, ou seja, 1 rastreador custaria por mês R\$1,21 (sem considerar os cursos de manutenção).

Considerando os custo de manutenção de R\$200 por ano, conforme anexo I, observa-se junto do consumo energético que o Rastreador custaria bem barato por ano.

Para um funcionamento de $24h/dia$ em um ano podemos, aproximando os valores, considerar o custo médio de R\$4200,00 (lucro mais custo de fabricação) para a compra de um rastreador solar e que ele custaria por ano com manutenção e funcionamento R\$200,00, considerando a despesa energética de 5 lâmpadas pontuais de R\$2100,00/ano, o rastreador, dessa forma, seria pago em aproximadamente 2 anos.

Pode-se observar que o rastreador solar é perfeitamente viável economicamente e que sua implementação é de grande atrativo para os mais diversos setores da sociedade, além de ajudar no meio ambiente ele ajuda na nossa busca por fontes de energia renováveis e garante uma economia mais saudável para pessoas, empresas e departamentos públicos.

5.3 Projetos Futuros

Como trabalhos futuros, podemos observar que o reprojeto de alguns componentes poderá ser utilizado para minizar os efeitos de alguns dos problemas encontrados no projeto. Inicialmente estava previsto a utilização do sensor Hall para evitar que o equipamento gire além do limite ($\pm 70^o$) necessário, caso isso ocorra o motor iria fazer força, atingindo seu consumo máximo de corrente, e poderia vir a avariar-se.

Dessa forma o sensor Hall foi pensado para ser posto no lugar do sensor de fim de curso (*Switch*), usado na primeira versão do rastreador solar, pois o Hall poderia ser posto dentro da caixa de forma que o sensor com atribuição de fim de curso não ficaria suscetível a intemperismos.

Mas devido ao fato do motor necessitar usar todo seu torque para mover o rastreador em alguns momentos (consequentemente atingindo seu pico de corrente) o uso do sensor Hall não se mostrou efetivo podendo acarretar no desligamento indevido do motor. Portanto, recomenda-se por esse fator e pelo alto grau de Zona Morta no redutor do motor, que seja estudada a compra de um motor com maior redução e qualidade de forma que seu torque aumente a baixas velocidades.

[2]Simulador de Conta da Light - `http://www.light.com.br/para-residencias/Simuladores/conta.aspx`, acessado em junho/2019

Apesar da eletrônica ter se mostrado eficiente para os testes, para diminuir a necessidade do uso do atraso na leitura do sensor, a mudança para uma eletrônica mais rápida deve ser revista, seja mudando para um modelo de Arduino melhor (como o Arduino Mega), criando uma eletrônica microcontrolada própria ou usando outros microcontroladores como o Raspeberry PI. O uso de uma eletrônica mais rápida permite que seja usado um sensor de posição menos ruidoso o que melhoria o controlador PID e a precisão no posicionamento.

Além dessas melhorias, é interessante analisar o custo-benefício do uso de um *Driver* com PID integrado. O *Driver* vem com a leitura do ângulo do motor no seu eixo, o que daria mais precisão em saber sua real posição e diminuiria problemas de ruído como ocorre no acelerômetro-giroscópio. Esse seria o mais adequado em termos de engenharia mas também o mais custoso.

Essas melhorias serão aplicadas em projetos futuros e devem ser pensadas para melhorar a segurança na planta, de forma que a mesma possa funcionar de forma autônoma com a menor necessidade de manutenção possível e com garantia de auto-correção no caso de haver intemperismos como vibração, chuva, vento e até mesmo oxidação de algumas peças, aumentando sua longevidade e diminuindo seus custos operacionais. Isso torna o projeto mais rentável, competitivo no mercado, acareetando em uma solução viável para a diminuição do custo de energia elétrica.

Capítulo 6

Anexos

6.1 Anexo I: Cálculos de custo benefício

6.1.1 Consumo Rastreador Solar

O consumo de energia do rastreador comparado a lâmpadas é encontrado neste anexo, para as contas foram usadas como referências dados retirados da primeira versão do rastreador e unidades básicas de referência de lâmpadas encontradas no mercado.

Conforme pode ser visto na refência [25] e no Anexo IV o motor consome $12V$ e $6A$ o que corresponde a uma potência de $72W$[1]. Considerando que a ponte H suporta pulsos de até $0{,}04ms$, segundo seu *datasheet*, isso significa que:

$$Energia_{consumida} = 72 * 0{,}04m = 2{,}88mW/pulso \tag{6.1}$$

Estimando (usando uma margem superior) que ele de até 4 pulsos cada vez que muda de referencial (essa contas foram feitas supondo que o motor oscile anti-horário e horário 4 vezes antes de estabilizar em uma posição, isso pode ser visto pelo gráfico de resposta ao impulso) correspondendo à:

$$Energia_{consumida} = 11{,}52mW \tag{6.2}$$

Que corresponde a energia consumida para cada vez que ele muda de referencial. Como ele muda de referencial a cada 4 minutos, considerando que mudará ao longo das 24horas e que o mês tem 30dias, isso corresponde à:

[1] A potência de 8,9W vista no *datasheet* corresponde a potência mecânica do torque que ele fornece.

$$Energia_{consumida/mês} = \frac{11{,}52mW * 24h * 60min * 30dias}{4min} = 124{,}4Wh/mês \quad (6.3)$$

O microcontrolador usado consome $5V$ de tensão, à $1MHz$ e 25^oC consome $0{,}2mA$ de corrente, o que significa que ele consome em média $1mW$ de energia. Considerando que toda a eletrônica irá trabalhar em seu modo ideal de operação, dado a baixa potência consumida pelo microcontrolador, será considerado que toda a eletrônica junta do microntrolador irá consumir $1W$ de energia garantindo uma boa margem de erro.

Conforme pode ser visto em [24] na primeira versão do Rastreador Solar foram testados para dois tipo de luminária uma pontual (que seria o correspondente a uma lâmpada normal) e uma difusa (que seria o correspondente a um lustre com duas lâmpadas em tubos). Para a lâmpada pontual foi obtido em torno de 1000 à $1200lux$[2]. Para a lâmpada difusa foi obtido de 35 à $40lux$, essa medidas foram feitas usando cerca de $7m$ de fibra ótica com 120 fibras em uma área de $16m^2$.

6.1.2 Consumo lâmpada

Segundo a empresa HTL Brasil [26] uma lâmpada pontual de led $50W$ (comummente usada em lustres e mesas) gera em média $3470lm$ que em uma área de $16m^2$ corresponde a $216lux$. Já uma lâmpada tubular de $9W$, segundo a fabricante *Good Lighting*, gera cerca de $1000lm$, o que em uma área de $16m^2$ corresponde a $62lux$.

Logo, uma lâmpada pontual consome por mês de luz elétrica:

$$Energia_{consumida/mês} = 60W * 24h * 30dias = 43200Wh/mês \quad (6.4)$$

Já uma lustre, que costuma ter duas lâmpadas, com lâmpada de tubo consome por mês de luz elétrica:

$$Energia_{consumida/mês} = 2 * 9W * 24h * 30dias = 12960Wh/mês \quad (6.5)$$

6.1.3 Custos de manutenção

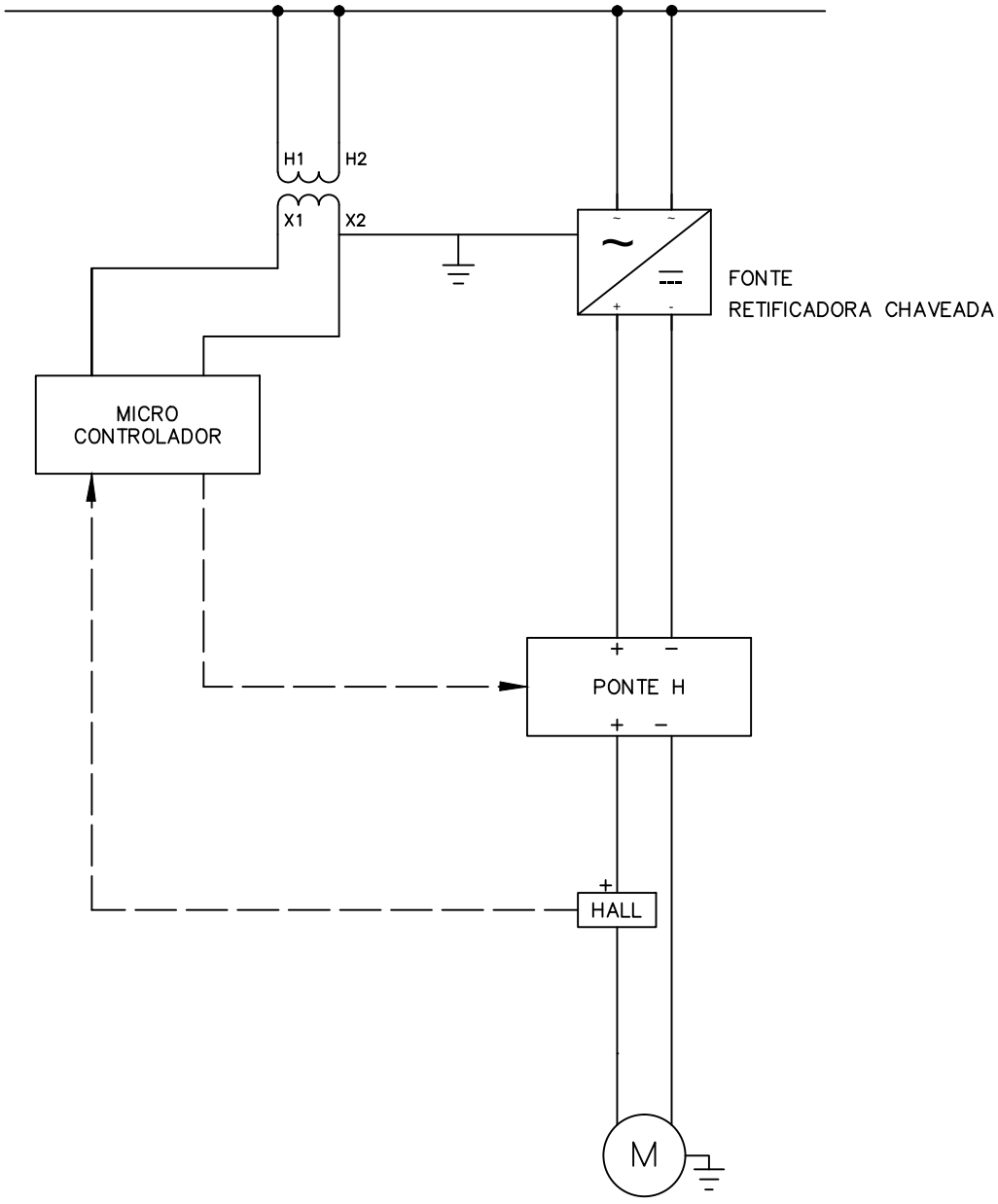
Considerando os preços médios dos componentes (conforme Anexo V), que seriam necessários dois técnicos, que ganham em média $R\$2000$ e que eles levaria cerca de 2horas para fazer as manutenções. O gasto seria de $R\$25$ por dia de forma que se

[2] $1lux = 1lmn/m^2$, $lm = lúmem$ é a unidade de fluxo luminoso que correspondem as unidades de densidade de iluminação.

tivesse que trocar todos os componentes ao longo de 1 ano, por ano o Rastreador custaria R\$200 por ano de manutenção.

6.2 Anexo II: Desenho do diagrama elétrico

O Desenho foi feito pela projetista Isabella Garcia Muniz da Silva da EletroNuclear no *software* AutoCad. E corresponde a ligação elétrica feita neste projeto.



UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA POLITÉCNICA - UFRJ

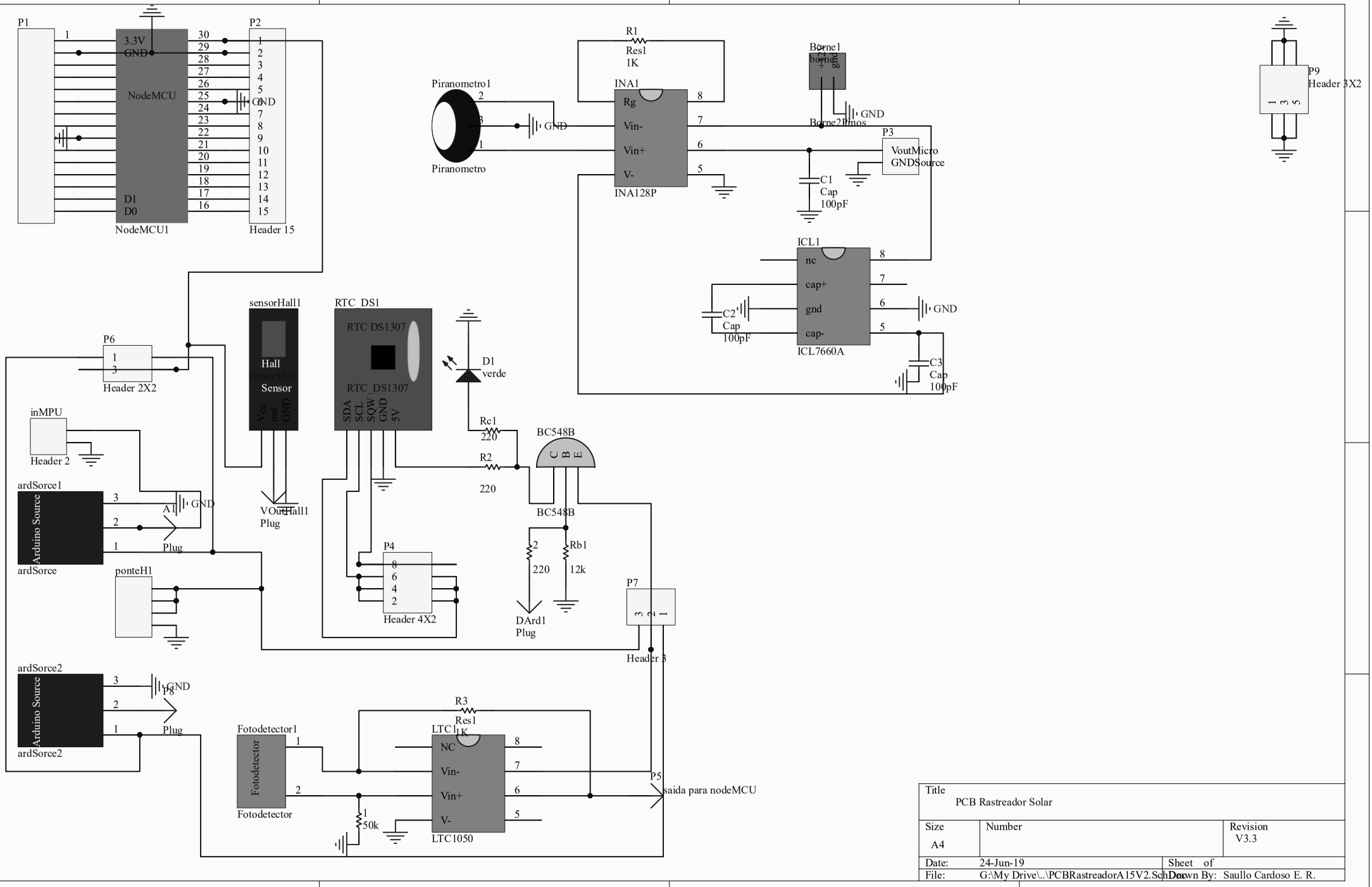
TÍTULO:

RASTREADOR SOLAR
ESQUEMÁTICO ELÉTRICO

ALUNO:	DESENHADO POR:	DATA:	ANEXO	ESCALA:
SAULLO CARDOSO ESTERQUE RODRIGUES	ISABELLA GARCIA	18/06/2019	2	S/ESC.

6.3 Anexo III: Desenho do diagrama eletrônico

O Desenho foi feito no *software* Altium e corresponde ao esquemático da eletrónica usada para projetar a PCB do Rastreador Solar.


P1
P2
3.3V
GND
NodeMCU
D1
D0
NodeMCU1
Header 15
R1
Res1
1K
Piranometro1
Piranometro
INA1
Rg
Vin-
Vin+
V-
INA128P
Borne1
Borne2Pinos
P3
VoutMicro
GNDSource
C1
Cap
100pF
ICL1
nc
cap+
gnd
cap-
ICL7660A
C2
C3
P9
Header 3X2
P6
Header 2X2
sensorHall1
Hall
Sensor
RTC_DS1
RTC DS1307
SDA
SCL
SQW
GND
5V
D1
verde
Rc1
220
R2
BC548B
inMPU
Header 2
ardSorce1
Arduino Source
ardSorce
A1
Plug
VOutHall1
P4
Header 4X2
ponteH1
Rb1
12k
DArd1
P7
Header 3
ardSorce2
P8
Fotodetector1
Fotodetector
R3
LTC1
LTC1050
50k
P5
saida para nodeMCU
Title
PCB Rastreador Solar
Size
A4
Number
Revision
V3.3
Date:
24-Jun-19
Sheet of
File:
G:\My Drive\..\PCBRastreadorA15V2.SchDoc
Drawn By:
Saullo Cardoso E. R.

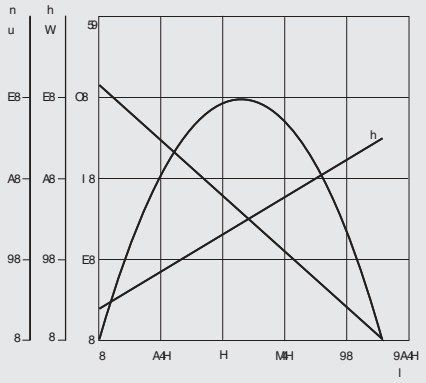
6.4 Anexo IV: Folha de Dados do Motor

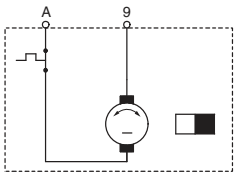
A folha de dados do motor fornecida pela empresa Bosch, referente a referência [25] na página 46, de $12V$ e 8,9W.

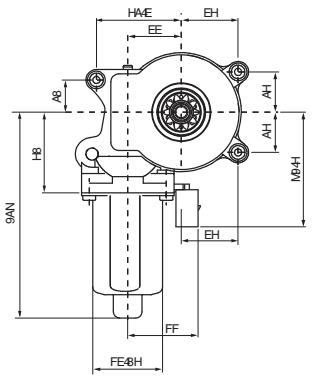
Wha

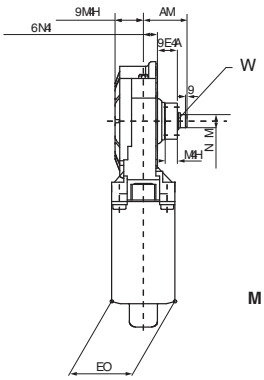
12 p 9,Ar

U	45 i
P	A0 k
n	A8
I	9 N
I .	583 N
M	4 V
M	45 V
i	: 6 L4
Rot.	S2 X
S	g5 18
IP	PW8m
kg	3893 r
R	0 130 921 A79
L	0 130 921 A7A







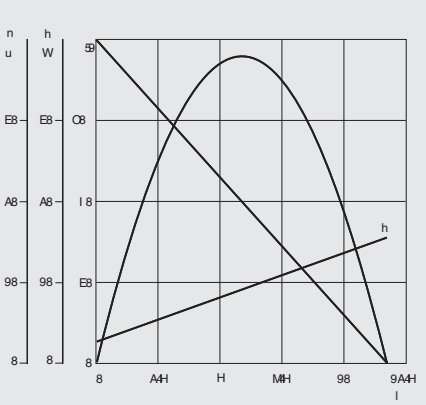


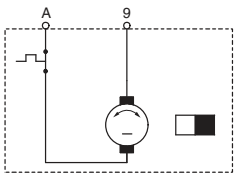
M m y y y yy y 6
e y 6y y 849
d Ø y y y y 6
g y 6y y 869
r y Ø á y 6
j y 6á y y á yáá yØ 869

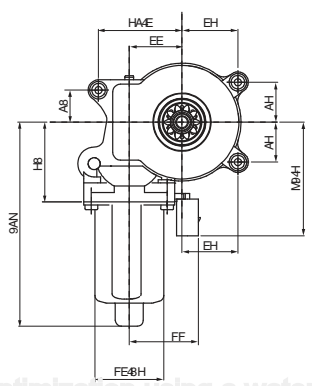
Wha

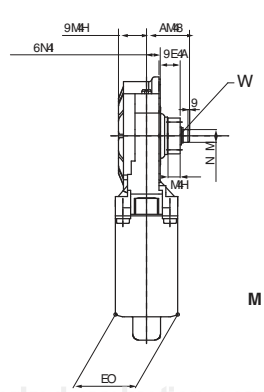
24 p 11,7 r

U	57 i
P	449 k
n	4430
I	7 N
I .	480 N
M	4 V
M	45 V
i	: 6 L4
Rot.	S2 X
S	g5 18
IP	PW8m
kg	3893 r
R	0 130 921 A89
L	0 130 921 A8A









M m y y y yy y 6
e y 6y y 849
d Ø y y y y 6
g y 6y y 869
r y Ø á y 6
j y 6á y y á yáá yØ 869

6.5 Anexo V: Precificação de componentes

Abaixo estão apresentados o preço médio dos componentes utilizados e os locais onde foram comprados, os preços no momento de leitura desta tese podem estar defasados do aqui apresentado por motivos econômicos-sociais e variações do dólar, tendo sido calculado para este projeto com o valor de compra no período que compreende os meses de 2017 e 2018. Também está representado o preço do projeto mecânico que consiste no desenho mecânico e montagem do primeiro protótipo, não é o custo de fabricação pois para a fabricação só é necessário que o desenho mecânico seja feito uma única vez, após este desenho o custo de fabricação só leva em conta as despesas de mão-de-obra, material e recursos para fabricar cada unidade.

Tabela 6.1: Preços componentes e locais de compras.

Componente:	Ponte H
Valor:	$R\$113$
Loja:	Baú da eletrônica
Tipo de loja:	e-Commerce
Componente:	Sensor Hall-$ACS712T/30A$
Valor:	$R\$21$
Loja:	Baú da eletrônica
Tipo de loja:	e-Commerce
Componente:	Motor CC
Valor:	$R\$40$
Loja:	Casa e Video
Tipo de loja:	Loja Física
Componente:	Sensor de Posição
Valor:	$R\$19$
Loja:	Robocore
Tipo de loja:	e-Commerce.
Componente:	Fibra ótica
Valor:	$R\$250/m$
Loja:	Mitsubishi
Tipo de loja:	e-Commerce
Componente:	Projeto mecânico
Valor:	$R\$27000$
Loja:	-
Tipo de loja:	Loja Física

6.6 Anexo VI: Códigos fonte

Abaixo estão apresentados os códigos utilizados para fazer o controle do Rastreador Solar, assim como a leitura dos sensores e acionamento do motor. Para tal, foi utilizado o compilador *open-sorce* do Arduino disponível em seu site gratuitamente. O Arduino utilizado foi um Arduino UNO R3, tudo foi programado na linguagem de programação $C++$. Para este projeto não foi utilizado o RTC, no seu lugar foi utilizado um código que usa o *clock* do Arduino para calcular o tempo e simulando a passagem de tempo.

6.6.1 Código mainFirmware.ino

Código abaixo é referente ao do arquivo "mainFirmware.ino", esse código é o arquivo principal que contém toda a lógica de funcionamento e leitura utilizada para fazer o projeto:

```
1  // 21/03/2019
2  // programmer: Saullo Cardoso E. R. // scardosoer@gmail.com ou
       saullorodrigues@poli.ufrj.br
3  // Projeto Rastreador POF 2 - LiF UFRJ - RJ, Brazil
4  //
5  // Connections MPU6050:VCC > 3.3V, SCL > A5, SDA > A4
6  // Connections MOTOR DC: GND - D6 - D5
7
8  #include "definicoes.h"
9  #include "math.h"
10 #include "PID_v1.h"
11 #include "SPI.h"
12 #include "Wire.h"
13 #include <TimeLib.h>
14 #include "TimeLord.h"
15
16 #define angle_limite 70 // angulos de seguranca
17 #define m1 6 //S1 // portas digitais de entrada do motor
18 #define m2 5 //S3 // portas digitais de entrada do motor
19 #define TIME_HEADER  "T"   // Header tag for serial time sync
      message
20 #define TIME_REQUEST  7    // ASCII bell character requests a
      time sync message
21
```

```
22 //para calculo do angulo do Sol, com biblioteca timeLord
23 const int PROGMEM TIMEZONE = -3;
24 const double PROGMEM LONGITUDE = -43.2311486;
25 const double PROGMEM LATITUDE = -22.8613427;
26
27 // ================================================
28 // declarando funcoes para leitura sensor MPU6050
29 // ================================================
30 void set_last_read_angle_data(unsigned long time, double x,
      double y, double z, double x_gyro, double y_gyro, double
      z_gyro);
31 inline unsigned long get_last_time();
32 inline double get_last_x_angle();
33 inline double get_last_y_angle();
34 inline double get_last_z_angle();
35 inline double get_last_gyro_x_angle();
36 inline double get_last_gyro_y_angle();
37 inline double get_last_gyro_z_angle();
38 int read_gyro_accel_vals(uint8_t* accel_t_gyro_ptr);
39
40 // ================================================
41 // my functions
42 // ================================================
43 void calibrate_sensors();
44 void leituraSensor(void);
45 void comandoMotor(double ref);
46 void motorSentidoHorario(double control);
47 void motorSentidoAntiHorario(double control);
48 void motorSentidoParado();
49 int Sunset (void);
50 int Sunrise (void);
51 double angulo (double x, double in_min, double in_max, double
      out_min, double out_max);
52 void printDigits(int digits);
53 void digitalClockDisplay();
54 void processSyncMessage();
55 void passo(double Setpoint);
56 void calculaHora(int taxa);
57 void printando(double Set);
58
59 // ================================================
```

```
60 // global variables
61 // ================================================
62 time_t requestSync();
63 double angle_x_kal;
64 SimpleKalmanFilter simpleKalmanFilter(2, 2, 0.01);
65 double Setpoint, Input, Output;//variaveis arduino PID
66 //double Kp = 0.08, Ki = 0.15, Kd = 0.00;
67 double Kp = 0.192, Ki = 10.739, Kd = 0.0;// por matlab
68
69 PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
70 TimeLord myLord;
71 long int Total_Sunrise, Total_Sunset, segundos_de_sol,
      horaTotal, hora_atual,metade_do_dia;
72 double angulo_de_sol,angulo_inicio,angulo_fim, angulo_desejado
      , angulo_real;
73 double erro = 0.5; //erro toleravel para parada motor
74 long int hora = hour();
75 long int minuto = minute();
76 long int segundos = second();
77 long int dia = day();
78 long int mes = month();
79 long int ano = year();
80 long int minuto_anterior=0, hora_anterior;//variavel usada
      para marcar quando havera mudanca de deslocamento.
81 double segd;
82
83
84 // ================================================
85 // Aqui comeca a logica de funcionamento,
86 // acima sao as variaveis/bibliotecas/funcoes usadas.
87 // ================================================
88
89 void setup() {
90   Serial.begin(19200);
91   Wire.begin();  // Initialize the 'Wire' class for the I2C-
        bus.
92   myPID.SetMode(AUTOMATIC);
93
94   myLord.TimeZone(TIMEZONE * 60);
95   myLord.Position(LATITUDE, LONGITUDE);
96
```

```
97    int error;
98    uint8_t c;
99
100   // default at power-up:
101   //    Gyro at 250 degrees second  //    Acceleration at 2g
102   //    Clock source at internal 8MHz  //    The device is in
         sleep mode.
103   error = MPU6050_read (MPU6050_WHO_AM_I, &c, 1);
104   // According to the datasheet, the 'sleep' bit  // should
         read a '1'. But I read a '0'.
105   // That bit has to be cleared, since the sensor  // is in
         sleep mode at power-up. Even if the bit reads '0'.
106   error = MPU6050_read (MPU6050_PWR_MGMT_2, &c, 1);
107   // Clear the 'sleep' bit to start the sensor.
108   MPU6050_write_reg (MPU6050_PWR_MGMT_1, 0);
109
110   // inicializa motor
111   pinMode(m1, OUTPUT); digitalWrite(m1, 0);
112   pinMode(m2, OUTPUT); digitalWrite(m2, 0);
113   calibrate_sensors();    // calibracao inicial do MPU6050
114   set_last_read_angle_data(millis(), 0, 0, 0, 0, 0, 0);
115
116   calculaHora(1); // calcula a hora sem usar RTC
117   mudaSetpoint(); // muda o setpoint do equipamento
118   printando(0); // printa valores no serial monitor
119   minuto_anterior = minute();
120   hora_anterior = hour();
121 }// end Setup
122
123 //uint32_t ts1=0, ts2=0;
124 uint32_t ts1=0, ts2=0,ts3,ts4;
125 // com isso usamos o clock do arduino e uma hora inicial
      obtida a partir do unix time para simular a hora e a
      passagem dela
126 //int unsigned long i = 1553787240;//http://www.
      onlineconversion.com/unix_time.htm
127 int unsigned long i = 1555003020;
128
129
130 void loop() {
131   leituraSensor();// le sensor mpu6050
```

```
132   calculaHora(1);// calcula a hora sem usar RTC
133   if (hora>hora_anterior ){// faz o angulo mudar a cada 3min
134       hora_anterior = hora;
135       minuto_anterior = minuto;
136   }
137   if (minuto>( minuto_anterior+3) ){// faz o angulo mudar a
          cada 3min
138       mudaSetpoint();// muda o setpoint do equipamento
139       minuto_anterior = minuto;
140   }
141
142   if(hora>=18 || hora<=7){
143     Setpoint=0;
144     if(hora=7 && minuto > 5){
145      Setpoint=-60;
146     }
147   }
148
149
150   printando(Setpoint);// printa valores no serial monitor
151   passo(Setpoint);// garante que o motor faca pqnos movimentos
          como um servo-motor.
152
153   if ( angle_x >= angle_limite || angle_x <= (-1 *
          angle_limite) ) { // impede girar de mais.
154       passo(5);// manda o motor para 0 graus novamente
155   } // end if protege girar de mais
156   else{
157       if (angle_x >= Setpoint - erro && angle_x <= Setpoint +
              erro) { //margem de erro de tolerancia
158           motorSentidoParado1();
159       }//end else if tolerancia
160       else {
161           comandoMotor(Setpoint);
162       }// end else protecao
163   }
164
165   delay(5);// in leituraSensor() have a delay(5)
166 }// end Loop
167
168 // ================================================
```

```
169 // Aqui termina a logica de funcionamento,
170 // abaixo sao as funcoes usadas.
171 // ================================================
172 // ================================================
173 // Declare Functions
174 // ================================================
175 void calculaHora(int taxa){
176   if (Serial.available()) {
177     processSyncMessage();
178   }
179 
180    setTime(i);
181    hora = hour();
182    minuto = minute();
183    segundos = second();
184    //segd=(double)segundos;
185    dia = day();
186    mes = month();
187    ano = year();
188     i=i+taxa;
189 }//end calculaHora();
190 
191 void mudaSetpoint(){
192   Sunrise();
193   Sunset();
194   segundos_de_sol = (Total_Sunset - Total_Sunrise);
195   angulo_de_sol = (segundos_de_sol/86400.)*360;
196   angulo_inicio = 90 - (angulo_de_sol/2.);
197   angulo_fim = 90 + (angulo_de_sol/2.);
198   long int hora_s = 3600 * hora;
199   long int minuto_s = 60 * minuto;
200   long int total = hora_s + minuto_s + segundos;//-420
201   hora_atual = total;
202   long int metade_do_dia = (Total_Sunset - Total_Sunrise)/2 +
        Total_Sunrise;
203   if ((hora_atual <= metade_do_dia) && (hora_atual > (
        Total_Sunrise - 2400))) {
204     angulo_desejado = angulo (hora_atual, Total_Sunrise,
          metade_do_dia, angulo_inicio, 90.);
205   }//end if
206   else if (hora_atual > metade_do_dia && hora_atual <
```

```
     Total_Sunset) {
207       angulo_desejado = angulo (hora_atual, metade_do_dia,
             Total_Sunset, 90., angulo_fim);
208   }//end else
209
210   Setpoint = angulo_desejado;
211 }// end mudaSetpoint()
212
213 double angulo (double x, double in_min, double in_max, double
     out_min, double out_max) {
214     return (((x - in_min) * (out_max - out_min)) / (in_max -
          in_min) + out_min -90.00);
215 }//end angulo
216
217 void printando(double Set){
218
219   Serial.print(Set); Serial.print("\t");
220   Serial.print(angle_x_kal); Serial.print("\t");
221
222   Serial.print(hora);Serial.print("\t");
223   Serial.print(minuto);Serial.print("\t");
224   Serial.print(segundos);Serial.print("\t");
225
226   Serial.print(dia);Serial.print("\t");
227   Serial.print(mes);Serial.print("\t");
228   Serial.print(ano); Serial.print("\t");
229   Serial.print(i); Serial.print("\t");
230
231   Serial.println();  }
232
233 void passo(double Setpoint){
234   int w =0;
235   int velocidade = 150;
236   ts1=0;
237   while(w<=velocidade){
238      ts2 = millis();
239      leituraSensor();
240       if (angle_x > Setpoint) {
241         motorSentidoParado2();
242       }
243       else if (angle_x <= Setpoint) {
```

```
244         motorSentidoParado1();
245       }
246       w++ ;
247       if( (ts2-ts1)>1000){
248          calculaHora(1);// calcula a hora sem usar RTC
249          ts1 = ts2;
250       }
251       printando(Setpoint);
252
253   }// end while
254 }// end passo()
255
256 void comandoMotor(double ref) {
257   Setpoint = ref;
258 // Input = angle_x_kal;
259  Input = angle_x;
260   myPID.Compute();
261   passo(Setpoint);
262   // isso faz a compara  o demorar e consequentemente ele
        oscilar sem estabilizar
263   if (angle_x > Setpoint) {
264     motorSentidoHorario(Output);
265   }
266   else if (angle_x < Setpoint) {
267     motorSentidoAntiHorario(Output);
268   }
269 }
270
271 void printDigits(int digits){
272   // utility function for digital clock display: prints
        preceding colon and leading 0
273   Serial.print(":");
274   if(digits < 10)
275     Serial.print('0');
276   Serial.print(digits);
277 }
278
279 void processSyncMessage() {
280   unsigned long pctime;
281   const unsigned long DEFAULT_TIME = 1357041600; // Jan 1 2013
282
```

```
283   if(Serial.find(TIME_HEADER)) {
284      pctime = Serial.parseInt();
285      if( pctime >= DEFAULT_TIME) { // check the integer is a
            valid time (greater than Jan 1 2013)
286        setTime(pctime); // Sync Arduino clock to the time
             received on the serial port
287      }}
288 }//end processSyncMessage()
289 time_t requestSync(){
290   Serial.write(TIME_REQUEST);
291   return 0; // the time will be sent later in response to
        serial mesg
292 }//end requestSync()
293
294 int Sunrise (void) {
295   byte day[] = {0, 0, 0, dia, mes, ano};
296   myLord.SunRise(day);
297   Total_Sunrise = (int) day[tl_hour]*3600 + (int) day[
        tl_minute]*60;}
298 int Sunset (void) {
299   byte day[] = {0, 0, 0, dia, mes, ano};
300   myLord.SunSet(day);
301   Total_Sunset = (int) day[tl_hour]*3600 + 65536 + (int) day[
        tl_minute]*60;}
302
303 // motorSentido(Anti)Horario(Parado) faz o controle do sentido
      de rotacao motor.
304 void motorSentidoHorario(double control) {
305   myPID.SetControllerDirection(REVERSE);
306   digitalWrite(m1, 0);
307   digitalWrite(m2, control);
308 }// end motorSentidoHorario
309 void motorSentidoAntiHorario(double control) {
310   myPID.SetControllerDirection(DIRECT);
311   digitalWrite(m1, control);
312   digitalWrite(m2, 0);
313 }// end motorSentidoAntiHorario
314 void motorSentidoParado1() {
315   myPID.SetControllerDirection( DIRECT );
316   digitalWrite(m1, 0);
317   digitalWrite(m2, 0);
```

```
318 }// end motorSentidoParado
319 void motorSentidoParado2() {
320   myPID.SetControllerDirection( REVERSE );
321   digitalWrite(m1, 0);
322   digitalWrite(m2, 0);
323 }// end motorSentidoParado
324
325 void leituraSensor() { // funcao leitura sensor mpu6050.
326   int error;
327   double dT;
328   accel_t_gyro_union accel_t_gyro;
329   // Read the raw values.
330   error = read_gyro_accel_vals((uint8_t*) &accel_t_gyro);
331
332   // Get the time of reading for rotation computations
333   unsigned long t_now = millis();
334
335   // The temperature sensor is -40 to +85 degrees Celsius. //
        It is a signed integer.
336   // According to the datasheet: //   340 per degrees Celsius,
         -512 at 35 degrees.
337   // At 0 degrees: -512 - (340 * 35) = -12412
338
339   // Convert gyro values to degrees/sec
340   double FS_SEL = 131;
341   double gyro_x = (accel_t_gyro.value.x_gyro - base_x_gyro) /
        FS_SEL;
342   double gyro_y = (accel_t_gyro.value.y_gyro - base_y_gyro) /
        FS_SEL;
343   double gyro_z = (accel_t_gyro.value.z_gyro - base_z_gyro) /
        FS_SEL;
344
345   // Get raw acceleration values  //double G_CONVERT = 16384;
346   double accel_x = accel_t_gyro.value.x_accel;
347   double accel_y = accel_t_gyro.value.y_accel;
348   double accel_z = accel_t_gyro.value.z_accel;
349
350   // Get angle values from accelerometer
351   double RADIANS_TO_DEGREES = 180.0 / PI;
352   // double accel_vector_length = sqrt(pow(accel_x,2) + pow(
        accel_y,2) + pow(accel_z,2));
```

```
353   double accel_angle_y = atan(-1 * accel_x / sqrt(pow(accel_y,
          2) + pow(accel_z, 2))) * RADIANS_TO_DEGREES;
354   double accel_angle_x = atan(accel_y / sqrt(pow(accel_x, 2) +
          pow(accel_z, 2))) * RADIANS_TO_DEGREES;
355   double accel_angle_z = atan(sqrt(pow(accel_y, 2) + pow(
         accel_x, 2)) / accel_z) * RADIANS_TO_DEGREES;
356
357   // Compute the (filtered) gyro angles
358   double dt = (t_now - get_last_time()) / 1000.0;
359   double gyro_angle_x = gyro_x * dt + get_last_x_angle();
360   double gyro_angle_y = gyro_y * dt + get_last_y_angle();
361   double gyro_angle_z = gyro_z * dt + get_last_z_angle();
362
363   // Compute the drifting gyro angles
364   double unfiltered_gyro_angle_x = gyro_x * dt +
         get_last_gyro_x_angle();
365   double unfiltered_gyro_angle_y = gyro_y * dt +
         get_last_gyro_y_angle();
366   double unfiltered_gyro_angle_z = gyro_z * dt +
         get_last_gyro_z_angle();
367
368   // Apply the complementary filter to figure out the change
         in angle - choice of alpha is  // estimated now.  Alpha
         depends on the sampling rate...
369   double alpha = 0.96;
370   angle_x = alpha * gyro_angle_x + (1.0 - alpha) *
         accel_angle_x;
371   angle_y = alpha * gyro_angle_y + (1.0 - alpha) *
         accel_angle_y;
372   angle_z = alpha * gyro_angle_z + (1.0 - alpha) *
         accel_angle_z;
373
374   // Update the saved data with the latest values
375   //  set_last_read_angle_data(t_now, angle_x, angle_y,
         angle_z, unfiltered_gyro_angle_x, unfiltered_gyro_angle_y
         , unfiltered_gyro_angle_z);
376   set_last_read_angle_data(t_now, angle_x, angle_y, angle_z,
         unfiltered_gyro_angle_x, unfiltered_gyro_angle_y,
         unfiltered_gyro_angle_z);
377   angle_x_kal = simpleKalmanFilter.updateEstimate(angle_x);
378   delay(5); // Delay so we don't swamp the serial port    }//
```

```
     end leituraSensor
379
380 // ================================================
381 // Funcoes para leitura sensor
382 // ================================================
383 // MPU6050_read  // This is a common function to read multiple
     bytes
384 // from an I2C device. // It uses the boolean parameter for
    Wire.endTransMission()
385 // to be able to hold or release the I2C-bus. // This is
    implemented in Arduino 1.0.1.
386 // Only this function is used to read. // There is no function
     for a single byte.
387 int MPU6050_read(int start, uint8_t *buffer, int size) {
388   int i, n, error;
389   Wire.beginTransmission(MPU6050_I2C_ADDRESS);
390   n = Wire.write(start);
391   if (n != 1) {
392     return (-10);
393   }
394   n = Wire.endTransmission(false);    // hold the I2C-bus
395   if (n != 0) {
396     return (n);
397   }
398   // Third parameter is true: relase I2C-bus after data is
      read.
399   Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);
400   i = 0;
401   while (Wire.available() && i < size) {
402     buffer[i++] = Wire.read();
403   }
404   if ( i != size) {
405     return (-11);
406   }
407   return (0);  // return : no error
408 }
409 // ----------------------------------------------------------
410 // MPU6050_write
411 // This is a common function to write multiple bytes to an I2C
     device.
412 // If only a single register is written,
```

```
413 // use the function MPU_6050_write_reg().
414 // Parameters:
415 //   start : Start address, use a define for the register
416 //   pData : A pointer to the data to write.
417 //   size  : The number of bytes to write.
418 // If only a single register is written, a pointer
419 // to the data has to be used, and the size is
420 // a single byte:
421 //   int data = 0;        // the data to write
422 //   MPU6050_write (MPU6050_PWR_MGMT_1, &c, 1);
423 int MPU6050_write(int start, const uint8_t *pData, int size) {
424   int n, error;
425   Wire.beginTransmission(MPU6050_I2C_ADDRESS);
426   n = Wire.write(start);        // write the start address
427   if (n != 1) {
428     return (-20);
429   }
430   n = Wire.write(pData, size);  // write data bytes
431   if (n != size) {
432     return (-21);
433   }
434   error = Wire.endTransmission(true); // release the I2C-bus
435   if (error != 0)   {
436     return (error);
437   }
438   return (0);         // return : no error
439 }
440 // --------------------------------------------------------
441 // MPU6050_write_reg
442 // An extra function to write a single register. // It is just
      a wrapper around the MPU_6050_write()
443 // function, and it is only a convenient function // to make
     it easier to write a single register.
444 int MPU6050_write_reg(int reg, uint8_t data) {
445   int error;
446   error = MPU6050_write(reg, &data, 1);
447   return (error);
448 }
449
450 void set_last_read_angle_data(unsigned long time, double x,
     double y, double z, double x_gyro, double y_gyro, double
```

```
   z_gyro) {
451   last_read_time = time;
452   last_x_angle = x;
453   last_y_angle = y;
454   last_z_angle = z;
455   last_gyro_x_angle = x_gyro;
456   last_gyro_y_angle = y_gyro;
457   last_gyro_z_angle = z_gyro;
458 }
459
460 inline unsigned long get_last_time() {
461   return last_read_time;
462 }
463 inline double get_last_x_angle() {
464   return last_x_angle;
465 }
466 inline double get_last_y_angle() {
467   return last_y_angle;
468 }
469 inline double get_last_z_angle() {
470   return last_z_angle;
471 }
472 inline double get_last_gyro_x_angle() {
473   return last_gyro_x_angle;
474 }
475 inline double get_last_gyro_y_angle() {
476   return last_gyro_y_angle;
477 }
478 inline double get_last_gyro_z_angle() {
479   return last_gyro_z_angle;
480 }
481
482 int read_gyro_accel_vals(uint8_t* accel_t_gyro_ptr) {
483   // Read the raw values.  // Read 14 bytes at once,
484   // containing acceleration, temperature and gyro.  // With
      the default settings of the MPU-6050,
485   // there is no filter enabled, and the values  // are not
      very stable.  Returns the error value
486
487   accel_t_gyro_union* accel_t_gyro = (accel_t_gyro_union *)
      accel_t_gyro_ptr;
```

```
488
489   int error = MPU6050_read (MPU6050_ACCEL_XOUT_H, (uint8_t *)
          accel_t_gyro, sizeof(*accel_t_gyro));
490
491   // Swap all high and low bytes.
492   // After this, the registers values are swapped,
493   // so the structure name like x_accel_l does no
494   // longer contain the lower byte.
495   uint8_t swap;
496 #define SWAP(x,y) swap = x; x = y; y = swap
497
498   SWAP ((*accel_t_gyro).reg.x_accel_h, (*accel_t_gyro).reg.
          x_accel_l);
499   SWAP ((*accel_t_gyro).reg.y_accel_h, (*accel_t_gyro).reg.
          y_accel_l);
500   SWAP ((*accel_t_gyro).reg.z_accel_h, (*accel_t_gyro).reg.
          z_accel_l);
501   SWAP ((*accel_t_gyro).reg.t_h, (*accel_t_gyro).reg.t_l);
502   SWAP ((*accel_t_gyro).reg.x_gyro_h, (*accel_t_gyro).reg.
          x_gyro_l);
503   SWAP ((*accel_t_gyro).reg.y_gyro_h, (*accel_t_gyro).reg.
          y_gyro_l);
504   SWAP ((*accel_t_gyro).reg.z_gyro_h, (*accel_t_gyro).reg.
          z_gyro_l);
505
506   return error;
507 }
508 // The sensor should be motionless on a horizontal surface
509 //  while calibration is happening
510 void calibrate_sensors() {
511   int                   num_readings = 10;
512   double                 x_accel = 0;
513   double                 y_accel = 0;
514   double                 z_accel = 0;
515   double                 x_gyro = 0;
516   double                 y_gyro = 0;
517   double                 z_gyro = 0;
518   accel_t_gyro_union    accel_t_gyro;
519
520   // Discard the first set of values read from the IMU
521   read_gyro_accel_vals((uint8_t *) &accel_t_gyro);
```

```
522
523   // Read and average the raw values from the IMU
524   for (int i = 0; i < num_readings; i++) {
525     read_gyro_accel_vals((uint8_t *) &accel_t_gyro);
526     x_accel += accel_t_gyro.value.x_accel;
527     y_accel += accel_t_gyro.value.y_accel;
528     z_accel += accel_t_gyro.value.z_accel;
529     x_gyro += accel_t_gyro.value.x_gyro;
530     y_gyro += accel_t_gyro.value.y_gyro;
531     z_gyro += accel_t_gyro.value.z_gyro;
532     delay(100);
533   }
534   x_accel /= num_readings;
535   y_accel /= num_readings;
536   z_accel /= num_readings;
537   x_gyro /= num_readings;
538   y_gyro /= num_readings;
539   z_gyro /= num_readings;
540
541   // Store the raw calibration values globally
542   base_x_accel = x_accel;
543   base_y_accel = y_accel;
544   base_z_accel = z_accel;
545   base_x_gyro = x_gyro;
546   base_y_gyro = y_gyro;
547   base_z_gyro = z_gyro;     }
```

Listing 6.1: Código do arquivo "mainFirmware.ino".

6.6.2 Código definicoes.h

O código abaixo é referente ao do arquivo "definicoes.h", contém algumas variáveis e funções utilizadas no código "mainFirmware.ino":

```
1 #include "arduino.h"
2
3
4 // MPU-6050 Accelerometer + Gyro
5 // -----------------------------
6 //
7 // By arduino.cc user "Krodal".
8 // June 2012
```

```
// Open Source / Public Domain
//
// Using Arduino 1.0.1
// It will not work with an older version,
// since Wire.endTransmission() uses a parameter
// to hold or release the I2C bus.
//
// Documentation:
// - The InvenSense documents:
//   - "MPU-6000 and MPU-6050 Product Specification",
//     PS-MPU-6000A.pdf
//   - "MPU-6000 and MPU-6050 Register Map and Descriptions",
//     RM-MPU-6000A.pdf or RS-MPU-6000A.pdf
//   - "MPU-6000/MPU-6050 9-Axis Evaluation Board User Guide"
//     AN-MPU-6000EVB.pdf
//
// The accuracy is 16-bits.
//
// Temperature sensor from -40 to +85 degrees Celsius
//   340 per degrees, -512 at 35 degrees.
//
// At power-up, all registers are zero, except these two:
//      Register 0x6B (PWR_MGMT_2) = 0x40  (I read zero).
//      Register 0x75 (WHO_AM_I)   = 0x68.
//

#include <Wire.h>


// The name of the sensor is "MPU-6050".
// For program code, I omit the '-',
// therefor I use the name "MPU6050....".


// Register names according to the datasheet.
// According to the InvenSense document
// "MPU-6000 and MPU-6050 Register Map
// and Descriptions Revision 3.2", there are no registers
// at 0x02 ... 0x18, but according other information
// the registers in that unknown area are for gain
// and offsets.
```

```
50 //
51 #define MPU6050_AUX_VDDIO          0x01   // R/W
52 #define MPU6050_SMPLRT_DIV         0x19   // R/W
53 #define MPU6050_CONFIG             0x1A   // R/W
54 #define MPU6050_GYRO_CONFIG        0x1B   // R/W
55 #define MPU6050_ACCEL_CONFIG       0x1C   // R/W
56 #define MPU6050_FF_THR             0x1D   // R/W
57 #define MPU6050_FF_DUR             0x1E   // R/W
58 #define MPU6050_MOT_THR            0x1F   // R/W
59 #define MPU6050_MOT_DUR            0x20   // R/W
60 #define MPU6050_ZRMOT_THR          0x21   // R/W
61 #define MPU6050_ZRMOT_DUR          0x22   // R/W
62 #define MPU6050_FIFO_EN            0x23   // R/W
63 #define MPU6050_I2C_MST_CTRL       0x24   // R/W
64 #define MPU6050_I2C_SLV0_ADDR      0x25   // R/W
65 #define MPU6050_I2C_SLV0_REG       0x26   // R/W
66 #define MPU6050_I2C_SLV0_CTRL      0x27   // R/W
67 #define MPU6050_I2C_SLV1_ADDR      0x28   // R/W
68 #define MPU6050_I2C_SLV1_REG       0x29   // R/W
69 #define MPU6050_I2C_SLV1_CTRL      0x2A   // R/W
70 #define MPU6050_I2C_SLV2_ADDR      0x2B   // R/W
71 #define MPU6050_I2C_SLV2_REG       0x2C   // R/W
72 #define MPU6050_I2C_SLV2_CTRL      0x2D   // R/W
73 #define MPU6050_I2C_SLV3_ADDR      0x2E   // R/W
74 #define MPU6050_I2C_SLV3_REG       0x2F   // R/W
75 #define MPU6050_I2C_SLV3_CTRL      0x30   // R/W
76 #define MPU6050_I2C_SLV4_ADDR      0x31   // R/W
77 #define MPU6050_I2C_SLV4_REG       0x32   // R/W
78 #define MPU6050_I2C_SLV4_DO        0x33   // R/W
79 #define MPU6050_I2C_SLV4_CTRL      0x34   // R/W
80 #define MPU6050_I2C_SLV4_DI        0x35   // R
81 #define MPU6050_I2C_MST_STATUS     0x36   // R
82 #define MPU6050_INT_PIN_CFG        0x37   // R/W
83 #define MPU6050_INT_ENABLE         0x38   // R/W
84 #define MPU6050_INT_STATUS         0x3A   // R
85 #define MPU6050_ACCEL_XOUT_H       0x3B   // R
86 #define MPU6050_ACCEL_XOUT_L       0x3C   // R
87 #define MPU6050_ACCEL_YOUT_H       0x3D   // R
88 #define MPU6050_ACCEL_YOUT_L       0x3E   // R
89 #define MPU6050_ACCEL_ZOUT_H       0x3F   // R
90 #define MPU6050_ACCEL_ZOUT_L       0x40   // R
```

```
91  #define MPU6050_TEMP_OUT_H          0x41   // R
92  #define MPU6050_TEMP_OUT_L          0x42   // R
93  #define MPU6050_GYRO_XOUT_H         0x43   // R
94  #define MPU6050_GYRO_XOUT_L         0x44   // R
95  #define MPU6050_GYRO_YOUT_H         0x45   // R
96  #define MPU6050_GYRO_YOUT_L         0x46   // R
97  #define MPU6050_GYRO_ZOUT_H         0x47   // R
98  #define MPU6050_GYRO_ZOUT_L         0x48   // R
99  #define MPU6050_EXT_SENS_DATA_00    0x49   // R
100 #define MPU6050_EXT_SENS_DATA_01    0x4A   // R
101 #define MPU6050_EXT_SENS_DATA_02    0x4B   // R
102 #define MPU6050_EXT_SENS_DATA_03    0x4C   // R
103 #define MPU6050_EXT_SENS_DATA_04    0x4D   // R
104 #define MPU6050_EXT_SENS_DATA_05    0x4E   // R
105 #define MPU6050_EXT_SENS_DATA_06    0x4F   // R
106 #define MPU6050_EXT_SENS_DATA_07    0x50   // R
107 #define MPU6050_EXT_SENS_DATA_08    0x51   // R
108 #define MPU6050_EXT_SENS_DATA_09    0x52   // R
109 #define MPU6050_EXT_SENS_DATA_10    0x53   // R
110 #define MPU6050_EXT_SENS_DATA_11    0x54   // R
111 #define MPU6050_EXT_SENS_DATA_12    0x55   // R
112 #define MPU6050_EXT_SENS_DATA_13    0x56   // R
113 #define MPU6050_EXT_SENS_DATA_14    0x57   // R
114 #define MPU6050_EXT_SENS_DATA_15    0x58   // R
115 #define MPU6050_EXT_SENS_DATA_16    0x59   // R
116 #define MPU6050_EXT_SENS_DATA_17    0x5A   // R
117 #define MPU6050_EXT_SENS_DATA_18    0x5B   // R
118 #define MPU6050_EXT_SENS_DATA_19    0x5C   // R
119 #define MPU6050_EXT_SENS_DATA_20    0x5D   // R
120 #define MPU6050_EXT_SENS_DATA_21    0x5E   // R
121 #define MPU6050_EXT_SENS_DATA_22    0x5F   // R
122 #define MPU6050_EXT_SENS_DATA_23    0x60   // R
123 #define MPU6050_MOT_DETECT_STATUS   0x61   // R
124 #define MPU6050_I2C_SLV0_DO         0x63   // R/W
125 #define MPU6050_I2C_SLV1_DO         0x64   // R/W
126 #define MPU6050_I2C_SLV2_DO         0x65   // R/W
127 #define MPU6050_I2C_SLV3_DO         0x66   // R/W
128 #define MPU6050_I2C_MST_DELAY_CTRL  0x67   // R/W
129 #define MPU6050_SIGNAL_PATH_RESET   0x68   // R/W
130 #define MPU6050_MOT_DETECT_CTRL     0x69   // R/W
131 #define MPU6050_USER_CTRL           0x6A   // R/W
```

```
132 #define MPU6050_PWR_MGMT_1          0x6B   // R/W
133 #define MPU6050_PWR_MGMT_2          0x6C   // R/W
134 #define MPU6050_FIFO_COUNTH         0x72   // R/W
135 #define MPU6050_FIFO_COUNTL         0x73   // R/W
136 #define MPU6050_FIFO_R_W            0x74   // R/W
137 #define MPU6050_WHO_AM_I            0x75   // R
138
139
140 // Defines for the bits, to be able to change
141 // between bit number and binary definition.
142 // By using the bit number, programming the sensor
143 // is like programming the AVR microcontroller.
144 // But instead of using "(1<<X)", or "_BV(X)",
145 // the Arduino "bit(X)" is used.
146 #define MPU6050_D0 0
147 #define MPU6050_D1 1
148 #define MPU6050_D2 2
149 #define MPU6050_D3 3
150 #define MPU6050_D4 4
151 #define MPU6050_D5 5
152 #define MPU6050_D6 6
153 #define MPU6050_D7 7
154
155 // AUX_VDDIO Register
156 #define MPU6050_AUX_VDDIO MPU6050_D7  // I2C high: 1=VDD, 0=
    VLOGIC
157
158 // CONFIG Register
159 // DLPF is Digital Low Pass Filter for both gyro and
    accelerometers.
160 // These are the names for the bits.
161 // Use these only with the bit() macro.
162 #define MPU6050_DLPF_CFG0     MPU6050_D0
163 #define MPU6050_DLPF_CFG1     MPU6050_D1
164 #define MPU6050_DLPF_CFG2     MPU6050_D2
165 #define MPU6050_EXT_SYNC_SET0 MPU6050_D3
166 #define MPU6050_EXT_SYNC_SET1 MPU6050_D4
167 #define MPU6050_EXT_SYNC_SET2 MPU6050_D5
168
169 // Combined definitions for the EXT_SYNC_SET values
170 #define MPU6050_EXT_SYNC_SET_0 (0)
```

```
171 #define MPU6050_EXT_SYNC_SET_1 (bit(MPU6050_EXT_SYNC_SET0))
172 #define MPU6050_EXT_SYNC_SET_2 (bit(MPU6050_EXT_SYNC_SET1))
173 #define MPU6050_EXT_SYNC_SET_3 (bit(MPU6050_EXT_SYNC_SET1)|bit
    (MPU6050_EXT_SYNC_SET0))
174 #define MPU6050_EXT_SYNC_SET_4 (bit(MPU6050_EXT_SYNC_SET2))
175 #define MPU6050_EXT_SYNC_SET_5 (bit(MPU6050_EXT_SYNC_SET2)|bit
    (MPU6050_EXT_SYNC_SET0))
176 #define MPU6050_EXT_SYNC_SET_6 (bit(MPU6050_EXT_SYNC_SET2)|bit
    (MPU6050_EXT_SYNC_SET1))
177 #define MPU6050_EXT_SYNC_SET_7 (bit(MPU6050_EXT_SYNC_SET2)|bit
    (MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))
178
179 // Alternative names for the combined definitions.
180 #define MPU6050_EXT_SYNC_DISABLED     MPU6050_EXT_SYNC_SET_0
181 #define MPU6050_EXT_SYNC_TEMP_OUT_L   MPU6050_EXT_SYNC_SET_1
182 #define MPU6050_EXT_SYNC_GYRO_XOUT_L  MPU6050_EXT_SYNC_SET_2
183 #define MPU6050_EXT_SYNC_GYRO_YOUT_L  MPU6050_EXT_SYNC_SET_3
184 #define MPU6050_EXT_SYNC_GYRO_ZOUT_L  MPU6050_EXT_SYNC_SET_4
185 #define MPU6050_EXT_SYNC_ACCEL_XOUT_L MPU6050_EXT_SYNC_SET_5
186 #define MPU6050_EXT_SYNC_ACCEL_YOUT_L MPU6050_EXT_SYNC_SET_6
187 #define MPU6050_EXT_SYNC_ACCEL_ZOUT_L MPU6050_EXT_SYNC_SET_7
188
189 // Combined definitions for the DLPF_CFG values
190 #define MPU6050_DLPF_CFG_0 (0)
191 #define MPU6050_DLPF_CFG_1 (bit(MPU6050_DLPF_CFG0))
192 #define MPU6050_DLPF_CFG_2 (bit(MPU6050_DLPF_CFG1))
193 #define MPU6050_DLPF_CFG_3 (bit(MPU6050_DLPF_CFG1)|bit(
    MPU6050_DLPF_CFG0))
194 #define MPU6050_DLPF_CFG_4 (bit(MPU6050_DLPF_CFG2))
195 #define MPU6050_DLPF_CFG_5 (bit(MPU6050_DLPF_CFG2)|bit(
    MPU6050_DLPF_CFG0))
196 #define MPU6050_DLPF_CFG_6 (bit(MPU6050_DLPF_CFG2)|bit(
    MPU6050_DLPF_CFG1))
197 #define MPU6050_DLPF_CFG_7 (bit(MPU6050_DLPF_CFG2)|bit(
    MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0))
198
199 // Alternative names for the combined definitions
200 // This name uses the bandwidth (Hz) for the accelometer,
201 // for the gyro the bandwidth is almost the same.
202 #define MPU6050_DLPF_260HZ    MPU6050_DLPF_CFG_0
203 #define MPU6050_DLPF_184HZ    MPU6050_DLPF_CFG_1
```

```
204 #define MPU6050_DLPF_94HZ     MPU6050_DLPF_CFG_2
205 #define MPU6050_DLPF_44HZ     MPU6050_DLPF_CFG_3
206 #define MPU6050_DLPF_21HZ     MPU6050_DLPF_CFG_4
207 #define MPU6050_DLPF_10HZ     MPU6050_DLPF_CFG_5
208 #define MPU6050_DLPF_5HZ      MPU6050_DLPF_CFG_6
209 #define MPU6050_DLPF_RESERVED MPU6050_DLPF_CFG_7
210
211 // GYRO_CONFIG Register
212 // The XG_ST, YG_ST, ZG_ST are bits for selftest.
213 // The FS_SEL sets the range for the gyro.
214 // These are the names for the bits.
215 // Use these only with the bit() macro.
216 #define MPU6050_FS_SEL0 MPU6050_D3
217 #define MPU6050_FS_SEL1 MPU6050_D4
218 #define MPU6050_ZG_ST   MPU6050_D5
219 #define MPU6050_YG_ST   MPU6050_D6
220 #define MPU6050_XG_ST   MPU6050_D7
221
222 // Combined definitions for the FS_SEL values
223 #define MPU6050_FS_SEL_0 (0)
224 #define MPU6050_FS_SEL_1 (bit(MPU6050_FS_SEL0))
225 #define MPU6050_FS_SEL_2 (bit(MPU6050_FS_SEL1))
226 #define MPU6050_FS_SEL_3 (bit(MPU6050_FS_SEL1)|bit(
      MPU6050_FS_SEL0))
227
228 // Alternative names for the combined definitions
229 // The name uses the range in degrees per second.
230 #define MPU6050_FS_SEL_250  MPU6050_FS_SEL_0
231 #define MPU6050_FS_SEL_500  MPU6050_FS_SEL_1
232 #define MPU6050_FS_SEL_1000 MPU6050_FS_SEL_2
233 #define MPU6050_FS_SEL_2000 MPU6050_FS_SEL_3
234
235 // ACCEL_CONFIG Register
236 // The XA_ST, YA_ST, ZA_ST are bits for selftest.
237 // The AFS_SEL sets the range for the accelerometer.
238 // These are the names for the bits.
239 // Use these only with the bit() macro.
240 #define MPU6050_ACCEL_HPF0 MPU6050_D0
241 #define MPU6050_ACCEL_HPF1 MPU6050_D1
242 #define MPU6050_ACCEL_HPF2 MPU6050_D2
243 #define MPU6050_AFS_SEL0   MPU6050_D3
```

```
244 #define MPU6050_AFS_SEL1    MPU6050_D4
245 #define MPU6050_ZA_ST       MPU6050_D5
246 #define MPU6050_YA_ST       MPU6050_D6
247 #define MPU6050_XA_ST       MPU6050_D7
248
249 // Combined definitions for the ACCEL_HPF values
250 #define MPU6050_ACCEL_HPF_0 (0)
251 #define MPU6050_ACCEL_HPF_1 (bit(MPU6050_ACCEL_HPF0))
252 #define MPU6050_ACCEL_HPF_2 (bit(MPU6050_ACCEL_HPF1))
253 #define MPU6050_ACCEL_HPF_3 (bit(MPU6050_ACCEL_HPF1)|bit(
    MPU6050_ACCEL_HPF0))
254 #define MPU6050_ACCEL_HPF_4 (bit(MPU6050_ACCEL_HPF2))
255 #define MPU6050_ACCEL_HPF_7 (bit(MPU6050_ACCEL_HPF2)|bit(
    MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0))
256
257 // Alternative names for the combined definitions
258 // The name uses the Cut-off frequency.
259 #define MPU6050_ACCEL_HPF_RESET  MPU6050_ACCEL_HPF_0
260 #define MPU6050_ACCEL_HPF_5HZ    MPU6050_ACCEL_HPF_1
261 #define MPU6050_ACCEL_HPF_2_5HZ  MPU6050_ACCEL_HPF_2
262 #define MPU6050_ACCEL_HPF_1_25HZ MPU6050_ACCEL_HPF_3
263 #define MPU6050_ACCEL_HPF_0_63HZ MPU6050_ACCEL_HPF_4
264 #define MPU6050_ACCEL_HPF_HOLD   MPU6050_ACCEL_HPF_7
265
266 // Combined definitions for the AFS_SEL values
267 #define MPU6050_AFS_SEL_0 (0)
268 #define MPU6050_AFS_SEL_1 (bit(MPU6050_AFS_SEL0))
269 #define MPU6050_AFS_SEL_2 (bit(MPU6050_AFS_SEL1))
270 #define MPU6050_AFS_SEL_3 (bit(MPU6050_AFS_SEL1)|bit(
    MPU6050_AFS_SEL0))
271
272 // Alternative names for the combined definitions
273 // The name uses the full scale range for the accelerometer.
274 #define MPU6050_AFS_SEL_2G  MPU6050_AFS_SEL_0
275 #define MPU6050_AFS_SEL_4G  MPU6050_AFS_SEL_1
276 #define MPU6050_AFS_SEL_8G  MPU6050_AFS_SEL_2
277 #define MPU6050_AFS_SEL_16G MPU6050_AFS_SEL_3
278
279 // FIFO_EN Register
280 // These are the names for the bits.
281 // Use these only with the bit() macro.
```

```
282 #define MPU6050_SLV0_FIFO_EN  MPU6050_D0
283 #define MPU6050_SLV1_FIFO_EN  MPU6050_D1
284 #define MPU6050_SLV2_FIFO_EN  MPU6050_D2
285 #define MPU6050_ACCEL_FIFO_EN MPU6050_D3
286 #define MPU6050_ZG_FIFO_EN    MPU6050_D4
287 #define MPU6050_YG_FIFO_EN    MPU6050_D5
288 #define MPU6050_XG_FIFO_EN    MPU6050_D6
289 #define MPU6050_TEMP_FIFO_EN  MPU6050_D7
290
291 // I2C_MST_CTRL Register
292 // These are the names for the bits.
293 // Use these only with the bit() macro.
294 #define MPU6050_I2C_MST_CLK0  MPU6050_D0
295 #define MPU6050_I2C_MST_CLK1  MPU6050_D1
296 #define MPU6050_I2C_MST_CLK2  MPU6050_D2
297 #define MPU6050_I2C_MST_CLK3  MPU6050_D3
298 #define MPU6050_I2C_MST_P_NSR MPU6050_D4
299 #define MPU6050_SLV_3_FIFO_EN MPU6050_D5
300 #define MPU6050_WAIT_FOR_ES   MPU6050_D6
301 #define MPU6050_MULT_MST_EN   MPU6050_D7
302
303 // Combined definitions for the I2C_MST_CLK
304 #define MPU6050_I2C_MST_CLK_0 (0)
305 #define MPU6050_I2C_MST_CLK_1  (bit(MPU6050_I2C_MST_CLK0))
306 #define MPU6050_I2C_MST_CLK_2  (bit(MPU6050_I2C_MST_CLK1))
307 #define MPU6050_I2C_MST_CLK_3  (bit(MPU6050_I2C_MST_CLK1)|bit(
    MPU6050_I2C_MST_CLK0))
308 #define MPU6050_I2C_MST_CLK_4  (bit(MPU6050_I2C_MST_CLK2))
309 #define MPU6050_I2C_MST_CLK_5  (bit(MPU6050_I2C_MST_CLK2)|bit(
    MPU6050_I2C_MST_CLK0))
310 #define MPU6050_I2C_MST_CLK_6  (bit(MPU6050_I2C_MST_CLK2)|bit(
    MPU6050_I2C_MST_CLK1))
311 #define MPU6050_I2C_MST_CLK_7  (bit(MPU6050_I2C_MST_CLK2)|bit(
    MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
312 #define MPU6050_I2C_MST_CLK_8  (bit(MPU6050_I2C_MST_CLK3))
313 #define MPU6050_I2C_MST_CLK_9  (bit(MPU6050_I2C_MST_CLK3)|bit(
    MPU6050_I2C_MST_CLK0))
314 #define MPU6050_I2C_MST_CLK_10 (bit(MPU6050_I2C_MST_CLK3)|bit(
    MPU6050_I2C_MST_CLK1))
315 #define MPU6050_I2C_MST_CLK_11 (bit(MPU6050_I2C_MST_CLK3)|bit(
    MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
```

```
316 #define MPU6050_I2C_MST_CLK_12 (bit(MPU6050_I2C_MST_CLK3)|bit(
        MPU6050_I2C_MST_CLK2))
317 #define MPU6050_I2C_MST_CLK_13 (bit(MPU6050_I2C_MST_CLK3)|bit(
        MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0))
318 #define MPU6050_I2C_MST_CLK_14 (bit(MPU6050_I2C_MST_CLK3)|bit(
        MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1))
319 #define MPU6050_I2C_MST_CLK_15 (bit(MPU6050_I2C_MST_CLK3)|bit(
        MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1)|bit(
        MPU6050_I2C_MST_CLK0))
320
321 // Alternative names for the combined definitions
322 // The names uses I2C Master Clock Speed in kHz.
323 #define MPU6050_I2C_MST_CLK_348KHZ MPU6050_I2C_MST_CLK_0
324 #define MPU6050_I2C_MST_CLK_333KHZ MPU6050_I2C_MST_CLK_1
325 #define MPU6050_I2C_MST_CLK_320KHZ MPU6050_I2C_MST_CLK_2
326 #define MPU6050_I2C_MST_CLK_308KHZ MPU6050_I2C_MST_CLK_3
327 #define MPU6050_I2C_MST_CLK_296KHZ MPU6050_I2C_MST_CLK_4
328 #define MPU6050_I2C_MST_CLK_286KHZ MPU6050_I2C_MST_CLK_5
329 #define MPU6050_I2C_MST_CLK_276KHZ MPU6050_I2C_MST_CLK_6
330 #define MPU6050_I2C_MST_CLK_267KHZ MPU6050_I2C_MST_CLK_7
331 #define MPU6050_I2C_MST_CLK_258KHZ MPU6050_I2C_MST_CLK_8
332 #define MPU6050_I2C_MST_CLK_500KHZ MPU6050_I2C_MST_CLK_9
333 #define MPU6050_I2C_MST_CLK_471KHZ MPU6050_I2C_MST_CLK_10
334 #define MPU6050_I2C_MST_CLK_444KHZ MPU6050_I2C_MST_CLK_11
335 #define MPU6050_I2C_MST_CLK_421KHZ MPU6050_I2C_MST_CLK_12
336 #define MPU6050_I2C_MST_CLK_400KHZ MPU6050_I2C_MST_CLK_13
337 #define MPU6050_I2C_MST_CLK_381KHZ MPU6050_I2C_MST_CLK_14
338 #define MPU6050_I2C_MST_CLK_364KHZ MPU6050_I2C_MST_CLK_15
339
340 // I2C_SLV0_ADDR Register
341 // These are the names for the bits.
342 // Use these only with the bit() macro.
343 #define MPU6050_I2C_SLV0_RW MPU6050_D7
344
345 // I2C_SLV0_CTRL Register
346 // These are the names for the bits.
347 // Use these only with the bit() macro.
348 #define MPU6050_I2C_SLV0_LEN0    MPU6050_D0
349 #define MPU6050_I2C_SLV0_LEN1    MPU6050_D1
350 #define MPU6050_I2C_SLV0_LEN2    MPU6050_D2
351 #define MPU6050_I2C_SLV0_LEN3    MPU6050_D3
```

```
352 #define MPU6050_I2C_SLV0_GRP     MPU6050_D4
353 #define MPU6050_I2C_SLV0_REG_DIS MPU6050_D5
354 #define MPU6050_I2C_SLV0_BYTE_SW MPU6050_D6
355 #define MPU6050_I2C_SLV0_EN      MPU6050_D7
356
357 // A mask for the length
358 #define MPU6050_I2C_SLV0_LEN_MASK 0x0F
359
360 // I2C_SLV1_ADDR Register
361 // These are the names for the bits.
362 // Use these only with the bit() macro.
363 #define MPU6050_I2C_SLV1_RW MPU6050_D7
364
365 // I2C_SLV1_CTRL Register
366 // These are the names for the bits.
367 // Use these only with the bit() macro.
368 #define MPU6050_I2C_SLV1_LEN0    MPU6050_D0
369 #define MPU6050_I2C_SLV1_LEN1    MPU6050_D1
370 #define MPU6050_I2C_SLV1_LEN2    MPU6050_D2
371 #define MPU6050_I2C_SLV1_LEN3    MPU6050_D3
372 #define MPU6050_I2C_SLV1_GRP     MPU6050_D4
373 #define MPU6050_I2C_SLV1_REG_DIS MPU6050_D5
374 #define MPU6050_I2C_SLV1_BYTE_SW MPU6050_D6
375 #define MPU6050_I2C_SLV1_EN      MPU6050_D7
376
377 // A mask for the length
378 #define MPU6050_I2C_SLV1_LEN_MASK 0x0F
379
380 // I2C_SLV2_ADDR Register
381 // These are the names for the bits.
382 // Use these only with the bit() macro.
383 #define MPU6050_I2C_SLV2_RW MPU6050_D7
384
385 // I2C_SLV2_CTRL Register
386 // These are the names for the bits.
387 // Use these only with the bit() macro.
388 #define MPU6050_I2C_SLV2_LEN0    MPU6050_D0
389 #define MPU6050_I2C_SLV2_LEN1    MPU6050_D1
390 #define MPU6050_I2C_SLV2_LEN2    MPU6050_D2
391 #define MPU6050_I2C_SLV2_LEN3    MPU6050_D3
392 #define MPU6050_I2C_SLV2_GRP     MPU6050_D4
```

```
393 #define MPU6050_I2C_SLV2_REG_DIS MPU6050_D5
394 #define MPU6050_I2C_SLV2_BYTE_SW MPU6050_D6
395 #define MPU6050_I2C_SLV2_EN      MPU6050_D7
396
397 // A mask for the length
398 #define MPU6050_I2C_SLV2_LEN_MASK 0x0F
399
400 // I2C_SLV3_ADDR Register
401 // These are the names for the bits.
402 // Use these only with the bit() macro.
403 #define MPU6050_I2C_SLV3_RW MPU6050_D7
404
405 // I2C_SLV3_CTRL Register
406 // These are the names for the bits.
407 // Use these only with the bit() macro.
408 #define MPU6050_I2C_SLV3_LEN0    MPU6050_D0
409 #define MPU6050_I2C_SLV3_LEN1    MPU6050_D1
410 #define MPU6050_I2C_SLV3_LEN2    MPU6050_D2
411 #define MPU6050_I2C_SLV3_LEN3    MPU6050_D3
412 #define MPU6050_I2C_SLV3_GRP     MPU6050_D4
413 #define MPU6050_I2C_SLV3_REG_DIS MPU6050_D5
414 #define MPU6050_I2C_SLV3_BYTE_SW MPU6050_D6
415 #define MPU6050_I2C_SLV3_EN      MPU6050_D7
416
417 // A mask for the length
418 #define MPU6050_I2C_SLV3_LEN_MASK 0x0F
419
420 // I2C_SLV4_ADDR Register
421 // These are the names for the bits.
422 // Use these only with the bit() macro.
423 #define MPU6050_I2C_SLV4_RW MPU6050_D7
424
425 // I2C_SLV4_CTRL Register
426 // These are the names for the bits.
427 // Use these only with the bit() macro.
428 #define MPU6050_I2C_MST_DLY0     MPU6050_D0
429 #define MPU6050_I2C_MST_DLY1     MPU6050_D1
430 #define MPU6050_I2C_MST_DLY2     MPU6050_D2
431 #define MPU6050_I2C_MST_DLY3     MPU6050_D3
432 #define MPU6050_I2C_MST_DLY4     MPU6050_D4
433 #define MPU6050_I2C_SLV4_REG_DIS MPU6050_D5
```

```
434 #define MPU6050_I2C_SLV4_INT_EN  MPU6050_D6
435 #define MPU6050_I2C_SLV4_EN      MPU6050_D7
436
437 // A mask for the delay
438 #define MPU6050_I2C_MST_DLY_MASK 0x1F
439
440 // I2C_MST_STATUS Register
441 // These are the names for the bits.
442 // Use these only with the bit() macro.
443 #define MPU6050_I2C_SLV0_NACK MPU6050_D0
444 #define MPU6050_I2C_SLV1_NACK MPU6050_D1
445 #define MPU6050_I2C_SLV2_NACK MPU6050_D2
446 #define MPU6050_I2C_SLV3_NACK MPU6050_D3
447 #define MPU6050_I2C_SLV4_NACK MPU6050_D4
448 #define MPU6050_I2C_LOST_ARB  MPU6050_D5
449 #define MPU6050_I2C_SLV4_DONE MPU6050_D6
450 #define MPU6050_PASS_THROUGH  MPU6050_D7
451
452 // I2C_PIN_CFG Register
453 // These are the names for the bits.
454 // Use these only with the bit() macro.
455 #define MPU6050_CLKOUT_EN       MPU6050_D0
456 #define MPU6050_I2C_BYPASS_EN   MPU6050_D1
457 #define MPU6050_FSYNC_INT_EN    MPU6050_D2
458 #define MPU6050_FSYNC_INT_LEVEL MPU6050_D3
459 #define MPU6050_INT_RD_CLEAR    MPU6050_D4
460 #define MPU6050_LATCH_INT_EN    MPU6050_D5
461 #define MPU6050_INT_OPEN        MPU6050_D6
462 #define MPU6050_INT_LEVEL       MPU6050_D7
463
464 // INT_ENABLE Register
465 // These are the names for the bits.
466 // Use these only with the bit() macro.
467 #define MPU6050_DATA_RDY_EN    MPU6050_D0
468 #define MPU6050_I2C_MST_INT_EN MPU6050_D3
469 #define MPU6050_FIFO_OFLOW_EN  MPU6050_D4
470 #define MPU6050_ZMOT_EN        MPU6050_D5
471 #define MPU6050_MOT_EN         MPU6050_D6
472 #define MPU6050_FF_EN          MPU6050_D7
473
474 // INT_STATUS Register
```

```
475 // These are the names for the bits.
476 // Use these only with the bit() macro.
477 #define MPU6050_DATA_RDY_INT   MPU6050_D0
478 #define MPU6050_I2C_MST_INT    MPU6050_D3
479 #define MPU6050_FIFO_OFLOW_INT MPU6050_D4
480 #define MPU6050_ZMOT_INT       MPU6050_D5
481 #define MPU6050_MOT_INT        MPU6050_D6
482 #define MPU6050_FF_INT         MPU6050_D7
483
484 // MOT_DETECT_STATUS Register
485 // These are the names for the bits.
486 // Use these only with the bit() macro.
487 #define MPU6050_MOT_ZRMOT MPU6050_D0
488 #define MPU6050_MOT_ZPOS  MPU6050_D2
489 #define MPU6050_MOT_ZNEG  MPU6050_D3
490 #define MPU6050_MOT_YPOS  MPU6050_D4
491 #define MPU6050_MOT_YNEG  MPU6050_D5
492 #define MPU6050_MOT_XPOS  MPU6050_D6
493 #define MPU6050_MOT_XNEG  MPU6050_D7
494
495 // IC2_MST_DELAY_CTRL Register
496 // These are the names for the bits.
497 // Use these only with the bit() macro.
498 #define MPU6050_I2C_SLV0_DLY_EN MPU6050_D0
499 #define MPU6050_I2C_SLV1_DLY_EN MPU6050_D1
500 #define MPU6050_I2C_SLV2_DLY_EN MPU6050_D2
501 #define MPU6050_I2C_SLV3_DLY_EN MPU6050_D3
502 #define MPU6050_I2C_SLV4_DLY_EN MPU6050_D4
503 #define MPU6050_DELAY_ES_SHADOW MPU6050_D7
504
505 // SIGNAL_PATH_RESET Register
506 // These are the names for the bits.
507 // Use these only with the bit() macro.
508 #define MPU6050_TEMP_RESET  MPU6050_D0
509 #define MPU6050_ACCEL_RESET MPU6050_D1
510 #define MPU6050_GYRO_RESET  MPU6050_D2
511
512 // MOT_DETECT_CTRL Register
513 // These are the names for the bits.
514 // Use these only with the bit() macro.
515 #define MPU6050_MOT_COUNT0      MPU6050_D0
```

```
516 #define MPU6050_MOT_COUNT1      MPU6050_D1
517 #define MPU6050_FF_COUNT0       MPU6050_D2
518 #define MPU6050_FF_COUNT1       MPU6050_D3
519 #define MPU6050_ACCEL_ON_DELAY0 MPU6050_D4
520 #define MPU6050_ACCEL_ON_DELAY1 MPU6050_D5
521
522 // Combined definitions for the MOT_COUNT
523 #define MPU6050_MOT_COUNT_0 (0)
524 #define MPU6050_MOT_COUNT_1 (bit(MPU6050_MOT_COUNT0))
525 #define MPU6050_MOT_COUNT_2 (bit(MPU6050_MOT_COUNT1))
526 #define MPU6050_MOT_COUNT_3 (bit(MPU6050_MOT_COUNT1)|bit(
       MPU6050_MOT_COUNT0))
527
528 // Alternative names for the combined definitions
529 #define MPU6050_MOT_COUNT_RESET MPU6050_MOT_COUNT_0
530
531 // Combined definitions for the FF_COUNT
532 #define MPU6050_FF_COUNT_0 (0)
533 #define MPU6050_FF_COUNT_1 (bit(MPU6050_FF_COUNT0))
534 #define MPU6050_FF_COUNT_2 (bit(MPU6050_FF_COUNT1))
535 #define MPU6050_FF_COUNT_3 (bit(MPU6050_FF_COUNT1)|bit(
       MPU6050_FF_COUNT0))
536
537 // Alternative names for the combined definitions
538 #define MPU6050_FF_COUNT_RESET MPU6050_FF_COUNT_0
539
540 // Combined definitions for the ACCEL_ON_DELAY
541 #define MPU6050_ACCEL_ON_DELAY_0 (0)
542 #define MPU6050_ACCEL_ON_DELAY_1 (bit(MPU6050_ACCEL_ON_DELAY0)
       )
543 #define MPU6050_ACCEL_ON_DELAY_2 (bit(MPU6050_ACCEL_ON_DELAY1)
       )
544 #define MPU6050_ACCEL_ON_DELAY_3 (bit(MPU6050_ACCEL_ON_DELAY1)
       |bit(MPU6050_ACCEL_ON_DELAY0))
545
546 // Alternative names for the ACCEL_ON_DELAY
547 #define MPU6050_ACCEL_ON_DELAY_0MS MPU6050_ACCEL_ON_DELAY_0
548 #define MPU6050_ACCEL_ON_DELAY_1MS MPU6050_ACCEL_ON_DELAY_1
549 #define MPU6050_ACCEL_ON_DELAY_2MS MPU6050_ACCEL_ON_DELAY_2
550 #define MPU6050_ACCEL_ON_DELAY_3MS MPU6050_ACCEL_ON_DELAY_3
551
```

```
552 // USER_CTRL Register
553 // These are the names for the bits.
554 // Use these only with the bit() macro.
555 #define MPU6050_SIG_COND_RESET MPU6050_D0
556 #define MPU6050_I2C_MST_RESET  MPU6050_D1
557 #define MPU6050_FIFO_RESET     MPU6050_D2
558 #define MPU6050_I2C_IF_DIS     MPU6050_D4   // must be 0 for
    MPU-6050
559 #define MPU6050_I2C_MST_EN     MPU6050_D5
560 #define MPU6050_FIFO_EN        MPU6050_D6
561
562 // PWR_MGMT_1 Register
563 // These are the names for the bits.
564 // Use these only with the bit() macro.
565 #define MPU6050_CLKSEL0      MPU6050_D0
566 #define MPU6050_CLKSEL1      MPU6050_D1
567 #define MPU6050_CLKSEL2      MPU6050_D2
568 #define MPU6050_TEMP_DIS     MPU6050_D3    // 1: disable
    temperature sensor
569 #define MPU6050_CYCLE        MPU6050_D5    // 1: sample and
    sleep
570 #define MPU6050_SLEEP        MPU6050_D6    // 1: sleep mode
571 #define MPU6050_DEVICE_RESET MPU6050_D7    // 1: reset to
    default values
572
573 // Combined definitions for the CLKSEL
574 #define MPU6050_CLKSEL_0 (0)
575 #define MPU6050_CLKSEL_1 (bit(MPU6050_CLKSEL0))
576 #define MPU6050_CLKSEL_2 (bit(MPU6050_CLKSEL1))
577 #define MPU6050_CLKSEL_3 (bit(MPU6050_CLKSEL1)|bit(
    MPU6050_CLKSEL0))
578 #define MPU6050_CLKSEL_4 (bit(MPU6050_CLKSEL2))
579 #define MPU6050_CLKSEL_5 (bit(MPU6050_CLKSEL2)|bit(
    MPU6050_CLKSEL0))
580 #define MPU6050_CLKSEL_6 (bit(MPU6050_CLKSEL2)|bit(
    MPU6050_CLKSEL1))
581 #define MPU6050_CLKSEL_7 (bit(MPU6050_CLKSEL2)|bit(
    MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))
582
583 // Alternative names for the combined definitions
584 #define MPU6050_CLKSEL_INTERNAL    MPU6050_CLKSEL_0
```

```
585 #define MPU6050_CLKSEL_X            MPU6050_CLKSEL_1
586 #define MPU6050_CLKSEL_Y            MPU6050_CLKSEL_2
587 #define MPU6050_CLKSEL_Z            MPU6050_CLKSEL_3
588 #define MPU6050_CLKSEL_EXT_32KHZ    MPU6050_CLKSEL_4
589 #define MPU6050_CLKSEL_EXT_19_2MHZ MPU6050_CLKSEL_5
590 #define MPU6050_CLKSEL_RESERVED     MPU6050_CLKSEL_6
591 #define MPU6050_CLKSEL_STOP         MPU6050_CLKSEL_7
592
593 // PWR_MGMT_2 Register
594 // These are the names for the bits.
595 // Use these only with the bit() macro.
596 #define MPU6050_STBY_ZG       MPU6050_D0
597 #define MPU6050_STBY_YG       MPU6050_D1
598 #define MPU6050_STBY_XG       MPU6050_D2
599 #define MPU6050_STBY_ZA       MPU6050_D3
600 #define MPU6050_STBY_YA       MPU6050_D4
601 #define MPU6050_STBY_XA       MPU6050_D5
602 #define MPU6050_LP_WAKE_CTRL0 MPU6050_D6
603 #define MPU6050_LP_WAKE_CTRL1 MPU6050_D7
604
605 // Combined definitions for the LP_WAKE_CTRL
606 #define MPU6050_LP_WAKE_CTRL_0 (0)
607 #define MPU6050_LP_WAKE_CTRL_1 (bit(MPU6050_LP_WAKE_CTRL0))
608 #define MPU6050_LP_WAKE_CTRL_2 (bit(MPU6050_LP_WAKE_CTRL1))
609 #define MPU6050_LP_WAKE_CTRL_3 (bit(MPU6050_LP_WAKE_CTRL1)|bit
      (MPU6050_LP_WAKE_CTRL0))
610
611 // Alternative names for the combined definitions
612 // The names uses the Wake-up Frequency.
613 #define MPU6050_LP_WAKE_1_25HZ MPU6050_LP_WAKE_CTRL_0
614 #define MPU6050_LP_WAKE_2_5HZ  MPU6050_LP_WAKE_CTRL_1
615 #define MPU6050_LP_WAKE_5HZ    MPU6050_LP_WAKE_CTRL_2
616 #define MPU6050_LP_WAKE_10HZ   MPU6050_LP_WAKE_CTRL_3
617
618
619 // Default I2C address for the MPU-6050 is 0x68.
620 // But only if the AD0 pin is low.
621 // Some sensor boards have AD0 high, and the
622 // I2C address thus becomes 0x69.
623 #define MPU6050_I2C_ADDRESS 0x68
624
```

```
625
626 // Declaring an union for the registers and the axis values.
627 // The byte order does not match the byte order of
628 // the compiler and AVR chip.
629 // The AVR chip (on the Arduino board) has the Low Byte
630 // at the lower address.
631 // But the MPU-6050 has a different order: High Byte at
632 // lower address, so that has to be corrected.
633 // The register part "reg" is only used internally,
634 // and are swapped in code.
635 typedef union accel_t_gyro_union
636 {
637   struct
638   {
639     uint8_t x_accel_h;
640     uint8_t x_accel_l;
641     uint8_t y_accel_h;
642     uint8_t y_accel_l;
643     uint8_t z_accel_h;
644     uint8_t z_accel_l;
645     uint8_t t_h;
646     uint8_t t_l;
647     uint8_t x_gyro_h;
648     uint8_t x_gyro_l;
649     uint8_t y_gyro_h;
650     uint8_t y_gyro_l;
651     uint8_t z_gyro_h;
652     uint8_t z_gyro_l;
653   } reg;
654   struct
655   {
656     int x_accel;
657     int y_accel;
658     int z_accel;
659     int temperature;
660     int x_gyro;
661     int y_gyro;
662     int z_gyro;
663   } value;
664 };
665
```

```
666 // Use the following global variables and access functions to
       help store the overall
667 // rotation angle of the sensor
668 unsigned long last_read_time;
669 double         last_x_angle;  // These are the filtered angles
670 double         last_y_angle;
671 double         last_z_angle;
672 double         last_gyro_x_angle;  // Store the gyro angles to
        compare drift
673 double         last_gyro_y_angle;
674 double         last_gyro_z_angle;
675
676
677 //  Use the following global variables and access functions
678 //  to calibrate the acceleration sensor
679 double    base_x_accel;
680 double    base_y_accel;
681 double    base_z_accel;
682
683 double    base_x_gyro;
684 double    base_y_gyro;
685 double    base_z_gyro;
686 // angle with filter
687 double angle_x;
688 double angle_y;
689 double angle_z;
```

Listing 6.2: Código do arquivo "definicoes.h".

6.6.3 Código TimeLord.cpp

O código abaixo é referente ao do arquivo "TimeLord.cpp", contém o código responsável por converter dia, mês, ano, hora, minuto, segundo, longitude e latitude no ângulo de referência do sol:

```
1 extern "C" {
2
3         #include <inttypes.h>
4         #include <Math.h> }
5
6 #include "TimeLord.h"
7
```

```
8  TimeLord::TimeLord(){
9                  latitude=27.0;
10                 longitude=-82.0;
11                 timezone=-300;
12                 DstRules(3,2,11,1, 60); // USA
13 }
14
15 bool TimeLord::TimeZone(int z){
16         if(Absolute(z)>720) return false;
17         timezone=z;
18         return true;
19 }
20
21 bool TimeLord::Position(float lat, float lon){
22         if(fabs(lon)>180.0) return false;
23         if(fabs(lat)>90.0) return false;
24         latitude=lat;
25         longitude=lon;
26         return true;
27 }
28
29 bool TimeLord::DstRules(uint8_t sm, uint8_t sw, uint8_t em,
      uint8_t ew, uint8_t adv){
30         if(sm==0 || sw==0 || em==0 || ew==0) return false;
31         if(sm>12 || sw>4 || em>12 || ew>4) return false;
32         dstm1=sm;
33         dstw1=sw;
34         dstm2=em;
35         dstw2=ew;
36         dstadv=adv;
37         return true;
38 }
39
40 void TimeLord::GMT(uint8_t * now){
41         Adjust(now,-timezone);
42 }
43
44 void TimeLord::DST(uint8_t *now){
45         if(InDst(now)) Adjust(now, dstadv);
46 }
47
```

```
48 bool TimeLord::SunRise(uint8_t * when){
49         return ComputeSun(when,true);
50 }
51
52 bool TimeLord::SunSet(uint8_t * when){
53         return ComputeSun(when,false);
54 }
55
56 float TimeLord::MoonPhase(uint8_t * when){
57 // the period is 29.530588853 days
58 // we compute the number of days since Jan 6, 2000
59 // at which time the moon was 'new'
60         long d;
61         float p;
62
63         d=DayNumber(2000+when[tl_year],when[tl_month],when[
              tl_day])-DayNumber(2000,1,6);
64         p=d/29.530588853; // total lunar cycles since 1/1/2000
65         d=p;
66         p-=d; // p is now the fractional cycle, 0 to (less
              than) 1
67         return p;
68
69 }
70
71 void TimeLord::Sidereal(uint8_t * when, bool local){
72         uint64_t second, d;
73         long minute;
74 /*
75          Based on US Naval observatory GMST algorithm
76          (http://aa.usno.navy.mil/faq/docs/GAST.php)
77          Adapted for Arduino
78          ------------------------------------------------
79          Since Arduino doesn't provide double precision
               floating point, we have
80          modified the algorithm to use (mostly) integer math.
81
82          This implementation will work until the year 2100
               with residual error +- 2 seconds.
83
84          That translates to +-30 arc-seconds of angular error,
```

```
            which is just
85          about the field of view of a 100x telescope, and well
            within the field of
86          view of a 50x scope.
87  */
88
89          // we're working in GMT time
90          GMT(when);
91
92          // Get number of days since our epoch of Jan 1, 2000
93          d=DayNumber(when[tl_year]+2000, when[tl_month], when[
               tl_day]) - DayNumber(2000,1,1);
94
95          // compute calendar seconds since the epoch
96          second=d*86400LL+when[tl_hour]*3600LL+when[tl_minute
               ]*60LL+when[tl_second];
97
98          // multiply by ratio of calendar to sidereal time
99          second*=1002737909LL;
100         second/=1000000000LL;
101
102         // add sidereal time at the epoch
103         second+=23992LL;
104
105         if(local){ // convert from gmt to local
106                 d=240.0*longitude;
107                 second+=d;
108         }
109
110         // constrain to 1 calendar day
111         second %= 86400LL;
112
113         // update the tl_time array
114         minute=second/60LL;
115         d=minute*60LL;
116
117         when[tl_second]=second-d;
118         when[tl_hour]=0;
119         when[tl_minute]=0;
120
121         Adjust(when,minute);
```

```
122 }
123
124 uint8_t TimeLord::_season(uint8_t * when){
125         if(when[tl_month]<3) return 0; // winter
126         if(when[tl_month]==3){
127                 if(when[tl_day]<22) return 0;
128                 return 1; // spring
129         }
130
131         if(when[tl_month]<6) return 1; // spring
132         if(when[tl_month]==6){
133                 if(when[tl_day]<21) return 1;
134                 return 2; // summer
135         }
136
137         if(when[tl_month]<9) return 2; // summer
138         if(when[tl_month]==9){
139                 if(when[tl_day]<22) return 2;
140                 return 3; // fall
141         }
142
143         if(when[tl_month]<12) return 3; // summer
144         if(when[tl_day]<21) return 3;
145
146         return 0; // winter
147 }
148
149 uint8_t TimeLord::Season(uint8_t * when){
150 uint8_t result;
151         result=_season(when);
152         if(latitude<0.0) result = (result+2) % 4;
153         return result;
154 }
155
156 uint8_t TimeLord::DayOfWeek(uint8_t * when){
157         int year;
158         uint8_t  month,day;
159
160         year=when[tl_year]+2000;
161         month=when[tl_month];
162         day=when[tl_day];
```

```
163
164         if (month < 3) {
165       month += 12;
166       year--;
167    }
168    day= ((13*month+3)/5 + day + year + year/4 - year/100 +
          year/400 ) % 7;
169    day=(day+1) % 7;
170    return day+1;
171 }
172
173 uint8_t TimeLord::LengthOfMonth(uint8_t * when){
174         uint8_t odd, mnth;
175         int yr;
176
177         yr=when[tl_year]+2000;
178         mnth=when[tl_month];
179
180         if(mnth==2){
181                 if(IsLeapYear(yr) ) return 29;
182                 return 28;
183         }
184         odd=(mnth & 1) == 1;
185         if (mnth > 7) odd  = !odd;
186         if (odd) return 31;
187         return 30;
188 }
189
190 bool TimeLord::IsLeapYear(int yr){
191         return ( (yr % 4 == 0 && yr % 100 != 0) || yr % 400 ==
              0);
192 }
193
194 bool TimeLord::InDst(uint8_t * p){
195         // input is assumed to be standard time
196         char nSundays, prevSunday, weekday;
197
198         if(p[tl_month]<dstm1 || p[tl_month]>dstm2) return
             false;
199         if(p[tl_month]>dstm1 && p[tl_month]<dstm2) return true
             ;
```

```
200
201         // if we get here, we are in either the start or end
                month
202
203         // How many sundays so far this month?
204         weekday=DayOfWeek(p);
205         nSundays=0;
206         prevSunday=p[tl_day]-weekday+1;
207         if(prevSunday>0){
208                 nSundays=prevSunday/7;
209                 nSundays++;
210         }
211
212         if(p[tl_month]==dstm1){
213                 if(nSundays<dstw1) return false;
214                 if(nSundays>dstw1) return true;
215                 if(weekday>1) return true;
216                 if(p[tl_hour]>1) return true;
217                 return false;
218         }
219
220         if(nSundays<dstw2) return true;
221         if(nSundays>dstw2) return false;
222         if(weekday>1) return false;
223         if(p[tl_hour]>1) return false;
224         return true;
225 }
226
227 //====Utility====================
228 // rather than import yet another library, we define sgn and
       abs ourselves
229 char TimeLord::Signum(int n){
230         if(n<0) return -1;
231         return 1;
232 }
233
234 int TimeLord::Absolute(int n){
235         if(n<0) return 0-n;
236         return n;
237 }
238
```

```
239 void TimeLord::Adjust(uint8_t * when, long offset){
240         long tmp, mod, nxt;
241 
242         // offset is in minutes
243         tmp=when[tl_minute]+offset; // minutes
244         nxt=tmp/60;                                     // hours
245         mod=Absolute(tmp) % 60;
246         mod=mod*Signum(tmp)+60;
247         mod %= 60;
248         when[tl_minute]=mod;
249 
250         tmp=nxt+when[tl_hour];
251         nxt=tmp/24;                                             //
              days
252         mod=Absolute(tmp) % 24;
253         mod=mod*Signum(tmp)+24;
254         mod %= 24;
255         when[tl_hour]=mod;
256 
257         tmp=nxt+when[tl_day];
258         mod=LengthOfMonth(when);
259 
260         if(tmp>mod){
261                 tmp-=mod;
262                 when[tl_day]=tmp+1;
263                 when[tl_month]++;
264         }
265         if(tmp<1){
266                 when[tl_month]--;
267                 mod=LengthOfMonth(when);
268                 when[tl_day]=tmp+mod;
269         }
270 
271         tmp=when[tl_year];
272         if(when[tl_month]==0){
273                 when[tl_month]=12;
274                 tmp--;
275         }
276         if(when[tl_month]>12){
277                 when[tl_month]=1;
278                 tmp++;
```

```
279         }
280         tmp+=100;
281         tmp %= 100;
282         when[tl_year]=tmp;
283 }
284
285 bool TimeLord::ComputeSun(uint8_t * when, bool rs) {
286   uint8_t  month, day;
287   float y, decl, eqt, ha, lon, lat, z;
288   uint8_t a;
289   int doy, minutes;
290
291   month=when[tl_month]-1;
292   day=when[tl_day]-1;
293   lon=-longitude/57.295779513082322;
294   lat=latitude/57.295779513082322;
295
296
297   //approximate hour;
298   a=6;
299   if(rs) a=18;
300
301   // approximate day of year
302   y= month * 30.4375 + day  + a/24.0; // 0... 365
303
304   // compute fractional year
305   y *= 1.718771839885e-02; // 0... 1
306
307   // compute equation of time... .43068174
308   eqt=229.18 * (0.000075+0.001868*cos(y)  -0.032077*sin(y)
         -0.014615*cos(y*2) -0.040849*sin(y* 2) );
309
310   // compute solar declination... -0.398272
311   decl=0.006918-0.399912*cos(y)+0.070257*sin(y)-0.006758*cos(y
         *2)+0.000907*sin(y*2)-0.002697*cos(y*3)+0.00148*sin(y*3);
312
313   //compute hour angle
314   ha=(  cos(1.585340737228125) / (cos(lat)*cos(decl)) -tan(lat
         ) * tan(decl)  );
315
316   if(fabs(ha)>1.0){// we're in the (ant)arctic and there is no
```

```
      rise(or set) today!
317         return false;
318   }
319
320   ha=acos(ha);
321   if(rs==false) ha=-ha;
322
323   // compute minutes from midnight
324   minutes=720+4*(lon-ha)*57.295779513082322-eqt;
325
326   // convert from UTC back to our timezone
327   minutes+= timezone;
328
329   // adjust the time array by minutes
330   when[tl_hour]=0;
331   when[tl_minute]=0;
332   when[tl_second]=0;
333   Adjust(when,minutes);
334         return true;
335 }
336
337 long TimeLord::DayNumber(uint16_t y, uint8_t m, uint8_t d){
338
339         m = (m + 9) % 12;
340         y = y - m/10;
341         return 365*y + y/4 - y/100 + y/400 + (m*306 + 5)/10 +
            d - 1 ;
342 }
```

Listing 6.3: Código do arquivo "TimeLord.cpp".

6.6.4 Código TimeLord.h

O código abaixo é referente ao do arquivo "TimeLord.h", contém algumas variáveis e funções utilizadas no código "TimeLord.cpp":

```
1 extern "C" {
2         #include <inttypes.h>
3         #include <Math.h> }
4
5 #define tl_second 0
6 #define tl_minute 1
```

```
7  #define tl_hour 2
8  #define tl_day 3
9  #define tl_month 4
10 #define tl_year 5
11
12 class TimeLord{
13         public:
14                 TimeLord();
15                 // configuration
16                 bool Position(float, float);
17                 bool TimeZone(int);
18                 bool DstRules(uint8_t,uint8_t,uint8_t,uint8_t,
                      uint8_t);
19
20                 // Political
21                 void GMT(uint8_t *);
22                 void DST(uint8_t *);
23
24                 //Solar & Astronomical
25                 bool SunRise(uint8_t *);
26                 bool SunSet(uint8_t *);
27                 float MoonPhase(uint8_t *);
28                 void Sidereal(uint8_t *, bool);
29                 uint8_t Season(uint8_t *);
30
31                 // Utility
32                 uint8_t DayOfWeek(uint8_t *);
33                 uint8_t LengthOfMonth(uint8_t *);
34                 bool IsLeapYear(int);
35
36         private:
37                 float latitude, longitude;
38                 int timezone;
39                 uint8_t dstm1, dstw1, dstm2, dstw2, dstadv;
40                 void Adjust(uint8_t *, long);
41                 bool ComputeSun(uint8_t *, bool);
42                 char Signum(int);
43                 int Absolute(int);
44                 long DayNumber(uint16_t, uint8_t, uint8_t);
45                 bool InDst(uint8_t *);
```

```
46                  uint8_t _season(uint8_t *); };
```

Listing 6.4: Código do arquivo "TimeLord.h".

Referências Bibliográficas

[1] SOLAR, P. "Quanto custa a energia solar fotovoltaica". *https://www.portalsolar.com.br/quanto-custa-a-energia-solar-fotovoltaica.html*, . Acessado em: 01/06/2019.

[2] SOLAR, P. "Governo cria programa de incentivo à geração de eneria solar (ProGD)". *encurtador.com.br/PQRW7*, . Acessado em: 01/06/2019.

[3] RODRIGUES, I. V. "Sistema de monitoramento remoto de um rastreador solar". 2015. Projeto de Graduação UFRJ.

[4] DORF, R. C., BISHOP, R. H. *Sistema de Controle Moderno*. 8 ed. Rio de Janeiro, LTC Editora, 2001.

[5] NARENDRA, KUMPATI S.ANNASWAMY, A. M. *Stable Adaptive Systems-dover Publication*. 1 ed. Minnesota, Englewood Cliffs, 2012.

[6] RIBEIRO, S. C., DO PRADO, P. P. L., GONÇALVES, J. B. "Projeto e Desenvolvimento de um RastreadorSolar para Painéis Fotovoltaicos." *IX Simpósio de excelência em gestão tecnológica*, 2012.

[7] AECWEB. "Tubos Reflexivos de Luz". *encurtador.com.br/drNRX*. Acessado em: 11/01/2020.

[8] CICLOPAK. "Tubos Reflexivos de Luz". *http://www.ciclopak.com.br/2012/tag/reutilizacao/page/3/*. Acessado em: 11/01/2020.

[9] SMARTFLOWER. "Smartflower empresa de rastreador biaxial". *https://smartflower.com/*. Acessado em: 11/01/2020.

[10] ALLIL, A. "Estudo, desenvolvimento e implementação do sistema de controle e automação de um rastreador solar". 2018. Projeto de Graduação UFRJ.

[11] ARDUINO. "PWM no Arduino". *https://www.arduino.cc/en/tutorial/PWM*. Acessado em: 11/01/2020.

[12] ANEEL. "Capacidade de geração do Brasil". *https://www2.aneel.gov.br/aplicacoes/capacidadebrasil/capacidadebrasil.cfm*. Acessado em: 01/06/2019.

[13] DA EDUCAÇÃO, M. "MEC libera *R*$60 milhões para instalação de usinas fotovoltaicas em instituições federais de educação tecnológica". *encurtador.com.br/lmIN2*. Acessado em: 14/12/2019.

[14] REGINA CÉLIA ALLIL, ALFREDO MANCHEGO, A. A. I. R. A. W. G. C. D. F. T. D. Y. R. M. W. "Solar tracker development applied to ambiences illumination and microalgae cultivation", 2017.

[15] N., G. C., DE MEDEIROS, J. L., DE QUEIROZ, O. *Modelagem e Controle na Produção de Petróleo: Aplicações em Matlab*. Edgar Blucher Ltda, 2010.

[16] BASILIO, J. C., DA SILVA VIANA, G., CARVALHO, L. K. "Laboratório de Sistemas de Controle I." 2014. Apostila UFRJ.

[17] BUENO, A. G., ROMANO, R. A. "Filtro complementar aplicado a medida de inclinação de plataformas móveis", *Instituto Mauá de Tecnologia*, pp. 1–12, 2018.

[18] NGO, H. T., NGUYEN, T. P. N. "A complete comparison to design complementary filterand kalman filter for aerial vehicle", *International Journal of Mechanical Engineering and Technology (IJMET)*, v. 9, n. 1, pp. 502–513, Apr 2018.

[19] TALASILA, D. V. "Complementary Filter". *https://www.youtube.com/watch?v=xg7z08wiP1Iweightage*. Acessado em: 01/06/2019.

[20] DOBRRE, O., ACHARD, G. "Optical simulation of lighting by hollow light pipe", *Ninth International IBPSA Conference*, 2005.

[21] BASILIO, J. C., MOREIRA, V. "Laboratório de Sistemas de Controle II." 2008. Apostila UFRJ.

[22] BASILIO, J. C., CORRÊA, C. S. "Comunicação entre Arduino e Simulink para controle de motor DC, Laboratório de controle e Automação (LCA)". Apostila UFRJ.

[23] GENE F. FRANKLIN, J. DAVID POWEL, M. L. W. *Digital Control of Dynamic Systems*. Addison Wesley.

[24] RODRIGUES, I. V. *Desenvolvimento de sistema de iluminação solar com fibrasópticas*. Ms. dissertation, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Rio de Janeiro, Brasil, 2018.

[25] BOSCH. "Catálogo de motores elétricos Bosch", pp. 46–47, 2014.

[26] HCM. "Tabela de equivalência Watts/Lumens". *https://www.hcm.pt/UserFiles/Image/Noticias/tabela_equivalencias_LED.pdf*. Acessado em: 14/07/2019.