Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Curso de Graduação em Engenharia Elétrica

LUCAS CANDEIA MEDEIROS MAIA

ESTIMATIVA DE TRAJETÓRIA EM VEÍCULOS AUTOGUIADOS ATRAVÉS DE ANÁLISE DE IMAGENS

Campina Grande, Paraíba
Dezembro de 2018

LUCAS CANDEIA MEDEIROS MAIA

ESTIMATIVA DE TRAJETÓRIA EM VEÍCULOS AUTOGUIADOS ATRAVÉS DE ANÁLISE DE IMAGENS.

Relatório de Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Controle e Automação

Orientador:
Rafael Bezerra Correia Lima, Dr.Sc.

LUCAS CANDEIA MEDEIROS MAIA

ESTIMATIVA DE TRAJETÓRIA EM VEÍCULOS AUTOGUIADOS ATRAVÉS DE ANÁLISE DE IMAGENS.

Relatório de Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Controle e Automação

Aprovado em _____ / _____ / _____

_____

Professor Rafael Bezerra Correia Lima, Dr.Sc.
Universidade Federal de Campina Grande
Orientador

_____

Professor Avaliador
Universidade Federal de Campina Grande

AGRADECIMENTO

Agradeço primeiramente à Deus e à minha família, em especial à minha mãe Alba Lucia e ao meu irmão David Candeia que sempre me apoiaram e me ajudaram ao longo de toda a graduação.

Agradeço, também, aos Professores Rafael Bezerra Correia Lima, Péricles Rezende Barros e George Acioli Júnior, pela oportunidade de realizar esse trabalho em colaboração com LIEC e pelas orientações prestadas ao longo da graduação.

Por fim, agradeço a todos os colegas do Laboratório de Instrumentação Eletrônica e Controle, em especial à Christian Charles Dias, Simões Toledo, Alequine Batista, Matheus Torres e tantos outros que sempre me ajudaram.

RESUMO

Esse trabalho contempla a estimativa de trajetória em veículos autoguiados através de análise de imagens. Para tal fim, foi utilizado uma webcam, um Raspberry Pi 3B e um SD card de 8 GB. Foi utilizada a metodologia de *Model-Based Design* para implementação desse projeto, possibilitando que todas as partes fossem testadas em simulação. Foram implementados três algoritmos distintos para detecção de faixa: Obtenção de ângulo de faixa por dois pontos, Método dos Mínimos Quadrados e Transformada de Hough. Estes algoritmos foram comparados e foi implementado o que obteve melhor resultado na versão final do veículo.

Palavras-chave: Veículo Autoguiado, Modelagem, Detecção de Faixa, MatLab, Raspberry Pi.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 Introdução

Este é o relatório do trabalho de conclusão de curso, o qual consiste no desenvolvimento de estimativa de trajetória em veículos autoguiados (do inglês, *Auto-Guided Vehicle*, AGV) através de análise de imagens.

Esse trabalho teve como proposta geral o desenvolvimento de um sistema de identificação de trajetória utilizando uma câmera RGB para verificação de percursos delimitados por faixas no chão. Essa forma de identificação será implementada em um AGV existente no Laboratório de Instrumentação Eletrônica e Controle (LIEC), como uma melhoria, uma vez que a identificação de faixa presente no AGV atualmente é com uso de sensores infravermelhos.

Como objetivos específicos, serão necessários realizar:

- Realizar o levantamento dos modelos do veículo, da câmera para detecção de faixa e o algoritmo para detecção de faixa usando o Simulink;
- Utilizar o Simulink para a geração de código para Raspberry Pi;
- Implementação do controle dos modelos levantados em Raspberry Pi;
- Testes para verificação da qualidade dos modelos e atuação da câmera.

A escolha da análise de imagem foi feita baseada na possibilidade de observar a qualidade da informação fornecida por uma câmera para possibilitar ao AGV o seguimento de uma faixa, a qual define o percurso que o AGV deve seguir.

Tem-se por finalidade ainda a implementação do *Model-Based Design* para teste e validação do código/sistema criado para a finalidade de análise de imagem.

Além disso, também relatar as atividades desenvolvidas e os resultados obtidos pelo aluno Lucas Candeia Medeiros Maia, durante o TCC, realizado em parceria com o Laboratório de Instrumentação Eletrônica e Controle (LIEC), situado na Universidade Federal de Campina Grande (UFCG), sob a orientação do professor Rafael Bezerra Correia Lima.

2 Fundamentação teórica

A elaboração desse trabalho foi planejada de forma a possibilitar a utilização de uma câmera para o uso desta para detecção de faixa pelo AGV e integração do micro controlador

utilizado para integração com o MatLab. Requerendo assim, a aplicação de conhecimentos adquiridos em diversas disciplinas ao longo do curso de graduação, como: Introdução à Programação, Técnicas de Programação, Eletrônica, Arquitetura de Sistemas Digitais e TEEE de IOT.

A seguir, serão comentados alguns pontos acerca dos pontos principais presentes nesse TCC afim de possibilitar um melhor entendimento sobre o que foi desenvolvido neste trabalho.

2.1 Veículo Auto-Guiado

A utilização de Veículos Autoguiados (*Auto-Guided Vehicle*, AGV, em inglês), é muito comum nos dias de hoje. Tal nomenclatura é utilizada para veículos que conseguem trafegar de forma autônoma, sem auxílio de um operador, com trajetórias pré-definidas. Os AGV's são muito utilizados em meios industriais para o transporte de produtos e armazenamento também (Gomes, 2016).

Eles são equipados com diversos sensores que os possibilitam perceber o ambiente ao redor e realizar os seus trajetos. Para verificação de trajetos existem algumas técnicas que podem utilizar faixas guias, lasers, sensores de distância (ultrassônico), etc.

Como já mencionado, pode ser utilizada a faixa guia no chão para que o AGV possa seguir um determinado percurso (Barros, 2016). Para a verificação de tal percurso podem ser utilizados sensores infravermelhos ou câmeras RGB (do inglês *Red, Green and Blue*) (Barros, 2016).

No caso dos sensores infravermelhos, pode-se utilizar um conjunto de sensores desta natureza para avaliar a cor do solo, pois quão mais escuro menos refletido pela superfície o sinal será (Barros, 2016). Sendo assim, o programador poderia utilizar esse conjunto de dados para estimar a localização de faixa.

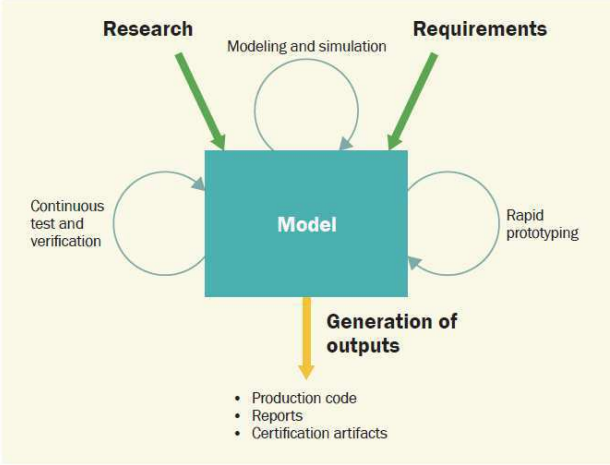
Mais atualmente, vem sendo realizados diversos testes com câmeras para o processamento de imagens e seguimento de faixa com os dados obtidos. Pode-se utilizar câmeras específicas para determinados micro controladores (que já fornecem muitas vezes uma gama de sensores inclusive com as bibliotecas disponíveis) ou câmeras USB's, por exemplo. Por essa linha de abordagem é necessário realizar a captura da imagem pela câmera acionada pelo micro controlador e, posteriormente, deve ser feito o processamento da imagem utilizando algum algoritmo de detecção de faixa.

No Brasil, os AGV's mais utilizados são os com sistemas óticos e magnéticos (Gomes, 2016). Tais ferramentas são tão utilizadas que existem empresas que atuam produzindo esses veículos específicos com várias funções, a exemplo da SINOVA que possui AGV's para rebocar, arrastar, empilhar, etc (SINOVA, 2018). Além de outras finalidades, como desenvolvimento de tecnologia para controle automático de veículos no trânsito (Wharton) e até o uso desses AGV's para ajudar pessoas que apresentem deficiência motora (Sucaria, Unicamp).

2.2 Model Based Design

O *Model-Based Design* (MBD) proporciona uma abordagem matemática e visual para o desenvolvimento de sistemas e controles complexos (Mathworks, whitepaper). Assim como a realização de testes dos sistemas. Dessa forma os desenvolvedores podem utilizar um simples modelo para análise de dados, modelo de visualização, simulação, teste e validação, com ou sem geração de código automático (eInfochips, MBD). Produzindo um fluxo como mostrado na Figura 1.

Figura 1 - Fluxo de trabalho para o MBD.



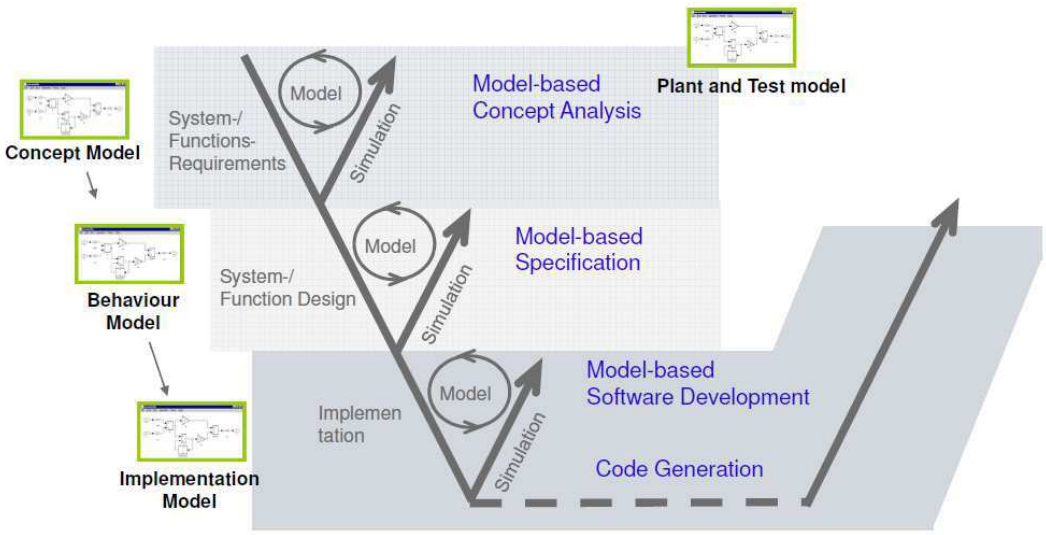
Fonte: Aarenstrup, R., Managing Model-Based Design.

Os engenheiros criam modelos para especificar o comportamento do sistema. Um modelo composto por diagrama de blocos, programas e outros elementos gráficos necessários (Mathworks, whitepaper).

MBD é muito utilizado para aplicação de sistemas em tempo real, pois permite a avaliação de múltiplas opções, predizer a performance do sistema, testes de funcionalidade pela imposição de entradas/saídas que são esperadas em funcionamento (eInfochips, MBD). Além disso, possibilita o refinamento do processo ou sistema, facilidade de testar novas ideias ou modelos pois não requer a construção de protótipo, modelos podem ser adaptador e reutilizados em subsistemas (Mathworks, whitepaper).

O MBD é um ótimo método a ser utilizado desde a concepção do modelo até a implementação do modelo, Figura 2, (Carmeq). Como dito anteriormente, os modelos podem ser reutilizados, ou seja, um modelo criado na concepção do modelo pode ser utilizado e reutilizado na implementação. As alterações e testes dessas alterações se tornam bem mais práticas e rápidas.

Figura 2 - Esquema de aplicação do MBD em um projeto.



Fonte: Carmeq.

O uso de MBD é muito importante na modelagem de sistemas complexos, como: sistema de orientação, sistema de motores, autopilotos, etc. Todavia, também pode ser utilizado efetivamente para modelagens menos complexas.

O modelo representa a dinâmica do sistema, na qual a resposta em qualquer momento é uma função matemática baseada nas entradas, no estado atual e no tempo atual. O MBD é composto por algumas etapas: modelagem, simulação, prototipagem rápida, desenvolvimento do sistema embarcado, teste em loop e integração das atividades (eInfochips, MBD).

De forma geral, inicia-se pela elaboração do modelo que é um processo iterativo que utiliza simulação, partindo de um sistema pouco fiel ao modelo desejado e alcançando uma

implementação que apresente uma representação fidedigna. Utilizando, assim de prototipagem virtual para validar o modelo antes da disponibilidade do hardware (Aarenstrup, R).

2.3 Estratégias para rastreamento de trajetória em AGV's

Atualmente existem algumas estratégias para a detecção de trajetória pelos AGV's, essas estratégias utilizam de sensores e atuadores que possibilitam aos AGV's perceberem o ambiente que os rodeiam. E em virtude dessa percepção o AGV será capaz de realizar as suas tarefas no ambiente no qual se encontre, desde o transporte local de material até tomada de alguma decisão, como seguir pela linha da direita ou da esquerda, por exemplo. Dentre as estratégias utilizadas, pode-se citar algumas, como: fita magnética, processamento de imagem, sensores ópticos.

2.3.1 Fita magnética

Uma forma de realizar a detecção de trajetória pode ser feita com a utilização de fita magnética e pinos magnéticos (spark). A fita apresenta um campo magnético que pode ser detectado por um sensor de efeito Hall, o qual oferece uma tensão na saída de acordo com o campo magnético detectado pelo sensor (smar).

Sendo assim, caso o veículo comece a se distanciar da fita para algum dos lados o sensor captará essa mudança de tensão e com auxílio do algoritmo utilizado para o seguimento de faixa, será gerada uma informação para os atuadores das rodas ou esteiras do AGV para que o mesmo possa voltar a se centralizar na faixa.

2.3.2 Processamento de imagem

Outra forma de realizar a verificação da posição da faixa é utilizando uma câmera para a coleta da imagem e realizar, posteriormente, o processamento desta imagem. No caso do artigo de Gomes, M. V. et al foi considerado um ponto fixo abaixo do AGV e a câmera era utilizada para encontrar um segundo ponto da faixa (Gomes, M. V. et al). Porém a possibilidade de utilização de algoritmo fica a cargo do programador. Atualmente já existem alguns softwares que apresentam interface e/ou funções para o processamento de imagem, como o MatLab, por exemplo.

2.3.3 Sensores ópticos

Em alguns casos são usados sensores baseados em emissão e recepção infravermelho para verificar se fita está abaixo do AGV e qual a posição dessa fita. Um ponto positivo dessa abordagem é que ele não sofre influência significativa da iluminação artificial (Ventriglio R., Universidade São Francisco.)

3 Plataforma experimental

3.1 Hardware utilizado

O hardware utilizado foi: uma webcam da Logitech HD 720p, Figura 3, para a coleta de imagens de faixa; um Raspberry Pi 3B, Figura 4; uma fonte, Figura 5; um cartão SD de 8GB para instalação do Raspibian, Figura 6; e um computador para a comunicação entre MatLab e o Raspberry e para o processamento das fotos de faixa no MatLab. Obtendo-se uma montagem como a mostrada na Figura 7.

Figura 3 - Webcam da Logitech HD 720p.

Fonte: Próprio autor.

Figura 4 – Raspberry Pi 3B.

Fonte: Amazon.

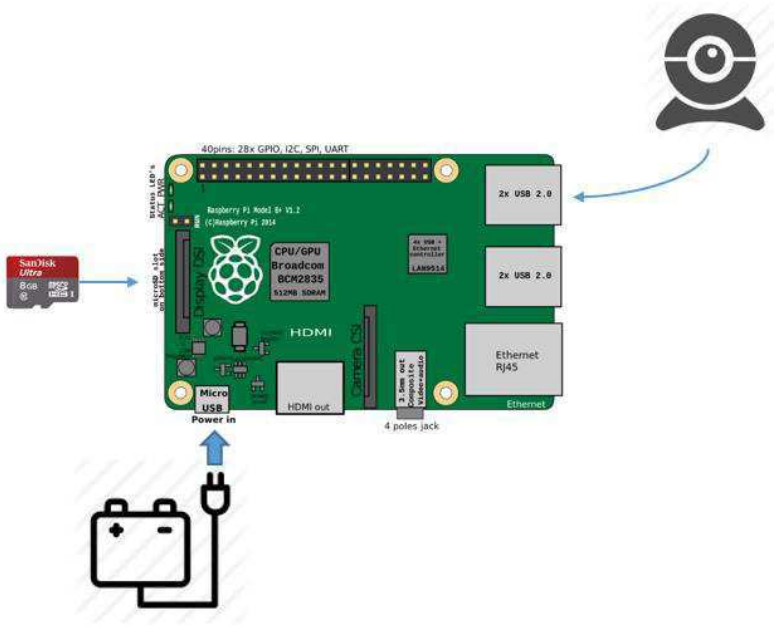
Figura 5 – Fonte alimentação, entrada: 127/220VAC – 50/60Hz – 350mA, saída: 5VDC – 2000mA.

Fonte: Próprio autor.

Figura 6 – Cartão SD de 8GB e adaptador para catão SD.

Fonte: Próprio autor.

Figura 7 – Raspberry Pi 3B com componentes utilizados.

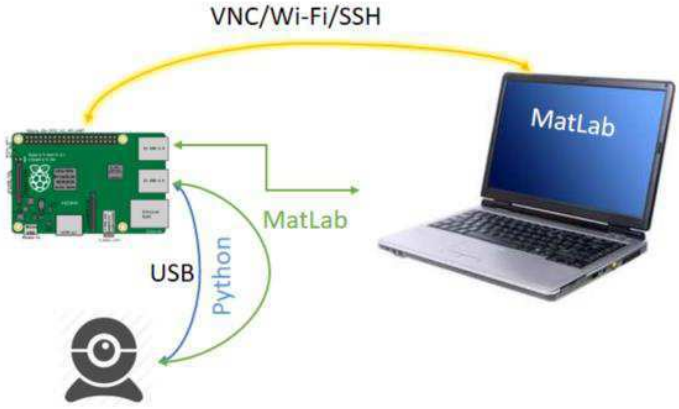


Fonte: Próprio autor.

3.2 Arquitetura de rede utilizada

Para a interação do MatLab com o Raspberry Pi 3B foi utilizado rede Wi-fi com SSH, Figura 8. Além disso, foi utilizado o VNC Viewer para a utilização e configuração/alterações no Raspberry sem necessidade de mouse e teclados específicos conectados a esse. A parte de comunicação de informações sobre os ângulos e distâncias foram feitas por meio de rede CAN.

Figura 8 – Diagrama de comunicação entre computador, placa e webcam.



Fonte: Próprio autor.

O SSH significa Secure Shell, uma rede na qual existe o tráfego de dados criptografados, o que garante uma melhor segurança para o usuário (Cybernet). Porém a comunicação de informações no Raspberry será realizada por meio da rede CAN.

CAN é um protocolo de comunicação serial síncrono. Nesse protocolo se tem vários módulos conectados a um barramento. A partir deste barramento os módulos podem checar se os demais módulos estão enviando mensagens e a prioridade desta mensagem. Em caso de recebimento de mensagens de maior prioridade, a mensagem com prioridade que estava sendo transmitida previamente cessa a sua transmissão (webautomotivo).

Este protocolo apresenta dois formatos de mensagem, as com identificador de 11 bits (CAN 2.0A) e a com identificador de 29 bits (CAN 2.0B). Os fundamentos que descrevem e especificam esse protocolo são descritos em duas normas: a ISO11898 que caracteriza uma rede trabalhando em alta velocidade de transmissão de dados e a ISO11519-2 que caracteriza uma rede trabalhando em baixa velocidade. As faixas de velocidade de transmissão são de 125Kbps a 1Mbps e 10Kbps a 125Kbps, para alta e baixa velocidade respectivamente (webautomotivo).

Esse protocolo ainda prevê a detecção e taxação de falha, colocando uma mensagem de erro no barramento, informando toda a rede do erro de determinada mensagem. Posteriormente é requisitado um novo envio do transmissor.

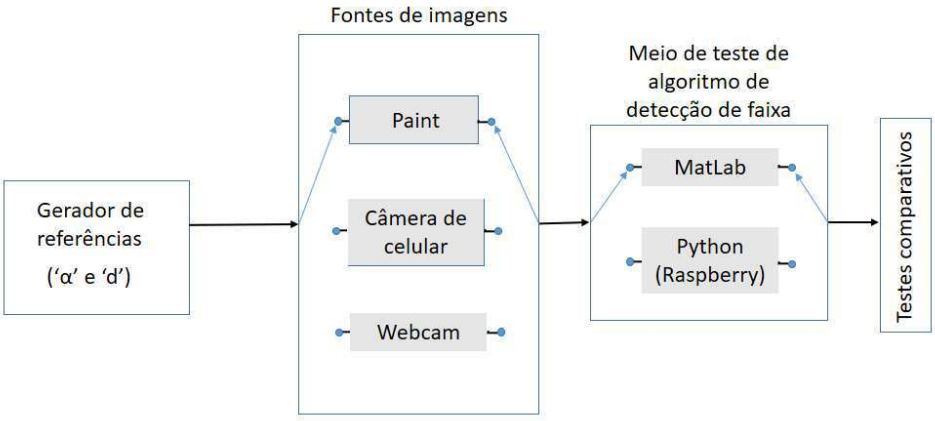
Uma rede com esse protocolo possuirá um valor máximo de conexões com módulo dependendo da norma utilizada. Todos esses módulos estarão ligados a um barramento, que nada mais são do que condutores elétricos para as linhas de comunicação e a forma como são conectados.

3.3 Sistemática geral para validação de algoritmos

Inicialmente, Figura 9, o teste de código foi realizado utilizando desenhos feitos em um software simples de computador, apenas para verificação do funcionamento parcial do código. Após isso, testou-se um primeiro código com fotos retiradas com uma câmera de celular de uma folha de papel com uma fita preta em diversas angulações distintas.

Por fim, quando já estava feita a comunicação entre MatLab e o Raspberry, foi coletada a imagem pela webcam conectada ao Raspberry com a finalidade de testar os códigos para detecção da faixa utilizando a imagem da webcam. Feito isso, foram comparados os dados obtidos afim de verificar qual seria o algoritmo que mais bem se adequasse a tarefa de detecção de faixa.

Figura 9 - Diagrama geral para teste de algoritmos.



Fonte: Próprio autor.

3.4 Pacotes do MatLab/Simulink para Raspberry Pi

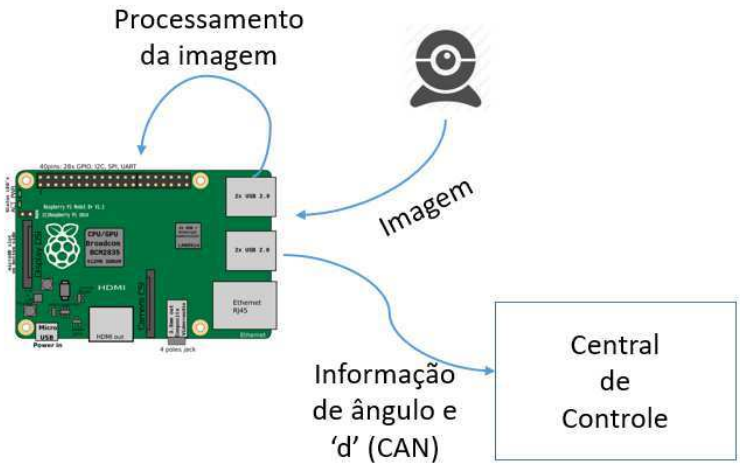
A Mathworks disponibiliza alguns pacotes de MatLab/Simulink para Raspberry Pi. Além disso, existem alguns exemplos disponíveis para uso de comunicação I2C, SPI e interface serial. Além de uso de GPIO e câmera USB.

Para ter acesso a esse material basta ter uma conta da Mathworks e ter o MatLab instalado no computador. E, posteriormente, fazer a instalação e as devidas configurações para uso das funções e blocos fornecidos. Existe uma gama de funções no MatLab para a detecção e processamento de imagem, assim como para aproximação de funções.

4 Sistema de navegação proposto e algoritmos

O sistema proposto engloba a detecção e processamento da imagem da câmera utilizada e posteriormente o envio da informação de angulação e distância da faixa ao eixo central para a central de controle do AGV por meio de rede CAN, Figura 10.

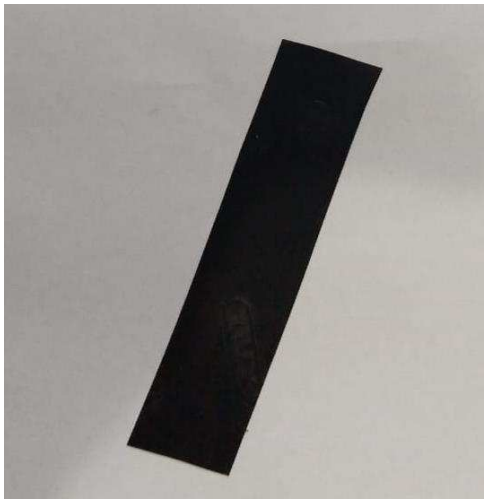
Figura 10 – Diagrama do projeto proposto.



Fonte: Próprio autor.

Sendo assim, acessa-se o Raspberry através do MatLab, realizando uma conexão pelo IP, usuário e senha do Raspberry. Feito isso, é enviado um comando, *"snapshot"*, para o Raspberry para que esse tire uma foto no mesmo instante, Figura 11, e os dados dessa foto na forma de um array é passado para o MatLab, onde será feito todo o processamento da imagem obtida. Posteriormente, também foi feita uma versão em Python do algoritmo que obteve o melhor desempenho na detecção dos ângulos e do offset da faixa.

Figura 11 – Foto retirada com a câmera USB utilizada.



Fonte: Próprio autor.

Após o processamento desta imagem, a informação cria um canal de comunicação em rede CAN, Figura 12, a qual irá repassar essa informação para o software central de funcionamento do AGV, fornecendo assim informações necessárias para que esse veículo possa realizar a ação necessária no motor das rodas do veículo.

Figura 12 – Funções usadas no MatLab para a criação dos canais de Rede CAN.

```
%Creating two virtual CAN channels
canch1 = canChannel('MathWorks','Virtual 1',1)
canch2 = canChannel('MathWorks','Virtual 1',2)
```

Fonte: Próprio autor.

4.1 Processamento de imagem

O processamento que foi feito foi tanto com o de imagens geradas em computador quanto de imagens obtidas pela webcam. Todo o carregamento de imagem ou obtenção de fotos foi feita através do MatLab. Os primeiros testes verificaram a necessidade de utilização de uma função do MatLab para aumentar o contraste entre claro e escuro, deixando a marca da faixa mais evidente e facilitando a identificação da faixa.

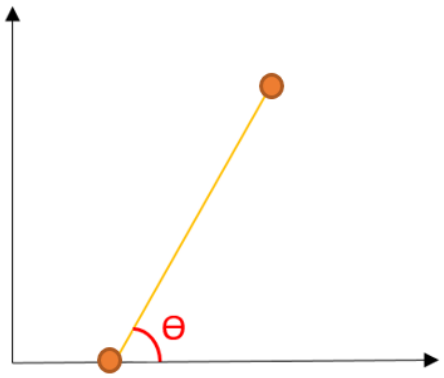
Para a real detecção da faixa foram testados alguns algoritmos, a fim de verificar qual apresentaria a informação mais próxima da real, utilizando assim a ideia do *Model-Based Design* para esses testes.

4.2 Algoritmos para detecção da faixa

Buscou-se encontrar o algoritmo que tivesse o melhor desempenho possível para a detecção da faixa captada pela webcam. Sendo assim, foram pensadas em três estratégias distintas para que se fosse possível compará-las e escolher a que teria o melhor desempenho.

A primeira estratégia utilizada se baseia na coleta de pontos da faixa através da imagem coletada pela câmera. Uma vez que é verificada a faixa na imagem, considera-se o primeiro e o último ponto da faixa (o primeiro ponto é localizado na borda superior esquerda da faixa e o último na borda inferior esquerda da faixa), de tal forma que é possível encontrar a equação da reta e consequentemente o ângulo da reta, Figura 13.

Figura 13 – Exemplo de reta com coeficiente angular.



Fonte: Próprio autor.

Todo esse processo é realizado no MatLab utilizando funções para tratamento de imagem já existentes no software, como a "*im2bw*" que realiza uma comparação das cores dos pixels de acordo com um fator escolhido que atendesse as situações testadas. Produzindo, a partir da imagem tirada pela webcam, um array com apenas valores binários, representando com um contraste melhor a imagem da faixa, Figura 14.

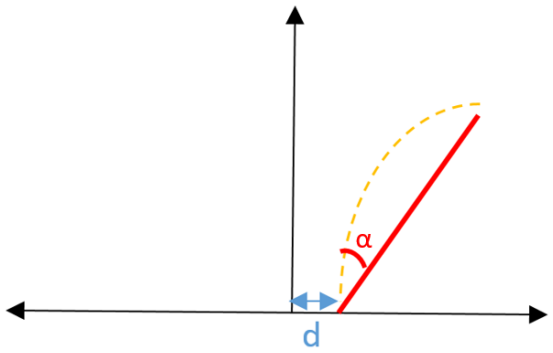
Figura 14 – Imagem de faixa após uso da função im2bw.

Fonte: Próprio autor.

Essa informação será passada para o sistema geral de controle do AGV seguindo o referencial indicado na Figura 15. São fornecidos ao sistema de controle informação de angulação alfa negativa ou positiva dependendo se a faixa se encontra a esquerda ou à direita

do eixo central, respectivamente. E ainda uma distância ‘d’ que será a distância entre o eixo central e a parte inferior da faixa.

Figura 15 – Exemplo de situação onde a faixa tem angulação para direita com relação a eixo central do AGV.



Fonte: Próprio autor.

A segunda estratégia utilizou o Método dos Mínimos Quadrados para a detecção da reta que se enquadrasse nos dados da imagem. Deseja-se estimar o valor de ‘y’. Para tal são considerados outros valores para uma variável ‘x’, se relacionando com ‘y’ de acordo com a Equação 1. Onde α é um parâmetro constante, β é o coeficiente de x e ε é o erro.

$$y = \alpha + \beta x + \varepsilon \qquad (1)$$

Para aplicação desse método é necessária uma base de dados com n valores. Nessa base de dados teremos diversos valores de x e y que estarão relacionados entre si e a partir da aplicação do Método dos Mínimos Quadrados será possível estimar α e β. Obtendo, assim, um modelo estimado, Equação 2.

$$y_i = a + bx_i + e_i \qquad (2)$$

O índice i indica as n situações existentes na base de dados utilizada. E o ‘e’ agora é o resíduo. Este é um procedimento matemático que visa encontrar a curva mais apropriada de forma que a soma dos quadrados dos resíduos, Equação 3, seja minimizada.

$$\sum_{i=1}^{n} e_i^2 \qquad (3)$$

Fazendo isso, procura-se encontrar os valores de ‘a’ e ‘b’ que trarão a menor diferença entre os valores reais e estimados de y. Isolando o resíduo na Equação 2 e substituindo em e, tem-se a Equação 4. E a minimização se obtém ao derivar com relação a ‘a’ e ‘b’ e igualando a zero. Obtendo a Equação 5 e 6.

$$S(a,b) = \sum_{i=1}^{n}(y_i - a - bx_i)^2 \qquad (4)$$

$$\frac{\partial S}{\partial a} = -2\sum_{i=1}^{n}(y_i - a - bx_i)^2 = 0 \qquad (5)$$

$$\frac{\partial S}{\partial b} = -2\sum_{i=1}^{n}(y_i - a - bx_i)^2 = 0 \qquad (6)$$

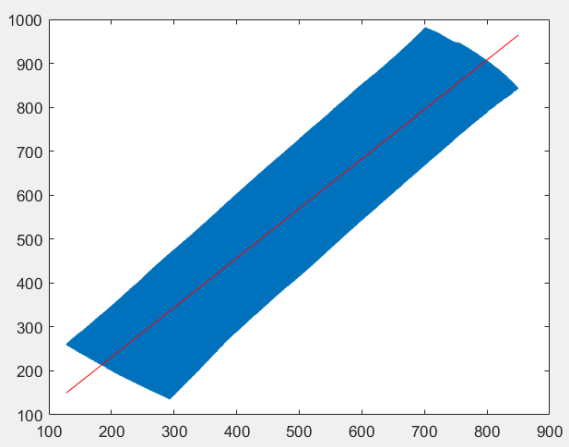
Após isso, é feita a distribuição e dividida a Equação 5 por 2n, obtendo a Equação 7, o qual será substituído na Equação 6, obtendo 8.

$$a = \bar{y} - b\bar{x} \qquad (7)$$

$$b = \frac{\sum_{i=1}^{n} x_i(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})} \qquad (8)$$

Com isso é possível descobrir a equação da reta que melhor se adequa aos pontos obtidos na matriz de pixels, sendo possível conhecer a angulação desta reta. A representação dos eixos mostrado na Figura 16 é feita com relação a matriz de pixels e pelo método é apresentada uma reta que melhor se adequa aos dados de array da imagem.

Figura 16 - Exemplo de aplicação do Método dos Mínimos Quadrados para uma foto de faixa com 45º de angulação.

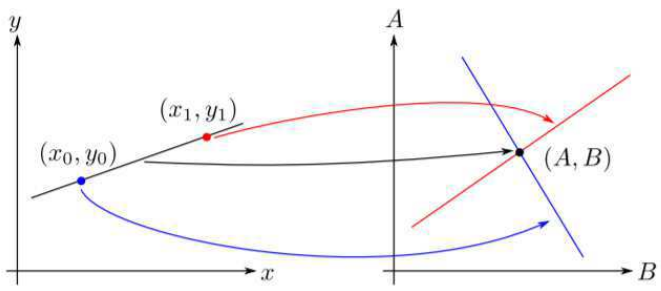


Fonte: Próprio autor.

A terceira abordagem foi utilizando a Transformada de Hough. Após uma pesquisa foi possível verificar que esse método é utilizado em problemas onde se deseja detectar características analiticamente representáveis em imagens binarizadas, como linhas, círculos e elipses (Visão Computacional, UFSC). Esse método utiliza do paradigma de dualidade, mapeando pontos do plano a imagem para curvas do plano de parâmetros (CERQUEIRA, UFMG). Essa transformada é aplicada após a imagem sofrer um pré-processamento, onde é possível identificar as bordas da forma (Visão Computacional, UFSC).

Sendo assim, é feito uma transformação de dualidade do plano xy para o plano dual ρθ. Quando existe ocorre a interceptação de curva no eixo dual isso significa que existe uma curva passando pelos pontos que seriam correspondentes no plano primal (xy) (CERQUEIRA, UFMG). Cada ponto (xi, yi) forma uma reta Ui no espaço dual (MARROQUIM, UFRJ), Figura 17.

Figura 17 – Representação da dualidade entre os planos xy e ρθ, representado por AB respectivamente.



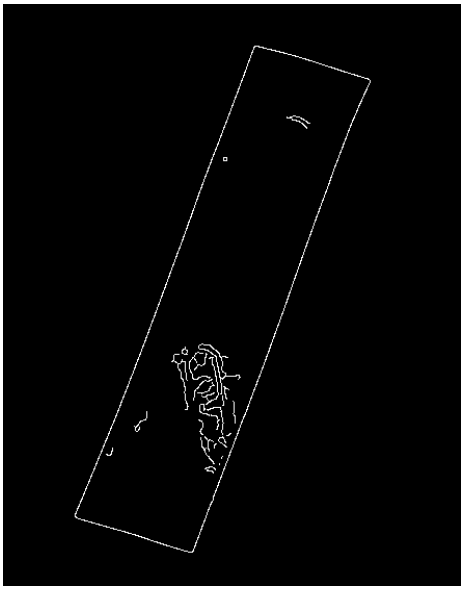
Fonte: MARROQUIM, A.O.R., "Processamento de Imagens", UFRJ.

O MatLab possui uma função que calcula os parâmetros da Transformada de Hough. A função utiliza a representação paramétrica de uma reta como visto na Equação 6. A função "hough" retorna a transformada padrão de Hough (como descrito no site da Mathworks) e os parâmetros ρ e θ.

$$\rho = x.cos\theta + y.sen\theta \qquad (6)$$

Antes de passar a matriz de pixels para a função é feito uma conversão utilizando a função “rgb2gray”, que converte a imagem, caso seja colorida, para tons de cinza. Em seguida, também é utilizada a função “edge”, a qual retorna uma imagem em preto e branco do mesmo tamanho da matriz de entrada, com 1’s onde for encontrado bordas e 0’s nos demais pontos, Figura 18. Para encontrar essas bordas existem algumas opções de métodos, os quais podem ser verificados nas documentações da Mathworks.

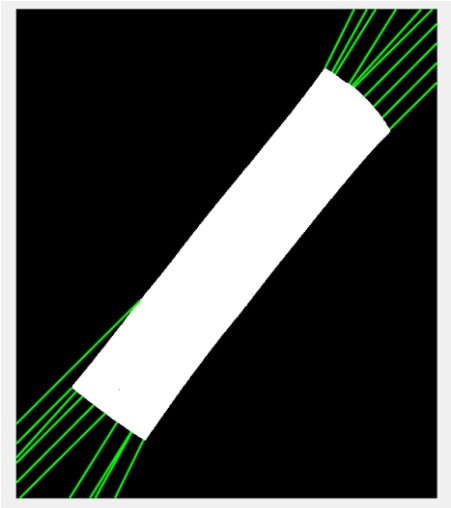
Figura 18 – Imagem após a utilização da função “edge” com método “canny”.



Fonte: Próprio autor.

O método de busca de borda utilizado foi o de Canny, o qual encontra as bordas devido a busca pela máxima local de gradiente na imagem. Posteriormente, são plotadas as retas na imagem, Figura 19, uma vez que são desenhadas diversas retas, afim de tentar obter a melhor referência foi feita uma média ponderada entre 10 primeiras retas desenhadas.

Figura 19 – Exemplo de aplicação de Transformada de Hough para uma foto de faixa com 45º de angulação.



Fonte: Próprio autor.

5 Resultados e análise de erros

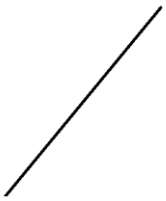
Baseado nos algoritmos previamente apresentados, foram feitas as comparações entre os valores de ângulos das imagens utilizadas como entrada para o código e os resultados apresentados após os testes com os algoritmos. Ou seja, foram plotados gráficos de erros baseados nas divergências do ângulo real e do ângulo calculado na detecção da faixa. Vale salientar que inicialmente, variou-se apenas o ângulo da faixa, mantendo a faixa no eixo central e com o 'd', distância entre a parte inferior da faixa com o eixo adotado igual a zero. Posteriormente foi analisado a influência dessa movimentação da faixa em 'd' e por fim a implicação da variação de ambos os fatores.

5.1 Resultados da estimativa de ângulos das faixas

Primeiramente, os testes foram realizados com as imagens geradas por computador nos três algoritmos desenvolvidos. Como sugestão do orientador, deveriam ser geradas em torno de 10 imagens para esse teste e esse número deveria ser mantido para os testes com a webcam.

A exemplo das linhas geradas em computador, tem-se a Figura 20 com uma linha de angulação de 45º.

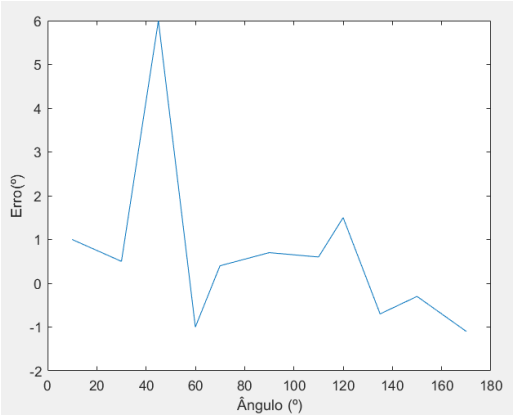
Figura 20 – Linha com angulação de 45º.



Fonte: Próprio autor.

Foram gerados, então 11 retas, com angulação distintas e calculado os erros relativos referentes ao valor de ângulo medido em cada um dos casos, plotando-os no gráfico na Figura 21. É perceptível que salvo um erro maior na medição do ângulo de 45 º, os demais foram bem baixos.

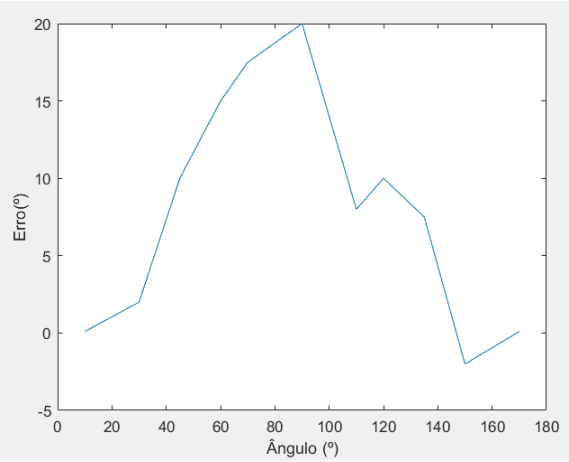
Figura 21 – Gráfico de Erro entre os ângulos obtidos pelo algoritmo 1 e os reais valores reais, linha feita em computador.



Fonte: Próprio autor.

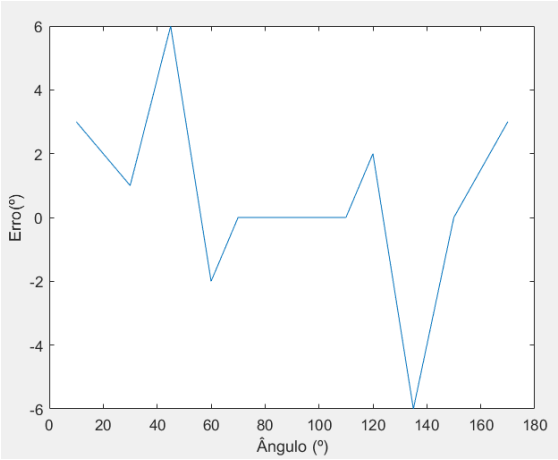
Tendo feito isso para o algoritmo 1, foi repetido o mesmo processo para os outros dois algoritmos, os quais são Método dos Mínimos Quadrados e Transformada de Hough, respectivamente Figura 22 e 23. Na Figura 22, é possível ver que a medida que o ângulo da reta se aproxima de 90 º o erro aumenta, provavelmente pelo fato de por esse método tentar-se enquadrar uma reta entre os diversos pontos. Na figura 34, fica-se evidente que os maiores erros ocorreram no ângulo de 45 º e 135 º.

Figura 22 - Gráfico de Erro entre os ângulos obtidos pelo algoritmo 2, Mínimos Quadrados, e os reais valores reais, linha feita em computador.



Fonte: Próprio autor.

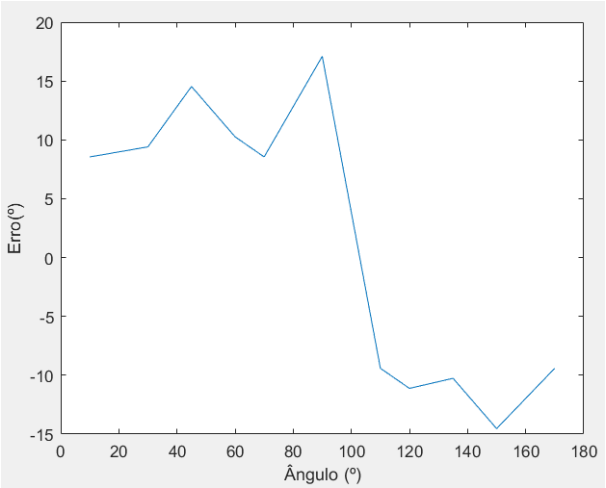
Figura 23 - Gráfico de Erro entre os ângulos obtidos pelo algoritmo 3, Transformada de Hough, e os reais valores reais, linha feita em computador.



Fonte: Próprio autor.

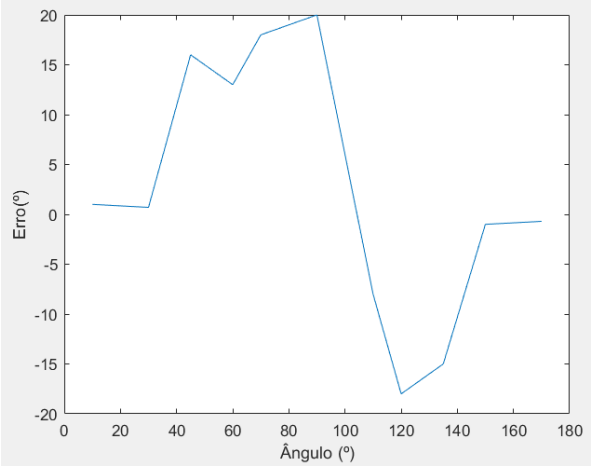
Feito isso, partiu-se para os testes com as imagens obtidas pela webcam. As imagens testadas apresentavam faixas com as mesmas angulações das testadas anteriormente, os gráficos com os erros correspondentes estão nas Figuras 22, 23 e 24. Os erros agora são maiores do que o apresentado nos testes com linhas. Isso provavelmente ocorre devido a largura de faixa, que influência bastante nos métodos dos Mínimos Quadrados e Transformada de Hough, principalmente.

Figura 24 - Gráfico de Erro entre os ângulos obtidos pelo algoritmo 1 e os reais valores reais, fotos webcam.



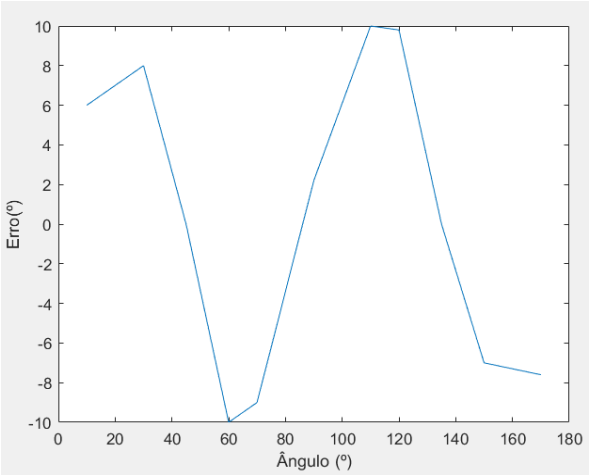
Fonte: Próprio autor.

Figura 25 - Gráfico de Erro entre os ângulos obtidos pelo algoritmo 2, Mínimos Quadrados, e os reais valores reais, fotos webcam.



Fonte: Próprio autor.

Figura 26 - Gráfico de Erro entre os ângulos obtidos pelo algoritmo 3, Transformada de Hough, e os reais valores reais, fotos webcam.



Fonte: Próprio autor.

Posteriormente, pensou-se em testar o desempenho desses algoritmos em casos onde ocorressem mudança de ângulo no percurso. O primeiro caso de reta com mudança de angulação é o apresentado na Figura 25. Como toda a faixa está presente na imagem, buscou-se associar tal imagem ao ângulo resultante que esta teria, ou seja, a resultante que o movimento total do AGV teria que realizar.

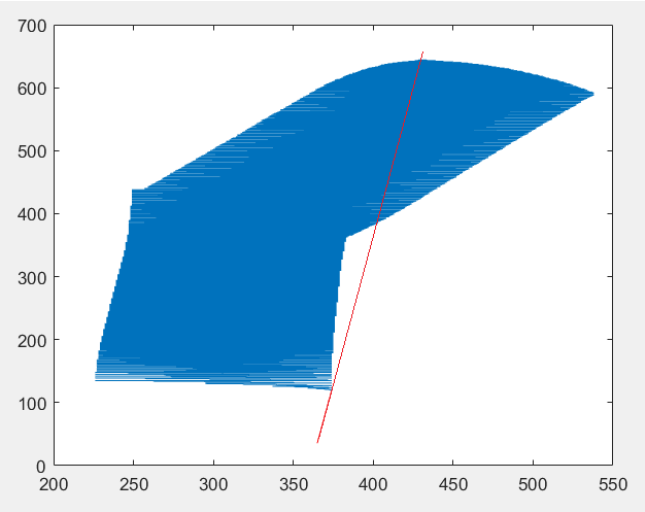
Figura 27 – Foto de faixa com mudança de angulação, resultante de 70º.

Fonte: Próprio autor.

Para o caso anterior também foi feita a avaliação pelos três algoritmos implementados em MatLab, obtendo os resultados mostrados a seguir. Com os gráficos das Figuras 28 a 30 é

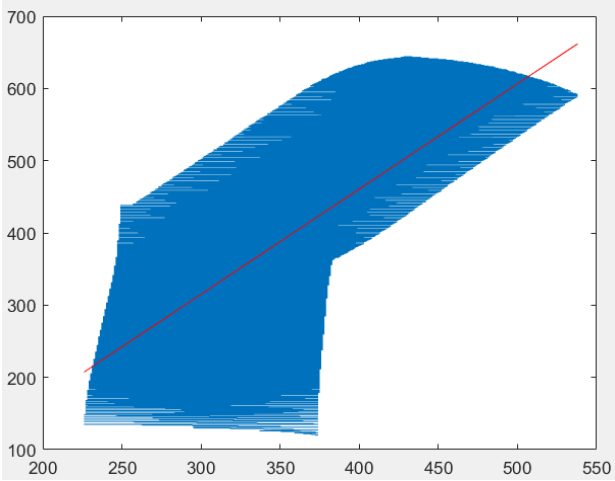
visível que houve uma variação, mas posteriormente veremos que em termos quantitativos foi bem pouco.

Figura 28 – Aplicação do algoritmo de cálculo de angulação com pontos inferior e superior.



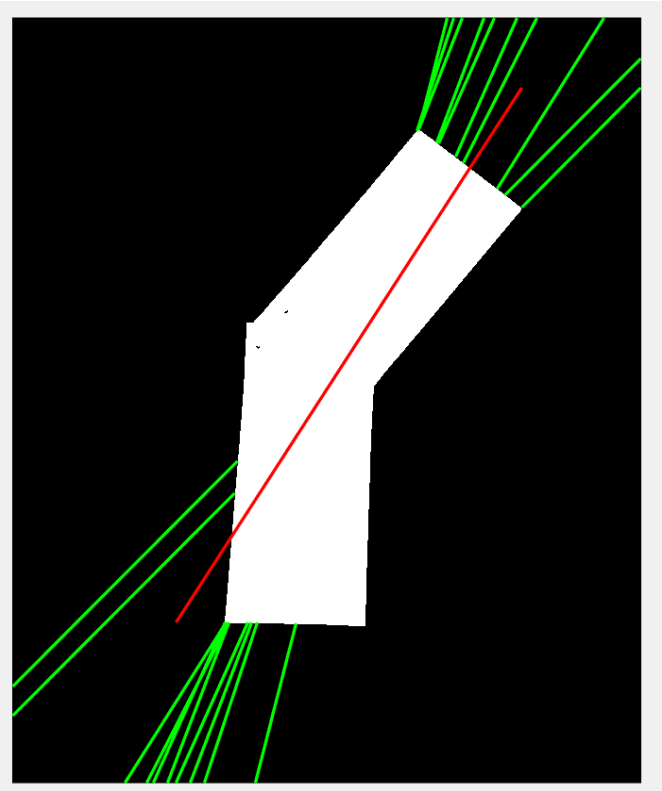
Fonte: Próprio autor.

Figura 29 - Aplicação do algoritmo de Mínimos Quadrados.



Fonte: Próprio autor.

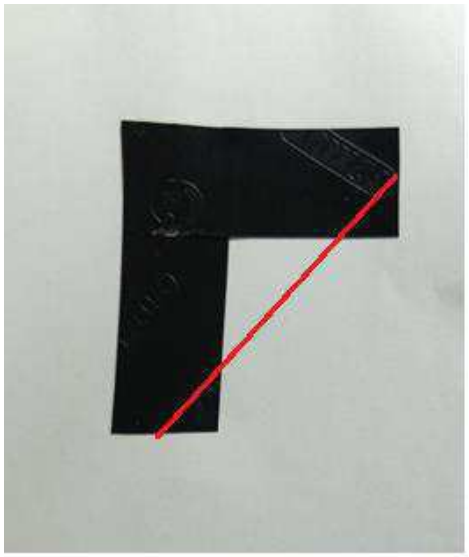
Figura 30 - Aplicação do algoritmo de Transformada de Hough, com plotagem de todas as linhas observadas pelo método e a resultante.



Fonte: Próprio autor.

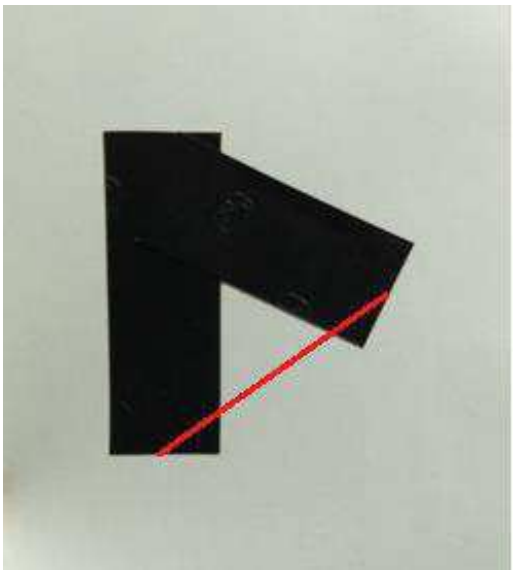
Também foram avaliados dois outros casos de mudança de angulação de faixa, mostrados nas Figuras 31 e 32. Foram avaliados os três algoritmos para os casos. Foi verificada uma variação considerável para cada algoritmo. A seguir é apresentado os erros referentes as três mudanças de angulação na Tabela 1.

Figura 31 - Foto de faixa com mudança de angulação, resultante de 40º.



Fonte: Próprio autor.

Figura 32 - Foto de faixa com mudança de angulação, resultante de 20º.



Fonte: Próprio autor.

A Tabela 1 apresenta a quantidade em graus que ultrapassou o valor do ângulo resultante, por exemplo, o último dado da tabela, o erro da Transformada de Hough para o caso do ângulo resultante de 70º indica que o erro da estimativa pelo algoritmo ultrapassou em 14º o valor correto, implicando numa estimativa de 84º, em vez de 70º.

Tabela 1 - Tabela que apresenta os erros de leitura em graus para cada caso de mudança de ângulo.

Ângulo Resultante	Erro Algoritmo 1	Erro Mínimos Quadrados	Erro Transformada de Hough
20º	79º	18,8º	61º
40º	54º	15º	5º
70º	13º	13º	14º

Fonte: Próprio autor.

5.2 Influência de estimativa de ângulo devido a variação de do offset da faixa

Os testes realizados em seguida foram feitos com imagens da câmera USB, mantendo a inclinação da faixa e variando apenas a distância ‘d’. Com intuito de verificar se essa variação tem alguma implicação para algum dos três métodos. Partiu-se de uma imagem onde se tinha o centro da faixa no centro da imagem e foi afastando essa faixa do centro a uma medida de 20 pixels para a direita, até que a ponta da faixa se encontrasse no limite da tela, Figura 33. Com o passar desse distanciamento foram utilizados os três métodos para cada instante para verificar se o erro variava com essa movimentação e em caso afirmativo, quanto ele viria a variar.

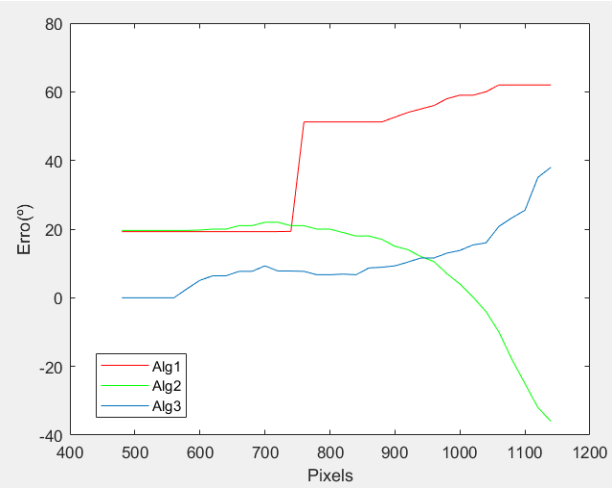
Figura 33 – Limite de teste com deslocamento de faixa para a direita.

Fonte: Próprio autor.

Sendo assim, percebeu-se que com o afastar da faixa do centro da imagem da câmera vai ocorrendo um aumento do erro na estimativa do ângulo da faixa. Em virtude dessa verificação, também foi implementada no algoritmo o cálculo da distância ‘d’. Os erros foram mensurados para os três algoritmos utilizados. Como visto previamente, o algoritmo 1 foi o de estimativa da reta por dois pontos da borda, o algoritmo 2 o Método dos Mínimos Quadrados e

o algoritmo 3 a Transformada de Hough que mais uma vez, apresentou os melhores resultados, Figura 34. Nessa imagem, fica claro que o algoritmo que sofre uma influência mais suave com relação a essa movimentação da faixa em ‘d’ é o da Transformada de Hough.

Figura 34 – Gráfico de erro de estimativa de ângulo em função do distanciamento da faixa do eixo central da câmera.

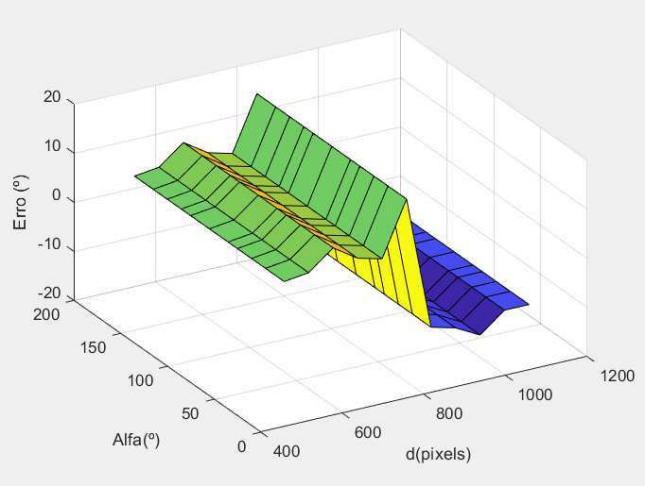


Fonte: Próprio autor.

5.3 Erro de estimativa na variação do ângulo e do offset da faixa

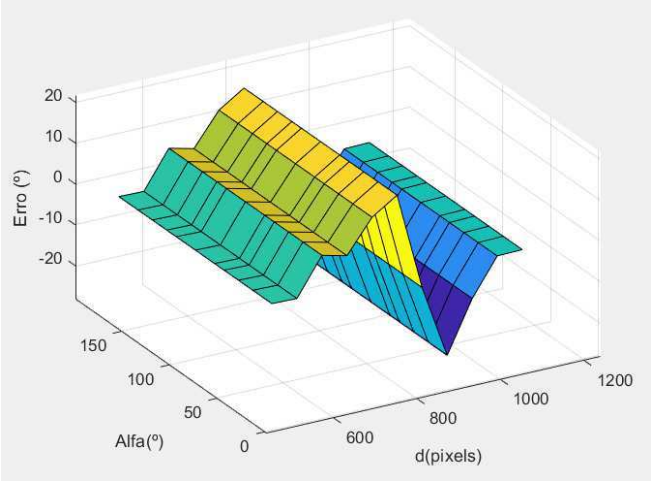
Percebido a significativa influência dessa variação de ‘d’ o próximo passo foi plotar o gráfico que englobasse todas essas informações, colocando no eixo ‘x’ a distância ‘d’, em ‘y’ o ‘∝’ e em ‘z’ o erro. Obtendo os gráficos presentes nas Figuras 35, 36 e 37. Pelos gráficos de superfície é possível ver que os erros de estimativa para um mesmo ‘d’ variam muito pouco. Todavia, quando se deseja estimar um mesmo ângulo e variasse bastante o ‘d’ são encontrados erros significativos. Os erros máximos permanecem praticamente os mesmos dos testes realizados com ‘d’ fixo, estes erros são mais significativos com o aumento da distância com relação ao eixo central, ‘d’.

Figura 35 – Gráfico 3d dos erros para o algoritmo 1.



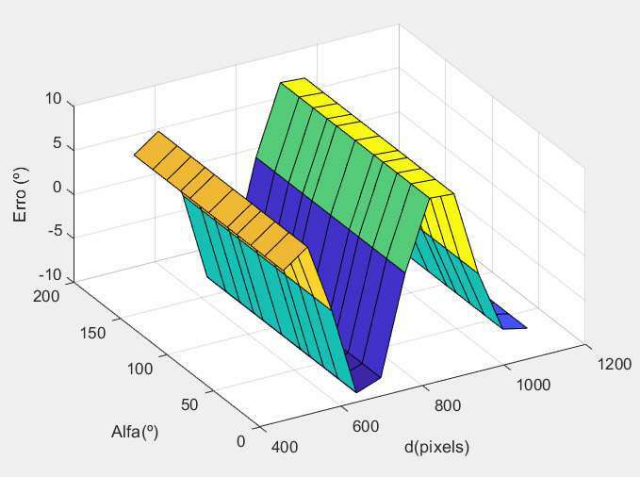
Fonte: Próprio autor.

Figura 36 - Gráfico 3d dos erros para o algoritmo 2.



Fonte: Próprio autor.

Figura 37 - Gráfico 3d dos erros para o algoritmo 3.



Fonte: Próprio autor.

5.4 Algoritmo em Python e resultados

Uma vez constatado que o algoritmo que apresenta a menor quantidade de erro na identificação dos erros, o algoritmo da Transformada de Hough, foi feita a implementação desse algoritmo em Python, com a finalidade de validar a qualidade do algoritmo direto no Raspberry Pi, onde será feito, de fato, a coleta dos dados.

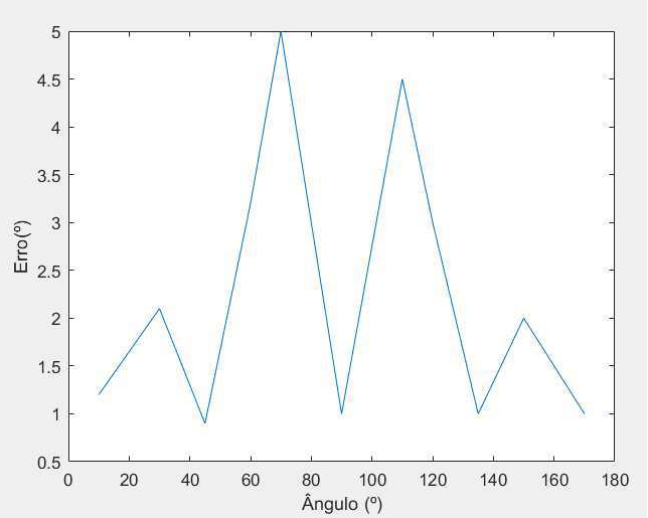
Devido a ideia inicial do projeto o algoritmo em Python deveria ter sido gerado automaticamente do Simulink. Porém, como ao longo do desenvolvimento do trabalho foi verificada uma dificuldade na operabilidade dos blocos do Simulink destinada ao Raspberry, escolheu-se implementar o algoritmo em Python sem auxílio do Simulink.

Existem algumas bibliotecas, também para Python, que fornecem a função de transformada de Hough, foi analisado se a função possuía estrutura similar a função de Transformada de Hough no MatLab, utilizando-a após.

Foram feitas algumas adaptações de informações e implementação de código para que as informações de ângulo e de ‘d’ fossem padronizadas.

Sendo assim, foi feito novamente os testes com as 11 imagens e plotado o gráfico de erro, dessa vez para o código já em Python, Figura 38. Verificando assim, que a faixa de erros para este algoritmo permanece praticamente a mesma.

Figura 38 - Gráfico de Erro entre os ângulos obtidos pelo algoritmo 3, Transformada de Hough, e os reais valores reais, fotos webcam, código em Python.



Fonte: Próprio autor.

Finalizando assim, a ideia de utilização do *Model-Based Design*, uma vez que após testes e alterações, implementou-se o código na linguagem a ser utilizada na placa, também, a ser utilizada.

6 Resultados experimentais

De acordo com os experimentos realizados foi possível perceber que existe uma variação distinta de erro entre os três métodos utilizados, obtendo erros maiores próximo aos 90º. Outra constatação verificada foi a de que as imagens geradas em computador apresentam um pouco menos de erros de medição, algo que de certa forma era esperado por apresentar uma imagem com branco e preto apenas, sem gradação de cor e sem interferência da iluminação do ambiente. Além de não sofrer tanto com a espessura da faixa, o que influência um pouco no erro da estimativa.

Quando os testes foram feitos com as imagens da faixa o algoritmo que apresentou um melhor resultado foi o da Transformada de Hough, com erros variando de 10 a -10 graus de erro, o menor erro dentre os três algoritmos utilizados.

No caso em que houve mudança na angulação na faixa os erros presentes foram bem maiores, como mostrado na Tabela 1. Todavia, o algoritmo 1 e de Mínimos Quadrados, nesse caso, foram os que apresentaram os erros mais baixos para uma mudança mais suave. Já no

caso de uma mudança de angulação 90º o melhor algoritmo seria o de Transformada de Hough e por fim, no caso de mudanças mais drásticas, o de Mínimos Quadrados foi a melhor opção.

Após a implementação em Python foi verificado que a qualidade do algoritmo de Transformada de Hough foi mantido, obtendo erros até um pouco mais baixos do que o mesmo algoritmo realizado em MatLab, possivelmente por algum detalhe implementado na biblioteca em Python ou por algum erro de medição.

Houveram algumas alterações no escopo inicial do projeto, como a construção do código em Simulink para geração automática sendo alterada para escrita de código em Python devido a problemas de operabilidade dos blocos no Simulink para Raspberry. A quantidade de faixas para testes não foi tão elevada quanto desejado inicialmente, porém foi possível ter uma noção do comportamento dos algoritmos.

7 Conclusão

Este trabalho teve como proposta de processamento de imagem com câmera para detecção de faixa. Em virtude de existirem diversas possibilidades para esse processamento, utilizou-se o MatLab em comunicação com o Raspberry Pi 3B com a implementação de três algoritmos diferentes que fossem capazes de realizar a determinação de angulação de faixas analisadas.

Além da análise dos três algoritmos foi analisada ainda a diferença de qualidade dessa verificação de angulação e da distância ‘d’ com as faixas criadas por programa e as faixas verificadas por fotos da webcam. Verificando que as imagens criadas em computador apresentam um pouco menos de erro por não sofrerem influência de iluminação, por exemplo.

A ideia do *Model-Based Design* foi utilizada com intuito de utilizar um software, como o MatLab, para testar os algoritmos e gerar os gráficos de erros atestando a qualidade desses algoritmos, e uma vez verificado qual o melhor algoritmo implementá-lo no uso final, que seria no Raspberry Pi em Python. Onde, também foram verificados bons resultados.

Por fim, foi verificado, também, que para os casos de mudança de angulação de faixa tem-se os algoritmos que melhor se enquadram na estimação dessa angulação resultando, dependendo do tamanho dessa variação.

8 Bibliografia

Aarenstrup, R., “Managing Model-Based Design”, Mathworks, 2015.

BARROS, C. P. B. "Desenvolvimento da instrumentação e comunicação para um veículo autoguiado". Relatório de Estágio, Universidade Federal de Campina Grande (UFCG), 2016.

CERQUEIRA, A. M. L. G. “Estudo e implementação de um algoritmo eficaz de detecção de retas”. Universidade Federal de Minas Gerais. Disponível em: https://www.researchgate.net/publication/228387894_Estudo_e_implementacao_de_um_algoritmo_eficaz_de_deteccao_de_retas. Acesso em: Novembro de 2018.

Cybernet. “O que é e como usar SSH?”. Diponível em: https://www.cybernetfx.com/clientes/knowledgebase.php?action=displayarticle&id=71. Acesso em: Novembro de 2018.

GOMES, M. V. et al "PID Control applied on a line-follower AGV using a RGB camera". IEEE 19th International Conference on Intelligent Transportation System (ITSC), 2016.

Leonel, R. S., Dissertação de Mestrado, “Estudo e Especificação de um Veículo Autoguiado para para Crianças com Deficiência Motora”. Disponível em: http://repositorio.unicamp.br/bitstream/REPOSIP/259273/1/Leonel_RicardoSucaria_M.pdf. Acesso em: Novembro de 2018.

Mathworks, White paper, “How small engineering teams adopt model based design”, 2014.

MARROQUIM, A. O. R., “Processamento de Imagens COS756/COC603”, UFRJ. Disponível em: https://www.lcg.ufrj.br/marroquim/courses/cos756/lessons/13-sift-hough.pdf. Acesso em: Novembro de 2018.

SINOVA. Disponível em: http://www.sinova.com.br/tipos-de-agvs/. Acesso em: Setembro de 2018.

Visão Computacional, “Reconhecimento de Formas: A Transformada de Hough”, UFSC. Disponível em: http://www.inf.ufsc.br/~aldo.vw/visao/2000/Hough/index.html. Acesso em: Novembro de 2018.

Webautomotivo. “Rede CAN – O que é e como funciona.” Disponível em: http://dicas.webautomotivo.com.br/rede-can-o-que-e-e-como-funciona/. Acesso em: Novembro de 2018.

Weber, M., “Model Based Developmente and Testing of Automotive Software – Some Experiences and Challenges”, Carmeq, 2011.

Wharton, “Por que a IA é fator decisivo no desenvolvimento dos carros autoguiados” .Disponível em: http://www.knowledgeatwharton.com.br/article/por-que-ia-e-fator-decisivo-no-desenvolvimento-dos-carros-autoguiados/. Acesso em: Novembro de 2018.

Wikipedia, “Método dos mínimos quadrados”. Disponível em: https://pt.wikipedia.org/wiki/M%C3%A9todo_dos_m%C3%ADnimos_quadrados. Acesso em: Dezembro de 2018.

Wolfram. Disponível em: http://mathworld.wolfram.com/LeastSquaresFitting.html. Acesso em: Novembro de 2018.

ANEXO

Código em Python:

```
import cv2
import numpy as np
import math

cam = cv2.VideoCapture(0)

cv2.namedWindow("test")

img_counter = 0

while True:
    ret, frame = cam.read()
    cv2.imshow("test", frame)
    if not ret:
        break
    k = cv2.waitKey(1)

    if k%256 == 32:
        # SPACE pressed
        img_name = "opencv_frame_{}.jpg".format(img_counter)
        cv2.imwrite(img_name, frame)
        print("{} written!".format(img_name))
        img_counter += 1

#Codigo da Transformada de Hough
    if k%256 == 27:
        image1 = cv2.imread('/home/pi/Desktop/opencv_frame_0.jpg')

        try:
            image1.shape
```

```
        print("checked for shape".format(image1.shape))
    except AttributeError:
        print("shape not found")
    #changing to gray scale
    gray=cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
    dst = cv2.Canny(gray, 50, 200)

    lines= cv2.HoughLines(dst, 1, math.pi/180.0, 100, np.array([]), 0, 0)
    cv2.imshow("Imagem",image1)

    a,b,c = lines.shape
    for i in range(a):
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        a = math.cos(theta)
        b = math.sin(theta)
        x0, y0 = a*rho, b*rho
        pt1 = ( int(x0+1000*(-b)), int(y0+1000*(a)) )
        pt2 = ( int(x0-1000*(-b)), int(y0-1000*(a)) )
        cv2.line(image1, pt1, pt2, (0, 0, 255), 2, cv2.LINE_AA)
        xtotal = xtotal + (pt1[0] + pt2[0])
        ytotal = ytotal + (pt1[1] + pt2[1])

xm = xtotal/((i+1)*2)
ym = ytotal/((i+1)*2)
print('xm',xm)
print('ym',ym)

xo=329
yo=-14.8

deltax=xm-xo
deltay=ym-yo
```

```
d = math.sqrt((deltax**2) + (deltay**2))

        print(theta)
        theta = theta*57.29
        print(theta)
        if theta>=90:
            theta = 90 - theta
        else:
            theta = -(-90+theta)

        cv2.imshow('image1',image1)
        print(theta)
        cv2.waitKey(0)
        cv2.destoryAllWindows(0)
```