Centro Federal de Educação Tecnológica de Minas Gerais

Campus Varginha

Luan Sousa Reis

Sistema para robô seguidor de linha utilizando OpenCV e Python.

Varginha

2019

Lista de ilustrações

Lista de tabelas

Sumário

1 Descrição do Projeto

O projeto proposto no programa de Bolsa de Iniciação Cientifica Júnior (BIC Júnior) foi planejar, simular e programar um robô de locomoção autônomo que possa, utilizando visão computacional, percorrer uma trajetória predeterminada por uma linha preta em um fundo branco. O projeto consiste na obtenção de imagens de uma câmera que, a partir de alguns métodos de processamento de imagens, determine qual a intensidade que os motores presentes no lado esquerdo e direito deve funcionar.

2 Introdução

Os robôs apesar de uma invenção do século XX, foram idealizados na antiguidade. Existem diversos relatos históricos da construção de autônomos como, por exemplo, o cachorro de brinquedo encontrado no Egito datado de 1390 AC à 1353 AC e os bonecos de madeiras japoneses *Karakuri* bem populares durante o século XVI.

Figura 1: Boneco Karakuri em exibição.

Fonte: Wikipédia (2019)

A alavanca para o desenvolvimento de máquinas autônomas surgiu com a Primeira Revolução Industrial, quando foram desenvolvidos dispositivos automáticos com a capacidade de manipulação de peças, automatizando a produção e consequentemente diminuindo o preço dos produtos.

Os teares mecânicos das indústrias têxteis foram as primeiras máquinas automatizadas e permitiram a produção em massa de tecidos e no aperfeiçoamento do produto final. A partir de então, com a crescente necessidade imposta pelo mercado e com a ajuda de novas tecnologias que surgiram ao longo do tempo, as máquinas automáticas vem se expandindo com o objetivo de desenvolver sistemas de produção mais eficazes e baratos.

A Visão Computacional é o processo de planejamento e replicação da visão humana usando *software* e *hardware*. A pesquisa da sobre visão computacional começou na década de 50 em três linhas distintas:

- Replicar o funcionamento do olho;

- Replicar o funcionamento do córtex visual;
- Reproduzir o funcionamento do resto de cérebro.

A parte em que tivemos o maior sucesso foi na de replicação do funcionamento do olho. Nas últimas décadas, foram criados sensores e processadores de imagem que combinam e, em alguns aspectos superam as capacidades do olho humano.

Nas indústrias atuais, é comum a utilização de sistemas que utilizam algum tipo de guia para o deslocamento de robôs, como fitas refletoras e canaletas.

3 Descrição do Projeto

Este projeto propõe a implementação de um sistema, utilizando visão computacional, capaz de seguir por um caminho predeterminado por uma fita e guiado por imagens obtidas de uma câmera.

3.1 Objetivo Geral

Desenvolver um sistema em capaz de se movimentar em uma trajetória demarcada por uma faixa escura em um fundo claro através de imagens com apenas duas cores chamadas de imagens binárias geradas a partir de imagens vindas de uma câmera.

3.2 Objetivos Específicos

- Entender o funcionamento da visão computacional;
- Entender o funcionamento dos GPIOs do *Raspberry Pi*;
- Programar um sistema capaz de gerar imagens binárias e determinar caminhos;
- Implementar o sistema em um *Raspberry Pi* e simular os motores;
- Testar o sistema no *Raspberry Pi* com os motores conectados.

4 Revisão Bibliográfica

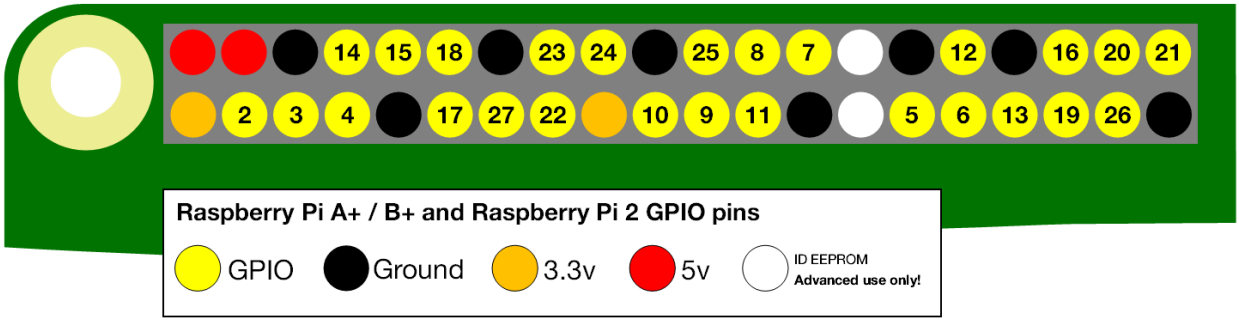
Com o intuito de uma melhor compreensão de componentes e procedimentos realizados neste trabalho, segue abaixo uma breve revisão bibliográfica dos principais temas e conceitos teóricos necessários para o entendimento do projeto.

4.1 *Raspberry Pi*

Raspberry Pi é uma linha de computadores de placa única com um tamanho reduzido.

4.1.1 *GPIOs* do *Raspberry Pi*

Figura 2: Numeração dos pinos GPIO do modelo 2 do Raspberry Pi.



Fonte: Documentação online do Raspberry Pi

Os GPIOs são uma linha de pinos de entrada e saída ao longo da borda superior da placa do *Raspberry Pi*.

4.2 *Python*

Python é uma linguagem de programação de alto nível, interpretada,, orientada a objetos, de tipagem dinâmica e forte.

4.3 *OpenCV*

OpenCV é uma biblioteca de visão computacional e aprendizado de máquina de código aberto.

4.4 Programação Orientada a Objetos

Programação Orientada a Objetos se trata de um padrão de desenvolvimento de software presentes em diversas linguagens de programação. Ela foi criada para aproximar o mundo real com a programação por meio de objetos uma vez que nosso mundo é composto por esses.

4.4.1 Objetos

Objetos são capazes de reter e oferecer uma série de informações . São os pilares da Programação Orientada a Objetos.

4.4.5 Classes

São os moldes dos objetos. Elas determinam qual informação um objeto pode conter e como ele pode manipulá-la.

4.4.5.1 Atributos

São as informações em si que o objeto pode apresentar.

4.4.5.2 Métodos

São procedimentos que realizam ações do objeto.

4.5 Imagem Binária

São imagens na qual cada pixel é totalmente preto ou totalmente branco, sendo assim, possuindo apenas duas cores. Elas são geradas a partir de imagens em escala de cinza utilizando métodos de limiarização.

Figura 3: Exemplo de imagem binária.

Fonte: Página da Universidade Estadual de Ohio (2019)

4.6 PWM

A modulação por largura de pulso (MLP) ou *Pulse-Width Modulation* PWM é uma técnica utilizada para controlar a energia fornecida a equipamentos elétricos. Com o PWM é possível controlar a tensão e corrente ao desligar e ligar o fornecimento de energia.

4.6.1 Duty Cycle

Duty Cycle é o tempo em que o fornecimento de energia permanece ativo.

5 Estudo sobre Visão Computacional e GPIO

Existem pela rede diversas bibliotecas de programação sobre visão computacional, porém a que apresenta maior conteúdo sobre si é a *OpenCV*. Ela é facilmente encontrada ao procurar projetos de programação relacionado a detecção de objetos em imagens e afins. Neste caso, ela foi encontrada ao procurar projetos semelhantes ao apresentado nesse relatório no site oficial do *Raspberry Pi*.

A biblioteca *OpenCV* é disponível em várias linguagens de programação como C++ e Lua, porém a linguagem escolhida foi Python, o motivo foi o ganho de tempo de desenvolvimento ao se utilizar ela já que se trata de uma linguagem de programação de alto nível ,sendo assim apresenta uma sintaxe mais próxima da nossa linguagem.

No início do projeto os códigos eram escritos de forma estruturada com o intuito de estudar as funções, mas posteriormente houve alterações que serão explicadas ao longo do relatório. O ambiente de desenvolvimento nesse processo foi Geany instalado em um computador com sistema operacional Windows na sua versão 7 e câmera usada foi a *webcam Powerpack VX-17*.

5.1 Obtenção e processamento de imagens

A biblioteca *OpenCV* oferece diversas maneiras para se detectar objetos. No caso deste projeto o processo foi de obtenção de uma imagem binária e a partir dela controlar os motores. Para isso obtêm-se a imagem da câmera com o método *read* da classe *VideoCapture* e depois esta imagem é, respectivamente, redimensionada para o tamanho de 160 pixeis por 60 pixeis transformada em escala de cinza, borrada para remover ruídos e transformada em uma imagem binária a partir das funções *cvtColor*, g*aussianBlur* e *threshold*.

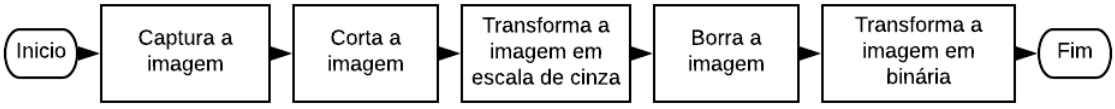
5.1.1 Binarização

A biblioteca *OpenCV* oferece diversas formas de criar uma imagem binária e após uma análise foi escolhido o método *Otsu*. A figura a seguir ilustra a diferença entre os métodos:

5.1.2 Conclusão

Após a idealização do processo de obtenção de imagem binária iniciou o processo de viu se a necessidade de usar a imagem binária para determinar como os motores de cada lado deviam funcionar.

Figura 4: Fluxograma do método de processamento da imagem.



Fonte: Própria (2019)

Ainda usando a biblioteca *OpenCV*, usando a função *findContours*, conseguimos obter os contornos da linha preta e com eles conseguimos achar a coordenada na imagem do meio da trilha. Com essa coordenada verificamos em que posição ela está por meio da comparação da coordenada *x* do meio da imagem, ou seja, 80. A saída desse processo são 3:

- Ir para a direita se a coordenada do contorno estiver a direita do meio da imagem.

- Ir para a esquerda se a coordenada do contorno estiver a esquerda do meio da imagem.
- Ir em frente se se a coordenada do contorno estiver no meio da imagem.

A partir de algumas pesquisas foi descoberta uma biblioteca nativa do *Raspberry Pi* que manipula as suas GPIOs. A partir daí viu se a necessidade de trocar o ambiente de programação para uma máquina virtual com o sistema operacional *Raspbian* do *Raspberry Pi* e assim foi porém mantendo o *Geany*.

Instalando uma biblioteca chamada *Fake Raspberry Pi* foi possível simular esses pinos de maneira digital e assim continuar com o projeto mas antes o código precisava de mudança para uma melhor compreensão foi então que ele foi alterado para o padrão Orientado a Objetos dividindo as tarefas entre classes de forma que cada classe tivesse funções semelhantes ficando dividido assim:

- Classe *Program* responsável por gerenciar as demais classes e executar o programa;
- Classe *Camera* responsável por capturar imagens da camera;
- Classe *ImageEffect* responsável por aplicar efeitos e transformações na imagem;
- Classe *LineDetect* responsável por encontrar os contornos da trilha;
- Classe *GpioController* responsável por determinar o DC de cada motor e controlar os pinos GPIOs;

5.2 Configurações

Com a troca do padrão de desenvolvimento, também veio a criação de uma classe que ficaria responsável por determinar valores pelo código com intuito de não ser preciso abrir o arquivo de cada classe para poder modificar tais e sim apenas o arquivo de configurações.

As configurações presentes nele são:

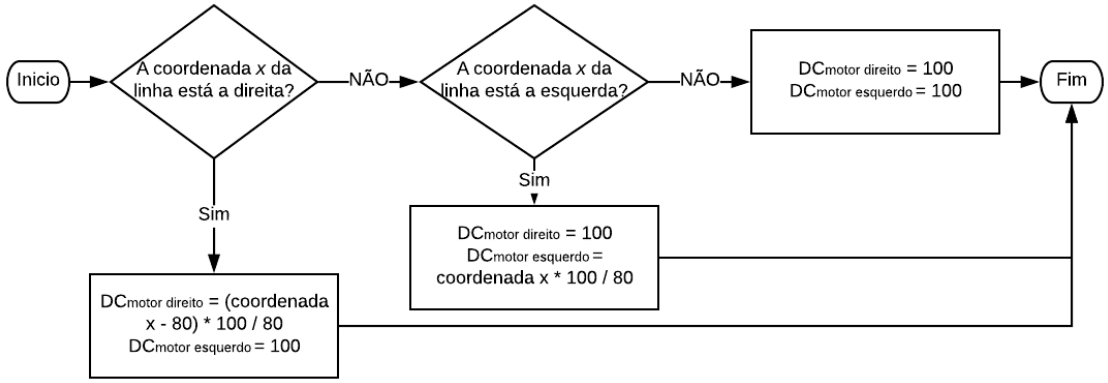
- O tamanho que a imagem deverá ser redimensionada e cortada;
- Qual câmera o código deve usar;
- Se quer que imagens e textos sejam exibidos durante a execução;
- Qual o modo de identificação dos pinos GPIOs será usado;
- Qual são pinos GPIOs que serão usados e quais as suas funções;
- Com qual frequência o código deverá funcionar.

5.3 GPIO

Após a organização, a classe *GpioControler* criada com o intuito de detectar como cada motor deve funcionar e fazer a conexão do código com as GPIOs recebeu um método com as saídas do *Duty Cycle* de cada motor para substituir a anterior que havia somente três saídas.

A figura – ilustra como esse novo método funciona:

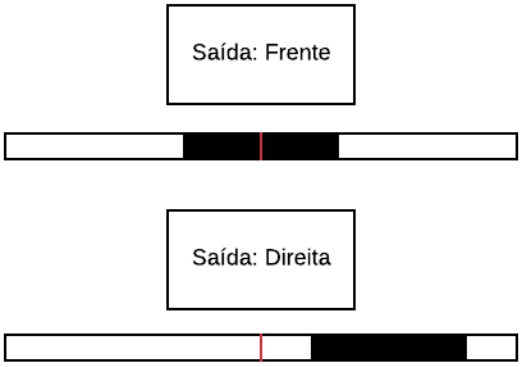
Figura 5: Método anterior de detecção de direção.



Fonte: Própria (2019)

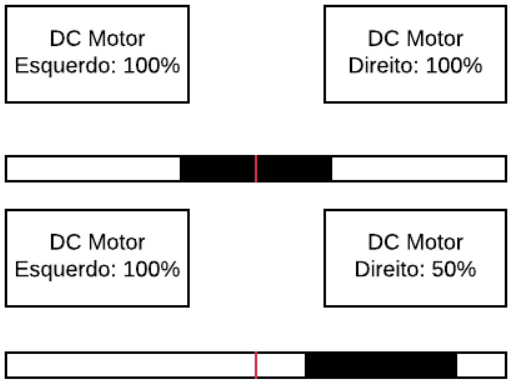
As seguintes figuras ilustram a diferença entre os dois métodos:

Figura 6: Método anterior de detecção de direção.



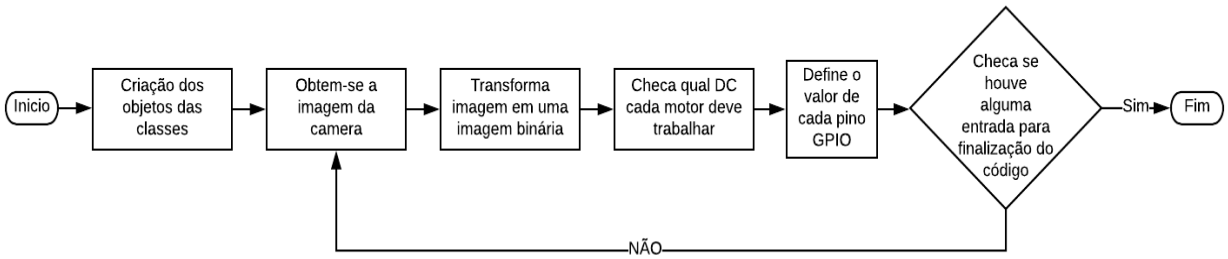
Fonte: Própria (2019)

Figura 7: Método atual de detecção de direção.



Fonte: Própria (2019)

Figura 8: Fluxograma do programa em execução.



Fonte: Própria (2019)

Após a conclusão de testes com imagens na câmera e modificações necessárias no código o programa passou a funcionar da maneira ilustrada na seguinte imagem:

6 Implementação

6.1 *Raspberry Pi*

Após a conclusão do código e teste, começou o processo de migração para o *Raspberry Pi* real que no caso foi a versão 2. Uma das diferenças sofridas com a migração foi o processador da placa. Enquanto que a máquina virtual usava o processador do computador, o *Raspberry Pi* usava um de arquitetura ARM (*Acorn RISC Machine*).

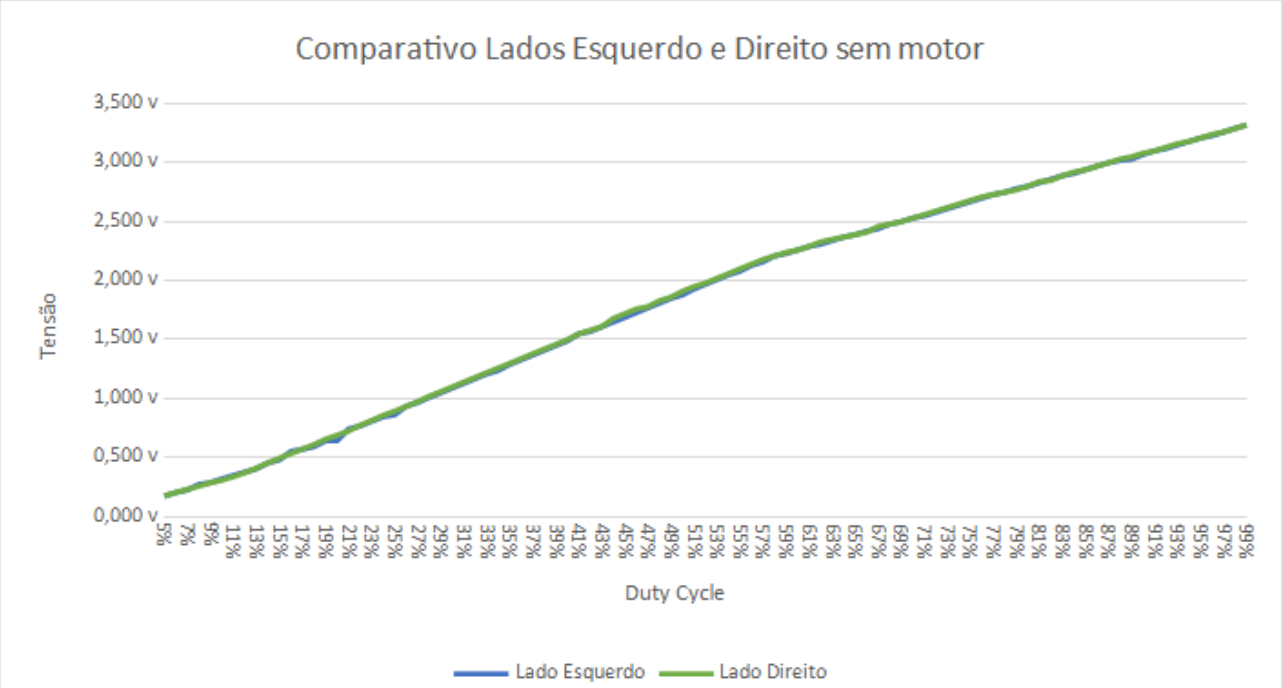
6.2 Configurações

Como não havia a biblioteca *OpenCV* disponível no gerenciador de pacotes padrão do *Raspberry Pi*, foi preciso lhe compilar. Além disso foi preciso instalar o Geany e a câmera NOIR, mas não houveram problemas durante a instalação. Com os pinos GPIOs físicos, a biblioteca de simulação não era necessária e portanto não foi instalada.

6.3 Teste dos Pinos

Após o código estar sendo executado normalmente no *Raspberry Pi*, com a ajuda de um osciloscópio, foram testados os pinos configurados anotando o DC usado junto a tensão. Os dados obtidos dos testes de cada motor ficaram bem próximo apenas, havendo uma pequena variação não grave nos dados finais, significando que os pinos usados com PWM estavam em perfeito estado e que a saída deles era também a esperada.

Figura 9: Gráfico de linha da tensão dos pinos em relação ao DC.



Fonte: Própria (2019)

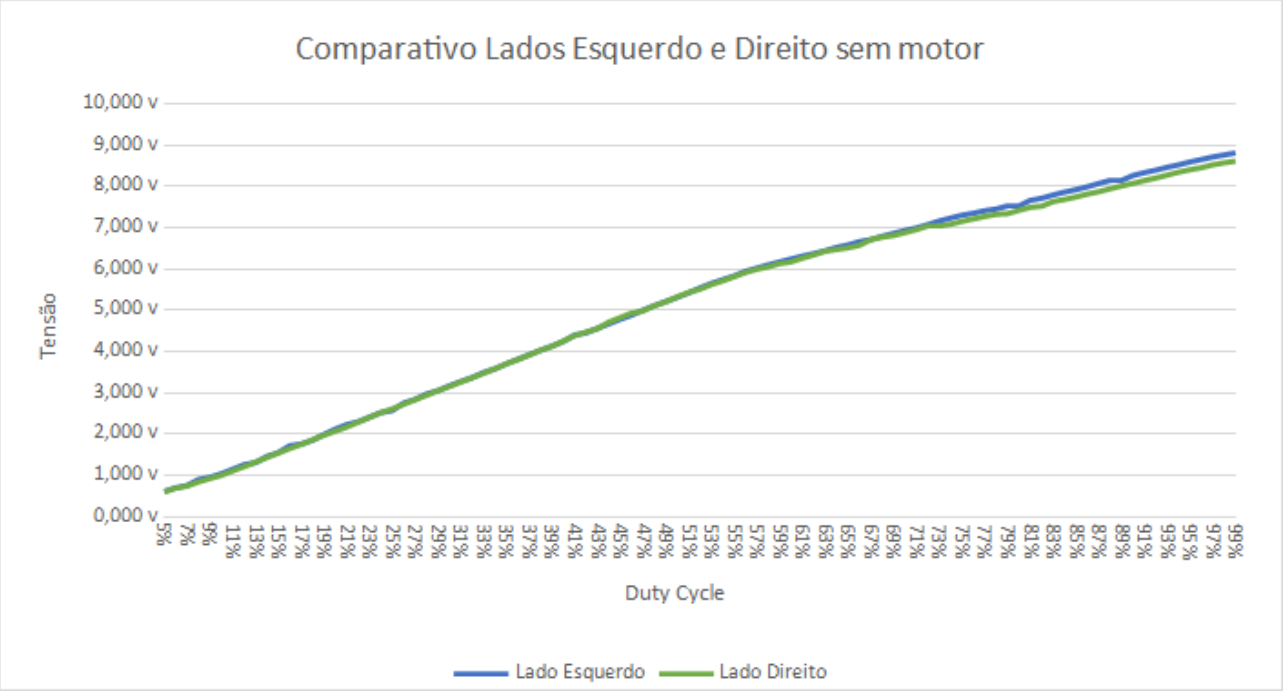
Tabela 1: Resultado estatístico da tensão dos pinos em relação ao DC.

Cálculo	Valor
Média do lado esquerdo	**5,142 v**
Menor do lado esquerdo	**0,614 v**
Maior do lado esquerdo	**8,810 v**
Desvio Padrão do lado esquerdo	**2,488 v**
Média do lado direito	**5,079 v**
Menor do lado direito	**0,599 v**
Maior do lado direito	**8,610 v**
Desvio Padrão do lado direito	**2,435 v**
Média de ambos os lados	**5,110 v**

6.4 Módulo *Driver* Ponte H

Módulo *Driver* Ponte H é um módulo de controle capaz de trabalhar com dois motores DC e portanto foi escolhido para ser usado com o sistema. Para que ele funcionasse eram precisos configurar dois pinos para trabalharem com PWM e mais 4 para definir em que direção os motores vão rodar. Ainda sem motor foram coletados os dados na saída de onde os motores estarão. Os dados obtidos dos testes de saída para motor ficaram bem próximo apenas, havendo uma pequena variação não grave nos dados finais, significando não haviam anormalidades que comprometeriam os resultados esperados.

Figura 10: Gráfico de linha da tensão da saída dos motores em relação ao DC.



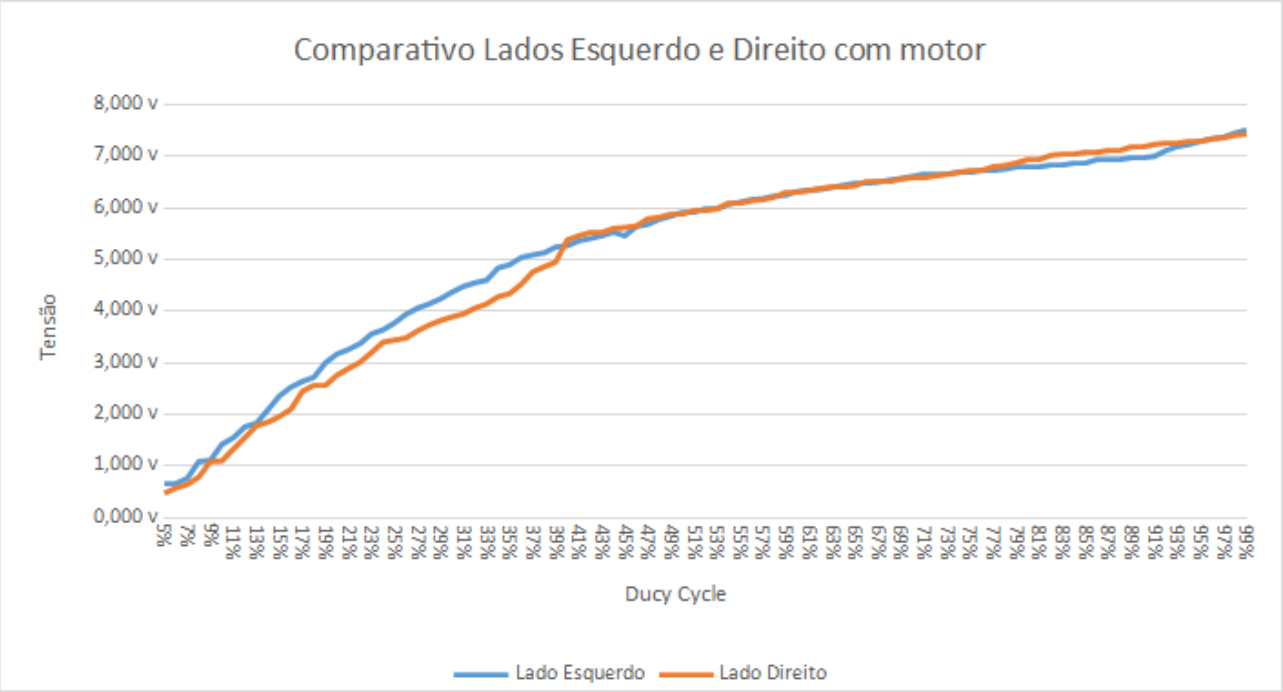
Fonte: Própria (2019)

Tabela 2: Resultado estatístico da tensão da saída dos motores em relação ao DC.

Cálculo	Valor
Média do lado esquerdo	**5,142 v**
Menor do lado esquerdo	**0,614 v**
Maior do lado esquerdo	**8,810 v**
Desvio Padrão do lado esquerdo	**2,488 v**
Média do lado direito	**5,079 v**
Menor do lado direito	**0,599 v**
Maior do lado direito	**8,610 v**
Desvio Padrão do lado direito	**2,435 v**
Média de ambos os lados	**5,110 v**

6.5 Motores

Figura 11: Gráfico de linha da tensão da saída dos motores ligados em relação ao DC.



Fonte: Própria (2019)

Uma vez que foram obtidos resultados foram resolutos, foram ligados os motores, mantendo o osciloscópio ligado na mesma saída, e os dados analisaram. Os resultados foram satisfatórios e em comparação com os resultados dos dois motores, se mantiveram semelhantes em vários momentos porém, houve uma variação do lado direito entre os períodos de DC de 13% e 41% causado pela baixa entrada de energia na Ponte H mas nada que comprometesse os resultados. Os motores também funcionaram perfeitamente como esperado.

.

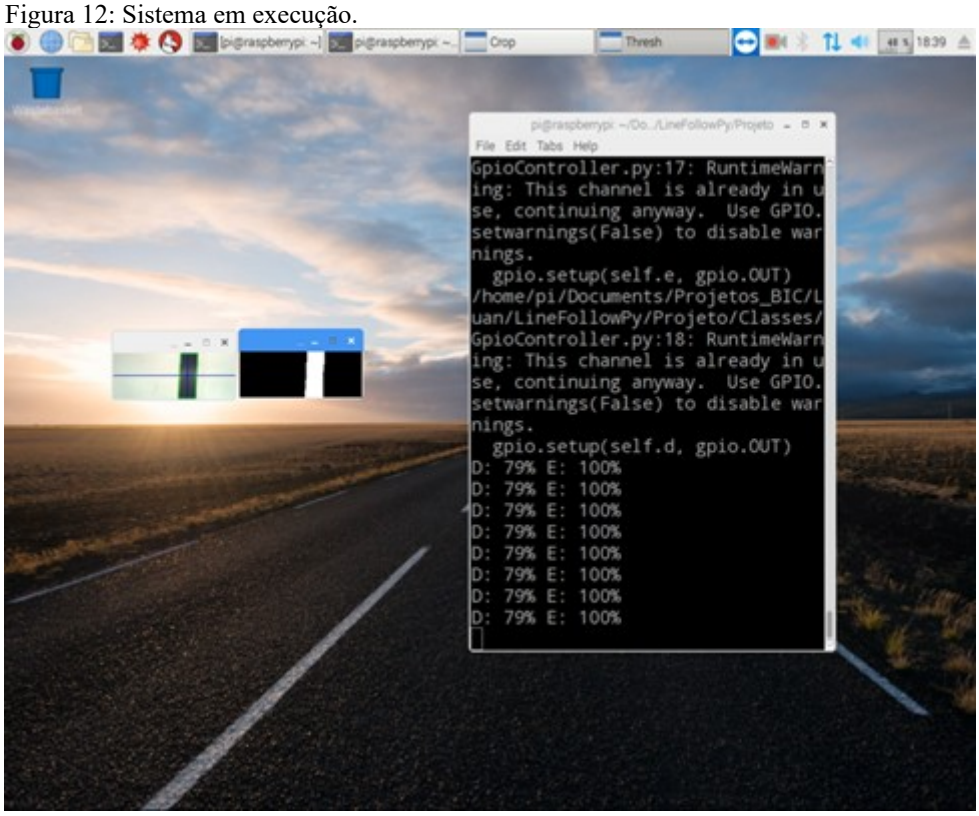
Tabela 3: Resultado estatístico da tensão da saída dos motores ligados em relação ao DC.

Cálculo	Valor
Média do lado esquerdo	**5,268 v**
Menor do lado esquerdo	**0,643 v**
Maior do lado esquerdo	**7,510 v**
Desvio Padrão do lado esquerdo	**1,876 v**
Média do lado direito	**5,178 v**
Menor do lado direito	**0,474 v**
Maior do lado direito	**7,430 v**
Desvio Padrão do lado direito	**2,014 v**
Média de ambos os lados	**5,223 v**

7 Resultados Finais

O projeto levou cerca de um ano para ser desenvolvido, durante esse período, várias alterações desde pequenas mudanças como comentários nos códigos e até a troca do método de detectar caminho surgiram porém mesmo as pequenas foram, no presente, um fator que acrescentou na qualidade do projeto.

Figura 12: Sistema em execução.



Fonte: Própria (2019)

O resultado final atendeu a expectativa no sentido de mostrar que a visão computacional é funcional na tarefa de detecção de objetos e análise de imagens e que organização é a chave de tudo. O projeto também foi capaz que é possível utilizar programação com o *Raspberry Pi* para controlar robôs e afins.

Figura 13: Código da classe de detecção da faixa.

```
1   import numpy as np
2   import cv2
3
4   class LineDetect:
5       """ Classe responssavel por detectar a trilha """
6
7       def findContour(self, thresh):
8           """ Procura pela linha e retorna """
9
10          _,contours,hierarchy = cv2.findContours(thresh.copy(), 1,\
11              cv2.CHAIN_APPROX_NONE)
12          return contours
13
14
15
16      def getMov (self,contours):
17          """ Encontra o meio da trilha e retorna as cordenadas """
18
19          if len(contours) > 0:
20              c = max(contours, key=cv2.contourArea)
21              M = cv2.moments(c)
22              if(M['m00']!=0):
23                  cx = int(M['m10']/M['m00'])
24                  cy = int(M['m01']/M['m00'])
25              else:
26                  cx = int(M['m10'])
27                  cy = int(M['m01'])
28
29          else:
30              #Nao consigo ver a linha
31              cx = 0
32              cy = 0
33
34          return cx,cy
```

Fonte: Própria (2019)

8 Sugestão para trabalhos futuros

No projeto atual se usou somente a parte de visão computacional da biblioteca *OpenCV*, porém essa biblioteca e outras mais oferecem também ferramentas para aprendizado de máquina que poderiam agregar mais ainda no projeto fazendo que invés de um algorítimo decidir qual caminho deve se tomar deixar que a máquina aprenda e ela determine qual o caminho.

Com o avanço da computação, a inteligência artificial e afins estão cada vez mais presentes, o que não falta também é material sobre e como utilizar essas ferramentas. Então, a meu ver, não seria de tamanha dificuldade em melhorar o código para que utilize esses princípios.

9 Conclusões

Seguindo conceitos apresentados no site da biblioteca *OpenCV* e no do *Raspberry Pi*, obteve-se êxito no desenvolvimento do software. A partir de algumas simulações e testes, foi possível modelar e analisar os códigos viabilizando o projeto.

Apesar de alguns imprevistos durante o desenvolvimento, o projeto foi concluído e mostrou que esses imprevistos sempre existirão e que necessário não tentar impedir que apareçam e sim se preparar para eles. Além disso, este projeto foi êxito em mostrar que a ciência da visão computacional apesar de nova, está cada vez mais próxima daquilo que será seu objetivo de imitar a visão humana.

Referência Bibliográfica

Mechanical Dog. Disponível em: <https://www.metmuseum.org/art/collection/search/544519> Acesso em jan. 2019.

Karakuri - Lean Manufacturing. Disponível em: <http://engenhariadeproducaoindustrial.blogspot.com/2009/08/karakuri-lean-manufacturing.html> Acesso em jan. 2019.

Visão Computacional. Disponível em: <https://pt.wikipedia.org/wiki/Vis %C3%A3o_computacional> Acesso em jan. 2019.

O que é Visão Computacional. Disponível em: <http://datascienceacademy.com.br/blog/o-que-e-visao-computacional/> Acesso em jan. 2019.

Computer Vision Is More Than Just Image Recognition. Disponível em: <https://www.forbes.com/sites/forbestechcouncil/2016/08/12/computer-vision-is-more-than-just-image-recognition/#4365c7c6065d> Acesso em jan. 2019.

GPIO. Disponível em: <https://www.raspberrypi.org/documentation/usage/gpio/> Acesso em jan. 2019.

Binary Image. Disponível em: <http://computervision.wikia.com/wiki/Binary_image> Acesso em jan. 2019.

Python. Disponível em: <https://pt.wikipedia.org/wiki/Python> Acesso em jan. 2019.

Image Thresholding. Disponível em: <https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html> Acesso em jan. 2019.

About Raspberry Pi. Disponível em: <https://www.raspberrypi.org/about/> Acesso em jan. 2019.