Universidade Estadual de Maringá
Centro de Ciências Exatas
Departamento de Física

Trabalho de Conclusão de Curso

VISÃO COMPUTACIONAL DO CARRO AUTÔNOMO APLICADO AO CARRO SEGUE FAIXA.

Acadêmico: Rodolfo Ricardo
Orientador: Prof. Dr. Breno Ferraz de Oliveira/UEM
Coorientador: Prof. Guilherme de Souza Rocha/FEITEP
Maringá, 10 de abril de 2022.

Universidade Estadual de Maringá
Centro de Ciências Exatas
Departamento de Física

Trabalho de Conclusão de Curso

VISÃO COMPUTACIONAL DO CARRO AUTÔNOMO APLICADO AO CARRO SEGUE FAIXA.

Trabalho de Conclusão de Curso ao Departamento de Física da Universidade Estadual de Maringá, sob orientação do professor Dr. Breno Ferraz de Oliveira, como parte dos requisitos para obtenção do título de Bacharel em Física.

Acadêmico: Rodolfo Ricardo
Orientador: Prof. Dr. Breno Ferraz de Oliveira/UEM
Coorientador: Guilherme de Souza Rocha/FEITEP
Maringá, 10 de abril de 2022.

SUMÁRIO

FIGURAS

QUADROS

AGRADECIMENTOS

Esse trabalho final de graduação foi o resultado da trajetória acadêmica na área de Física. Também, pela participação no desenvolvimento do projeto institucional entre a Universidade Estadual de Maringá (UEM) em parceria com empresa de Maringá do ramo de tecnologia e robótica. Assim, meus agradecimentos:

- Aos participantes do projeto institucional que deram a oportunidade de obter o conhecimento prático no desenvolvimento do protótipo do Carro Segue Faixa: Prof. Dr. Breno Ferraz de Oliveira (Coordenador/projeto-UEM), Sr. Francisco Alexandre Ribeiro de Alencar (UEM) e equipe; Dra. Denise Adorno Lopes, Sr. Guilherme de Souza Rocha, Sr. Rafael de Souza Rocha (Sigma *Robotics*) e equipe; os alunos participantes Renê Sarrão Moura, Vinicius de Souza Paulus e outros.
- Ao corpo docente e técnico do Departamento de Física da Universidade Estadual de Maringá pela transmissão do conhecimento e da experiência profissional, a qual, lembrarei com carinho no transcurso da minha vida profissional. Em especial, ao meu orientador Prof. Dr. Breno Ferraz de Oliveira, pela dedicação em ensinar os caminhos da pesquisa e na resolução prática, nos contratempos que sugiram durante o desenvolvimento dessa pesquisa.
- A Universidade Estadual de Maringá, que oportuniza a oferta gratuita e de qualidade do ensino, da pesquisa e da extensão.
- A minha família, meu alicerce e minha estrutura de vida.

Rodolfo Ricardo.

RESUMO

Esse trabalho foi desenvolvido por meio de participação de pesquisa realizada em 2019, pela Universidade Estadual de Maringá (UEM), em parceria, com empresa de Maringá do ramo de tecnologia e robótica. O propósito do projeto realizado pelos pesquisadores da UEM foi desenvolver tecnologia para ser aplicada em carros autônomos utilizando-se da confecção de um protótipo Carro Segue Faixa. O objetivo geral desse trabalho teve como recorte delimitador, estudar o conceito da visão computacional para automóveis autônomos, com base o aplicado ao Carro Segue Faixa, no projeto/UEM. A linguagem tecnológica computacional aplicada nesse trabalho é a *Python*, com base na utilização do dispositivo computacional *Raspberry Pi 3*, para as codificações do Carro Segue Faixa. Foi possível compreender algumas das principais dificuldades da visão computacional aplicada nos carros autônomos sendo uma delas a identificação de objetos quando o sistema é feito com cores. Essa dificuldade está relacionada as cores em si. O computador entende leves sombras como cores totalmente diferentes, tornando necessário criar uma maneira de fazer o programa entender, que essas pequenas sutilezas nas corres podem ser ignoradas ou reajustadas para a interpretação do objeto em questão.

Palavra-chave: Visão Computacional. Carro Autônomo. Carro Segue Faixa.

ABSTRACT

This work was developed through participation in research carried out in 2019 by the State University of Maringá (UEM) in partnership with a company from Maringá in the field of technology and robotics. The purpose of the project developed by the researchers at UEM was to develop technology to be applied in autonomous cars, using the production of a prototype of Line Follows Robot. The general objective of this work was to study the concept of computer vision for autonomous cars, based on what was applied to the Line Follows Robot, in the project/UEM. The computational technological language applied in this work was Python, based on the use of the Raspberry Pi 3 computational device, for the coding of the Line Follows Robot. It was possible to understand some of the main difficulties of computer vision applied to autonomous cars, one of them being the identification of objects when the system is made with colors. This difficulty is related to the colors themselves. The computer understands light shadows as totally different colors, making it necessary to create a way to make the program understand that these small subtleties in color can be ignored or readjusted to the interpretation of the object in question.

Keyword: Computer Vision. Autonomous Car. Line Follows Robot.

1 INTRODUÇÃO.

O carro autônomo é um dos avanços tecnológicos mais esperados não só pelos pesquisadores, mas, também, pela população em geral, pois com a sua fabricação há uma grande expectativa pela melhora no trânsito e conforto na locomoção de seus passageiros. Segundo Mataric, “a criação de máquinas inteligentes pode alimentar nossa imaginação, criatividade e desejo de fazer a diferença” (MATARIC, 2014, p. 14). Assim, toda a tecnologia desenvolvida na área robótica desenvolve um papel-chave para o futuro tecnológico.

E, no futuro tecnológico o desenvolvimento da tecnologia voltada ao carro autônomo. Muitas são as pesquisas desenvolvidas, protótipos básicos desse conceito de carro autônomo (nível 5 de automação), com projetos criando miniatura de carro com a capacidade de seguir a faixa e parar caso necessário, chamado de Carro Segue Faixa. Todos os seguimentos dos Carros Segue Faixa são com base em resultados das pesquisas realizadas pela Robótica.

A Robótica, segundo Mataric (2014) “é o estudo dos robôs, o que significa que é o estudo da capacidade de sentir e agir no mundo físico de forma autônoma e intencional” (MATARIC, 2014, p. 21). O termo robótica, de acordo com Mataric (2014), vem da origem tcheco *rabota* (trabalhos obrigatórios) e robotnik (servo), sendo popularizada em 1921, pelo dramaturgo Karel Capek, com o termo robô.

O robô vem com o conceito de ideia da máquina que poderia ajudar as pessoas. E esse conceito vêm muito antes de Karel Capek. Atualmente, o entendimento dessa máquina, a qual, é conhecida como robô, teve a sua ampliação e, hoje é de acordo com Mataric, entendido de forma simples “um sistema autônomo que existe no mundo físico, pode sentir o seu ambiente e pode agir sobre e para alcançar alguns objetivos” (MATARIC, 2014, p. 19).

A área que abrange o conhecimento da robótica, é ampla, não limitando-se ao que se é possível no campo mecânico ou computacional. Conforme o campo das pesquisas e da tecnologia avançam, não se é possível prever o alcance que poderá ser realizada no futuro.

A autonomia do robô é o alcance que a ciência e tecnologia podem chegar. De acordo com a definição de Mataric (2014), um robô contem sistemas autônomos, o qual, possui uma base que o faz tomar as suas próprias decisões e, não seja controlado pelo ser humano. Dessa forma, o autor diz que o robô existe em um

mesmo mundo físico como os animais, as pessoas, natureza e meio ambiental, em todo o tempo e, ele tem desafios reais no sentir o ambiente e interagir nele. E, essa interação, são pelos seus sensores.

Os sensores é um mecanismo do robô para perceber o que está a sua volta e obter informações do mundo, do ambiente, em que está interagindo. E, para Mataric (2014), para alcance dos objetivos, a inteligência artificial (AI) se torna o componente mais importante, conforme a sua aplicação pode ser para fins: pessoais, de serviços operacionais, de campo e industriais. A aplicação do conhecimento da IA no robô propicia ele a tomar decisões próprias e autônomas.

Com o avanço do conhecimento científico e tecnológico, a aplicação da robótica se amplia e, a Robótica Móvel é um campo que se aprofunda no entendimento dos sistemas de condução autônoma. Em, 1969, foi criado um dos primeiros robôs móveis a utilizar a inteligência artificial nos controles dos seus movimentos, utilizando-se de câmeras e sensores táteis binários. Na década de 70, foi desenvolvido o Lunar Rover, projeto da NASA, e no final dessa década, em Stanford, no Laboratório de Inteligência Artificial, o robô CART. Em 1986, o robô Hilare, um robô multissensorial. Com a experiências adquiridas na robótica móvel em 2006, a Universidade de Stanford em colaboração com a Volkswagen, desenvolveu o robô Stanley, criado para competição da DARPA (Stanford Racing Team), tendo como base o veículo Volkswagen Touareg R5 TDI e uma plataforma de computação, com seis processadores, atuadores e sensores. Essa pesquisa realizada em colaboração entre a Instituição de Ensino Superior e a Empresa Automobilística, teve objetivo de aumentar a capacidade de velocidade do veículo em terrenos variados e não pré-determinado. (COUTINHO, 2014, p.11-13)

Os resultados dessa pesquisa em 2006, teve a expansão dos conceitos da robótica móvel aplicada ao segmento do automobilismo e, desse modo, a possibilidade de estudar os principais conceitos que poderiam ser aplicados aos veículos e torná-los um carro autônomo, assim como, os seus componentes.

Também, se expandiu os conceitos no desenvolvimento do Carro Segue Faixa (semi-robô móvel), cujo principal componente do carro, o computador, é usado para processar as informações geradas pelos sensores e comandá-lo. No estudo dos componentes dos carros autônomos, o computador está ligado ao processo das informações e a linguagem computacional da máquina. Quando aplicado ao Carro

Segue Faixa, os diversos componentes, como uma pequena câmera para capitar a imagem do percurso, o motor elétrico e um sensor de ultrassom.

O micro computador que pode ser aplicado ao Carro Segue Faixa, é o *Raspberry Pi* 3. Para a construção do código do programa utilizado no *Raspberry Pi*, aplicado ao Carro Segue Faixa, muitos pesquisadores apontam a utilização da linguagem computacional *Python.*

O *Python* é uma linguagem com várias aplicações. De acordo com Franco (2016), a linguagem de programação *Python* foi criada em 1991 por Guido Van Rossumem, com a finalidade de ser uma linguagem simples e de fácil compreensão. Apesar de simples, para Franco (2016), *Python* é uma linguagem poderosa, que pode ser usada para desenvolver e administrar grandes sistemas.

O computador *Raspberry Pi* 3 e a linguagem de programação Python podem contribuir, também, para o tratamento da imagem, a visão computacional de um carro autônomo e o Carro Segue Faixa. De acordo com Marengoni (2009), a visão computacional é o processamento de uma imagem, conforme um conjunto de dados extraindo as informações da imagem real, no caso do carro autônomo à imagem seria um *frame* do vídeo transmitido pela câmera.

No processo de visão computacional, necessita-se de um pré-processamento, no qual, se coloca filtro para excluir os possíveis ruídos acarretados pela extração da imagem. Assim, nota-se que há complexidade, na criação da visão computacional de um carro autônomo, no elo entre os componentes necessários à interpretação do cenário que está a sua volta.

Nesse sentido, mediante os problemas dessa complexidade que envolve o desenvolvimento da visão computacional aplicado ao carro autônomo, a questão de pesquisa se concentra em entender: qual a importância da visão computacional para os automóveis autônomos.

Como objetivo geral, estudar-se o conceito da visão computacional para automóveis autônomos, aplicado ao Carro Segue Faixa. Para o alcance desse objetivo têm-se os objetivos específicos: a) compreender as Teorias aplicadas a Física, a identificação dos componentes do carro segue faixa e as suas funções; b) entender a linguagem computacional utilizada; c) Analisar a visão computacional aplicada no Carro Segue Faixa; d) Analisar o código do carro.

Esse trabalho tem a sua base de estudo de caso na pesquisa desenvolvida para a criação de um Carro Segue Faixa, realizada por meio de projeto desenvolvido

em parceria entre a empresa do ramo de tecnologia e robótica e a UEM, em 2019. A linguagem tecnológica computacional aplicada no projeto foi a *Python*, com a utilização do dispositivo computacional *Raspberry Pi 3*. Também, para a elaboração desse trabalho foi realizado levantamento da pesquisa qualitativa para a compreensão das Teorias multidisciplinares, incluída a da Física, que envolvem o objeto de estudo, análise das codificações parciais apresentadas no projeto UEM/Sigma e a visão computacional.

2 ALGUNS CONCEITOS DE TEORIAS APLICADAS COM BASE NA FÍSICA.

Em todo os processos práticos aplicados as diversas atividades, os conceitos da Física são estudados por serem considerados um conhecimento multidisciplinar. Assim, entende-se a Física como ciência voltada a estudar os fenômenos naturais, dentro das bases teóricas que foram formadas, por meio, da observação e experimentos realizados com fatores extraídos da natureza. Portanto, a área da Física estuda a natureza, as propriedades da matéria.

Dentro desse universo multidisciplinar, os conceitos aplicados à Física, auxiliam para explicar as barreiras existentes entre o ser humano e a máquina. Como no caso da visão humana e a simulada em computadores exigem estudos de algoritmos, as quais são técnicas baseadas no conceito de visão computacional que têm intuito de desenvolver procedimentos e processos para o computador (máquina) ter a capacidade de fazer leituras, interpretações de vídeos e imagens igual ao olho e cérebro humano.

No entanto, ambos os olhos humanos e as telas das câmeras têm um objeto intrínseco em comum com a imagem, no qual sua capacidade de capitação está delimitada pelas fontes de luz do ambiente. Essa fonte de luz incide sobre os objetos ao redor e, pela capacidade dos objetos de absorver, refratar ou refletir dos materiais, que compõem esses objetos, os faz coloridos. Fisicamente, “a cor é definida como uma sensação produzida por certas organizações nervosas sob a ação da luz, isto é, ainda sem a interpretação humana” (SILVEIRA, 2015, p.44). Dessa forma, cientificamente, ondas de luz “alcançam os olhos através de uma transmissão (da fonte de luz para o objeto, e deste para o observador) ou quando o objeto é a própria fonte de luz, resultando na sensação cromática” (SILVEIRA, 2015, p.44).

O conhecimento da Física se aplica ao estudar a natureza da Luz em três áreas/partes: a ótica geométrica; a ótica física; e a ótica quântica. A ótica geométrica tem a fundamentação na trajetória dos raios da luz; a ótica física a fundamentação se encontra nas radiações eletromagnéticas (a luz como ondas); e na quântica o estudo da luz com base na sua construção formada por partículas. Segundo Silveira, “adota-se hoje, então, o conceito de luz como uma dualidade de propriedades ondulatórias e corpusculares” (SILVEIRA, 2015, p.45). Complementa Silveira que:

> Pensando somente pelo lado da Física, a interação entre a luz e o objeto gera o fenômeno da cor percebida nos corpos. A luz incide sobre os átomos componentes das substâncias, interagindo e gerando a coloração dos objetos. A capacidade de absorver, refratar ou refletir determinados raios luminosos incidentes nos objetos os faz coloridos. Assim, não podemos dizer que as substâncias possuem cor, mas sim somente esta capacidade. (SILVEIRA, 2015, p.45).

A interação entre a luz e o objeto é a absorção da luz, que se define como fenômeno ótico, ou seja, a luz incide sobre a superfície de algum material e parte dessa luz fica retida nela, e a outra parte é refletida ou refratada, assim, gerando a luminescência capitada pelo olho humano.

A parte de estudo da Física que compreende a geometria ótica considera a luz como formação de raios de luz, que correspondem a linhas orientadas, as quais, representam a direção e o sentido de propagação da luz.

2.1 ÓTICA GEOMÉTRICA

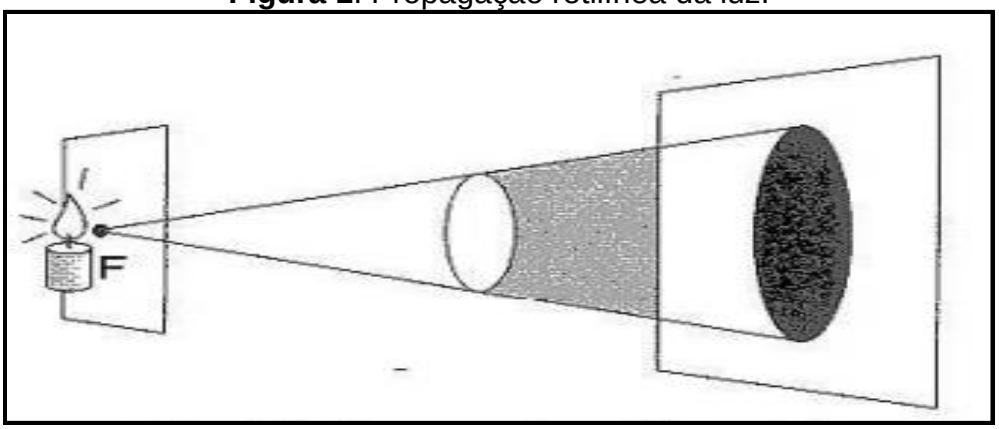
A ótica geométrica é compatível com duas teorias que se diferenciam entre elas, todavia, a mais utilizada pela sua eficácia e capacidade efetiva de resultados na Física Moderna, prevalece a Teoria Ondulatória da Luz. Segundo Nussenzveig, “o triunfo da teoria ondulatória sobreveio no início do século passado, com os trabalhos de Thomas Young e Augustin Fresnel sobre os efeitos de interferência e difração” (NUSSENZVEIG, 1998, p.1). Complementa o autor que “levando a conclusão que as ondas de luz são transversais e não longitudinais como as do som” (NUSSENZVEIG, 1998, p.1).

Dessa forma, a ótica geométrica é compreendida como a parte da ótica responsável a propagação da luz por meio dos raios e partículas de luz, com os

fenômenos: propagação retilínea da luz; reflexão; refração; e o comportamento de espelhos e lentes.

Na propagação retilíneas os raios de luz têm como princípios, sempre se propagar em linha reta e não há interferência entre os mesmos, isto é, eles são independentes. A luz é reversível e sempre será capaz de realizar o caminho na direção oposta. Na figura 1, têm-se a demonstração da propagação retilínea da luz.

Figura 1: Propagação retilínea da luz.



Fonte: NUSSENZVEIG, 1998, p.3.

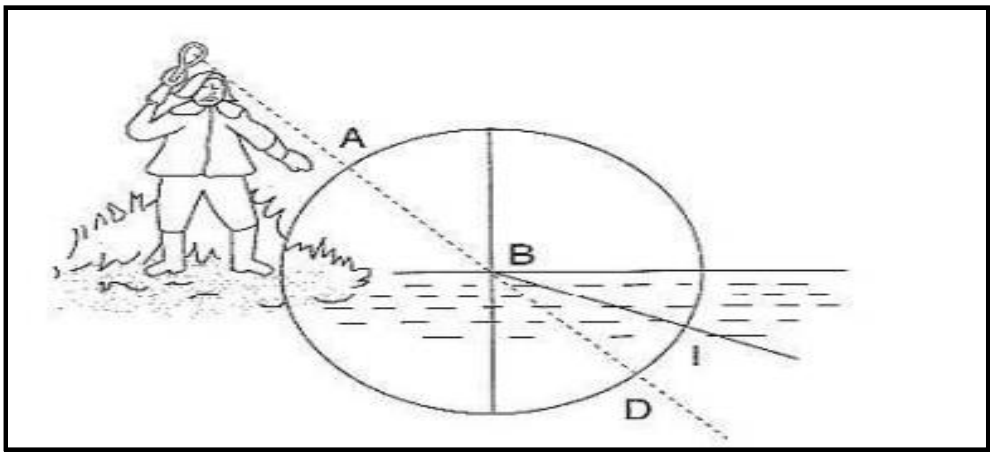
A fonte [F] gera, de acordo com a figura 1, parte do raio de luz retido pelo obstáculo gerando a sombra, com contorno nítido e definido pelo conceito de propagação retilínea da Luz.

Na incidência do raio de luz sobre uma superfície, retorna-se para o meio sofrendo uma mudança na sua direção e sentido, que se dá a reflexão. Segundo Nussenzveig:

> É fácil explicar a reflexão pela Teoria Corpuscular: se uma bola de tênis sobre uma reflexão elástica no solo, a componente vertical e sua velocidade se inverte, sem que a componente horizontal se altere, o que implica na igualdade entre o ângulo de reflexão e incidência". (NUSSENZVEIG, 1998, p. 8).

A refração é quando o raio de luz atravessa uma superfície opaca e ocorre uma mudança de direção nesse feixe, devido à mudança (do meio), de acordo com a figura 2:

Figura 2: Demonstração corpuscular da refração



Fonte: NUSSENZVEIG, 1998, p. 8.

Pela figura 2, é fácil explicar a refração pela Teoria Corpuscular, segundo Nussenzveig (1998). Se uma bola entrar do ar para a água (trajetória de A para B), desvia-se da sua trajetória original, por perder parte do componente vertical da sua velocidade, deslocando-se em menor velocidade (D).

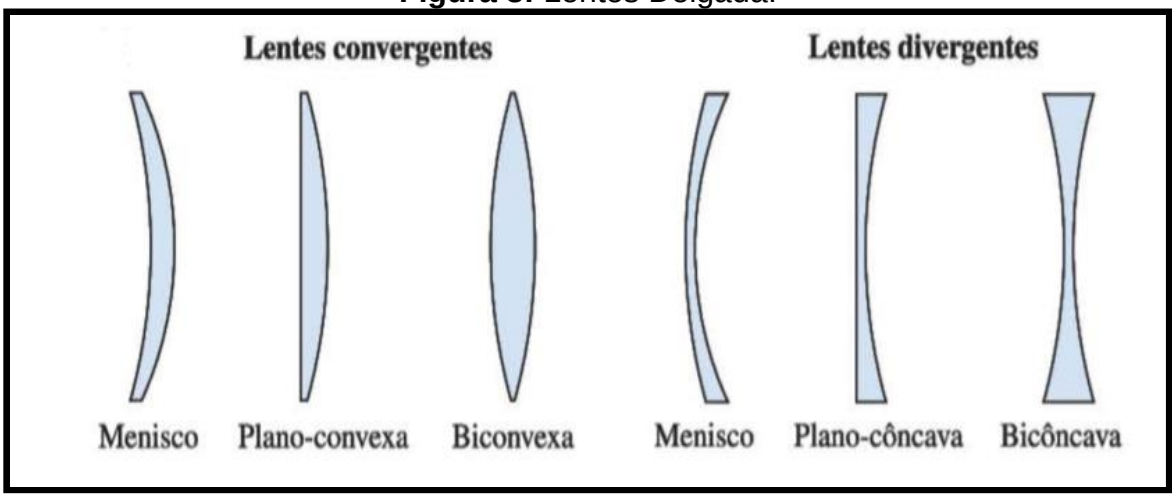
O comportamento de espelho e lente se deve ao formato da superfície do espelho e da lente. Os espelhos planos e curvos e lentes são secções de uma esfera cortada, o qual gera várias lentes distintas uma da outra. Para aplicação desse conceito da ótica geométrica ao Carro Segue Faixa é importante o entendimento básico do funcionamento de lentes delgadas.

Lentes delgadas são objetos que refrata os raios luminosos duas vezes, uma ao entrar na lente e outro ao sair, segundo Nussenzveig (1998) “tem duas superfícies refratoras, de raios de curvatura, respectivamente na ordem em que são encontradas pela luz incidente, e com sinais definidos em uma superfície de eixo e negativas” (Nussenzveig, 1998, p. 27). Para Sears (et al., 2016):

> A distância focal de uma lente divergente é uma grandeza negativa, e a lente também é chamada de lente negativa. Os focos de uma lente negativa estão em posições invertidas em relação aos focos de uma lente convergente. O segundo foco, de uma lente divergente, é o ponto a partir do qual os raios que estavam originalmente paralelos ao eixo parecem divergir depois da refração. Os raios incidentes que convergem para o primeiro foco, emergem da lente formando um feixe paralelo a seu eixo. [...] uma lente divergente apresenta a mesma relação com uma lente convergente que um espelho convexo tem com um espelho côncavo. (SEARS, et al., 2016, p. 59).

Uma lente mais grossa nas bordas e finas no centro é uma lente divergente; e uma lenta grossa no centro e fina nas bordas é uma lente convergente, segundo os autores Sears (et al. 2016). Na figura 03, a demonstração da lente delgada:

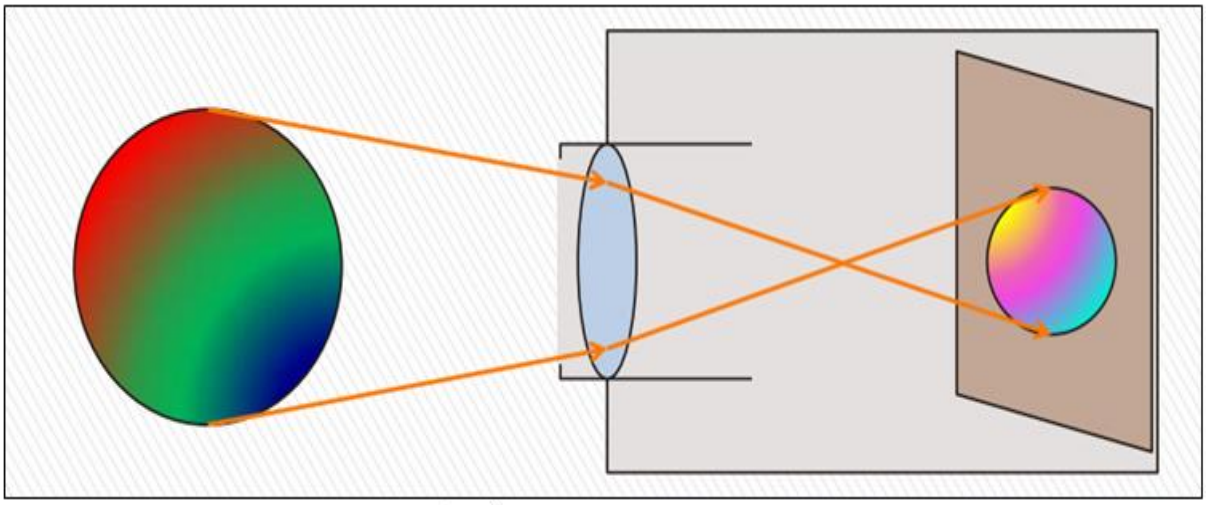
Figura 3: Lentes Delgada.



Fonte: Sears & Zemansky; Young & Freedman, 2016, p. 59.

A lente convergente é uma lente que quando o raio luminoso passa por ela esse raio é convergido ao foco da lente, segundo Nussenzveig, “uma das principais aplicações do espelho esférico é aumentar ou diminuir o tamanho da imagem” (Nussenzveig, 1998, p. 21). Na figura 4, demonstra-se a lente convergente:

Figura 4: Espelho Convexo.



Fonte: Vieira, 2013, p. 31.

As lentes de bordas finas apresentam comportamento convergente, no geral o comportamento óptico de uma lente é condizente com a forma, o material empregado na sua criação e o meio onde ela está inserida.

2.2 ONDAS SONORAS

O som se propaga por meio da matéria, o sólido, o liquido e o gasoso. Segundo Nussenzveig (2002), “o fato de que o som se propaga através de um meio material, sem que haja transporte de matéria de um ponto a outro, já é uma indicação de sua natureza ondulatória” (NUSSENZVEIG, 2002, p. 122).

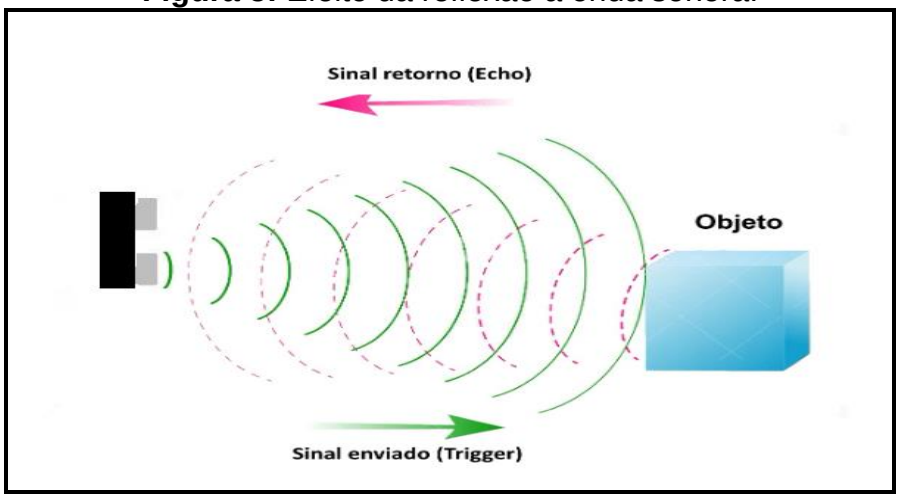
Entende-se que o som é uma onda que tem como necessidade da matéria para a sua existência e propagação, isto é, não há propagação no vácuo, no qual não há matéria. Assim, de acordo com Nussenzveig (2002), o som tem certas propriedades, tais como: a velocidade, reflexão, interferência e refração. A “velocidade finita de propagação do som é evidenciada pelo intervalo de tempo decorrido entre o clarão de um relampado e o ruido do trovão que o acompanha. A reflexão do som também é um efeito familiar, manifestado na produção de ecos” (NUSSENZVEIG, 2002, p. 122).

A Reflexão, ocorre quando a onda atinge uma superfície e retorna ao mesmo meio de origem; mantém a velocidade, o comprimento de onda e a sua frequência, iguais a onda de origem (incidente). Para Nussenzveig (2002),

> [...] quando uma onda unidimensional se encontra uma superfície de descontinuidade separando dois meios diferentes, ela parcialmente refletida e parcialmente transmitida. O mesmo acontece para as ondas de duas ou três dimensões. Nesse caso, as direções de propagação das ondas refletidas e transmitidas dependem da direção da onda incidente. (NUSSENZVEIG, 2002, p. 143).

Dessa forma a Refração, acontece quando a onda muda o seu meio de propagação, como exemplo o ouvir uma música em baixo d’agua, considerando que a fonte de origem do som está fora d’água. Em síntese, a Reflexão o Som bate em uma superfície e é refletida de volta de acordo com a figura 5 e a Refração é quando a Onda Sonora adentra um novo meio.

Figura 5: Efeito da reflexão a onda sonora.



Fonte: com base na imagem disponível em https://www.filipeflop.com/blog/sensor-ultrassonico-hc-sr04-ao-arduino/, acesso 04/2022

Levando os conceitos abordados de Som, observa-se uma limitação do ouvido humano. Assim, de acordo com Nussenzveig (2002), a “oscilações harmônicas podem produzir sons audíveis pelo ouvido do humano somente num intervalo limitado de frequência, aproximadamente entre 20 Hz e 20 KHz” (NUSSENZVEIG, 2002, p. 122).

3 A VISÃO COMPUTACIONAL.

Para o ser humano é simples a identificação de estruturas em três dimensões. A percepção das três dimensões, de acordo com Szelisk (2010), é tão vívida que basta olhar a um vaso de plantas, é possível notar a forma e a transluscência de cada pétala, pelos sutis padrões de luz e sombra que estão sobre a superfície de cada flor. Na figura 6, demonstra-se uma flor e as suas dimensões vistas a olho nu.

Figura 6: Dimensões e efeitos de sombra e luz.

Fonte: Autor/2022.

A visão computacional criada no início dos anos 70, segundo Szelisk (2010), era vista como um componente de um conceito ambicioso de imitar a inteligência humana e colocá-la em robôs inteligentes. Na época, se acreditava pelos primeiros pioneiros da inteligência artificial e robótica que resolver o problema de identificação visual, seria um passo fácil, na longa jornada de resolver problemas mais complexos, tais como: raciocínio e planejamento.

Atualmente, se percebe que o problema da visão não é algo tão simplório. Há distinção entre a visão computacional e outras formas de processamentos de imagens já existente. E, essa distinção está, no fato da visão computacional, buscar a formação de estruturas em três dimensões nas imagens e usar isso como um passo para a compreensão total da cena. E, assim, criam-se modelos avançados de visão computacional.

No entanto, segundo Sonka (et al., 2015), a visão computacional em sua forma mais crua se entrelaça com o processamento de imagem digital, nos quais, em ambos os casos seguem os seguintes passos: a) a imagem for capturada por um sensor, como uma câmera e digitalizada; b) o computador pré processa a imagem removendo algum ruido ou realçando algo do objeto ou característica que são relevantes para o entendimento da imagem; c) ocorre a segmentação da imagem, o computador tenta separar os objetos do fundo da imagem de si, ocorrendo a classificação e descrição dos objetos obtidos pelas imagens segmentadas.

Complementa Sonka (et al., 2015), que a diferença entre processamento de imagem e visão computacional está na forma de utilização dos dados obtidos perante uma imagem. Enquanto o processamento visa a imagem original representada por matrizes de luz e valores similares, a visão computacional origina na seleção dos dados relevantes ao objetivo da visão, sendo eles importantes para o conteúdo dos objetos na imagem como por exemplo: o tamanho, a forma e a relação entre os objetos na imagem.

A ciência da Física é a base para o desenvolvimento desses modelos. De acordo com Szelisk (2010), os modelos mais avançados usados na visão computacional são usualmente desenvolvidos utilizando conceitos da Física como a radiometria, a ótica, e o design de sensores, junto com a computação gráfica.

A visão computacional, não alcançou o seu objetivo final de construir um mundo 3D perante o uso de imagens. Segundo Sonka (et al., 2015), uma das maiores complicações, por não alcançar o objetivo, se deve a inabilidade de

automatizar uma forma, no qual, os próprios programas identifiquem os objetos em geral em uma imagem. A inabilidade de automatizar uma forma, ainda, exige a ação do ser humano em dizer quais os objetos que são relevantes.

Porém, de acordo com Szelisk (2010), a visão computacional está em estado avançado, no qual, é possível ver suas aplicações no mundo real, como: a) o reconhecimento de caracteres: ler os códigos postais escritos a mão em correspondência e a identificação automáticas de placas de automóveis; b) as lojas de varejo – venda do produto: identifica objetos para uma automação de linhas de saída do produto em estoque; c) os modelos em 3D: vistas panorâmicas de construções usados em sistemas como Google Maps; d) a segurança automobilística: a detecção de pedestre na rua; e) em filmes: usado imagens geradas por computador [CGI] com as imagens gravadas durante as cenas incluindo modelos em 3D, tanto, quanto, cenários de fundo, como, personagens totalmente computacionais.

4 OS CONCEITOS GERAIS DE CARRO AUTÔNOMO.

Para conceituar carro autônomo deve-se entender o termo atribuído a autonomia. De acordo com Ozguner (et al.,2007), é definido autonomia de carro como esse carro pudesse fazer decisões próprias sem a intervenção do Ser Humano. Para o autor, a autonomia existe em vários aspectos do carro atual, como por exemplo o ‘modo cruzeiro’ e os sistemas de prevenção de travamento das rodas (ABS – *Antilock Brake Systems*). Também, descreve que há existência de vários suportes autônomos nos sistemas que compõem os carros atuais.

Observa-se que o pensamento do carro autônomo no decorrer dos anos foram transformando-se, para alcançar o objetivo principal que é dar autonomia total aos veículos. À medida que as pesquisas foram avançando alguns sistemas foram sendo criados no meio automobilístico. De acordo com Weber, “na década de 1960, entusiastas da inteligência artificial (IA) em computadores começaram a sonhar com carros inteligentes o suficiente para navegar por ruas comuns por conta própria’ (WEBER, 2014, s/p).

Para Weber (2014), era assustador o desafio da época, o qual comparava-se os processos de IA a engenharia reversa do movimento de um animal, dado a necessidade de construir mecanismos para a IA operar os sistemas, na qual,

houvesse a forma de detectar, o de processar e o de reagir, automaticamente, ou seja, “1) Sensoriamento 2) Processamento (modelagem do mundo exterior, tomada de decisões) 3) Reação, com movimento apropriado” (WEBER, 2014,s/p). O autor esclarece que existia a parte conhecida e partes desconhecidas dos processos, na qual, a desconhecida se dava pela falta de inteligência da máquina necessária para o processamento. De acordo com Anderson (et al., 2016):

> Em geral, os sistemas robóticos, incluindo AVs, usam um design “*sense-plan-act*”. Para detectar o ambiente, os AVs usam uma combinação de sensores, incluindo lidar (detecção e alcance de luz), radar, câmeras, ultrassônico e infravermelho. Um conjunto de sensores combinados pode complementar um ao outro e compensar quaisquer deficiências em qualquer tipo de sensor. Embora os sistemas robóticos sejam muito bons em coletar dados sobre o ambiente, dar sentido a esses dados continua sendo provavelmente a parte mais difícil do desenvolvimento de um sistema ultraconfiável. (ANDERSON, et al., 2016, p. 19).

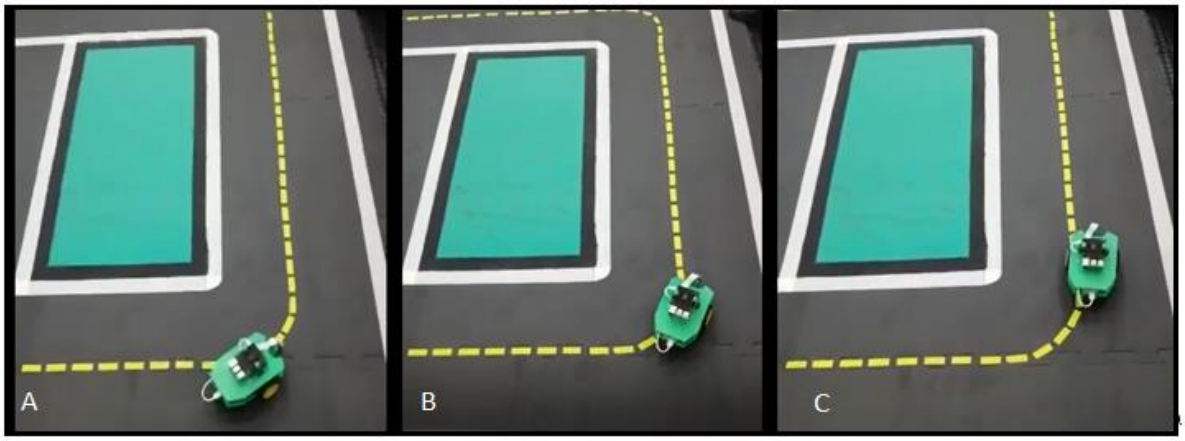
Os autores Anderson (et al 2016), complementam que mesmo com a tecnologia atual, o carro autônomo, ainda, não pode ser considerável um projeto totalmente seguro, dado a possibilidade de ocorrer falha no sistema e o carro não ter a capacidade de identificar a falha e se deslocar de maneira segura. Os carros autônomos devem conter dispositivos de funcionalidade de tomada de decisão própria, capaz de navegar em determinados ambientes com certo nível de autonomia que não venha a colocar a vida do Ser Humano em risco.

4.1 INTRODUÇÃO AOS CONCEITOS DO CARRO SEGUE FAIXA.

Um carro segue faixa em conceito mais simples é um robô sobre rodas. De acordo com Pscheidt (2007), a definição de robô “é máquina controlada por computador e são programadas para mover, manipular objetos e realizar diversos trabalhos” (PSCHEIDT, 2007, p.4). No qual, segundo Nunes (et al.), cada vez mais a “robótica aliada às técnicas de inteligência artificial e várias formas de controle, têm evoluído e se tornado uma potente geradora de oportunidades para o avanço da ciência” (NUNES, et al., 2014, p. 1). Complementa os autores que “atualmente, os robôs já não são mais fixos no seu ambiente e tampouco exclusivos na indústria” (NUNES, et al., 2014, p. 1).

O Carro Segue Faixa tem a finalidade de seguir uma faixa ou feixe desenhado no chão. A complexibilidade do carro se dá na intensidade da captação de informações de uma câmera acoplada no carro e o programa desenvolvido para estudar as informações capitadas. E, desta maneira, mais simples, detectar um feixe infravermelho, um certo tipo de cor ou, até mesmo, detectar formas geométricas no seu modo mais avançado.

Figura 7: Movimento do Carro Segue Faixa em uma curva.



Fonte: Autor, com base no projeto desenvolvido pelo projeto UEM/2019.

Na visualização da imagem na figura 7, é possível notar o movimento do Carro Segue Faixa. Para entender como esse movimento é realizado, a compreensão dos conceitos de robótica se faz necessário. Nos conceitos de robótica, entende-se que é “a ciência dos que interagem com o mundo real com poucas ou mesmo nenhuma intervenção humana” (MARTINS, 2007, p.3).

A robótica é uma ciência multidisciplinar, pela sua própria aplicação, a qual transpassa da formação simplesmente de um robô para aplicação em equipamentos indústrias, máquinas agrícolas e carros, especialmente o de segue faixa. De acordo com Martins, “para engendrar os vários dispositivos robóticos e seus movimentos, a robótica exige dosagens de conhecimentos da microeletrônica, da engenharia mecânica e da Física”, (MARTINS, 2007, p.3).

No entanto, a funcionalidade desse carro está em estudar, de maneira básica, os conceitos introdutórios de robótica, visão computacional e lógica. Na figura 8, na demonstração do protótipo do Carro Segue Faixa desenvolvido por meio de projeto da Universidade Estadual de Maringá, em 2019.

Figura 8: Protótipo do Carro Segue Faixa – Projeto UEM

Fonte: Autor, com base no projeto desenvolvido pelo projeto UEM/2019.

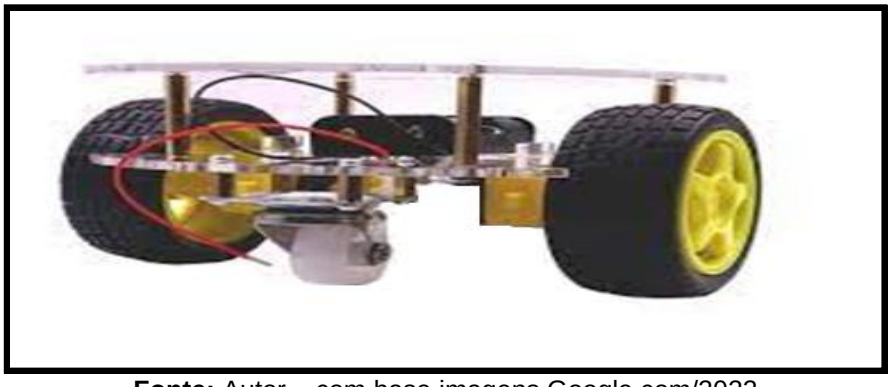
O Carro Segue Faixa, segue de forma mecânica o percurso estabelecido de forma autônoma. Ao estudar os movimentos, códigos e processos do Carro Segue Faixa, além de alguns incrementos que podem ser aperfeiçoados, é possível simular um carro autônomo e os seus principais conceitos, e dificuldades, encontrados pelos pesquisadores hoje em dia. Cada componente do Carro Segue Faixa é formado para complementar e completar o conjunto de peças e tecnologias necessárias para dar autonomia para ele.

4.1.1 Dos Componentes do Carro Segue Faixa.

Na composição de um carro são vários os seus componentes, tais como o chassi, computador, câmera, sensor, motor elétrico, entre outros.

A montagem do carro começa com a montagem do chassi. O chassi utilizado no carro em estudo foi criado exclusivamente para esse projeto, no entanto um chassi 2WD de Arduíno pode ser utilizado como substituto. Na figura 9, a demonstração de um modelo de chassi 2WD:

Figura 9: Modelo de chassi 2WD



Fonte: Autor – com base imagens Google.com/2022

Após a montagem do chassi é fixado o *Raspberry Pi* na parte superior do mesmo, a partir desse ponto todos os componentes são fixados no chassi e conectados ao computador nas suas respectivas entradas.

O *Raspberry Pi* de acordo com Oliveira (et al, 2019) “é um dispositivo computacional com tamanho reduzido, reunindo recursos antes encontrados somente em computadores tradicionais” (OLIVEIRA, et al, 2019, p.1). A ideia inicial desse computador surge em 2006, mas somente em 2008 com a instituição da fundação *Raspberry Pi*, ela foi desenvolvida, sendo em 2011 o surgimento do primeiro modelo distribuído ao público. Segundo Oliviera (et al):

> O *Raspberry Pi* é um dispositivo computacional do tamanho aproximado de um cartão de crédito, formado por uma única placa que reúne diversos recursos que eram encontrados somente em dispositivos maiores como os notebooks e desktops tradicionais. A Fundação *Raspberry Pi* é a responsável pela criação e desenvolvimento do computador de mesmo nome. (FUNDAÇÃO RASPBERRY PI, 2018 apud OLIVEIRA, et al, 2019, p.1).

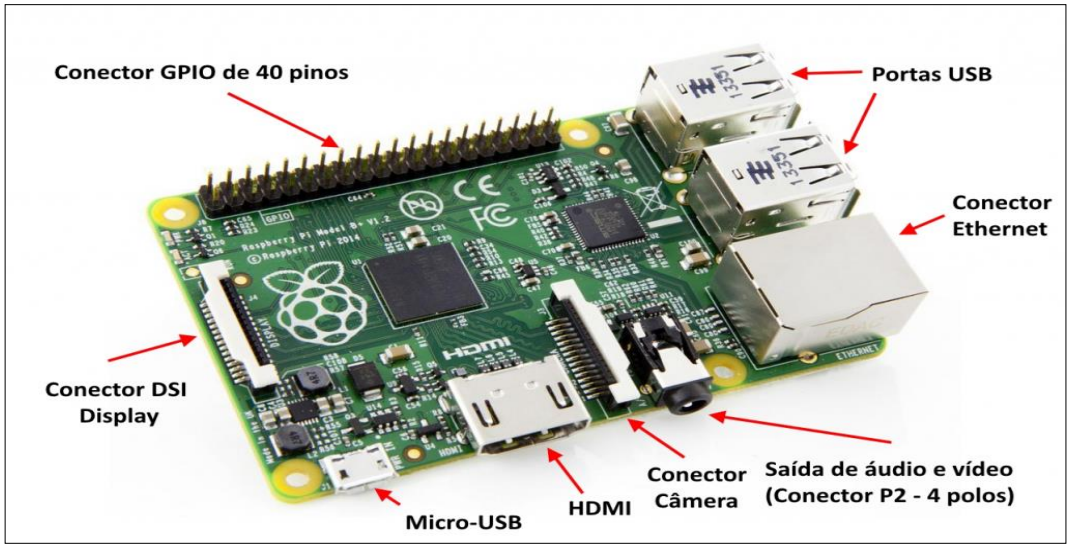
A princípio o dispositivo computacional foi desenvolvido para uso educacional, de baixo valor. No entanto, após o seu lançamento observou-se que o seu uso estava sendo utilizado, além do educacional, pelo seu baixo valor de aquisição e de fácil acesso de pessoas não especializadas em informática, como uma ferramenta corporativa e de uso pessoal. De acordo com Oliveria (et al., 2019):

> O valor de aquisição de um Raspberry Pi é muito baixo, quando comparado a outros computadores. Este trabalho visando comprovar sua viabilidade de utilização para substituição de computadores tradicionais básicos, efetuou testes usando os sistemas operacionais Raspbian, Lubuntu, Q4OS e Fedora. Os resultados demonstraram que o dispositivo não apresenta problemas graves, e que é possível afirmar que a substituição de um computador tradicional por um Raspberry Pi, além de viável tecnicamente, pode representar grande economia, em tempos de escassez de recursos. (OLIVEIRA, at al, 2019, p.1).

A utilização dessa ferramenta, em termos comparativos com outros da época, poderia trazer redução de custos corporativos nas aquisições dos *hardwares* e dos *softwares*, o que poderia substituir em muitos casos o computador de mesa comum. Conforme Oliveira (et al), “(…) com suas medidas diminutas, o dispositivo poderia ser acoplado aos próprios monitores fazendo com que houvesse um ganho de

espaço físico considerável, (OLIVEIRA, at all,2019, p. 2). Na figura 10 o modelo do dispositivo computacional Raspberry Pi 3 - modelo B+.

Figura 10: Modelo de dispositivo computacional *Raspberry Pi* 3 - modelo B+.



Fonte: ARDUINO e Cia, 2014.

São várias as versões do *Raspberry Pi*, o estudo de caso analisado por essa pesquisa foi desenvolvido no modelo da placa *Raspberry Pi 3* – modelo B+. De acordo com a especificação do produto o *Raspberry Pi* 3 – modelo B+ a placa (figura 10) se constitui de uma única placa de silício, com todos os componentes acoplados: CPU de quatro núcleos 1.2GHz Broadcom BCM2837 64bit, com 1 GB de RAM; dispositivo de WiFi e Bluetooth, placa GPIO de 40 pinos, 4 portas USB 2, saída estéreo de 4 polos e porta de vídeo composto, HDMI normal, porta de câmera CSI, porta de micro SD.

Em seguida são conectados os motores elétricos a placa GPIO de 40 pinos do *Raspberry Pi*. Essa conexão é feita utilizando jumpers macho-fêmea ou jumpers macho-macho, de acordo com as especificações dos motores acoplados no chassi.

No modelo Carro Segue Faixa do projeto observado por essa pesquisa, como o chassi usado continha uma bateria com entrada USB, a bateria foi conectada ao computador por meio de um cabo adaptador USB para micro USB. No entanto, se for utilizado um chassi de Arduíno, geralmente acompanhado por suporte de uma bateria movida a 4 pilhas AA, se torna necessário um *powerpack* no qual alimentará o computador, por meio, de um cabo adaptador USB para micro USB e as pilhas irão exclusivamente alimentar os motores

Outro componente importante do Carro Segue Faixa é a Câmera (figura 11) cujo o objetivo é de captar informações por meio de vídeo. Essa informação quando captada é transmitida para o computador, em que a câmera está acoplado e cabe ao computador fazer a análise e uso dessa informação.

Figura 11: Modelo de Câmera *Raspberry Pi* - Rev 1.3.

Fonte: https://www.robocore.net/acessorios-raspberry-pi/camera-para-raspberry-pi-rev-1-3

A câmera utilizada no Carro Segue Faixa, do projeto estudado é a *Raspberry Pi* Rev 1.3. Ela possui lente de foco fixo, capaz de fornecer resolução de 2592 x 1944 pixeis para imagens estáticas e 1080p30, 720p60 e 640x480p60/90 para vídeos.

Por ser uma câmera moderna, é composta por várias lentes projetando uma imagem dentro dela e essa imagem é capitada por um *chip* e pelo efeito foto elétrico transformada em sinais eletrônicos são enviados ao computador.

A câmera é conecta através da porta CSI e fixada na parte frontal do chassi, deixando um pouco inclinada para frente para a câmera poder capitar melhor a parte frontal próxima ao carro. Se a câmera estiver colocada verticalmente pode ocorre de somente capitar uma visão periférica longe de onde o carro irá dar o próximo passo. De preferência, fixar a câmera em um suporte móvel para poder ser realizados pequenos ajustes no ângulo vertical, se necessário.

O Sensor é um dispositivo que responde a estímulos, produzindo um sinal que pode ser transformado em outra grandeza, afins de medição e monitoramento, esses estímulos são físicos ou químicos. Na figura 12, a demonstração do sensor de distância ultrassônico:

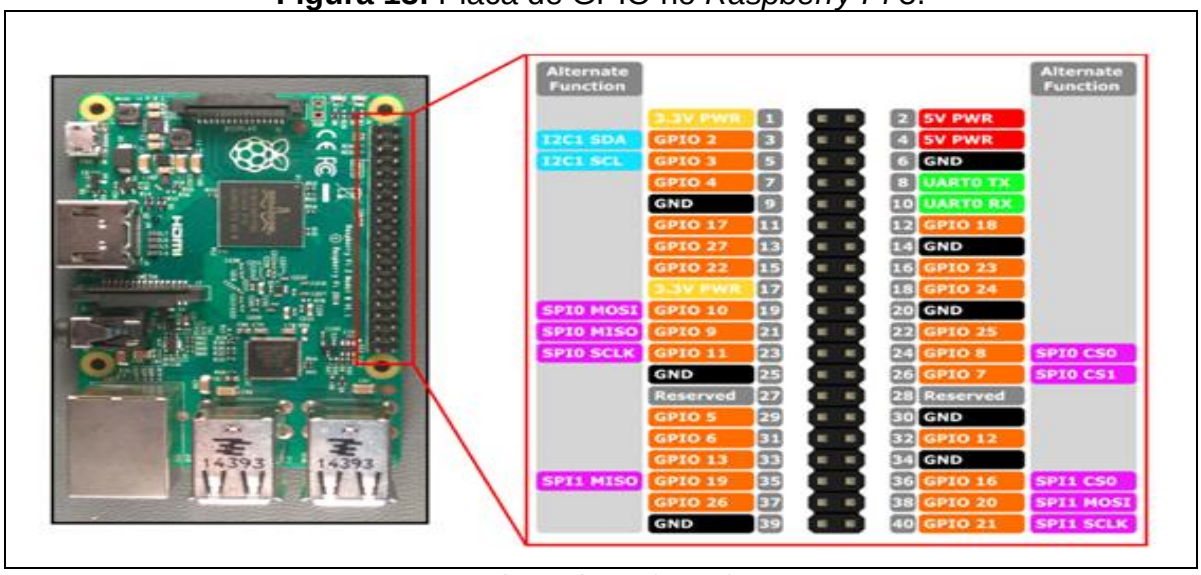
Figura 12: Modelo de Sensor de Distancia Ultrassônico

Fonte: Modulo Sensor de Distancia Ultrassônico, Chipsce, Hc-Sr04

No caso do sensor ultrassônico, ele gera um sinal sonoro na faixa do ultrassom; a onda sonora gerada pelo sensor se reflete na superfície mais próxima ao mesmo e, pelo tempo decorrido do disparo da onda pelo sensor até a onda refletida voltar ao sensor, o mesmo calcula a distância do objeto que foi detectado pelo sensor.

O sensor utilizado para simular o mecanismo de prevenção de colisões, no projeto estudado, foi o Modulo Sensor de Distancia Ultrassônico, Chipsce, Hc-Sr04, cujo, o qual, é um sensor sonoro. Ele é fixado na parte frontal do carro de preferência em um local que não seja capitado pela câmera e a sua parte frontal não esteja obstruída, a fim de evitar a ocorrência de interferências com o chassi do carro e, é conectado ao computador pela placa GPIO.

Figura 13: Placa de GPIO no *Raspberry Pi 3*.

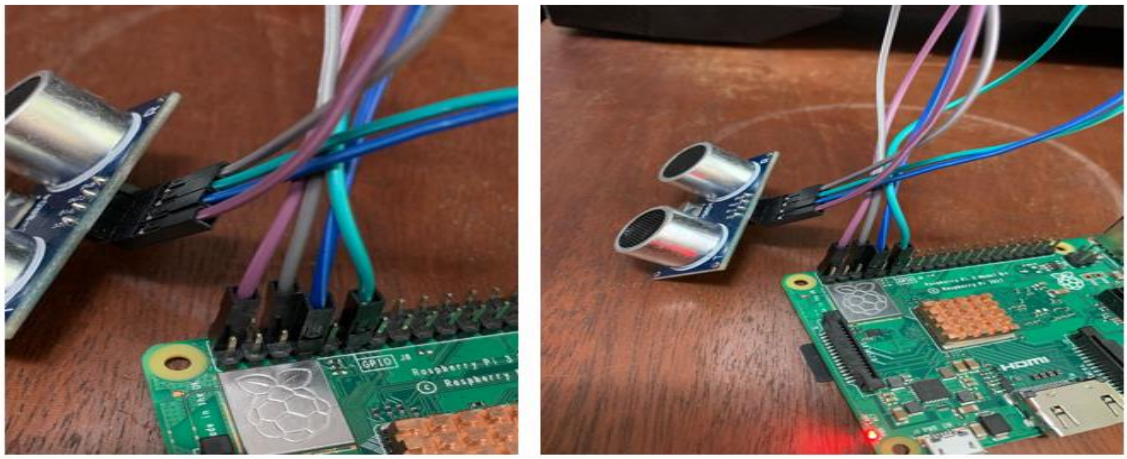


Fonte: docs.microsoft.com/ 2022

Na placa GPIO, representada na figura 13, nota-se que ela está dívida em duas colunas com 20 pinos cada coluna. A coluna da esquerda, que está para dentro do computador, é composta pelos pinos de numeração ímpar; a coluna da direita é a coluna dos pinos com numeração par.

Para ligar os componentes no GPIO, é usado os cabos *jumpers. A* ligação é bem simples, deve-se localizar no programa o número dos pinos utilizados e conectar, esses pinos, aos seus respectivos componentes. Verificar se o componente precisa estar em algum dos pinos terra (GND) ou ao pino de tenção (PWR). Na figura 14, sequência de fotos do sensor ligado aos pinos:

Figura 14: Foto do sensor ligado aos pinos do *Raspberry Pi 3*.



Fonte: Autor/2022.

Na figura 14, está representado como dever ser ligado os pinos do sensor, nos quais, os pinos são 7 e 11 (conforme figura 13) utilizados pelo programa do sensor e os pinos 2 e 6 (conforme figura 13), são respectivamente o de tensão e de terra, necessários para o funcionamento do sensor. No anexo 3, demonstra-se o circuito do Carro Segue Faixa.

5 A LINGUAGEM PROGRAMAÇÃO PYTHON.

A linguagem de Programação *Python* é uma linguagem computacional de fácil compreensão e segundo DONAT (2018), indica processos ou segmentos de fácil operação, ou seja, “uma linguagem de *script*” (DONAT, 2018, p.23). No entanto,

mesmo sendo uma linguagem simples, contem abrangência que de acordo com Menezes:

> Python é software livre, ou seja, pode ser utilizada gratuitamente […], pode utilizar Python em praticamente qualquer arquitetura de computadores ou sistema operacional, como Linux2, FreeBSD3, Microsoft Windows ou Mac OS X4. (MENEZES,2019, p.26).

Por ser um software livre, é um produto de fácil acesso e está disponível grande quantidade de materiais bibliográficos que fala a respeito, tanto em conceitos gerais como em técnicos. Também, programas já prontos ou segmentos de códigos já funcionais para diversas áreas, dessa forma o *python* é uma ferramenta que pode ser utilizada em uma área abrangente e com uma facilidade de aprendizado. Para DONAT, 2018, "[…] Alguns podem questionar se é uma linguagem de programação ou uma linguagem de script". (DONAT,2018, p.23)

A linguagem de script é segundo Donat (2018), uma linguagem lida e colocada em prática, cada vez que são executadas, diferente da linguagem de programação. A linguagem de programação segundo Donat (2018),

> [...] são compilados, ao contrário de linguagens de script. Linguagens comuns como C, C ++, Java e deve ser compilado por um compilador. O processo de compilação resulta em um arquivo de código de máquina, ilegível por seres humanos, que o computador possa ler e seguir. Quando você escreve um programa de C e compilar, o arquivo. O resultante é o que é lido pelo computador. Um dos efeitos colaterais / resultados deste é que as linguagens de programação *podem* produzir mais rápidos programas de ambos porque a compilação só acontece uma vez e muitas vezes porque o compilador otimiza o código durante o processo de compilação, tornando-se mais rápido do que seria como originalmente escrito. (DONAT, 2018, p. 31).

Quando se utiliza da linguagem de programação o computador faz a compilação e cria um arquivo que apenas o computador faz a leitura e com maior rapidez da ação exigida dele. A linguagem de script, "por outro lado, é lida, interpretadas e postas em prática cada vez que se executá-los. Eles não produzem um arquivo compilado, e as instruções são seguidas exatamente como escrito" (DONAT, 2018, p. 31), dessa forma segundo o autor, "[…] se você escrever código desleixado, você obtém resultados desleixado. Por esta razão, as linguagens de script podem resultar em programas mais lentos" (DONAT, 2018, p. 31). Ou seja, na linguagem de programação, ao construir um programa e ao copilar o computador

cria o chamado código de máquina, uma linguagem que somente compressível ao computador e muitas vezes otimizada por ele.

Na linguagem de script isso não ocorre a construção do código de máquina. Nessa linguagem ela vai seguir exatamente o que está escrito, não se importando com a otimização do processo. Mas segundo DONAT (2018), com o desenvolvimento computacional atual é raro o caso que a linguagem de programação é mais rápida do que a linguagem script.

O *Python*, por ser uma linguagem computacional de fácil entendimento, o torna mais vantajoso. Para Menezes:

> O Python é uma linguagem completa, contando com bibliotecas para acessar bancos de dados, processar arquivos XML, construir interfaces gráficas e mesmo jogos, podemos utilizar muitas funções já existentes escrevendo poucas linhas de código. Isso aumenta a produtividade do programador, pois, ao utilizarmos bibliotecas, usamos programas desenvolvidos e testados por outras pessoas. Isso reduz o número de erros e permite que você se concentre realmente no problema que quer resolver. (MENEZES,2019, p.27).

Por isso o *Python* tem vantagem em relação a outras linguagens de computação. Uma grande vantagem de *Python* “é a legibilidade dos programas escritos nessa linguagem. Outras linguagens de programação utilizam inúmeras marcações, como ponto (.) ou ponto e vírgula (;)” (MENZES,2019, p.26), e, também, no fim de cada linha, “além dos marcadores de início e fim de bloco como chaves ({ }) ou palavras especiais (*begin/end*). Esses marcadores tornam os programas um tanto mais difícil de ler e felizmente não são usados em *Python*.” (MENZES,2019, p.26).

A estruturação de um código *Python,* mesmo que simples, ainda, deve-se seguir um certo número de regras de digitação que permite a leitura do código na plataforma de desenvolvimento computacional. Entre as mais simples estão a diferença entre letras maiúsculas e minúsculas, sempre que abrir aspas ou parênteses eles devem ser fechados e os espaços no *Python* são muito importantes, entre outros aspectos de digitação.

Outra parte importante da estruturação de um código em *Python* são as próprias estruturas que o código deve ter, ou seja, regras simples que podem gerar os mais complexos programas, apresentadas no quadro 1.

Quadro 1: Estrutura de codificação *PYTHON*.

VARIÁVEIS	São pequenos espaços de memória, utilizados para armazenar e manipular dados. Em Python, os tipos de dados básicos são: tipo inteiro (armazena números inteiros), tipo float (armazena números em formato decimal), e tipo string (armazena um conjunto de caracteres). Cada variável pode armazenar apenas um tipo de dado a cada instante.
STRINGS	É uma sequência de caracteres simples. Na linguagem Python, as strings são utilizadas com aspas simples ('... ') ou aspas duplas ("...").
NÚMEROS	Os quatro tipos numéricos simples, utilizados em Python, são números inteiros (int), números longos (long), números decimais (float) e números complexos (complex).
LISTAS	É um conjunto sequencial de valores, onde cada valor é identificado através de um índice. O primeiro valor tem índice 0.
TUPLAS	É um conjunto sequencial de valores, onde cada valor é identificado através de um índice. A principal diferença entre elas é que as tuplas são imutáveis, ou seja, seus elementos não podem ser alterados
DICIONÁRIOS	É um conjunto de valores, onde cada valor é associado a uma chave de acesso
BIBLIOTECAS	Armazenam funções pré-definidas, que podem ser utilizados em qualquer momento do programa.
ESTRUTURAS DE DECISÃO	Permitem alterar o curso do fluxo de execução de um programa, de acordo com o valor (Verdadeiro/Falso) de um teste lógico
ESTRUTURAS DE REPETIÇÃO	É utilizada para executar uma mesma sequência de comandos várias vezes. A repetição está associada ou a uma condição, que indica se deve continuar ou não a repetição, ou a uma sequência de valores, que determina quantas vezes a sequência deve ser repetida. As estruturas de repetição são conhecidas também como laços (loops).
FUNÇÕES	São pequenos trechos de código reutilizáveis. Elas permitem dar um nome a um bloco de comandos e executar esse bloco, a partir de qualquer lugar do programa.

Fonte: Grupo PET-ADS/IFSP- Campus de São Carlos/ 2016.

A memória de um computador é um arcabouço de informações com gavetas destinadas para cada item de dados. As variáveis são utilizadas em programação para “armazenar valores e para dar nome a uma área de memória do computador onde armazenamos dados” (MENEZES, 2019, p. 46).

Na variável *strings*, são armazenadas cadeias de caracteres, intitulado de cadeia de caracteres, ou seja, uma sequência de símbolos, sinais de pontuação, no qual cada bloco tem um significado, uma sequência lógica.

Na estrutura de decisão serve para selecionar qual a parte de um programa deve ser ativada e quando. Assim como, também, quando, deve ser ignorado. Na

Estrutura de repetição, é utilizado para a execução de repetições de procedimentos quantas vezes for necessário para um ou mais processos de parte do programa. E nas bibliotecas, do projeto estudado, foram utilizadas:

- as **RPi.GPIO,** permite a configuração, a leitura, e comandar os pinos da placa GPIO;
- a **Matplotlib.pyplot**, que é uma coleção de funções para fazer a biblioteca *matplotlib* trabalhar igual ao programa MATLAB, no qual, MATLAB é um software interativo de alta performance voltado para o cálculo numérico, e o *matplotlib* é uma biblioteca de software voltada a criação de gráficos e visualização de dados em geral. Cada função do *pyplot* permite trabalhar com imagens plotando áreas nessa figura, criando linhas nas áreas plotadas e decorando essas regiões conforme o interesse do usuário;
- a **Numpy**, o qual é o pacote fundamental para computação científica em Python. É uma biblioteca *Python* que fornece um objeto matricial multidimensional, vários objetos derivados e uma variedade de rotinas para operações rápidas em matrizes, incluindo matemática, lógica, manipulação de formas, classificação, seleção, transformadas discretas de *Fourier*, álgebra linear básica, operações estatísticas básicas, simulação aleatória e muito mais;
- a **OpenCV,** é uma biblioteca para a visão computacional cujo o objetivo é criar uma infraestrutura de simples uso para a visão computacional para facilitar a construção de devidamente sofisticadas aplicação a visão de forma rápida.

Para instalar o *Raspbian* no *Raspberry Pi,* é necessário um cartão microSD, de preferência, que tenha no mínimo 8 (oito) GB e um computador com estrada para o microSD. Na sequência, fazer o download do *Raspbian deszipe* e instalá-lo no microSD; abra o terminal, após colocar o microSD no *Raspiberry*, e digite: **sudo raspi config**. Esse comando abrirá um diretório; nesse diretório deve ir na opção **sete** (**advanced options**) e selecionar; ao proceder com esse comando, abrirá outra porta, no qual, deverá ser selecionado a primeira opção **A1**(**expand fliesystem**), esse passo é feito para ter certeza que o *Pi* esteja usando todo o espaço disponível no microSD; após o término dos comandos, basta reiniciar a máquina; escreva no terminal **sudo reboot.** Para verificar que o *Raspbian* está totalmente atualizado no terminal, deve-se digitar **sudo apt-get update,** aperte o comando [Enter] e digite

apt-get upgrade. Com o *Raspbian* atualizado, está pronto para as instalações dos programas necessários.

Para a instalação do *Python* é necessário abrir o terminal e digitar **sudo apt update**. Proceda com o comando [Enter] e digite **sudo apt install python3 idle3,** novamente dê o comando [Enter] e aguardar o *Raspberry* terminar de instalar o *Python*. Para instalar as bibliotecas basta digitar os seguintes comandos no terminal:

- verificação de atualização do Raspbian: **sudo apt-get update**;
- o RPi.GPIO: **sudo apt-get install rpi.gpio**;
- o matplotlib.pyplot: **sudo apt-get install python3-matplotlib**;
- o numpy: **sudo apt-get install python3-numpy**; e
- o OpenCV: **sudo pip install opencv-contrib-python.**

Das bibliotecas, a única que pode causar alguma confusão é o **openCV**. Assim, caso ocorra algum problema após a instalação do comando da biblioteca **openCV**, siga os passos adiante: para instalar o openCV, primeiro, instala-se o Cmake no terminal escrevendo os seguintes pacotes:

```
sudo apt install snapd ,
sudo snap install cmake –classic
```

Instale o *python3*, caso já não o tenha feito: **sudo apt-get install python3-dev**. Na sequência, abaixe os pacotes do **openCV** digitando no terminal: **get -O opencv.zip https://github.com/opencv/opencv/archive/4.0.0.zip**

Observa-se que o **openCV,** possuem pacotes pré-montados para o Python. Assim, é importante fazer o *download*: **Wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.0.0.zip**

Em seguida, desempacotar os arquivos baixados do **openCV**, com seguinte comando: **unzip opencv.zip, unzip opencv_contrib.zip**. Como o **openCV** requer o *numpy* para funcionar instale-o**: pip install numpy** .

Siga as seguintes linhas de comando no terminal e termine a instalação do **openCV:**

```
cd ~/opencv-4.0.0
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-4.0.0/modules \
-D ENABLE_NEON=ON \
-D ENABLE_VFPV3=ON \
-D BUILD_TESTS=OFF \
-D WITH_TBB=OFF \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D BUILD_EXAMPLES=OFf
Make –j4
```

Caso ocorra algum erro ao usar **Make –j4**, tente **Make –j1** e continue com a instalação: **sudo apt-get install libopencv-devpython-opencv.** Com esse procedimento a biblioteca **openCV** será instalada.

Para verificar a instalação escreva no terminal:

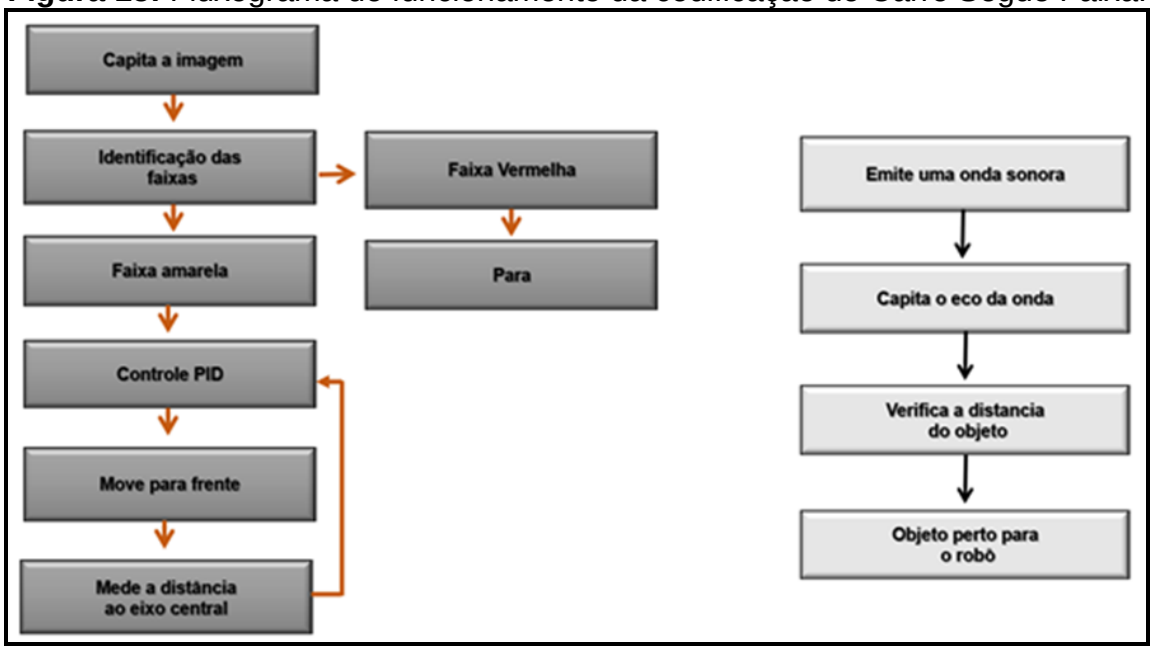
```
python       [após dar Enter]
import cv2  [após das Enter]
```

Se instalado corretamente, o terminal responderá com a versão do *Python* e, não deverá devolver nenhum tipo de erro.

Também, existe outro caminho para instalar as outras bibliotecas, ou seja, a método manual. Todavia, não será abordado esse método para outras bibliotecas, além a do **openCV,** devido ao excessivo número de passos necessários, a ser operacionalizado, para o processo de cada instalação.

No menu do *Raspberry Pi* na aba PREFERENCIAS, *Raspberry Pi* configurações. Dentro das CONFIGURAÇÕES, habilite na aba INTERFACES o SPI e o I2C para poder liberar a câmera e os outros componentes. Na figura 15, se apresenta o fluxograma do funcionamento da codificação do Carro Segue Faixa:

Figura 15: Fluxograma do funcionamento da codificação do Carro Segue Faixa.



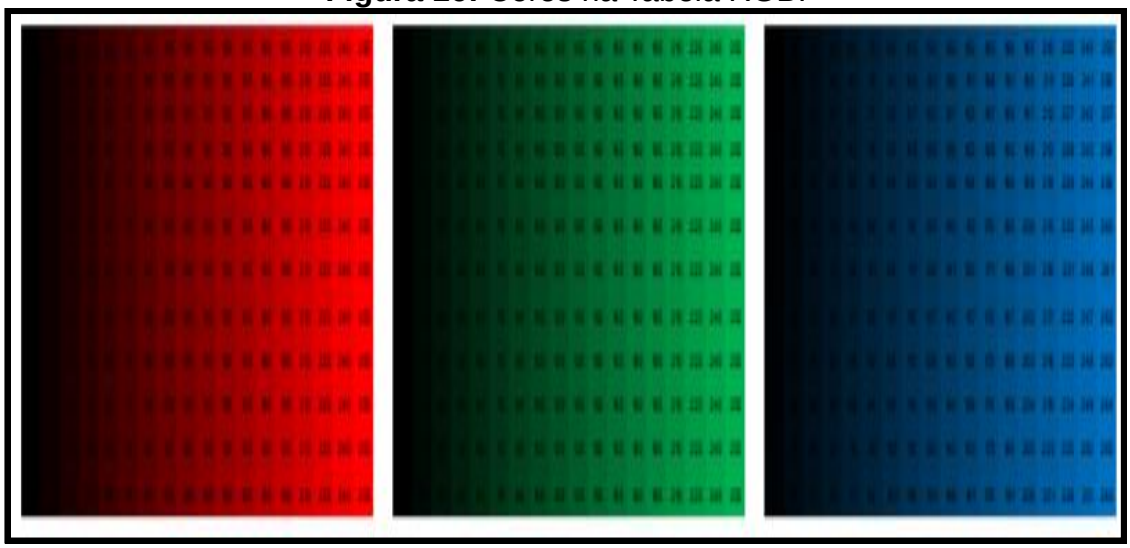
Fonte: Autor/2022.

Na figura 15, demonstra-se a captação da imagem e logo após o tratamento. No qual, são identificadas as faixas amarelas e as vermelhas. Ao ser identificada a faixa vermelha, o Carro Segue Faixa para a sua movimentação por um determinado tempo. Caso não for identificado a cor vermelha, o Carro Segue Faixa seguirá a faixa amarela, conforme o calculo do PID. Com relação ao sensor, caso for detectado um objeto próximo ao carro, o mesmo para a sua movimentação até o objeto ser retirado.

5.1 TRATAMENTO DE IMAGEM.

Na aplicação prática, de acordo com Antonello (2017), cada pixel em uma imagem colorida é constituído por três matrizes de duas dimensões, representando uma das cores do sistema RGB (Red; Green; Blue) de 8 Bits, para cada pixel, dando ao todo um total de 16,7 milhões de combinações sendo (0,0,0) o preto, e (255,255,255) o branco. Na figura 16, a demonstração das Tabelas RGB e as suas respectivas cores:

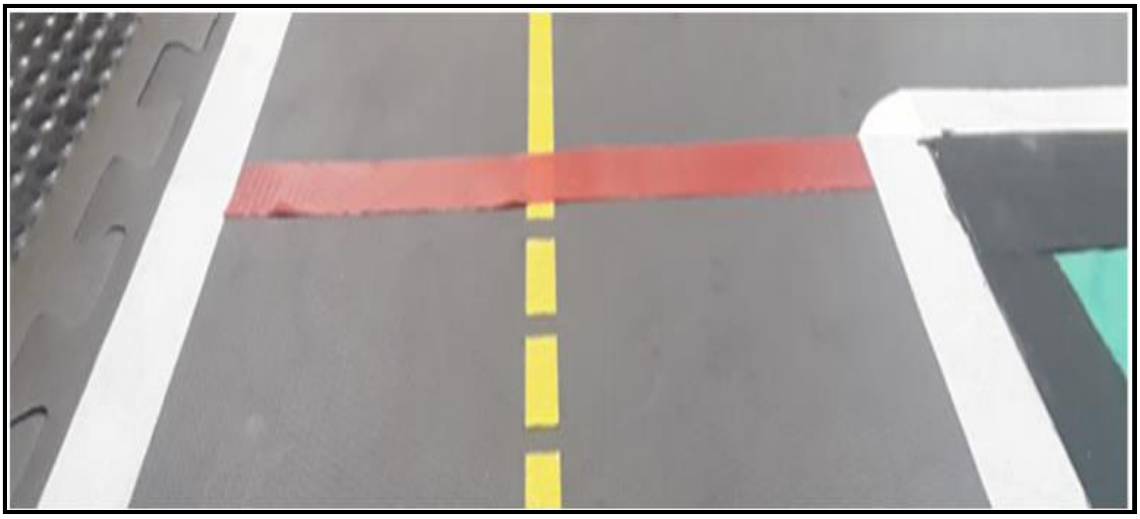
Figura 16: Cores na Tabela RGB.



Fonte: ANTONELLO, 2017, p.08.

O computador procede com a leitura do pixel da esquerda para direita e de cima apara baixo, nessa ordem. Assim, é possível classificar o pixel igualmente como feito em uma matriz, sendo possível ler e saber a informação individual de cada pixel e a ordem de leitura dos mesmos pelo computador. Na figura 17 a demonstração do percurso.

Figura17: O percurso.



Fonte: Autor/2022.

Uma das maneiras de tratamento de imagem é chamada de ‘mascara’, cujo o intuito é de isolar certas matrizes de cores geradas pelo sistema RGB, fazendo assim, a imagem ser constituível apenas pelas matrizes que foram isoladas, facilitando a identificação de objetos e formas na imagem. A imagem criada pela

máscara é a imagem utilizada para a criação e identificação dos objetos na visão computacional.

Utilizando a biblioteca do **OpenCV**, primeiramente **cv2.cvtcolor( )** é o método utilizado para converter de um espaço de cor para outro, nesse caso para o espaço RGB.

Na sequência, se utiliza de **array** para separar as matrizes de maior e menor valor, as quais englobam a cor desejada: **np.array ([, ,], dstype = “unit8”)**. No final para a apresentação da imagem: **cv2_imshow( )**.

A Figura 18, mostra a imagem da figura 17 apresentada após o tratamento de imagem feito por uma máscara, para a identificação e isolamento das matrizes da cor amarela.

Figura 18: Modelo de Máscara aplicada e a sua codificação.

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt

img = cv2.imread("20190909_151751.png")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
low_yellow = np.array([15, 52, 132], dtype = "uint8")
up_yellow = np.array([50, 255, 255], dtype = "uint8")
mask_yellow = cv2.inRange(hsv, low_yellow, up_yellow)
cv2_imshow(mask_yellow)
```

Fonte: Autor/2022.

Em seguida, para a construção da máscara para o isolamento da cor vermelha é necessário fazer o uso de quatro matrizes de isolamento, pois a cor vermelha está localizada nas primeiras e últimas matrizes. Na figura 19, demonstra-se a máscara para a cor vermelha e a sua codificação:

Figura 19: Modelo de Máscara para a cor vermelha e a sua codificação.

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt

img = cv2.imread("20190909_151751.png")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

rangeMin = np.array([0, 50, 50], np.uint8)
rangeMax = np.array([10, 255, 255], np.uint8)
mask1 = cv2.inRange(hsv, rangeMin, rangeMax)

rangemin = np.array([160, 50, 50], np.uint8)
rangemax = np.array([180, 255, 255], np.uint8)
mask2 = cv2.inRange(hsv, rangemin, rangemax)

mask= mask1+mask2
cv2_imshow(mask)
```

Fonte: Autor/2022.

Da codificação apresentada na figura 18 (amarela) para a da figura 19 (vermelha) a diferença está na construção de uma máscara a mais, utilizando dois *arrays* em cada uma e, finalmente, somando essas duas máscaras criando uma terceira, a máscara da cor vermelha.

É possível criar contornos na imagem original nas regiões onde foi detectado a matriz de cor que a máscara isola e desses contornos, por meio de restrição de tamanhos de áreas, selecionados certos contornos para serem trabalhados. Nesse contorno é criado um retângulo e um círculo; no centro desse retângulo para facilitar na aproximação da localização do centro desse contorno selecionado, esse processo é feito do seguinte modo.

Para estabelecer o contorno utiliza: **cnts, hierarchy= cv2.FindCountours(, ,)**, o qual, *hierarchy* significa que haverá uma ordem hierárquica que escolherá os contornos; os três argumentos dentro do **cv2.FindCountours( )** são em ordem a imagem fonte, o modo o qual o contorno será retirado (de forma hierárquica

utilizada); no caso do **cv2,RETR_ESTERNAL** faz o programa procurar contornos na parte inferior da imagem e, a forma de aproximação utilizada para desenhar o contorno.

Para o desenho do contorno da imagem, cor e tamanho do contorno, a codificação dever ser: **cv2.drawContours(imagem, contorno, ,(, , ,),)**. Exemplo do código:

```
cnts,hierarchy,=cv2.findContours(mask_yellow.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
 cv2.drawContours(img,cnts,-1,(255,0,255),2)
```

O **contorno.reverse()** vem para inverter a ordem e forma de obtenção dos contornos, fazendo o programa agora procurar os contornos na parte superior da imagem. Exemplo código:

```
cnts.reverse()
```

É criada estrutura de repetição utilizando “for” para colocar duas estruturas de condição: a primeira estrutura, se o contorno for maior que o contorno mínimo e esse contorno será desenhado. Se, o contorno for menor que o contorno mínimo, esse contorno será ignorado.

A segunda estrutura é a de condição: se a imagem conter algum contorno e for detectado pela primeira condição, será traçado uma linha do eixo de referência vertical e criado um ponto no centro desse contorno utilizando o **cv2.line( )**. Exemplo da codificação:

```
for c in cnts:
  if cv2.contourArea(c) < AreaContornoLimiteMin:
    continue

if (QtdeContornos > 0):
    cv2.line(img,PontoCentralContorno,(round(width/2),CoordenadaYCentroContorno),(0,255,0),1)
```

Observe que a linha central vertical de referência cujo o objetivo é ser usada como um guia é: criar pela linha de texto **cv2.line( ),** nos quais seus argumentos são: a ponte de início; o ponto final; a cor da linha criada; e a grossura da linha. Exemplo da codificação:

```
cv2.line(img,(round(width/2),0),(round(width/2),round(height)),(255,0,0),2)
```

No caso de existir algum contorno na imagem cuja imagem tenha uma área maior que a área mínima, nele será somado mais um [+1] na variável quantidade de contornos. Exemplo da codificação:

```
QtdeContornos = QtdeContornos + 1
```

Na sequência usa-se **o cv2.boundingRect()** para ser criado um retângulo que encaixe de melhor maneira possível dentro desse contorno. Exemplo da codificação:

```
(x, y, w, h) = cv2.boundingRect(c)
```

Os argumentos da função **cv2.rectangle()** são em ordem a imagem utilizada, as coordenadas iniciais do retângulo, as coordenadas finais do retângulo, a cor, e a grossura da sua borda, conforme o exemplo da codificação:

```
cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

Para encontrar o centro desse retângulo criado é usado uma conta matemática de geometria, conforme o exemplo da codificação:

```
CoordenadaXCentroContorno = round((x+x+w)/2)
CoordenadaYCentroContorno = round((y+y+h)/2)
PontoCentralContorno = (CoordenadaXCentroContorno,CoordenadaYCentroContorno)
```

No próximo passo é utilizado essa coordenada do ponto central, para fazer um círculo nesse ponto central usando **cv2.circle()**, no qual, os argumentos são respectivamente a imagem, a localização desse ponto central, o tamanho do raio desse círculo, a cor do círculo e a grossura do mesmo, de acordo com a codificação:

```
cv2.circle(img, PontoCentralContorno, 1, (0, 0, 0), 5)
```

Da biblioteca do **numpy np.size( ),** essa função conta os elementos de um certo eixo e seus complementos são os dados inicias, ao longo do qual os elementos (linhas ou colunas), são contados. Por padrão, se deve fornecer o número total de elementos em uma matriz. Exemplo da codificação:

```
height = np.size(img,0)
width= np.size(img,1)
```

O código **AreacontronoLimiteMin** como o nome já diz é uma variável empírica que tem a função de dar o valor utilizado como a área mínima do contorno; o código **QtdeContornos** é uma variável dizendo de quantos contos a quando o programa é iniciado. Segue exemplo da codificação:

```
AreaContornoLimiteMin = 750
QtdeContornos = 0
```

No quadro 2, demonstra-se a aplicação da codificação, ao fazer a junção sequencial de todas as linhas de código. Assim, para ter a cor amarela, por exemplo, apresenta-se a formatação do seguinte código:

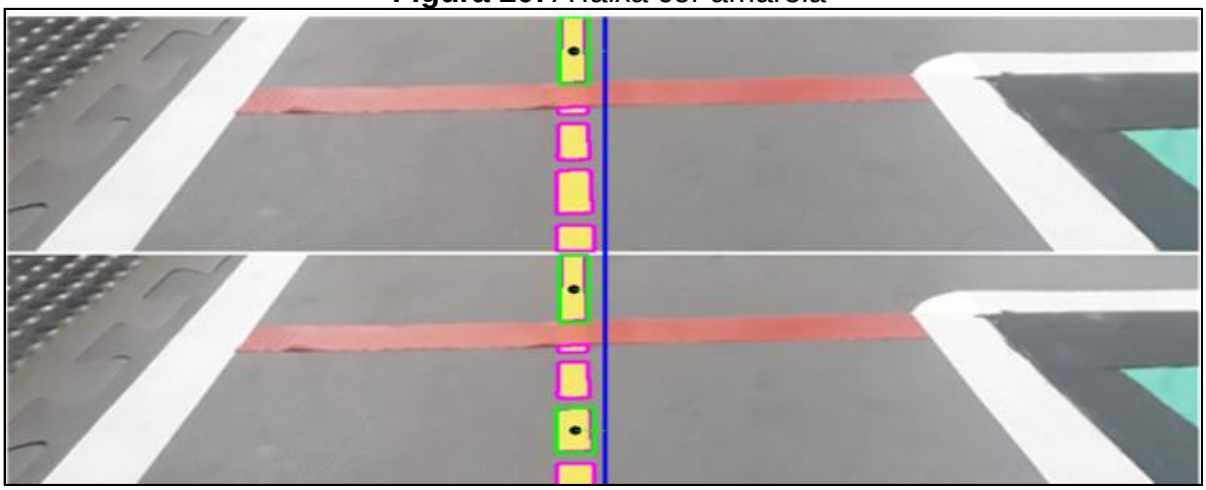
Quadro 2: Codificação para a Cor Amarela

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

AreaContornoLimiteMin = 750
img = cv2.imread("20190909_151751.png")

height = np.size(img,0)
width= np.size(img,1)
QtdeContornos = 0

hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
low_yellow = np.array([15, 52, 132], dtype = "uint8")
up_yellow = np.array([50, 255, 255], dtype = "uint8")
mask_yellow = cv2.inRange(hsv, low_yellow, up_yellow)
cv2.waitKey(1)
cnts, hierarchy, = cv2.findContours(mask_yellow.copy(), cv2.RETR_EXTERNAL, cv2
.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img,cnts,-1,(255,0,255),2)

cnts.reverse()
for c in cnts:

 if cv2.contourArea(c) < AreaContornoLimiteMin:
  continue
 QtdeContornos = QtdeContornos + 1
 (x, y, w, h) = cv2.boundingRect(c)

 cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
 CoordenadaXCentroContorno = round((x+x+w)/2)
 CoordenadaYCentroContorno = round((y+y+h)/2)
 PontoCentralContorno = (CoordenadaXCentroContorno,CoordenadaYCentroContorno)
 cv2.circle(img, PontoCentralContorno, 1, (0, 0, 0), 5)

 cv2.line(img,(round(width/2),0),(round(width/2),round(height)),(255,0,0),2)

 if (QtdeContornos > 0):
  cv2.line(img,PontoCentralContorno,(round(width/2),CoordenadaYCentroContorno)
,(0,255,0),1)
  cv2_imshow(img)
```

Fonte: Autor/2022.

Ao fazer a aplicação da codificação apresentada no quadro 2, ao final gera a imagem:

Figura 20: A faixa cor amarela



Fonte: Autor/2022.

A figura 20, representa o resultado do *loop* gerado pelo programa do quadro 2. É identificado as áreas de amarelo e as áreas, cujo o tamanho, se adéqua ao código. Dentro da área selecionada será traçado um retângulo e calculado o centro desse retângulo. No centro desse retângulo é criado um ponto para representar essa coordenada central. E, do ponto central até a linha de referência, é calculado a distância (entre o ponto central e a referência), para que essa distância seja utilizada mais tarde na parte responsável pela locomoção do carro.

É definido toda esse tratamento de imagem como uma função e acrescentado a variável **DireçãoASerTomada,** que tem como objetivo o de definir a distância que o carro se mantêm do eixo vertical com relação ao centro do retângulo, criado no tratamento da imagem. Exemplo da codificação:

```
DirecaoASerTomada = CoordenadaXCentroContorno - (width/2) + 90
```

E, assim, como é feito para o amarelo é definido o contorno para o vermelho, no entanto, para o vermelho é apenas criado um contorno simples para ser identificado a sua área, conforme segue:

```
conts, _, = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
   conts = sorted(conts, key = cv2.contourArea, reverse = True)[:1]
   areas = [ ]
for c in conts:
areas.append(cv2.contourArea(c))
Areacalc = np.asarray(areas)
```

5.2 CODIFICAÇÃO DO CARRO SEGUE FAIXA.

Para o procedimento da codificação do Carro Segue Faixa, se faz necessário executar alguns procedimentos para cada etapa:

a) Dos procedimentos-padrão:

Para iniciar a codificação dentro do Carro Segue Faixa, primeiro é necessário a importação das bibliotecas utilizadas em todo o código. A codificação deve seguir os procedimentos:

```
import cv2
import numpy as np
import RPi.GPIO as GPIO
import time
```

Na sequência, são construídos as variáveis e constantes globais. Globais, pois podem ser utilizadas em todo o código, como segue:

```
limiarBinarizacao = 127
areaContorno_LimiteMinimo = 5
kd = 0.6
kp = 1.3
ki = 0.001
lastError = 0
totalError = 0
statusParada = 0
```

Logo em seguida, é configurado a câmera fazendo ele começar a captura de imagens em *stream;* os valores da função representam a resolução da área capturada pela câmera; áreas grandes fornecem precisão maior, porém, deixam o código mais lento. Segue a configuração:

```
camera = cv2.VideoCapture(0)
camera.set(3,320)
camera.set(4,240)
```

b) Da função do GPIO:

Encerrados os procedimentos descritos no item dos procedimentos-padrão, define-se uma função, na qual, nela estará toda a configuração da GPIO e as variáveis que podem ser utilizadas para controlar os motores e o sensor, dado por meio de:

```
def setupGPIO():
    global motorDireito
    global motorEsquerdo
    global velocidade_motorA
    global velocidade_motorB
```

Explica-se que o termo global, antes das variáveis, faz elas se tornarem ‘variáveis globais’, utilizadas em outras funções e, ao mesmo tempo, utilizada dentro da função do GPIO, para poder, controlar a placa GPIO.

As variáveis responsáveis pela velocidade dos motores e, a qual, o programa (código) irá utilizar para manobrar o carro segue faixa, são as duas variáveis: **velocidade_motorA** e **velocidade_motorB**.

As outras duas variáveis **motorDireito** e **motorEsquerdo** são as responsáveis em controlar os motores e os manter funcionando. Na sequência, demonstra-se a configuração das portas da GPIO, tanto, para os motores do carro, quanto, para o sensor.

```
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)

GPIO.setup(7, GPIO.OUT)
GPIO.setup(11, GPIO.IN)

GPIO.setup(33, GPIO.OUT)
motorDireito = GPIO.PWM(33, 100)
GPIO.setup(31, GPIO.OUT)
GPIO.setup(29, GPIO.OUT)

GPIO.setup(32, GPIO.OUT)
motorEsquerdo = GPIO.PWM(32, 100)
GPIO.setup(35, GPIO.OUT)
GPIO.setup(36, GPIO.OUT)

motorDireito.start(0)
motorEsquerdo.start(0)

velocidade_motorA = 60
velocidade_motorB = 60
```

Na sequência, é realizado a codificação do tratamento de imagem. No programa vem a função do tratamento de imagem que já foi explicado detalhadamente nas páginas 33 a 42.

c) Da função de verificar a parada do carro no vermelho:

A função para verificar a parada do carro no vermelho, segue o código:

```
def verificarParada(areaFaixaVermelha):
```

No qual, é chamado e criado novas variáveis globais:

```
global motorDireito
 global motorEsquerdo
 global velocidade_motorA
 global velocidade_motorB
 global statusParada
 global valorParada
```

Caso o carro veja uma área pré estabelecida de vermelho, o programa faz os motores pararem e, após um tempo pré determinado, nesse caso, quatro segundos, o carro volta a poder andar. A seguir a codificação necessária:

```
if (statusParada == 0 and areaFaixaVermelha >= 1500):

    motorDireito.ChangeDutyCycle(0)
    motorEsquerdo.ChangeDutyCycle(0)
    time.sleep(4)

    motorDireito.ChangeDutyCycle(velocidade_motorA)
    motorEsquerdo.ChangeDutyCycle(velocidade_motorB)
    statusParada = 1
    elif(statusParada == 1 and areaFaixaVermelha < 100):
    statusParada = 0
```

A variável **statusParada,** tem o objetivo de evitar que o carro fique parando na mesma faixa de vermelho. Ao passar o carro pela faixa de vermelho, ele restaura o valor inicial da variável **statusParada**, assim, reinicia a função para verificar a parada do carro na faixa vermelha.

d) Da função do PID:

A função do PID (*Proportional Integral Derivative*), é responsável por fazer o cálculo da velocidade necessária, para os motores manterem o carro em constante

movimento e estabilidade, ao seguir a linha central da pista de percurso. Para esse procedimento, são necessários os seguintes códigos:

```
def PID(valorAjuste):
    global velocidade_motorA
    global velocidade_motorB
    global motorDireito
    global motorEsquerdo
    global kd
    global kp
    global ki
    global error
    global totalError
    global lastError
```

As variáveis **kd, kp, ki**, são as variáveis, respectivamente, utilizadas pelo PID para fazer a derivada, o produto e a integral.

As últimas três variáveis **error**, **totalError** e **lastError,** são utilizadas para construir uma função junto ao PID e com o resultado do tratamento de imagem, o qual, é recebido os parâmetros que o carro deve estabelecer perante a faixa central amarela, que ele usa como guia.

```
if (valorAjuste <> 0):
        error = valorAjuste
        totalError = totalError + error

        proporcional = kp * error
        derivada = kd * (error - lastError)
        integral = ki * totalError

        velocidade_motorA = 60 - proporcional - integral - derivada
        velocidade_motorB = 60 + proporcional + integral + derivada
        lastError = error
```

Na codificação anterior, se deve após o programa saber a distância que ele está da faixa central amarela (central da pista), calcular a velocidade necessária para o carro se manter seguindo a faixa amarela.

Pode ocorrer uma correção na velocidade, caso o programa comande algum dos motores: ‘ir muito rápido’ ou ‘muito devagar’. Com isso, se evita de o carro girar e perder a faixa central amarela da pista. Segue a codificação:

```
if(velocidade_motorA > 100):
    velocidade_motorA = 100
elif(velocidade_motorA < 10):
    velocidade_motorA = 10

if(velocidade_motorB > 100):
    velocidade_motorB = 100
elif(velocidade_motorB < 10):
    velocidade_motorB = 10
```

Dada a codificação descrita ao PID, o programa coloca os valores das variáveis de velocidades dentro dos controladores do motor, como segue:

```
motorDireito.ChangeDutyCycle(velocidade_motorA)
motorEsquerdo.ChangeDutyCycle(velocidade_motorB)
GPIO.output(31, GPIO.LOW)
GPIO.output(29, GPIO.HIGH)
GPIO.output(35, GPIO.HIGH)
GPIO.output(36, GPIO.LOW)
```

Caso o carro não veja a faixa amarela ele irá parar. Assim,

```
def verificaTrajeto(faixaAmarela):
    if(faixaAmarela == False):
        GPIO.output(31, GPIO.LOW)
        GPIO.output(29, GPIO.LOW)
        GPIO.output(35, GPIO.LOW)
        GPIO.output(36, GPIO.LOW)
```

e) Da função do Sensor:

A função para o sensor faz o sensor produzir um pulso ultrassônico por um curto período de tempo. Assim, o sensor capta qualquer eco gerado na frequência do pulso produzido. A codificação é:

```
def distance():

    GPIO.output(7, GPIO.LOW)

    time.sleep(0.0001)

    GPIO.output(7, GPIO.HIGH)

    pulseendtime = time.time()
    pulsestarttime = time.time()
    time.sleep(0.00001)

    GPIO.output(7, GPIO.LOW)
```

O primeiro eco captado pelo sensor é utilizado para o cálculo da distância que o carro está de um possível objeto, de acordo com a codificação:

```
    while GPIO.input(11)==0:
        pulse_start_time = time.time()
    while GPIO.input(11)==1:
        pulse_end_time = time.time()

        pulse_duration = pulse_end_time - pulse_start_time
        SensorParada = round(pulse_duration * 17150, 2)
```

Caso for detectado algum objeto perto, a menos de cinco centímetros do carro, o mesmo parará e permanecerá parado, até o objeto ser retirado da frente dele.

```
if(SensorParada < 5)
    motorDireito.ChangeDutyCycle(0)
    motorEsquerdo.ChangeDutyCycle(0)
    time.sleep(4)

    motorDireito.ChangeDutyCycle(velocidade_motorA)
    motorEsquerdo.ChangeDutyCycle(velocidade_motorB)
```

A partir de todas as funções criadas, o programa principal utilizará de todas as funções, criando um *loop* ou um processo infinito. Para fazer o carro executar as funções preestabelecidas para *frame* de imagens captado ou cada Eco detectado pelo sensor. Segue a codificação final:

```
setupGPIO()

while True:
    (grabbed, frame) = camera.read()

    if(grabbed):
        valorAjuste,faixaAmarela,areaFaixaVermelha = tratalmagem(frame)
        Sensor()
        PID(valorAjuste)
        verificarParada(areaFaixaVermelha)
        verificaTrajeto(faixaAmarela)

exit(1)
```

No anexo I, apresenta-se toda a codificação na sua sequência para aplicação direta ao Carro Segue Faixa.

CONCLUSÕES

O Carro Segue Faixa pode ser considerado um pseudo robô ou semi-robô. Ele, tem quase todas as características de um robô. A característica que falta nele, é

a capacidade do carro fazer algum tipo de escolha autônoma sem a ação ou controle de um ser humano.

Contudo, embora o Carro Segue Faixa não seja um robô em sua plena forma, é possível utilizá-lo como protótipo para estudar os principais sistemas operacionais e componentes do robô móvel para se chegar, mais especificamente, aos componentes que deve ter um carro autônomo.

Nota-se a importância da observação dos componentes do robô móvel, a dificuldade ocorrida no processamento das informações recolhidas necessárias para o desenvolvimento da movimentação correta e segura, para entender a complexidade dos carros autônomos em seguir, por exemplo, simples leis de trânsito.

Essa pesquisa concentrou-se em entender, aplicando o conceito em um Carro Segue Faixa, qual a importância da visão computacional para os automóveis autônomos. Assim, com o estudo do Carro Segue Faixa é possível compreender algumas das principais dificuldades da visão computacional aplicada nos carros autônomos, sendo uma delas, a identificação de objetos quando o sistema é feito com cores.

Observou-se que essa dificuldade está relacionada as cores em si. O computador entende leves sombras como cores totalmente diferentes, tornando necessário criar uma maneira de fazer o programa entender que essas pequenas sutilezas nas corres podem ser ignoradas ou reajustadas para a interpretação do objeto em questão.

Em virtude de o carro estar sempre em movimento, há necessidade de outras formas de detecção, tendo em vista, o mesmo não ter a capacidade cognitiva igual atribuída ao cérebro humano de ter a percepção de detectar objetos fora do ‘comum’ e a sua distância especifica sem certo referencial, como por exemplo, uma folha caindo perto da câmera ou uma pedra grande jogada no meio do caminho, ou seja, tomada de decisões rápidas pela diversidades que possam ocorrer sem estar programado, a sensibilidade. Para isso é utilizado outros meios de sensoriamento para estabelecer uma visão completa e totalmente segura para que possa ocorrer o deslocamento do carro autônomo com total segurança, tanto, para o ambiente interno, como, externo do carro.

Em análise da visão computacional aplicada no Carro Segue Faixa, observa-se a dificuldade dos acadêmicos da área de definir com certeza o que é a visão computacional. É possível classificar ela como um tipo de processamento de

imagem, com a finalidade de identificar e categorizar ou catalogar objetos em 3D pré selecionados. Com isso, fazer a máquina ou robô compreender que aquele objeto possui três dimensões, de forma lógica.

Dado o entendimento da visão computacional, na sua essência, não é possível falar que na composição do Carro Segue Faixa tem contido em seu código o que é chamado de visão computacional. Pois, o código do carro identifica a pista como um objeto em duas dimensões para ser mais específico:

a) ele identifica retângulos que são pressupostos ser a faixa central da pista e a faixa de pedestre do percurso utilizado. No entanto, é possível utilizar o Carro Segue Faixa para identificar um grande problema da visão computacional quando utilizado apenas as informações obtidas pela câmera.
b) há dificuldade de estabelecer o conceito de profundidade no sentido que, o carro, não entende a diferença no qual existem faixas que então próximas dele e faixas que estão distantes. O carro apenas entende que existe faixas e que elas têm tamanhos diferentes. Se for colocado algum tipo de obstrução na frente do carro, mas ele ainda conseguir ver as faixas a frente da obstrução, ele irá colidir. Para evitar essa colisão, torna-se necessário atribuir formas diferentes de detecção, no qual o carro deva seguir além das captadas e trabalhadas sobre a câmera.

Muito, ainda, tem que ser pesquisado na área da visão computacional, dada a sua importância para aplicação nos carros autônomos. Seguindo o pensamento de que autonomia significa capacidade de tomar decisões precisas, e mais próximas do ideal. A visão computacional é um componente que leva a condução segura do carro autônomo, como também, a percepção de evitar de colocar-se em risco ou risco de outro, dentro da autonomia que lhe é atribuída. Como seres humanos, têm-se a visão como um componente básico e necessário para a percepção das ações do ambiente externo, a visão computacional é o ponto chave para o carro chegar próximo a autonomia tão desejada pelas empresas automobilísticas.

Segue o link para acesso ao código desenvolvido do Carro Segue Faixa:
https://github.com/rodolfo-lab/mascaras/blob/main/carro_segue_faixa.ipynb

REFERENCIAS:

ALMEIDA, Lucas Felipe Araújo. **Veículo Auto Guiado (Agv – Automated Guided Vehicle) - Protótipo seguidor se linha.** Monografia. Belo Horizonte: Centro Federal de Educação Tecnológica de Minas Gerais, 2016.

ANDERSON, James M.; KALRA, Nidhi; STANLEY, Karlyn D.; SORESEN, Paul; SAMARAS, Constantine; OLUWATOLA, Oluwatobi A. **Autonomous Vehicle Technology - A Guide for Policymakers**. Santa Monica/Calif: Rand Corporation, 2016. Disponível em: hhttps://wiscav.org/wp-content/uploads/2017/03/RAND_RR443-2_Guide_Policymakers.pdf. Acesso em: 22 de maio de 2022.

ANTONELLO. Ricardo. **Introdução a Visão Computacional com Python e OpenCV. 8**ª Versao. Disponível em https://professor.luzerna.ifc.edu.br/ricardo-antonello/wp-content/uploads/sites/8/2017/02/Livro-Introdu%C3%A7%C3%A3o-a-Vis%C3%A3o-Computacional-com-Python-e-OpenCV-3.pdf, aceso em agosto de 2021.

BRASDKI, Gary; KAEHLER, Adrian. **Learning OpenCV.** EUA: O'Reilly Media,2008.
CARVALHO JUNIOR, Helton Hugo de. **Métodos Inteligentes de Navegação e Desvio de Obstáculos.** Dissertação: Programa de Pós-Graduação em Engenharia Elétrica. Universidade Federal de Itajaí, 2007. Disponível em http://livros01.livrosgratis.com.br/cp048302.pdf , acesso em agosto de 2021.

COUTINHO, César Valentino Ribeiro. **Robótica Móvel – Sistema de Condução Autónoma.** Dissertação: Portugal: Instituto Superior de Engenharia de Lisboa (ISEL). Pós graduação de Engenharia eletrônica, 2014. Disponível em <https://repositorio.ipl.pt/bitstream/10400.21/3624/1/Disserta%C3%A7%C3%A3o.pdf >, acesso em: 23 de maio de 2022.

DONAT, Wolfram. **Learn Raspberry Pi Programming with Python**. EUA: Editora Apress,2014.

FRANCO, João Luiz. **Introdução à programação com Python**. São Carlos: Instituto Federal de Educação, Ciência e Tecnologia de São Paulo. Apostila desenvolvida pelos integrantes do Programa de Educação Tutorial do curso de Tecnologia em Análise e Desenvolvimento de Sistemas - grupo PET ADS / IFSP São Carlos, 2016. Disponível em< http://antigo.scl.ifsp.edu.br/portal/arquivos/2016.05.04_Apostila_Python_-_PET_ADS_S%C3%A3o_Carlos.pdf> acesso em 23/08/2021,

LERBROCK, Rafael. **Absorção da luz**. *Brasil Escola*. Disponível em: https://brasilescola.uol.com.br/fisica/absorcao-da-luz.htm. Acesso em 24 de março de 2022.
MARENGONI, Mauricio; STRINGHINI, Denise. **Tutorial: Introdução à Visão Computacional usando OpenCV. Artigo. Revista** RITA • Volume XVI • Número 1 • 2009

MATARIC, Maja J. **Introdução a Robótica.** 1. ed. Sao Paulo: Editora Unesp/Blucher, 2014. Traducao de Humberto Ferasol Filho, Jose reinaldo Silva e silas Franco dos Reis alves.

MENEZES, Nilo Ney Coutinho. **PYTHON Algoritmos e lógica de programação para iniciantes, 3 Edição.** São Paulo: Editora Novatec,2019.

NUSSENZVEIG, Herch Moysés. **Curso de Física Básica 2 Fluidos, Oscilações e ondas, Calor, 4 Edição.**São Paulo: Editora Blucher, 2002

NUSSENZVEIG, Herch Moysés. **Curso de Física Básica Vol.4/H.Moysés Nussenzveig – 1 Edição - -**São Paulo:Editora Blucher,1998.

OLIVEIRA, Guilerme Godoy de; CAMPOS, Eduarda Pains; PESSOA, Nathan Guedes**. Sistemas operacionais para utilização do Raspberry Pi como substituto a computadores tradicionais.** Artigo apresentado no X congresso Integrado da Tecnologia da Informação. Campos: Instituo Federal Fluminense, novembro de 2019.

OZGUNER, By Umit; STILLER, Christoph; REDMILL, Keith. S**ystems for Safety and Autonomous Behavior in Cars: The DARPA Grand Challenge Experience.** Vol. 95, No. 2, February 2007. Proceedings of the IEEE. Disponível em <https://www.mrt.kit.edu/z/publ/download/Oezguener_al2007ieeeproc.pdf>, acesso em Acesso em: 22 de maio de 2022.

RODRIGUES, Leonardo Cavalheiros. **Fundamentos, Tecnologias e Aplicações de Veículos Autônomos.** Ponta Grossa: UTPR, 2017. Disponível em< http://repositorio.utfpr.edu.br:8080/jspui/bitstream/1/16201/2/PG_COELE_2017_2_19.pdf>, acesso em agosto de 2021.

SILVA, Guilherme Damasceno **.Desenvolvimento de um Sistema de Visao Computacional para o Estudo do Comportamento de Animais Experimentais.** Belo Horizonte: Universidade Federal de Minas Gerais, 2008. Dissertação de Mestrado, 98p.

SILVEIRA, Luciana Martha. **Introdução à teoria da cor**. 2. ed. Curitiba: Ed. UTFPR, 2015.

SONKA, Milan; HLAVAC, Vaclav; BOYLE, Roger. **Image Processing, Analysis, and Machine Vision.** Fourth Edition. Boston/Massachusetts-EUA: Cengage Learning, 2015.

UPTON, Eben; HALFACREE, Gareth. **Raspberry Pi Guia do Usuário,4 Edição**. Rio de Janeiro: Editora Alta Books,2017.

WEBER, M. (2014). **Where to? A history of autonomous vehicles. Computer History Museum**, 2014. Disponível em: http://www.computerhistory.org/atchm/where-to-a-history-ofautonomous-vehicles/. Acesso em: 22 de maio de 2022

ANEXO I

CODIFICAÇÃO SEQUENCIAL DO CARRO

```
import cv2
import numpy as np
import RPi.GPIO as GPIO
import time


#------------------------------
# CONSTANTES E VARIAVEIS GLOBAIS
#------------------------------
limiarBinarizacao = 127
areaContorno_LimiteMinimo = 5

#PID
kd = 0.6 #0.8
kp = 1.3 #1
ki = 0.001
lastError = 0
totalError = 0

statusParada = 0



#------------------------------
# BLOCO DE METODOS  -----------
#------------------------------
camera = cv2.VideoCapture(0)
camera.set(3,320)
camera.set(4,240)


def setupGPIO():
    global motorDireito
    global motorEsquerdo
    global velocidade_motorA
    global velocidade_motorB

    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BOARD)

    GPIO.setup(7, GPIO.OUT)
    GPIO.setup(11, GPIO.IN)
```

```
    GPIO.setup(33, GPIO.OUT)
    motorDireito = GPIO.PWM(33, 100)
    GPIO.setup(31, GPIO.OUT)
    GPIO.setup(29, GPIO.OUT)

    GPIO.setup(32, GPIO.OUT)
    motorEsquerdo = GPIO.PWM(32, 100)
    GPIO.setup(35, GPIO.OUT)
    GPIO.setup(36, GPIO.OUT)

    motorDireito.start(0)
    motorEsquerdo.start(0)

    velocidade_motorA = 60
    velocidade_motorB = 60

def trataImagem(img):
    height = np.size(img,0)
    width= np.size(img,1)
    qtdContornos = 0
    valorAjuste = 0

    #tratamento da imagem
    img = cv2.flip(img, -1)
    img = img[:115,:]
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    rangeMin = np.array([0, 50, 50], np.uint8)
    rangeMax = np.array([10, 255, 255], np.uint8)
    mask1 = cv2.inRange(hsv, rangeMin, rangeMax)

    rangemin = np.array([160, 50, 50], np.uint8)
    rangemax = np.array([180, 255, 255], np.uint8)
    mask2 = cv2.inRange(hsv, rangemin, rangemax)

    mask= mask1+mask2
    #cv2.imshow('vermelho', mask)

    conts, _, = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    #cnts = imutils.grab_contours(cnts)
    conts = sorted(conts, key = cv2.contourArea, reverse = True)[:1]
    areas = []
    for c in conts:
        areas.append(cv2.contourArea(c))
    areaFaixaVermelha = np.asarray(areas)
    #print (areaFaixaVermelha)
    #areaFaixaVermelha = map(int, areas)
    low_yellow = np.array([15, 52, 132], dtype = "uint8")
    up_yellow = np.array([50, 255, 255], dtype = "uint8")
```

```
    mask_yellow = cv2.inRange(hsv, low_yellow, up_yellow)

    #cv2.imshow('F.B.',mask_yellow)

    cnts, hierarchy, = cv2.findContours(mask_yellow.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIM
PLE)
    cv2.drawContours(img,cnts,-1,(255,0,255),2)



    cnts.reverse()
    for c in cnts:
        #se a area do contorno capturado for pequena, nada acontece
        if cv2.contourArea(c) < areaContorno_LimiteMinimo:
            continue

        qtdContornos = qtdContornos + 1



        (x, y, w, h) = cv2.boundingRect(c)   #x e y: coordenadas do vertice superior esquerdo
                                             #w e h: respectivamente largura e altura do retangulo

        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

        CoordenadaXCentroContorno = (x+x+w)/2
        CoordenadaYCentroContorno = (y+y+h)/2
        PontoCentralContorno = (CoordenadaXCentroContorno,CoordenadaYCentroContorno)
        cv2.circle(img, PontoCentralContorno, 1, (0, 0, 0), 5)

        valorAjuste = CoordenadaXCentroContorno - (width/2) + 83  #em relacao a linha central


        cv2.line(img,(width/2,0),(width/2,height),(255,0,0),2)

        if (qtdContornos > 0):
            cv2.line(img,PontoCentralContorno,(width/2,CoordenadaYCentroContorno),(0,255,0),1)


        #cv2.imshow('Analise de rota',img)

    return valorAjuste, qtdContornos, areaFaixaVermelha



def verificarParada(areaFaixaVermelha):
    global motorDireito
    global motorEsquerdo
```

```
    global velocidade_motorA
    global velocidade_motorB
    global statusParada
    global valorParada


    if (statusParada == 0 and areaFaixaVermelha >= 1500):
        #desativa motores
        motorDireito.ChangeDutyCycle(0)
        motorEsquerdo.ChangeDutyCycle(0)
        time.sleep(4)
        #reativa motores
        motorDireito.ChangeDutyCycle(velocidade_motorA)
        motorEsquerdo.ChangeDutyCycle(velocidade_motorB)
        statusParada = 1

    elif(statusParada == 1 and areaFaixaVermelha < 100): #aguarda saida da faixa
        statusParada = 0



def PID(valorAjuste):
    global velocidade_motorA
    global velocidade_motorB
    global motorDireito
    global motorEsquerdo
    global kd
    global kp
    global ki
    global error
    global totalError
    global lastError


    if (valorAjuste <> 0):
        #print ("Distancia da linha de referencia: "+str(abs(valorAjuste))+" pixels a direita")
        error = valorAjuste
        totalError = totalError + error

        proporcional = kp * error
        derivada = kd * (error - lastError)
        integral = ki * totalError

        velocidade_motorA = 60 - proporcional - integral - derivada
        velocidade_motorB = 60 + proporcional + integral + derivada
        lastError = error

      #correcao velocidade
        if(velocidade_motorA > 100):
```

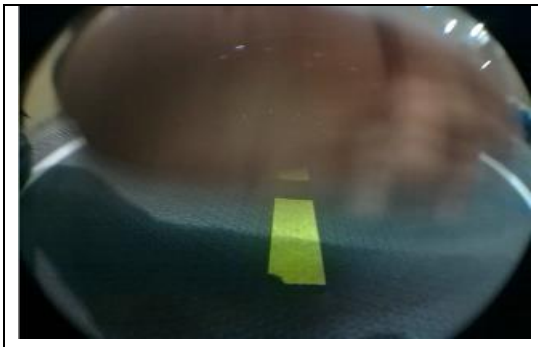
```
            velocidade_motorA = 100
        elif(velocidade_motorA < 10):
            velocidade_motorA = 10

        if(velocidade_motorB > 100):
            velocidade_motorB = 100
        elif(velocidade_motorB < 10):
            velocidade_motorB = 10


        motorDireito.ChangeDutyCycle(velocidade_motorA)
        motorEsquerdo.ChangeDutyCycle(velocidade_motorB)
        GPIO.output(31, GPIO.LOW)
        GPIO.output(29, GPIO.HIGH)
        GPIO.output(35, GPIO.HIGH)
        GPIO.output(36, GPIO.LOW)


def verificaTrajeto(faixaAmarela):
    if(faixaAmarela == False):
        GPIO.output(31, GPIO.LOW)
        GPIO.output(29, GPIO.LOW)
        GPIO.output(35, GPIO.LOW)
        GPIO.output(36, GPIO.LOW)


def distance():


    GPIO.output(7, GPIO.LOW)

      #print (" Aguardando o sensor estabilizar")

    time.sleep(0.0001)

      #print (" Cálculo de distância ")

    GPIO.output(7, GPIO.HIGH)

    pulse_end_time = time.time()
    pulse_start_time = time.time()
    time.sleep(0.00001)

    GPIO.output(7, GPIO.LOW)

      while GPIO.input(11)==0:
            pulse_start_time = time.time()
      while GPIO.input(11)==1:
            pulse_end_time = time.time()
```

```
    #print "Distance:",SensorParada ,"cm"
    pulse_duration = pulse_end_time - pulse_start_time
    SensorParada = round(pulse_duration * 17150, 2)

    if(SensorParada < 5)
        motorDireito.ChangeDutyCycle(0)
        motorEsquerdo.ChangeDutyCycle(0)
        time.sleep(4)
        #reativa motores
        motorDireito.ChangeDutyCycle(velocidade_motorA)
        motorEsquerdo.ChangeDutyCycle(velocidade_motorB)
    #print "Distance:",distance,"cm"


#------------------------------
#  PROGRAMA PRINCIPAL  --------
#------------------------------
setupGPIO()

while True:
    (grabbed, frame) = camera.read()

    if(grabbed):
        valorAjuste,faixaAmarela,areaFaixaVermelha = trataImagem(frame)
        Sensor()
        PID(valorAjuste)
        verificarParada(areaFaixaVermelha)
        verificaTrajeto(faixaAmarela)

exit(1)
```

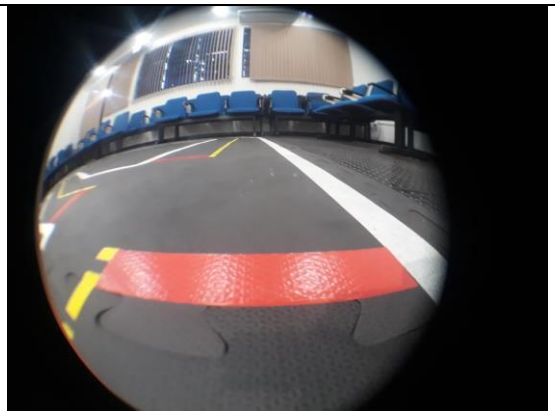
ANEXO II

IMAGENS DO CARRO/PROJETO INSTITUCIONAL/UEM

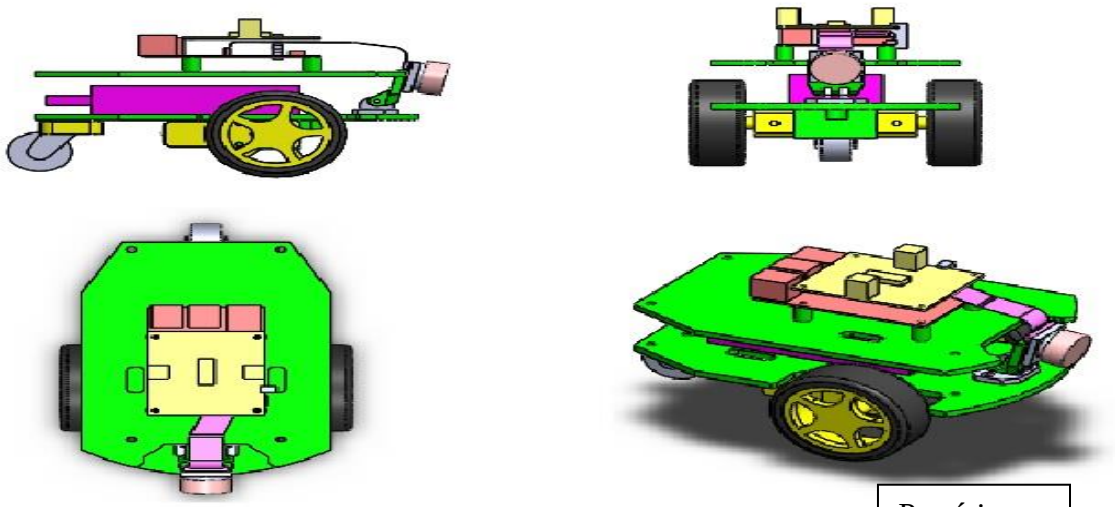
a) *Frame* – Saída com lente - Projeto Institucional/UEM/2019



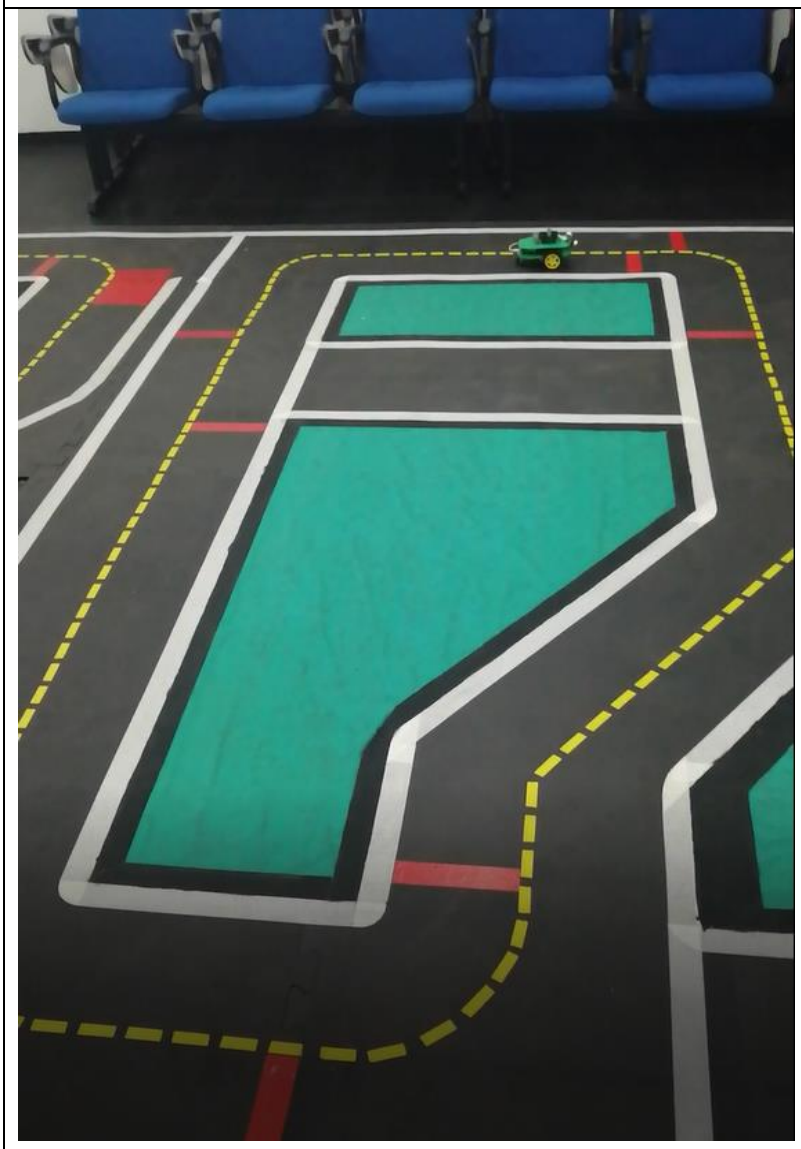


b) Apresentação do Carro desenvolvido – Projeto Institucional/UEM/2019

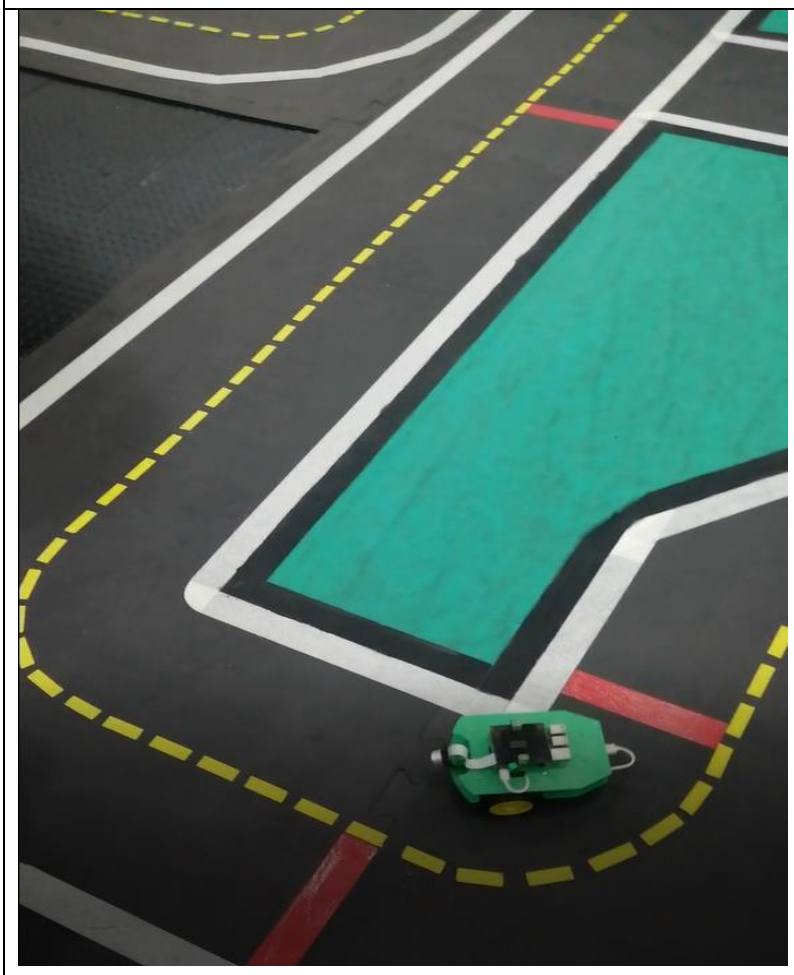


Protótipo



Apresentação Carro
Segue Faixa
Projeto /2019/UEM

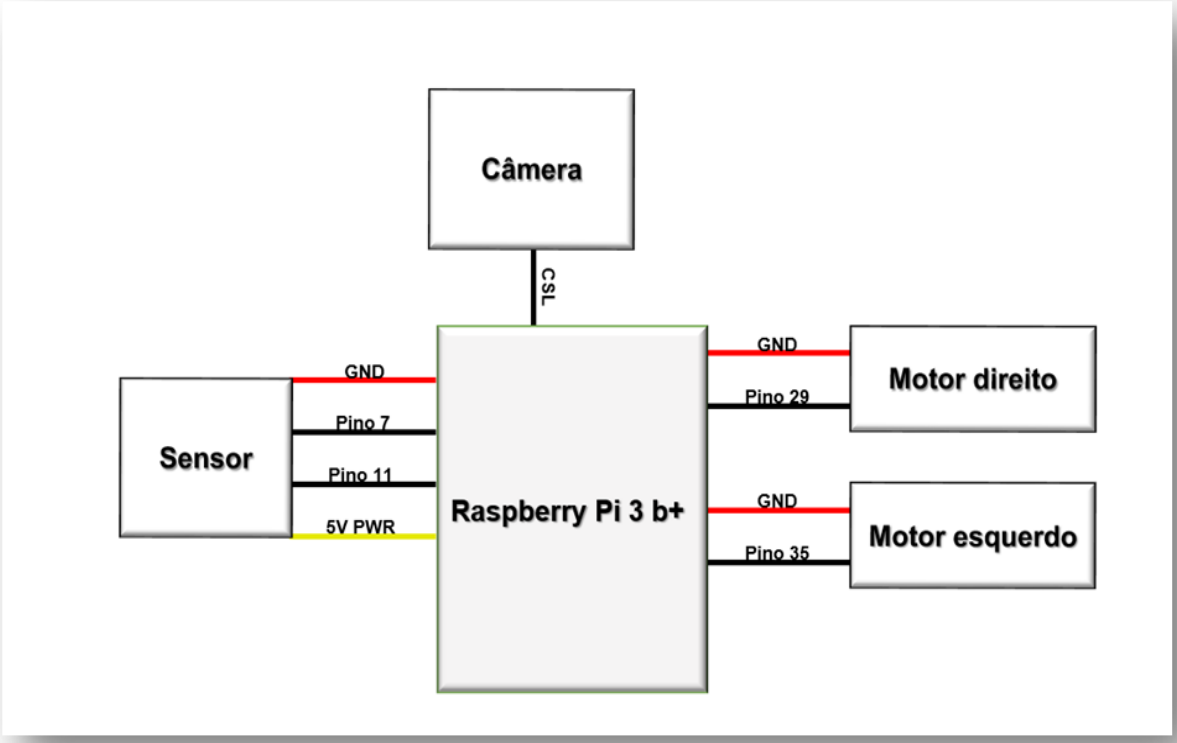
Apresentação Carro
Segue Faixa
Projeto/2019/UEM



Apresentação Carro
Segue Faixa
Projeto/2019/UEM

ANEXO III

CIRCUITO DO CARRO SEGUE FAIXA



Fonte: Autor/2022