UNIVERSIDADE DO ESTADO DO AMAZONAS
ESCOLA SUPERIOR DE TECNOLOGIA - EST

CARLOS EMANUEL DA COSTA SOUSA

DESENVOLVIMENTO DE PROTÓTIPO DE SISTEMA PARA AUXILIAR A LOCOMOÇÃO DE DEFICIENTES VISUAIS NO AMBIENTE URBANO UTILIZANDO SENSORES E VISÃO COMPUTACIONAL ATRAVÉS DO RASPBERRY PI.

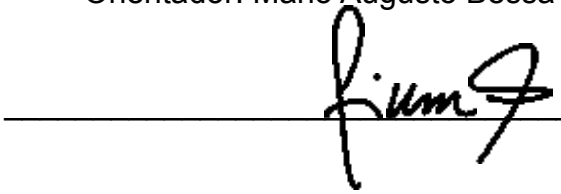
Manaus
2022

CARLOS EMANUEL DA COSTA SOUSA

DESENVOLVIMENTO DE PROTÓTIPO DE SISTEMA PARA AUXILIAR A LOCOMOÇÃO DE DEFICIENTES VISUAIS NO AMBIENTE URBANO, UTILIZANDO SENSORES E VISÃO COMPUTACIONAL ATRAVÉS DO RASPBERRY PI.

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Eletrônica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheiro em Eletrônica.

Orientador: Mario Augusto Bessa de Figueiredo



Manaus

2022

Universidade do Estado do Amazonas – UEA
Escola Superior de Tecnologia - EST

Reitor(a):
André Luiz Nunes Zogahib
Vice-Reitor(a):
Kátia do Nascimento Couceiro
Diretor(a) da Escola Superior de Tecnologia:
Ingrid Sammyne Gadelha Figueiredo
Coordenador do Curso de Engenharia Eletrônica:
Bruno da Gama Monteiro

Banca Avaliadora composta por: *Data da defesa: 09/05/2022.*
Prof. Mario Augusto Bessa de Figueiredo, MSc *(Orientador)*
Prof. Fábio de Sousa Cardoso, Dr
Prof. Daniel Guzmán Del Río, Dr

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).
Sistema Integrado de Bibliotecas da Universidade do Estado do Amazonas.

D111dd Sousa, Carlos Emanuel da Costa
Desenvolvimento de protótipo de sistema para auxiliar a locomoção de deficientes visuais no ambiente urbano utilizando sensores e visão computacional através do raspberry pi. / Carlos Emanuel da Costa Sousa. Manaus : [s.n], 2022.
55 f.: color.; 30 cm.

TCC - Graduação em Engenharia Eletrônica - Bacharelado - Universidade do Estado do Amazonas, Manaus, 2022.
Inclui bibliografia
Orientador: de Figueiredo, Mario Augusto Bessa

1. deficiente visual. 2. sensor ultrassônico. 3. Visão Computacional. 4. OpenCV. 5. YOLO. I. de Figueiredo, Mario Augusto Bessa (Orient.). II. Universidade do Estado do Amazonas. III. Desenvolvimento de protótipo de sistema para auxiliar a locomoção de deficientes visuais no ambiente urbano utilizando sensores e visão computacional através do raspberry pi.

Elaborado por Jeane Macelino Galves - CRB-11/463

CARLOS EMANUEL DA COSTA SOUSA

DESENVOLVIMENTO DE PROTÓTIPO DE SISTEMA PARA AUXILIAR A LOCOMOÇÃO DE DEFICIENTES VISUAIS NO AMBIENTE URBANO UTILIZANDO SENSORES E VISÃO COMPUTACIONAL ATRAVÉS DO RASPBERRY PI.

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Eletrônica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheiro em Eletrônica.

Nota obtida: 10,0 (DEZ)

Aprovada em 09 / 05 /2022

Área de concentração: Sistemas Embarcados, Sensores e atuadores, Visão computacional

BANCA EXAMINADORA

Orientador: Mario Augusto Bessa de Figueiredo

Avaliador: Fábio de Sousa Cardoso

Avaliador: Daniel Guzmán Del Río

Manaus 2022

RESUMO

O presente trabalho teve como objetivo o desenvolvimento de um protótipo com capacidade de auxiliar a locomoção de deficientes visuais durante uma caminhada em um ambiente urbano. Inicialmente é apresentada a fundamentação teórica para o desenvolvimento do presente trabalho: conceito de deficiência visual, acessibilidade e de mobilidade, pesquisas sobre as dificuldades enfrentadas por pessoas com essa deficiência mesmo usando bengala ou cão-guia, estudo sobre o sensor ultrassônico que detecta obstáculos em ambientes adversos. Além disso, é explorada a linguagem de programação Python juntamente com a biblioteca OpenCV para o uso da tecnologia de Visão Computacional, visando utilizar o método de detecção de objetos, YOLO, e também realizar o cálculo da distância entre o deficiente e os objetos detectados. Posteriormente, são apresentadas as etapas e materiais necessários à construção do protótipo, seguida das ferramentas utilizadas, uma descrição detalhada dos experimentos, e implementação do que foi citado anteriormente. Como resultado desses experimentos, são exibidas as imagens onde são mostrados os objetos detectados e a distância que eles estão do deficiente visual. Por fim, os dados obtidos mostraram que o protótipo desenvolvido atende aos requisitos propostos ao informar ao deficiente visual quais são os obstáculos ao seu redor e a que distância eles se encontram dele, auxiliando-o significativamente durante uma caminhada em ambiente urbano.

Palavras-chave: deficiente visual. sensor ultrassônico. Visão Computacional. OpenCV. YOLO.

ABSTRACT

This work aimed to develop a prototype capable of assisting the locomotion of visually impaired people during a walk in an urban environment. Initially, the theoretical basis for the development of this work will be presented: concept of visual impairment, accessibility and mobility, research on the difficulties faced by people with this disability even using a walking stick or guide dog, study on the ultrasonic sensor that detects obstacles in adverse environments. In addition, the Python programming language is explored together with the OpenCV library for the use of Computer Vision technology, aiming to use the object detection method, YOLO, and also to calculate the distance between the visually impaired and the detected objects. Subsequently, the steps and materials necessary for the construction of the prototype are presented, followed by the tools used, a detailed description of the experiments, and implementation of what was mentioned above. As a result of these experiments, images are displayed showing the detected objects and the distance they are from the visually impaired. Finally, the data obtained showed that the prototype developed meets the proposed requirements by informing the visually impaired about the obstacles around them and how far they are from them, significantly helping them during a walk in an urban environment.

Keywords: visually impaired. ultrasonic sensor. Computer vision. OpenCV. YOLO.

LISTA DE FIGURAS

SUMÁRIO

INTRODUÇÃO

O trabalho que será apresentado tem como tema o desenvolvimento de protótipo de sistema para auxiliar a locomoção de deficientes visuais no ambiente urbano utilizando sensores e visão computacional através do Raspberry Pi.

O problema que inspirou a escolha desse tema foi que, as pessoas que não enxergam ou possuem algum grau de perda visual relatam dificuldades em trafegar em uma grande metrópole. Rampas inadequadas, passeios com buracos e desníveis, e travessias perigosas são dificuldades rotineiras na vida dessas pessoas. Mesmo com o uso de um cão-guia ou de uma bengala, alguns acabam se machucando e, diante da falta de acessibilidade, precisam da ajuda de outras pessoas para se locomoverem. Há a necessidade de desenvolvimento de sistema para auxílio de pessoas com deficiência visual, já que hoje não temos no mercado algo voltado, especificamente, para essa questão.

Dito isso, a hipótese levantada foi que, é possível auxiliar a locomoção de um deficiente visual no ambiente urbano utilizando sensores ultrassônicos do tipo HC-SR04, uma webcam, um mini alto-falante e motores vibratórios presos ao corpo do usuário para indicar onde há obstáculos, e com o Raspberry Pi é possível processar as imagens captadas pela câmera, monitorar os sensores e controlar os motores.

O objetivo é projetar e montar um protótipo em laboratório, localizado na Escola Superior de Tecnologia (EST) da Universidade do Estado do Amazonas (UEA), baseado no microcomputador Raspberry Pi, que receberá como entrada as imagens captadas pela webcam junto com os sensores ultrassônicos e realizará o processamento desses dados. Os resultados de saída serão disponibilizados pelo protótipo via periféricos, em forma de áudio e vibração.

Esse trabalho se justifica pois, ao deficiente visual é necessário conceder as mesmas oportunidades de frequentar ambientes que as pessoas consideradas normais frequentam. E com segurança. E com a utilização deste protótipo, será possível relatar ao usuário, em tempo real, os obstáculos e situações adversas ao seu redor, tornando mais fácil a sua locomoção em ambiente urbano.

Usarei os conhecimentos adquiridos durante o curso de Engenharia Eletrônica, principalmente das disciplinas: Microprocessadores e Microcontroladores, Sensores e Instrumentação Eletrônica, Sistemas Eletrônicos de Áudio, Processamento Digital de Sinais, Sistemas Eletrônicos de Imagens e Sistemas Operacionais Embarcados.

O trabalho se baseou em pesquisas sobre visão computacional, sensores eletrônicos, microcontroladores e microcomputadores com o intuito de entender como funcionam e como mesclar essas tecnologias.

1 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados alguns conceitos e uma revisão de literatura envolvendo todas as tecnologias necessárias para a elaboração do protótipo, que ajudarão a entender melhor como o problema em questão será resolvido.

1.1 DEFICIENTES VISUAIS

É considerada portadora de deficiência visual aquela pessoa que apresenta acuidade visual que seja igual ou menor que 0,1 no melhor olho, logo após a melhor correção, ou campo visual inferior a 20º na tabela de Snellen (CÂMARA DOS DEPUTADOS, 2009).

1.2 ACESSIBILIDADE E MOBILIDADE

> A acessibilidade, definida pela Lei 10.098, de 19/12/2000, como a possibilidade e condição de alcance para utilização, com segurança e autonomia, dos espaços, mobiliários e equipamentos urbanos, das edificações, dos transportes e dos sistemas e meios de comunicação, por pessoa portadora de deficiência ou com mobilidade reduzida (art 2°, I)," é uma importante garantia de que os cidadãos nessa condição possam exercer o seu direito de ir e vir e viver normalmente em sociedade. (CÂMARA DOS DEPUTADOS, 2009, p.53).

> A deficiência visual, em qualquer grau, compromete a capacidade da pessoa de se orientar e de se movimentar no espaço com segurança e independência. Na idade pré-escolar, quando a criança está desenvolvendo sua capacidade de socialização, isso prejudica (ou até mesmo impede) o conhecimento do mundo ao seu redor e seu relacionamento com outras pessoas. É um momento em que ela gosta de ter amigos, brincar junto e compartilhar os brinquedos. Se estiver impossibilitada de desempenhar esses papéis, ficará insatisfeita e isolada, e isso trará prejuízos a sua aprendizagem. Para alguns autores, a limitação na orientação e na mobilidade pode ser considerada o efeito mais grave da cegueira. (GIL, 2000, p.12).

1.3 CÃO-GUIA

> Estima- se que em todo o Brasil, há no máximo 80 animais como o Uiait. Ao mesmo tempo, cerca de cinco mil pessoas com deficiência visual estão na fila a espera de um cão. Por que há tão poucos animais como este no país? É que adestrar um cão guia exige mão- de- obra qualificada; leva tempo, de um a dois anos, e é caro. Custa, em média, R$ 30 mil. (GLOBO NEWS, 2011).

> A mobilidade urbana de São Paulo é um problema para pessoas com deficiência visual. Além de conviver com as calçadas irregulares, os deficientes visuais ainda passam por outros problemas como as pessoas que tiram a atenção do cão-guia e com o cancelamento de corridas de aplicativo devido à presença do animal. (G1, 2019).

1.4 TECNOLOGIA ASSISTIVA

> Tecnologia Assistiva é um termo ainda novo, utilizado para identificar todo o arsenal de Recursos e Serviços que contribuem para proporcionar ou ampliar habilidades funcionais de pessoas com deficiência e consequentemente promover Vida Independente e Inclusão. (SARTORETTO; BERSCH, 2021).

> Proporcionar à pessoa com deficiência maior independência, qualidade de vida e inclusão social, através da ampliação de sua comunicação, mobilidade, controle de seu ambiente, habilidades de seu aprendizado, trabalho e integração com a família, amigos e sociedade. (SARTORETTO; BERSCH, 2021).

1.5 MICROCOMPUTADOR

1.5.1 Raspberry Pi

O protótipo precisa de um dispositivo que tenha uma fácil conexão com sensores, com câmera, seja portátil, leve e que tenha um rápido processamento de imagem. Diante disso, um estudo sobre o microcomputador *Raspberry Pi 3 Modelo B+* foi necessário, pois ele cumpre todos os requisitos.

> O Raspberry Pi é um minicomputador, semelhante ao PC ou notebook que você tem em casa ou no trabalho. A diferença é que este dispositivo é compacto e possui todos os principais componentes de um computador numa pequena placa do tamanho de um cartão de crédito. (OLHAR DIGITAL, 2019).

1.6 MICROCONTROLADOR

1.6.1 Arduino Nano

Uma parte do protótipo requer apenas um microcontrolador que tenha uma fácil conexão com sensores e atuadores, e que também seja portátil e leve. Por esses motivos, foi realizada uma revisão sobre o funcionamento do Arduino Nano.

> O Arduino Nano é uma placa pequena, completa e compatível com protoboards baseada no ATmega328 (Arduino Nano 3.x). Ele tem mais ou menos a mesma funcionalidade do Arduino Duemilanove, mas em um pacote diferente. Falta apenas um conector de alimentação DC e funciona com um cabo USB Mini-B em vez de um padrão. (ARDUINO, 2021).

1.7 SENSORES

1.7.1 Sensor Ultrassônico

Para a detecção de obstáculos e a medição de distâncias, o sensor ultrassônico foi escolhido. Outro fator que influenciou essa escolha foi o fato de suas medições não serem afetadas pela transparência, poeira, sujeira ou vapores/gases presentes no ambiente.

> Os sensores ultrassônicos podem medir variáveis como enchimento e altura sem ter que entrar em contato com os elementos do meio, o que é uma grande vantagem quando comparado com outros tipos de sensores. Uma outra vantagem é que o sensor ultrassônico não possui sua operação prejudicada pela transparência, poeira, sujeira ou vapores/gases presentes no ambiente. Desde que o objeto reflita as ondas sonoras, é possível usar um sensor ultrassônico independentemente de seu acabamento superficial ou cor. Existem sensores que podem medir distâncias de dezenas de metros com ótima precisão. Devido a todas essas características, os sensores ultrassônicos são amplamente utilizados na indústria e em várias aplicações de robótica e automação. (BLOG ELETROGATE, 2021).

> O princípio de funcionamento do HC-SR04 consiste na emissão de sinais ultrassônicos pelo sensor e na leitura do sinal de retorno (reflexo/eco) desse mesmo sinal. A distância entre o sensor e o objeto que refletiu o sinal é calculada com base no tempo entre o envio e leitura de retorno. ”Sinais Ultrassônicos são ondas mecânicas com frequência acima de 40 KHz“. Como o ouvido humano só consegue identificar ondas mecânicas até a frequência de 20KHz, os sinais emitidos pelo sensor Ultrassônico não podem ser escutados por nós. (BLOG ELETROGATE, 2021).

1.7.2 Sensor Acelerômetro e Giroscópio MPU6050

Em alguns momentos, veremos que será necessário saber a inclinação do usuário em relação ao solo. Pesquisas sobre o funcionamento do *Acelerômetro e Giroscópio MPU6050* foram feitas, justamente porque este sensor fornece essa informação com alta precisão e possui uma fácil conexão com os microcontroladores.

> O Acelerômetro e Giroscópio 3 Eixos contém em um único chip um acelerômetro e um giroscópio tipo MEMS. São 3 eixos para o acelerômetro e 3 eixos para o giroscópio, sendo ao todo 6 graus de liberdade (6DOF). Não bastasse isso, esta placa GY-521 tem um sensor de temperatura embutido no CI MPU6050, permitindo medições entre -40 e +85 ºC. Possui alta precisão devido ao conversor analógico digital de 16-bits para cada canal. Portanto o sensor captura os canais X, Y e Z ao mesmo tempo. (FILIPEFLOP, 2022).

1.8 VISÃO COMPUTACIONAL

1.8.1 O que é?

Uma das tecnologias que têm grande importância neste trabalho é a Visão Computacional. Pois, através dela que conseguiremos não só detectar mas também identificar os obstáculos. Por essa razão, um estudo sobre essa tecnologia foi realizado, para conhecer mais a fundo o que é e como funciona.

> Visão computacional é a visão de máquinas, é possível obter informações de imagens, sejam elas astronômicas, microscópicas ou em tamanho natural, podemos utilizar algoritmos computacionais para descrever e analisar o conteúdo de qualquer imagem digitalizada. Essa prática é cada vez mais comum na indústria no controle de qualidade de processos e orientação de robôs, a visão computacional é capaz de realizar análises com precisão e velocidades que o olho humano não poderia alcançar, traz um novo leque de possibilidades como navegação de veículos autônomos, descoberta de novos planetas e análises biológicas em células. (PASSARELLI, 2017).

1.8.2 OpenCV

Essa biblioteca é uma ferramenta de suma importância, pois facilita bastante a vida de quem está trabalhando com visão computacional. Por esse motivo, uma pesquisa mais aprofundada sobre essa biblioteca se fez necessária.

> OpenCV é a principal biblioteca de código aberto para a visão computacional, processamento de imagem e aprendizagem de máquina, e agora apresenta a aceleração de GPU para operação em tempo real. OpenCV é liberado sob uma licença de BSD e daqui é livre para o uso acadêmico e comercial. Possui interfaces C++, C, Python e Java e suporta Windows, Linux, Mac OS, iOS e Android. OpenCV foi projetado para eficiência computacional e com um forte foco em aplicações em tempo real. Escrito em C/C++ otimizado, a biblioteca pode aproveitar o processamento multi-core. Adotado em todo o mundo, OpenCV tem mais de 47 mil pessoas da comunidade de usuários e número estimado de downloads superior a 6 milhões. O uso varia de arte interativa, a inspeção de minas, costura mapas na web ou através de robótica avançada. (PASSARELLI, 2017).

1.8.3 Detecção de objetos com YOLO

YOLO é uma técnica que é usada para a detecção de objetos, e sua característica principal é que, diferente de alguns algoritmos de detecção de objetos, como R-CNN ou Faster R-CNN, ele precisa olhar somente uma vez pela imagem para enviar os dados para a rede neural. Por isso o nome (You Only Look Once –

“Você só olha uma vez”). Dessa maneira o YOLO é capaz de uma velocidade de detecção muito maior do que outras técnicas, isso tudo mantendo uma ótima acurácia.

> Enquanto as técnicas mais precisas levavam aproximadamente 0.5 segundos (ou mais) para processar uma imagem, o YOLO conseguia detectar com o mesmo nível de precisão em menos de 0.05 segundos, o que permitiu ser utilizado em aplicações de tempo real, sendo capaz de rodar até em uma taxa de 30 frames por segundo. Outro motivo do sucesso do YOLO é o fato de ser totalmente código aberto e livre de licenças de uso. Ou seja, desde o código fonte, até a arquitetura da rede neural e os pesos pré-treinados, tudo isso pode ser usado por qualquer um e de qualquer forma. (EXPERT ACADEMY, 2022).

1.9 PYTHON

Para utilizar a biblioteca OpenCV com mais tranquilidade e rapidez, Python é a linguagem de programação mais indicada, pois é de alto nível, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Dito isso, uma revisão na documentação dessa linguagem de programação foi realizada.

> Python é poderoso... e rápido; joga bem com os outros; corre em todos os lugares; é amigável e fácil de aprender; está aberto. Essas são algumas das razões pelas quais as pessoas que usam Python preferem não usar mais nada. (PYTHON, 2022)

2 METODOLOGIA UTILIZADA

Será feita uma revisão de literatura nas áreas de sensores e atuadores, microcontroladores e microprocessadores, visão computacional utilizando a biblioteca OpenCV, microcomputador Raspberry Pi e linguagem de programação Python.

A construção do protótipo será separada em quatro etapas:

- A primeira etapa será destinada à aquisição do material necessário para a montagem, sabendo que será utilizado a placa de desenvolvimento do Raspberry Pi ligada diretamente à uma webcam, aos sensores ultrassônicos e aos motores vibratórios. Para realizar a montagem, serão feitas pesquisas para saber como posicionar os componentes junto ao corpo do usuário de tal forma que não haja desconforto para o mesmo.
- A segunda etapa será responsável pela instalação da biblioteca OpenCV no Raspberry Pi, com o intuito de conhecer e escolher os melhores modelos fornecidos na área da visão computacional, que consigam detectar e analisar obstáculos e situações adversas do ambiente urbano que possam ser uma barreira para a mobilidade de pessoas com deficiência visual.
- A terceira etapa será para desenvolver a lógica de programação que irá receber os dados dos sensores ultrassônicos e detectar obstáculos próximos à pessoa com deficiência visual e, em seguida, acionar os motores vibratórios junto com um mini alto-falante para que o usuário saiba onde se encontram os obstáculos.
- Na quarta e última etapa, será realizada a junção das lógicas de detecção de obstáculos, tanto através da visão computacional quanto através dos sensores ultrassônicos. Feito essa junção, serão realizados testes controlados em laboratório. Após os ajustes necessários, serão realizados testes em ambiente urbano. Os testes terão o objetivo de relatar ao usuário o cenário à sua volta e auxiliá-lo para que ele se locomova sem sofrer qualquer tipo de acidente.

3 IMPLEMENTAÇÃO DO PROJETO

Neste capítulo veremos o que realmente foi feito na pesquisa. Para isso, será feita uma descrição detalhada do que foi realizado em cada etapa, com foco no objetivo projetado.

3.1 AQUISIÇÃO DO MATERIAL

Vejamos a lista dos materiais necessários para a montagem do protótipo:

- 1 - Raspberry Pi 3 Model B+;
- 1 - Arduino Nano;
- 4 - Sensor Ultrassônico HC-SR04;
- 1 - Sensor Acelerômetro e Giroscópio MPU6050;
- 1 - Webcam HD Logitech C270;
- 2 - Placa 10x10 Universal Padrão Perfurada Ilhada Fibra Fenolite;
- 3 - Motor de Vibração Vibracall 1027;
- 6 - Bateria Li-ion 18650 9800mah 4.2v - Recarregável Original;
- 1 - Regulador de Tensão LM2596 Conversor DC-DC Step Down.
- 1 - Módulo Amplificador de Áudio Com LM386
- 2 - Mini Alto Falante 8 Ohm 0.5W
- 1 - Ferro de solda
- 1 - Tubo de solda (estanho)

3.2 POSICIONAMENTO E COMPORTAMENTO DO PROTÓTIPO

Foi feita uma pesquisa para saber onde posicionar o protótipo junto ao corpo do usuário de tal forma que não haja desconforto para o mesmo. Essa pesquisa se baseou no feedback de alguns deficientes visuais. A eles foi perguntado quais seriam os melhores locais para acoplar o protótipo. A maioria respondeu que, para uma caminhada tranquila, o melhor local seria na bengala, e que se o protótipo tivesse um molde semelhante ao de um óculos, seria bem interessante. Diante do resultado dessa pesquisa, foi definido que o protótipo se dividiria em duas partes.

3.2.1 Primeira parte do protótipo

A primeira parte do protótipo terá um molde semelhante a um óculos de realidade virtual e será responsável pela detecção e identificação de obstáculos, através de um sensor ultrassônico e uma webcam respectivamente, Também será responsável pelo cálculo da distância até os mesmos. O fato de estar fixada ao rosto do usuário, esta parte acompanhará o movimento de sua cabeça, isso significa que, o próprio usuário escolherá a direção que deseja ter informações, pois o sensor ultrassônico e a webcam sempre estarão na direção em que o seu nariz estiver apontado. Essa parte será controlada pelo microcomputador Raspberry Pi 3 Model B+.

3.2.2 Segunda parte do protótipo

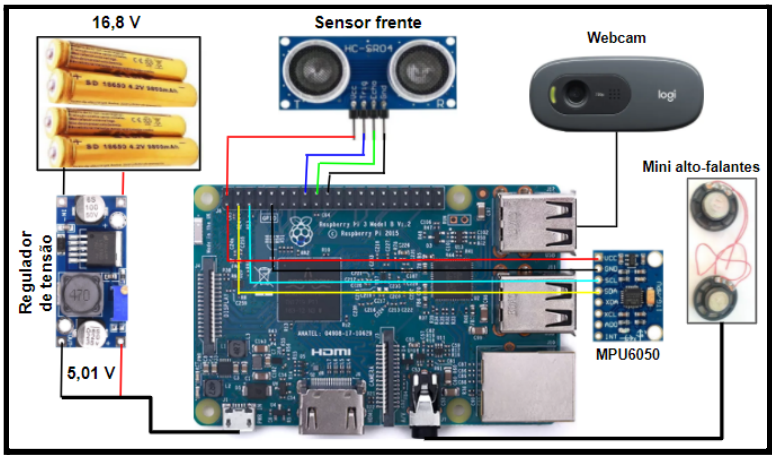
A segunda parte do protótipo ficará fixada na bengala do deficiente visual e terá o comportamento semelhante a um sonar, que detecta e localiza obstáculos próximos ao usuário e que estejam na linha da sua cintura. Será composta de três sensores ultrassônicos, um ficará detectando obstáculos à esquerda, outro detectando obstáculos à frente e outro detectando obstáculos à direita. Também serão usados três motores vibratórios, um motor vibrará se for detectado obstáculo à esquerda, outro vibrará se for detectado obstáculo à frente e outro vibrará se for detectado obstáculo à direita. Isso significa que o usuário saberá a localização dos obstáculos e poderá seguir uma rota alternativa, evitando assim uma possível colisão. Essa parte será controlada pelo microcontrolador Arduino Nano.

3.3 CONFECÇÃO E IMPRESSÃO 3D DO PROTÓTIPO

Como o protótipo se divide em duas partes, cada parte foi confeccionada separadamente. Os circuitos eletrônicos e as modelagens 3D foram desenvolvidos de forma personalizada, visando ter uma boa estética e ocupar o menor espaço possível. Os circuitos foram montados sobre placas de fenolite e as modelagens 3D foram desenvolvidas através do software AutoCAD 2015.

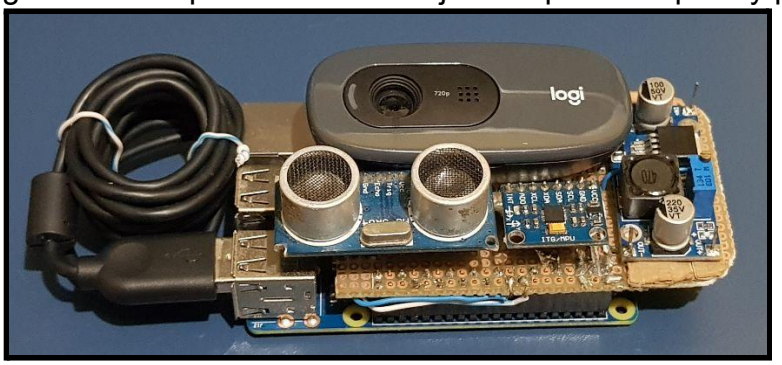
A Primeira parte do protótipo conterá uma Webcam HD Logitech C270, um Sensor Ultrassônico HC-SR04, um Sensor Acelerômetro e Giroscópio MPU6050, quatro Baterias Li-ion 18650 9800mah 4.2v, um Regulador de Tensão LM2596 Conversor DC-DC Step Down e uma placa Raspberry Pi 3 Model B+.

Figura 1 - Conexões elétricas da primeira parte do protótipo



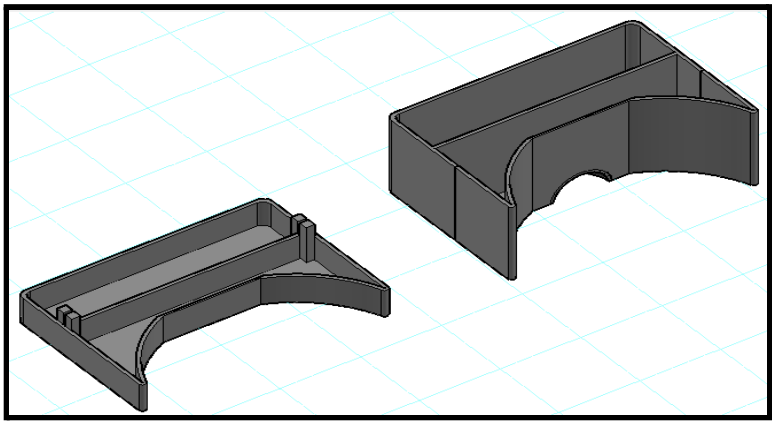
Fonte: O próprio autor

Figura 2 - Componentes fixados junto à placa raspberry pi



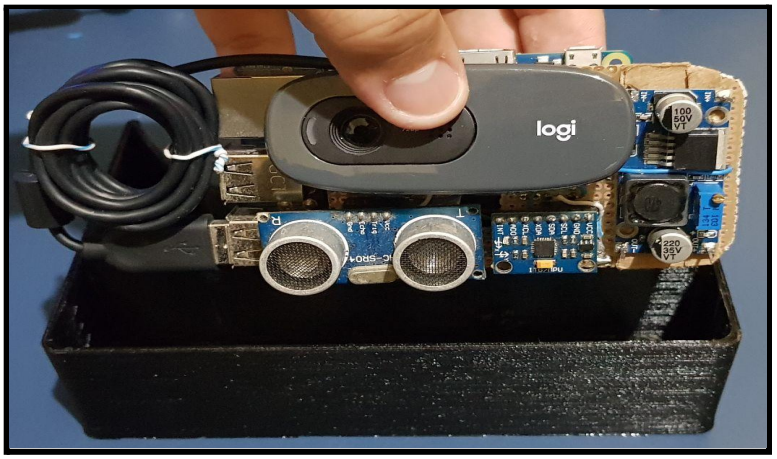
Fonte: O próprio autor

Figura 3 - Case em 3D para a primeira parte do protótipo



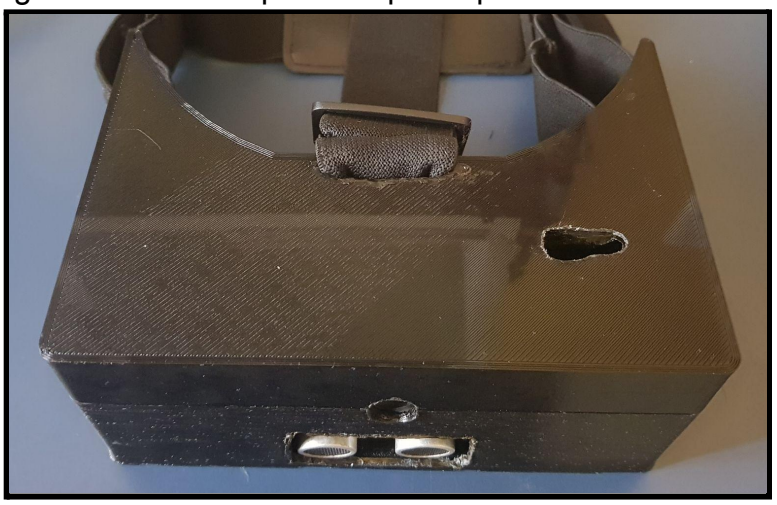
Fonte: O próprio autor

Figura 4 - Case em 3D junto ao protótipo



Fonte: O próprio autor

Figura 5 - Primeira parte do protótipo totalmente vestível



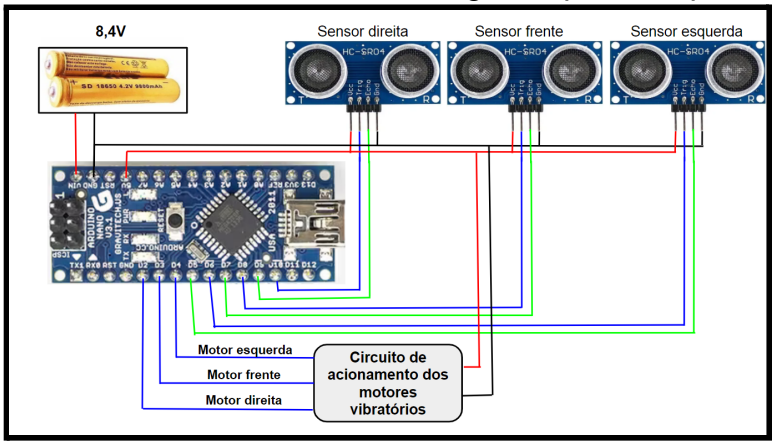
Fonte: O próprio autor

Figura 6 - Fixação do protótipo junto ao usuário

Fonte: O próprio autor

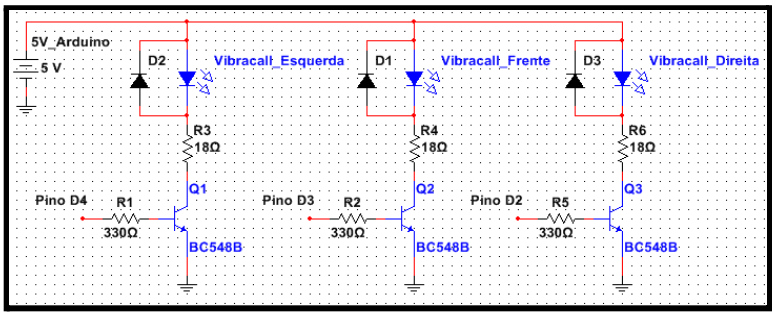
A segunda parte do protótipo conterá três Sensores Ultrassônicos HC-SR04, três Motores de Vibração Vibracall 1027, duas Baterias Li-ion 18650 9800mah 4.2v, uma placa Arduino Nano e um circuito de acionamento dos motores de vibração.

Figura 7 - Conexões elétricas da segunda parte do protótipo



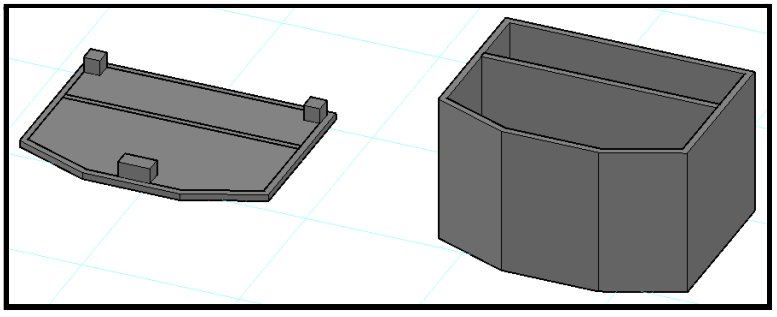
Fonte: O próprio autor

Figura 8 - Circuito de acionamento dos motores vibratórios



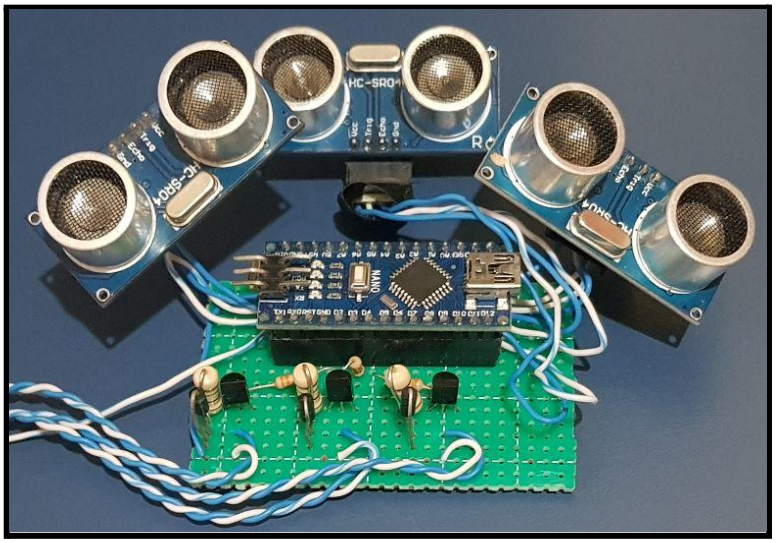
Fonte: O próprio autor

Figura 9 - Case em 3D para a segunda parte do protótipo



Fonte: O próprio autor

Figura 10 - Componentes fixados junto à placa arduino nano



Fonte: O próprio autor

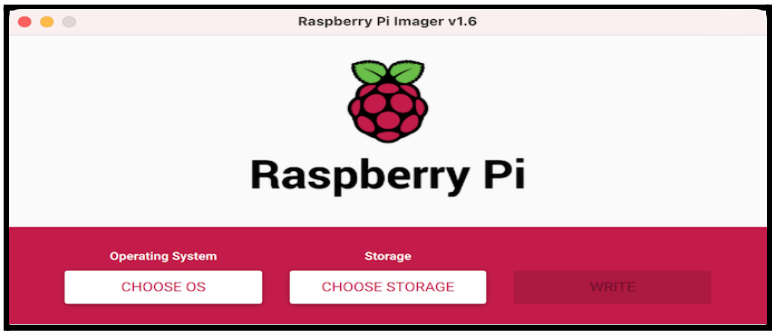
3.4 DETECÇÃO DE OBJETOS E CÁLCULO DE DISTÂNCIA ATRAVÉS DE VISÃO COMPUTACIONAL

A detecção de objetos através de visão computacional só ocorre na primeira parte do protótipo. Tendo isso em vista, veremos a seguir todos os passos realizados.

3.4.1 Preparação do ambiente no raspberry pi

O desenvolvimento da primeira parte do protótipo se baseará na placa Raspberry Pi 3 Model B+, então será necessário configurá-la. O Raspberry Pi precisa de um sistema operacional para funcionar. Atualmente, o sistema operacional oficial suportado é o ‘Raspberry Pi OS’. As placas Raspberry Pi não possuem HD, por isso, vamos instalar o sistema operacional e armazenar os arquivos no cartão microSD de 64GB. Foi utilizado o ‘Raspberry Pi Imager’ para instalar esse sistema operacional. Para evitar qualquer tipo de problema, foi seguido o tutorial de instalação fornecido pelo próprio site da raspberry, https://www.raspberrypi.com/software/.

Figura 11 - Interface do instalador ‘Raspberry Pi Imager’



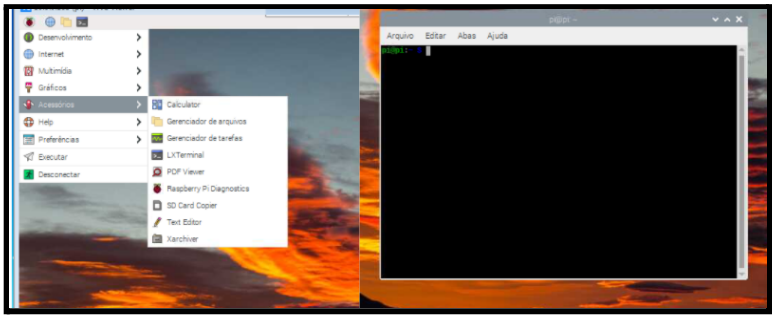
Fonte: O próprio autor

Após a instalação do sistema operacional no cartão microSD, colocaremos o cartão na parte de trás da placa raspberry. Conectaremos também ao raspberry um monitor, um mouse e um teclado. Ao ligar na tomada, depois de alguns segundos a sua tela principal será apresentada. Logo em seguida, o terminal de comandos foi aberto.

Figura 12 - Tela inicial do sistema operacional ‘Raspberry Pi OS’

Fonte: O próprio autor

Figura 13 - Terminal do Raspberry Pi OS



Fonte: O próprio autor

3.4.2 Instalação da biblioteca opencv

Antes de instalar qualquer biblioteca no raspberry, foi feita a atualização do software da placa. Para isso, foi preciso conectar a placa em uma rede wifi e logo em seguida, via terminal, executar os seguinte comandos.

sudo apt-get update

sudo apt-get upgrade

sudo reboot

Após o último comando, a placa será reiniciada e estará pronta para uso.

Para a instalação da biblioteca OpenCV, foi preciso executar apenas um comando no terminal.

pip3 install opencv-python

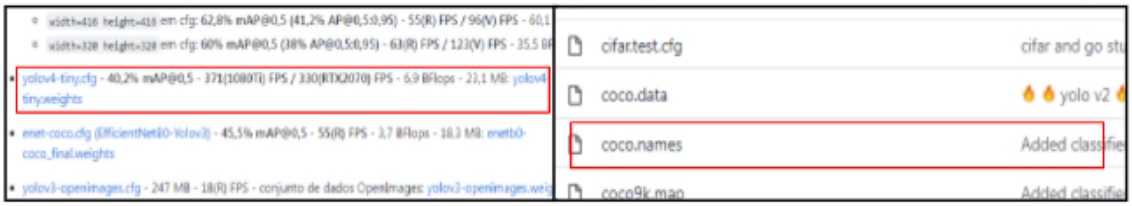
3.4.3 Escolha de um método de detecção de objetos

No mundo da visão computacional, existem vários métodos de detecção de objetos. Alguns dos melhores métodos necessitam de um computador com um nível de processamento muito alto, e isso afeta diretamente a velocidade de detecção. Como já foi dito, utilizaremos o microcomputador Raspberry Pi 3 Model B+. Seu nível de processamento é menor do que o de um computador normal. Diante disso, foi escolhido um método de detecção YOLO, pois, diferente de outros métodos, como R-CNN ou Faster R-CNN, ele só precisa olhar a imagem uma única vez para enviar para a rede neural. Por isso o nome (You Only Look Once – Você só olha uma vez). Esse método é capaz de uma velocidade na detecção muito maior do que as técnicas concorrentes, é mais leve e consegue manter uma ótima acurácia.

3.4.4 Detecção de objetos com darknet yolo v4 utilizando um modelo pré-treinado

Teremos que baixar três arquivos diretamente do Git Darknet Yolo: https://github.com/AlexeyAB/darknet. O primeiro é o arquivo *yolov4-tiny.config*, que contém todas as configurações, o segundo é o arquivo *yolov4-tiny.weights*, que contém os pesos pré definidos, e por último o arquivo *coco.names*, que contém a lista de todas as classes (objetos) que o modelo consegue detectar. Esse último arquivo se encontra na pasta cfg do Git.

Figura 14 - Arquivos do modelo pré-treinado do yolo v4



Fonte: O próprio autor

Utilizaremos a linguagem de programação python, através do Thonny Python IDE que já vem instalado por padrão, para rodar o primeiro programa que realizará as primeiras detecções, e iremos analisar qual o tipo de informação que o modelo fornece. Esse primeiro programa se chamará *teste_com_YOLO.py.* Será preciso armazenar esses três arquivos, que foram baixados, na mesma pasta em que iremos salvar o programa de teste. Veremos agora um código de teste e a sua saída.

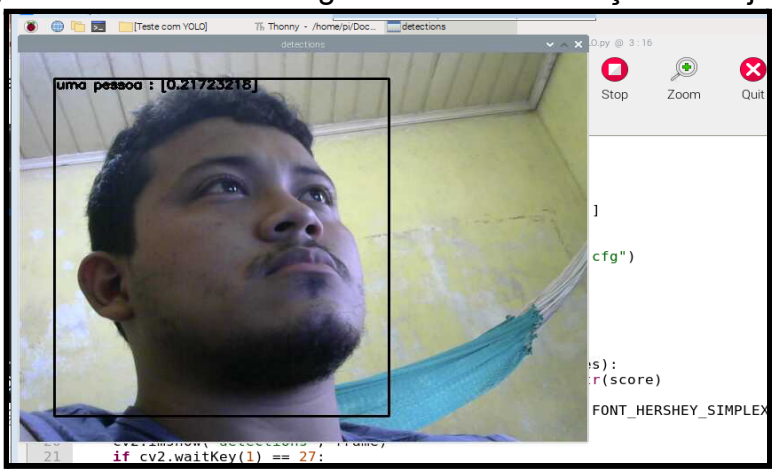
Figura 15 - Código de teste de detecção de objetos

teste_com_YOLO.py

```
import cv2

class_name = []
with open("coco.names", "r") as f:
    class_name = [cname.strip() for cname in f.readlines()]

cap = cv2.VideoCapture(0)
net = cv2.dnn.readNet("yolov4-tiny.weights", "yolov4-tiny.cfg")
model = cv2.dnn_DetectionModel(net)
model.setInputParams(size=(416, 416), scale=1/255)

while True:
    _, frame = cap.read()
    classes, scores, boxes = model.detect(frame, 0.1, 0.2)
    for (classid, score, box) in zip(classes, scores, boxes):
        label = str(class_name[int(classid)]) + ' : ' + str(score)
        cv2.rectangle(frame, box, (0, 0, 0), 2)
        cv2.putText(frame,label,(box[0]+2, box[1]+12),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,0,0),2)

    cv2.imshow("detections", frame)
    if cv2.waitKey(1) == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

Fonte: O próprio autor

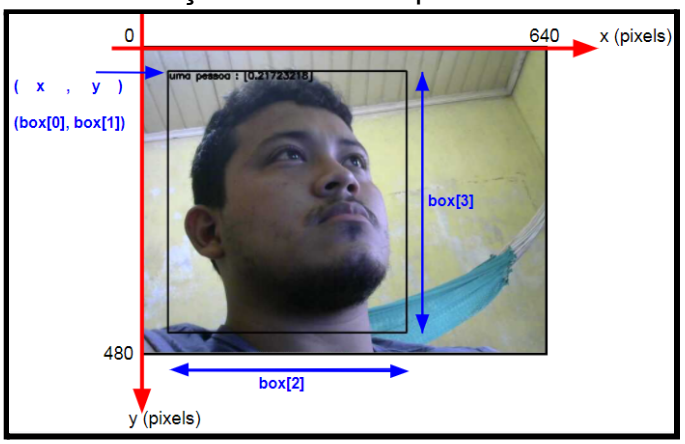
Figura 16 - Saída do código de teste de detecção de objetos



Fonte: O próprio autor

Analisando a saída do programa minuciosamente, podemos obter várias informações. A imagem é plotada em um plano cartesiano, onde os primeiros pixels estão localizados no canto superior esquerdo. A imagem tem uma resolução de 480p (480 pixels na vertical e 640 pixels na horizontal). O modelo de detecção retorna o nome do objeto detectado, a porcentagem de certeza da detecção e as coordenadas do retângulo desenhado ao redor do objeto, indicando onde o objeto está localizado. As coordenadas são representadas por *box[0]*, *box[1]*, *box[2]* e *box[3]*.

Figura 17 - Informações fornecidas pelo modelo de detecção

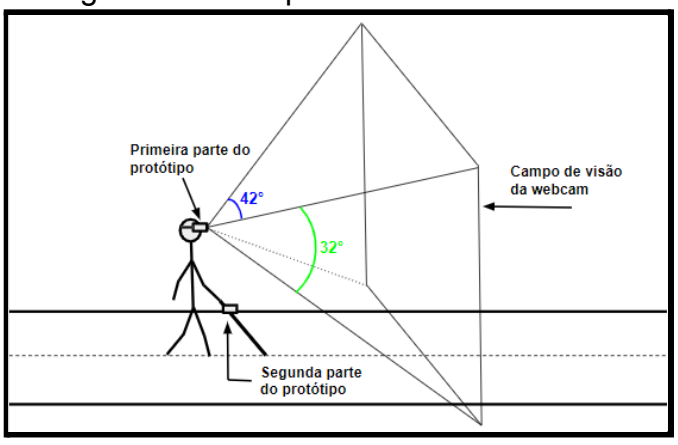


Fonte: O próprio autor

3.4.5 Localização do objeto detectado

Após alguns testes, identificou-se os ângulos de visão da webcam. O ângulo de visão vertical da webcam é aproximadamente 32°, e o ângulo de visão horizontal da webcam é aproximadamente 42°. Com essas informações, podemos estimar o campo de visão de suas capturas. Podemos entender melhor com a ilustração abaixo.

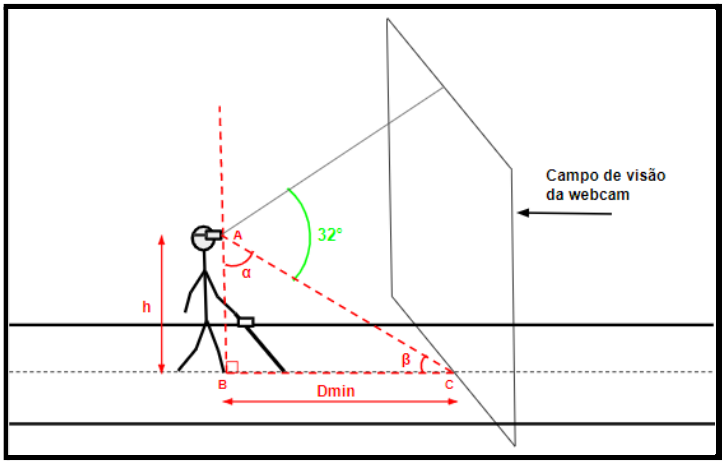
Figura 18 - Campo de visão da webcam



Fonte: O próprio autor

Através da trigonometria conseguimos calcular a que distância a webcam começa a capturar as imagens. Essa distância será chamada de *Dmin* (Distância Mínima Detectável). Vejamos abaixo uma ilustração, onde a variável *h* representa a altura que o primeiro protótipo está em relação ao solo. Dessa maneira chegamos a primeira equação.

Figura 19 - Ilustração da Distância Mínima Detectável

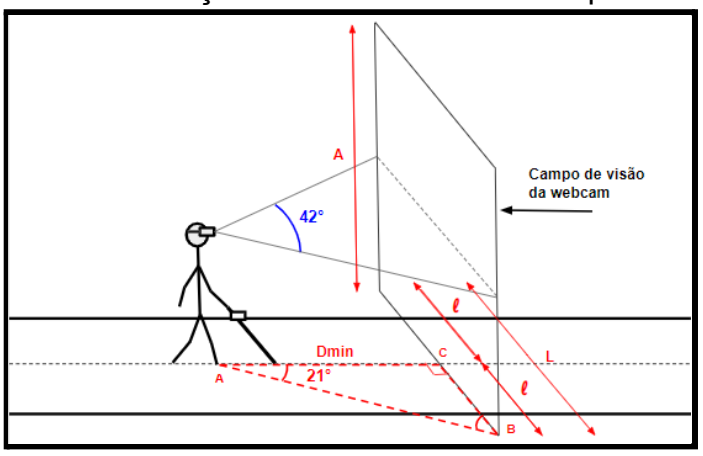


Fonte: O próprio autor

$$Dmin = h \,.\, tan(\alpha) \qquad \text{(Equação 1)}$$

Após o cálculo da distância mínima detectável, conseguimos calcular a largura (*L*) e a altura (*A*) do campo de visão da webcam. Para isso, assumimos que a razão entre o máximo pixel horizontal (640 pixels) e o máximo pixel vertical (480 pixels) é igual a razão entre a largura do campo de visão (*L*) e a altura do campo de visão (*A*). Vejamos abaixo uma ilustração e o desenvolvimento das equações.

Figura 20 - Ilustração das dimensões do campo de visão



Fonte: O próprio autor

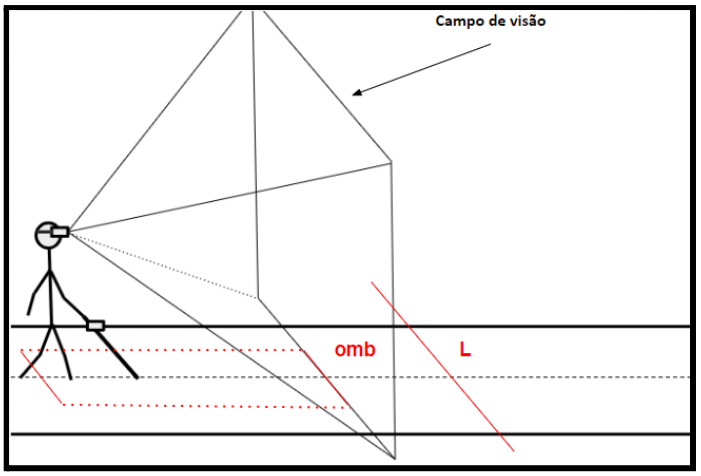
$$l = Dmin\,.\,tan(21°) \quad \text{(Equação 2)}$$

$$L = 2\,.\,l \quad \text{(Equação 3)}$$

$$A = L\,.\,(480/640) \quad \text{(Equação 4)}$$

Como o objeto já foi detectado, precisamos agora saber a sua localização em relação ao usuário. Mas antes disso, analisaremos a imagem capturada e delimitaremos as áreas que iremos considerar como esquerda, frente e direita. Para isso, vamos admitir que a largura do ombro do usuário mede metade de sua altura, dessa forma podemos calcular quantos pixels essa medida representa na imagem através da regra de três simples, considerando que a imagem tem 640 pixels na horizontal. Abaixo veremos uma ilustração disso e as equações geradas.

Figura 21 - Medida do ombro em pixel



Fonte: O próprio autor

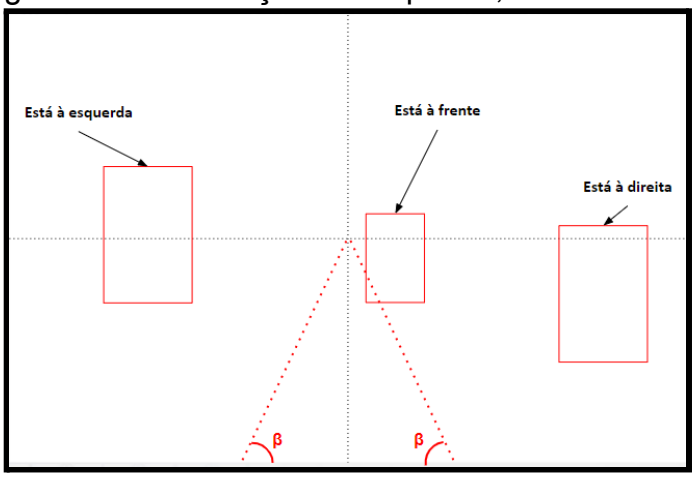
$$l = Dmin\,.\,tan(21°) \quad \text{(Equação 5)}$$

$$omb = h/2 \quad \text{(Equação 6)}$$

$$Pomb = 640 * omb/L \quad \text{(Equação 7)}$$

Sabemos que quanto mais longe o objeto estiver da câmera, menor ele aparecerá na imagem. E isso irá acontecer com a medida do nosso ombro se for projetada para mais longe. Se projetarmos nosso ombro até o meio da imagem, essa medida tenderá ao horizonte. Para representar isso, traçamos duas retas até o meio da imagem com uma inclinação β. Dessa forma, conseguimos delimitar as áreas que representarão a esquerda, a frente e a direita.

Figura 22 - Delimitação da esquerda, frente e direita



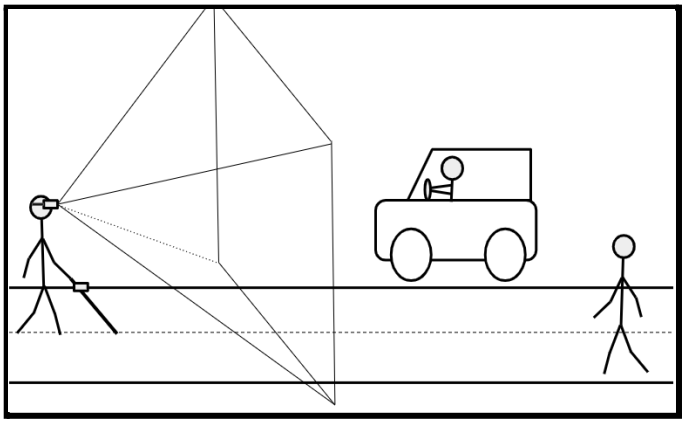
Fonte: O próprio autor

$$\beta = arctan(2 * h/omb) \qquad \text{(Equação 8)}$$

3.4.6 Cálculo da distância até o objeto detectado

Agora iremos simular uma situação real do dia a dia no ambiente urbano. Na ilustração abaixo, veremos um deficiente visual utilizando a primeira parte do protótipo, e à sua frente há um carro à esquerda e uma pessoa à direita.

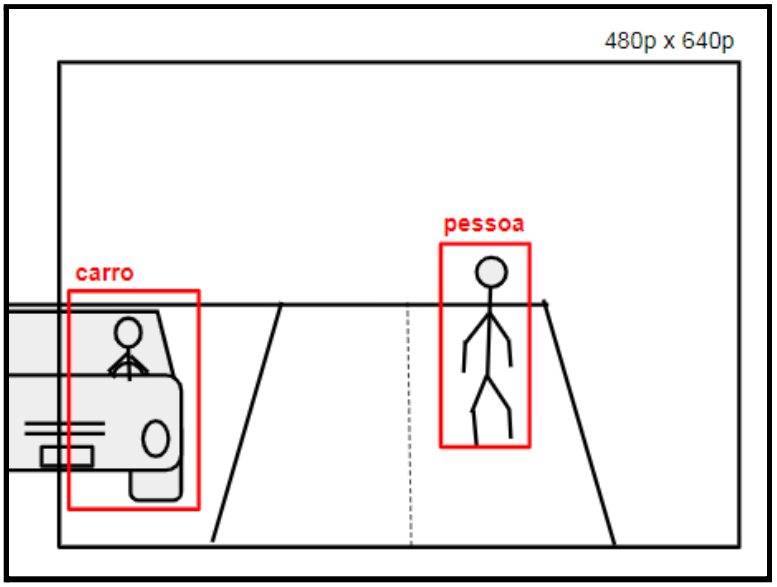
Figura 23 - Simulação de uma situação real do dia a dia



Fonte: O próprio autor

Logo a seguir, veremos uma simulação da captura da webcam, onde a janela plotada tem uma resolução de 480p. Além da captura da imagem, assumimos que o protótipo realize a detecção do carro e da pessoa. Lembrando que o modelo de detecção informa o nome do objeto, informa a porcentagem de certeza e as coordenadas do retângulo desenhado ao redor do objeto detectado.

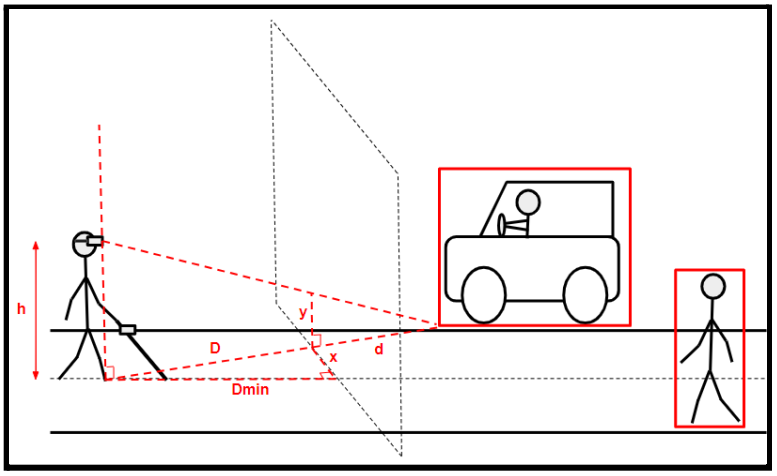
Figura 24 - Simulação da detecção feita pelo protótipo



Fonte: O próprio autor

A próxima ilustração pretende tornar mais fácil o raciocínio de como será feito o cálculo da distância entre o deficiente visual e os objetos detectados.

Figura 25 - Ilustração da distância até o objeto detectado



Fonte: O próprio autor

Assumimos que a menor distância entre eles é uma reta, a distância até o objeto é a soma *D* + *d*. Dito isso, iremos calcular primeiro o valor de *D*.

$$D = \sqrt{Dmin^2 + x^2} \quad \text{(Equação 9)}$$

Agora iremos calcular o valor de *d* usando a semelhança de triângulos.

$$\frac{h}{y} = \frac{D+d}{d} \quad \text{(Equação 10)}$$

$$\frac{h}{y} = \frac{D}{d} + 1 \quad \text{(Equação 11)}$$

$$d = \frac{D \,.\, y}{h-y} \quad \text{(Equação 12)}$$

Após o cálculo das variáveis *D* e *d*, somos capazes de chegar a uma única equação, que nos dará a distância entre o deficiente visual e os obstáculos detectados. Vamos chamar essa distância de *Dobj*.

$$Dobj = D + d \quad \text{(Equação 13)}$$

$$Dobj = D + \frac{D \,.\, y}{h-y} \quad \text{(Equação 14)}$$

$$Dobj = D \,.\, (1 + \frac{y}{h-y}) \quad \text{(Equação 15)}$$

$$Dobj = D \,.\, (\frac{h-y+y}{h-y}) \quad \text{(Equação 16)}$$

$$Dobj = D \,.\, (\frac{h}{h-y}) \quad \text{(Equação 17)}$$

A Equação 13 nos dá a medida de distância em metros. Precisamos converter essa medida para a quantidade de passos correspondente, pois a única unidade de medida que os deficientes visuais conseguem contar com facilidade é a quantidade de passos. Admitindo que um passo mede aproximadamente 0,82 metros, podemos chegar a uma nova equação.

$$Dobj = (D \,.\, (\frac{h}{h-y})) \,/\, 0,82 \quad \text{(Equação 18)}$$

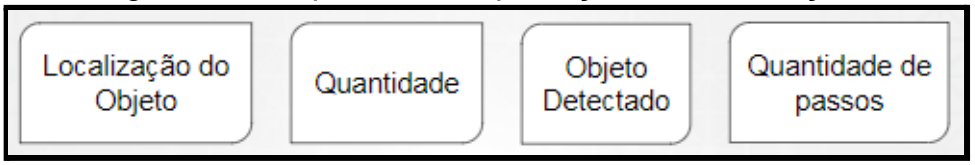
3.4.6 Informar o usuário através de áudio

Feita a detecção, a localização e o cálculo da distância até o objeto, iremos passar essas informações para o deficiente visual através de áudios. Para isso, foi feita uma gravação de áudio para objeto que o modelo de detecção consegue detectar. Através da Equação 8, sabemos a distância, em passos, até o objeto e também se o objeto está a uma distância menor do que *Dmin* (Mínima Distância Detectável).

Para informar que há objetos perto, foram feitas três gravações de áudio contendo as respectivas frases: “logo a frente tem”, “logo a esquerda tem” e ”logo a direita tem”. Para informar que há objetos longe, foram feitas três gravações de áudio contendo as respectivas frases: “a frente tem”, “a esquerda tem” e ”a direita tem”. Também foram gravados as frases: “à 1 passo”, “à 2 passos”, “à 3 passos”, …, até “à 30 passos”. Se a distância até o objeto for maior que 30 passos, o protótipo informará a seguinte frase “à mais de 30 passos”.

Na figura abaixo, veremos como os áudios são organizados e reproduzidos após as detecções dos objetos.

Figura 26 - Sequência de reprodução das informações



Fonte: O próprio autor

3.5 LÓGICA DE DETECÇÃO DE OBSTÁCULO COM ULTRASSÔNICO

Como mencionado anteriormente, o protótipo se divide em duas partes, e as duas partes utilizarão o sensor ultrassônico para detectar obstáculos.

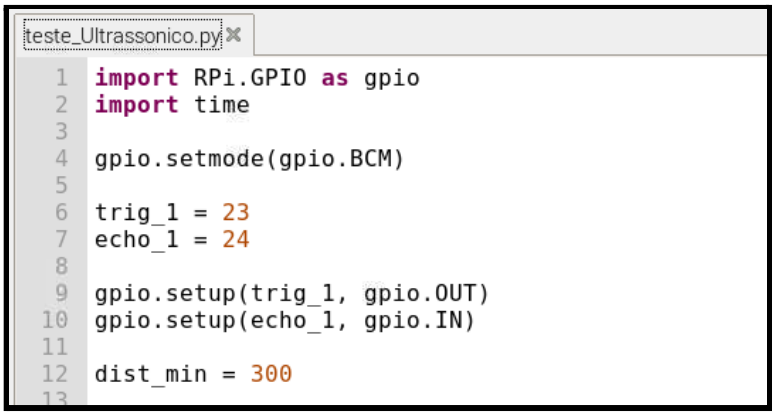
3.5.1 Ultrassônico na primeira parte do protótipo

Aqui utilizaremos apenas um sensor ultrassônico, que ficará centralizado e apontando sempre para a frente. Usaremos o raspberry pi para analisar essa leitura. Como.o alcance do sensor é de 400 centímetros, definiremos uma distância de

detecção segura, essa distância será de 300 centímetros. Com essa distância o deficiente visual terá mais tempo para analisar a situação e desviar do obstáculo.

Logo abaixo veremos as bibliotecas importadas e a definição dos pinos utilizados pelo sensor. Também podemos ver que a variável *dist_min* armazena a distância de detecção.

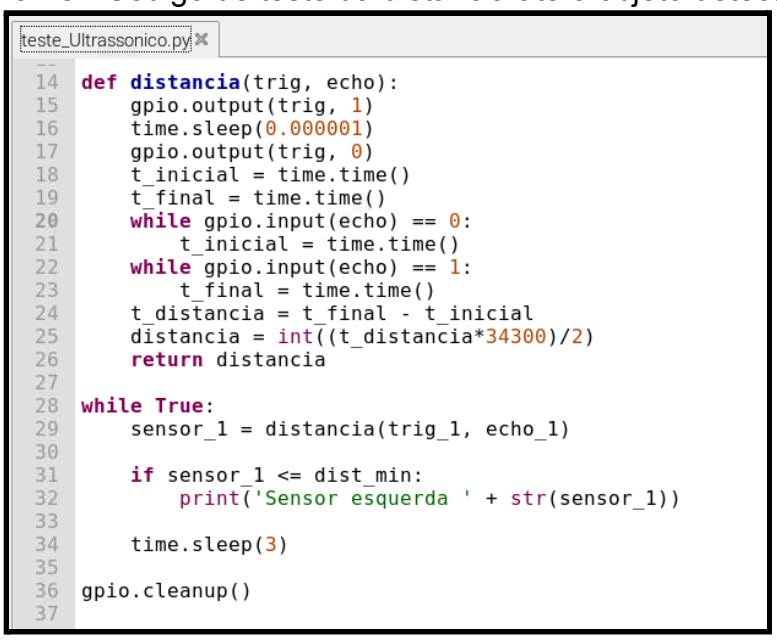
Figura 27 - Código de teste da distância até o objeto detectado



Fonte: O próprio autor

Aqui veremos a função *distancia()*, que é responsável pelo cálculo da distância até o obstáculo.

Figura 28 - Código de teste da distância até o objeto detectado



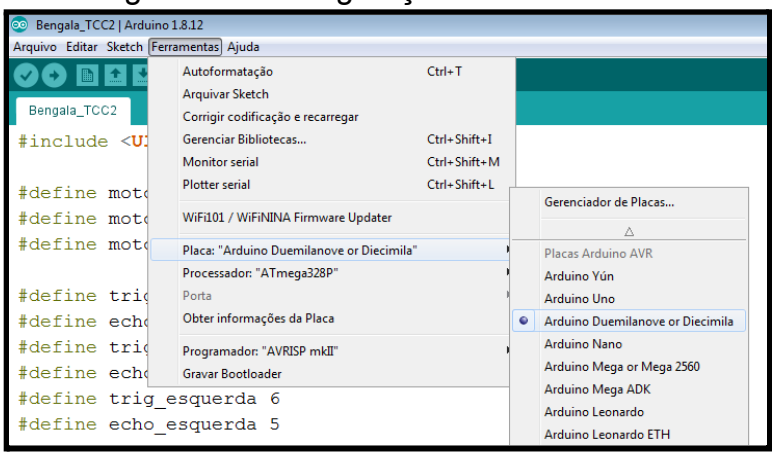
Fonte: O próprio autor

3.5.2 Ultrassônico na segunda parte do protótipo

Nessa parte do protótipo não é necessário um alto nível de processamento, então utilizaremos o Arduino Nano. O código será desenvolvido através da Arduino IDE, usaremos três sensores ultrassônicos e três motores vibratórios que ficarão fixados na bengala do usuário.

Primeiro foi feita uma configuração necessária dentro da Arduino IDE.

Figura 29 - Configuração da Arduino IDE



Fonte: O próprio autor

Aqui veremos que foi utilizada a biblioteca *Ultrassonic.h*. Também foram definidos os pinos utilizados pelos sensores e pelos motores. Também podemos ver que a variável *dist_detec* armazena a distância de detecção, só que adotaremos outra medida aqui, definiremos uma distância de 150 cm. Na parte de *setup()* foram definidos os pinos dos motores como saída (OUTPUT).

Figura 30 - Código fonte da segunda parte do protótipo

```
#include <Ultrasonic.h>

#define motor_direita 2
#define motor_frente 3
#define motor_esquerda 4

#define trig_direita 10
#define echo_direita 9
#define trig_frente 8
#define echo_frente 7
#define trig_esquerda 6
#define echo_esquerda 5

Ultrasonic ultrasonic_direita(trig_direita, echo_direita);
Ultrasonic ultrasonic_frente(trig_frente, echo_frente);
Ultrasonic ultrasonic_esquerda(trig_esquerda, echo_esquerda);

int dist_detec = 150;
unsigned long time_direita=0, time_frente=0, time_esquerda=0;
void setup() {
  Serial.begin(9600);
  Serial.println("Lendo dados do sensor...");
  pinMode(motor_direita, OUTPUT);
  pinMode(motor_frente, OUTPUT);
  pinMode(motor_esquerda, OUTPUT);
  digitalWrite(motor_direita, LOW);
  digitalWrite(motor_frente, LOW);
  digitalWrite(motor_esquerda, LOW);
}
```

Fonte: O próprio autor

Abaixo foram criadas as variáveis que receberão as medidas em centímetros. Dentro do *loop()* as medidas são realizadas e guardadas em suas respectivas variáveis. Logo em seguida as medidas são impressas no *serial monitor*.

Figura 31 - Código fonte da segunda parte do protótipo

```
float cm_direita=0, cm_frente=0, cm_esquerda=0;
void loop() {
  cm_direita = ultrasonic_direita.convert(ultrasonic_direita.timing(), Ultrasonic::CM);
  delay(10);
  cm_frente = ultrasonic_frente.convert(ultrasonic_frente.timing(), Ultrasonic::CM);
  delay(10);
  cm_esquerda = ultrasonic_esquerda.convert(ultrasonic_esquerda.timing(), Ultrasonic::CM);
  delay(10);
  Serial.print("Sensor 1: ");
  Serial.println(cm_direita);
  Serial.print("Sensor 2: ");
  Serial.println(cm_frente);
  Serial.print("Sensor 3: ");
  Serial.println(cm_esquerda);
  Serial.println();
```

Fonte: O próprio autor

Aqui as medidas lidas pelos sensores são comparadas com a medida de detecção segura, e se as medidas lidas forem menores, os motores respectivos são acionados durante 0,5 segundos, e só estarão disponíveis novamente após 1 segundo.

Figura 32 - Código fonte da segunda parte do protótipo

```
  if((cm_direita <= dist_detec) && (time_direita == 0)){
    digitalWrite(motor_direita, HIGH);
    time_direita = millis();
  }
  else if( (millis()-time_direita) >= 500 ){
    digitalWrite(motor_direita, LOW);
    if ( (millis()-time_direita) >= 1500 ){
      time_direita = 0;
    }
  }
  if((cm_frente <= dist_detec) && (time_frente == 0)){
    digitalWrite(motor_frente, HIGH);
    time_frente = millis();
  }
  else if( (millis()-time_frente) >= 500 ){
    digitalWrite(motor_frente, LOW);
    if ( (millis()-time_frente) >= 1500 ){
      time_frente = 0;
    }
  }
  if((cm_esquerda <= dist_detec) && (time_esquerda == 0)){
    digitalWrite(motor_esquerda, HIGH);
    time_esquerda = millis();
  }
  else if( (millis()-time_esquerda) >= 500 ){
    digitalWrite(motor_esquerda, LOW);
    if ( (millis()-time_esquerda) >= 1500 ){
      time_esquerda = 0;
    }
  }
}
```

Fonte: O próprio autor

4 RESULTADOS OBTIDOS

Vamos analisar os resultados obtidos separadamente, já que o protótipo se dividiu em duas partes.

4.1 RESULTADOS DA PRIMEIRA PARTE DO PROTÓTIPO

Primeiramente, iremos juntar a lógica de detecção de objeto realizada tanto pela visão computacional quanto pelos sensores ultrassônicos. Após essa junção, são feitas algumas adaptações para que as duas lógicas funcionem em conjunto.

O objetivo do novo código fonte gerado é usar o sensor ultrassônico para detectar os obstáculos aéreos e, através de dois mini alto-falantes, emitir uma mensagem de aviso. O outro objetivo é identificar os objetos ao redor, informar sua localização e a distância até eles. Essas informações também serão emitidas através dos mesmos mini alto-falantes.

Nessa primeira parte, o código fonte estará rodando apenas no Raspberry Pi 3 Model B+, e mesmo que o nível de processamento desse raspberry seja considerado bom, a velocidade de detecção fica em torno de 5 segundos. Isso quer dizer que a cada 5 segundos a detecção é realizada.

Para a realização dos testes, o protótipo foi fixado a uma altura de 170 centímetros do solo. Vejamos abaixo as capturas feitas pelo protótipo.

Figura 33 - 1ª Detecção realizada pelo primeiro protótipo

Fonte: O próprio autor

Figura 34 - 2ª Detecção realizada pelo primeiro protótipo

Fonte: O próprio autor

Figura 35 - 3ª Detecção realizada pelo primeiro protótipo

Fonte: O próprio autor

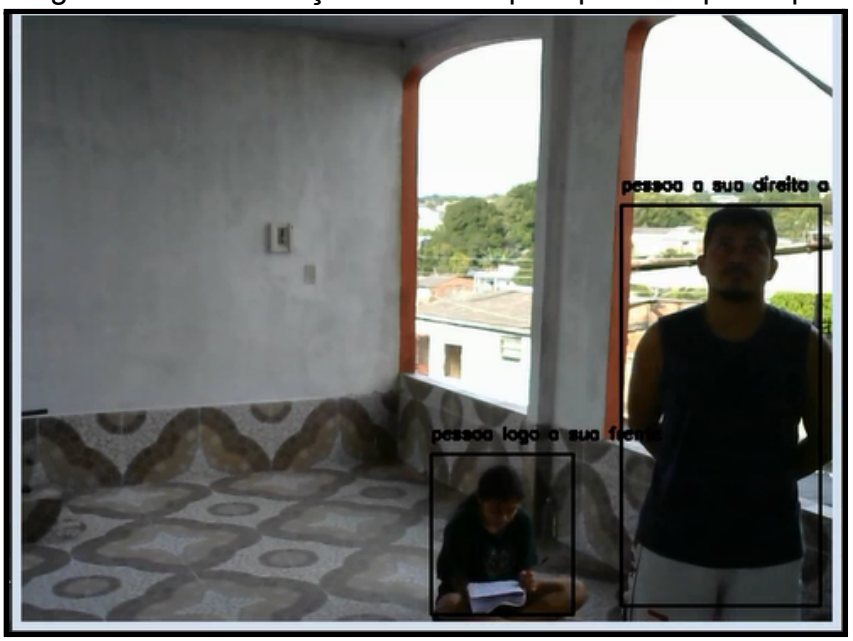
Figura 36 - 4ª Detecção realizada pelo primeiro protótipo

Fonte: O próprio autor

Figura 37 - 5ª Detecção realizada pelo primeiro protótipo

Fonte: O próprio autor

Figura 38 - 6ª Detecção realizada pelo primeiro protótipo



Fonte: O próprio autor

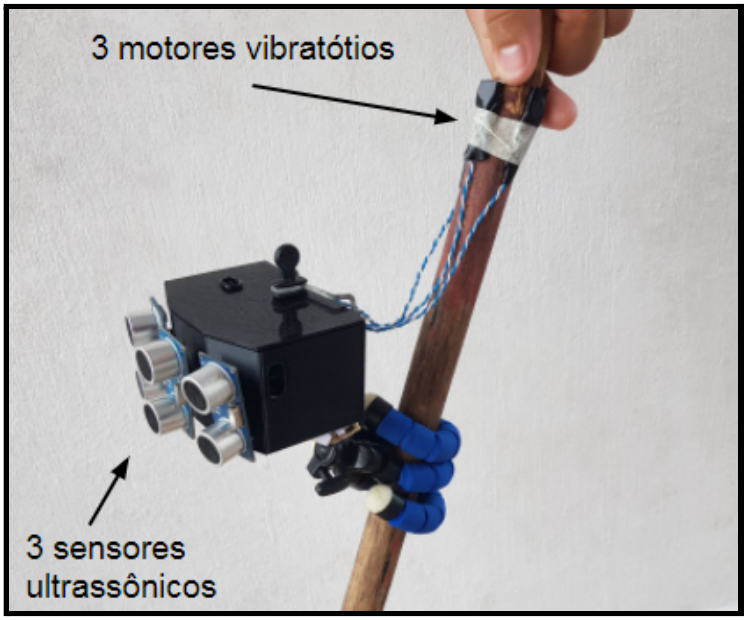
Figura 39 - 7ª Detecção realizada pelo primeiro protótipo

Fonte: O próprio autor

4.2 RESULTADOS DA SEGUNDA PARTE DO PROTÓTIPO

Nessa segunda parte, o código fonte estará rodando apenas no Arduino Nano. Para a realização dos testes, o protótipo foi fixado em um cabo de vassoura, para simular a fixação em uma bengala.

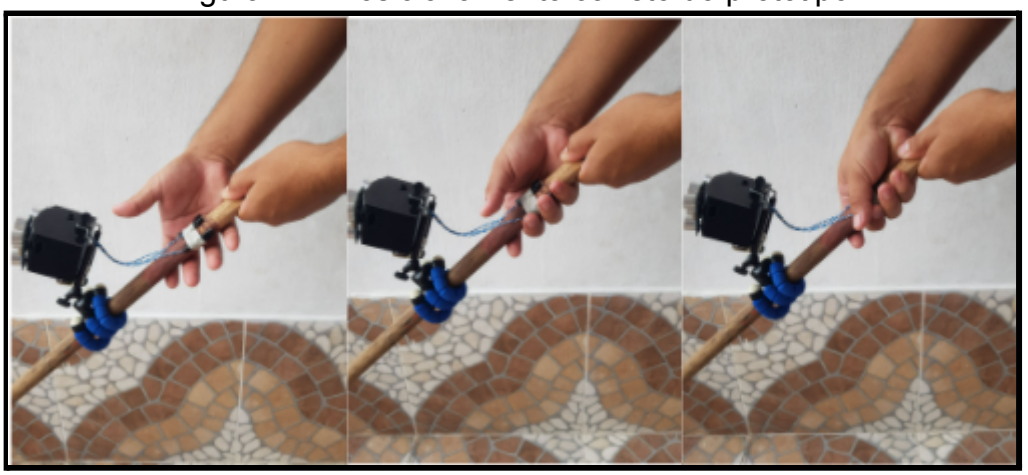
Figura 40 - Segunda parte do protótipo



Fonte: O próprio autor

De acordo com os testes, para o uso correto do protótipo, o usuário deve segurar a bengala no mesmo local onde os motores vibratórios estão fixados. Dessa maneira o usuário perceberá com mais clareza as vibrações. Vejamos abaixo.

Figura 41 - Posicionamento correto do protótipo



Fonte: O próprio autor

Durante os testes realizados, o protótipo mostrou uma boa detecção. O intuito destes testes foi usar os sensores ultrassônicos para detectar os obstáculos na linha da cintura e acionar o motor vibratório correspondente.

Figura 42 - Posicionamento correto do protótipo

Fonte: O próprio autor

4.3 RESULTADOS DE TESTES EM AMBIENTE URBANO

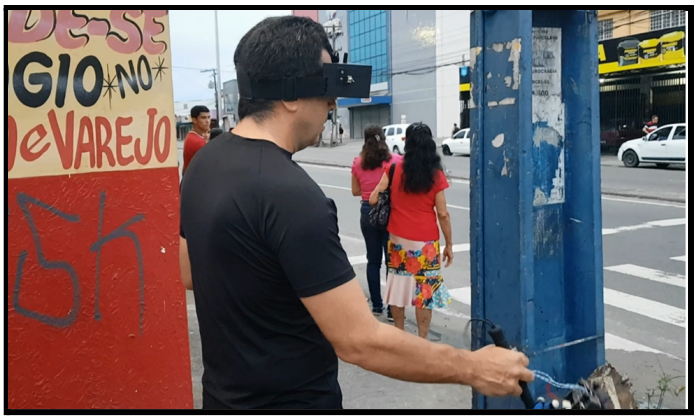
Finalmente foram realizados testes em ambiente urbano, onde as variáveis não podiam ser controladas. Contamos com a participação do Williams, que é um deficiente visual total. Ao final dos testes, perguntamos a ele como foi a experiência de usar o protótipo. Ele se mostrou muito impressionado com o desempenho do protótipo, e ainda forneceu algumas dicas para melhoria do mesmo.

Figura 43 - 1º Teste em ambiente urbano

Fonte: O próprio autor

Figura 44 - 2º Teste em ambiente urbano

Figura 45 - 3º Teste em ambiente urbano



Fonte: O próprio autor

CONCLUSÃO

Durante o desenvolvimento desta pesquisa foram realizados estudos sobre as dificuldades que as pessoas com deficiência visual enfrentam em um ambiente urbano durante uma simples caminhada. Além disso, pesquisas sobre detecção de objetos através de sensores e de Visão Computacional foram realizadas, a fim de detectar os obstáculos ao redor do deficiente visual e informá-lo sobre a localização de tais obstáculos, assim como a distância até os mesmos. Após algumas pesquisas, observou-se a necessidade de dividir o protótipo em duas partes. Uma parte ficaria no rosto do usuário para identificar objetos e detectar obstáculos aéreos, e outra parte ficaria fixa na bengala para detectar os obstáculos na linha da cintura.

Com base nos resultados obtidos, foi possível confirmar a hipótese de que é possível utilizar sensores ultrassônicos e Visão Computacional para auxiliar um deficiente visual, durante uma caminhada em um ambiente urbano, através de vibrações e mensagens de áudio. Mas o protótipo apresenta algumas limitações, como por exemplo: só funcionará corretamente se o local em que o usuário estiver for plano e que tenha uma iluminação mínima.

O modelo de detecção utilizado possui uma grande quantidade de classes de objetos detectáveis, só que a maioria dessas classes normalmente não está presente em um ambiente urbano, isso significa que o modelo perde muito tempo verificando classes desnecessárias. Uma sugestão para trabalhos futuros é utilizar um modelo que contenha, em sua maioria, classes de objetos que estão realmente presentes no ambiente urbano, para assim realizar uma detecção mais rápida.

REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO. **Arduino Nano.** 2021. Disponível em: <https://store-usa.arduino.cc/products/arduino-nano?selectedStore=us>. Acesso em: 2 de fevereiro de 2022.

BLOG ELETROGATE. **Sensor Ultrassônico HC-SR04 com Arduino.** 2021. Disponível em: <https://blog.eletrogate.com/sensor-ultrassonico-hc-sr04-com-arduino/>. Acesso em: 5 de outubro de 2021.

CÂMARA DOS DEPUTADOS, **Portador de Deficiência Visual Guia Legal.** 2. ed. Brasília. 2009. Disponível em: <https://bd.camara.leg.br/bd/bitstream/handle/bdcamara/1944/guia_legal.pdf>. Acesso em 20 de setembro de 2021.

EXPERT ACADEMY. **Detecção de Objetos com YOLO – Uma abordagem moderna.** 2022. Disponível em: <https://iaexpert.academy/2020/10/13/deteccao-de-objetos-com-yolo-uma-abordagem-moderna/>. Acesso em: 20 de fevereiro de 2022.

FILIPEFLOP. **Acelerômetro e Giroscópio 3 Eixos 6 DOF MPU-6050.** 2022. Disponível em: <https://www.filipeflop.com/produto/acelerometro-e-giroscopio-3-eixos-6-dof-mpu-6050/>. Acesso em: 15 de fevereiro de 2022.

GIL, M.(org). ***Deficiência Visual*. Brasília: MEC. Secretaria de Educação a Distância, 2000.** 80 p. (Cadernos da TV Escola. 1. ISSN 1518-4692). Disponível em: <http://portal.mec.gov.br/seed/arquivos/pdf/deficienciavisual.pdf>. Acesso em: 25 de setembro de 2021.

GLOBO NEWS. **Preparar um cão-guia custa em média R$ 30 mil e demora cerca de 2 anos.** 2011. Disponível em: <http://g1.globo.com/globo-news/noticia/2011/08/preparar-um-cao-guia-custa-em-media-r-30-mil-e-demora-cerca-de-2-anos.html>. Acesso em: 25 de setembro de 2021.

G1. **Anda SP: Deficientes visuais com cão-guia enfrentam desafios de mobilidade em SP.** 2019. Disponível em: <https://g1.globo.com/sp/sao-paulo/noticia/2019/07/15/anda-sp-deficientes-visuais-com-cao-guia-enfrentam-desafios-de-mobilidade-em-sp.ghtml>. Acesso em: 25 de setembro de 2021.

OLHAR DIGITAL. **Raspberry Pi: o que é, pra que serve e como comprar.** 2019. Disponível em: <https://olhardigital.com.br/2019/02/18/noticias/raspberry-pi-o-que-e-para-que-serve-e-como-comprar/>. Acesso em: 30 de setembro de 2021.

PASSARELLI, Leandro. **Embarcados: Aplicação de visão computacional com OpenCV.** 2017. Disponível em: <https://www.embarcados.com.br/aplicacao-de-visao-computacional-com-opencv/>. Acesso em: 5 de outubro de 2021.

PYTHON. **Sobre Python.** 2022. Disponível em: <https://www.python.org/about/>. Acesso em: 15 de janeiro de 2022.

SARTORETTO, M. L.; BERSCH, R. **Assistiva: Tecnologia e Educação.** 2021. Disponível em: <http://www.assistiva.com.br/tassistiva.html>. Acesso em: 25 de setembro de 2021.

APÊNDICE A – CÓDIGO FONTE DA PRIMEIRA PARTE DO PROTÓTIPO

```
from mpu6050 import mpu6050
import cv2
import vlc
import time
import RPi.GPIO as gpio
import math

gpio.setmode(gpio.BCM)
mpu = mpu6050(0x68)

trig_1 = 23
echo_1 = 24
gpio.setup(trig_1, gpio.OUT)
gpio.setup(echo_1, gpio.IN)

altura = 0

def distancia(trig, echo):
   gpio.output(trig, 1)
   time.sleep(0.000001)
   gpio.output(trig, 0)
   t_inicial = time.time()
   t_final = time.time()
   while gpio.input(echo) == 0:
       t_inicial = time.time()
   while gpio.input(echo) == 1:
       t_final = time.time()
   t_distancia = t_final - t_inicial
   distancia = int((t_distancia*34300)/2)
   return distancia

def angulo():
   acel = mpu.get_accel_data()
   x = acel['x']
   z = acel['z']
   if x==0:
       ang = 0
   elif z==0:
       ang = 90
   else:
       ang = (math.atan(z/x) * 180 / math.pi)
   return ang

def Localizacao_Objeto(h, px1, px2, py):
 PixAlt = 480
 PixLar = 640
```

```
AngV = 32
AngH = 42
omb = h/2
passo = 50
a = 90 - (AngV/2) + angulo()
b = math.atan(2*h/omb) * 180 / math.pi

px = int((px1+px2)/2)
Dmin = h * math.tan(a*math.pi/180)
L = 2 * Dmin * math.tan((AngV/2)*math.pi/180)
A = L * PixAlt/PixLar
x = math.fabs((PixLar - 2*px)) * L / (2*PixLar)
D = math.sqrt((Dmin*Dmin + x*x))
y = (PixAlt-py) * A / PixAlt
Dobj = int((D * h / (h-y)) / passo)

Pomb = PixLar * omb / L
Pe = int(PixLar - Pomb) / 2
Pd = int(PixLar + Pomb) / 2

posicao = ''
if (py+10) >= int(PixAlt - (Pomb * math.tan(b*math.pi/180) / 2)):
    if (px2 <= int(PixLar/2)):
        if px2 < Pe:
            posicao = 'e'
        else:
            px = px2
    else:
        if (px1 > Pd):
            posicao = 'd'
        else:
            if (px2 < Pd):
                px = px2
            else:
                if (px1 > Pe):
                    px = px1

    if posicao == '':
        if (px > Pe) and (px < Pd):
            lim = int((Pomb -
2*(PixAlt-py)/math.tan(b*math.pi/180)) / 2)
            if (int(math.fabs(px - (PixLar/2))) <= lim):
                posicao = 'f'
            else:
                if (px < int(PixLar/2)):
                    posicao = 'e'
                else:
                    posicao = 'd'
```

```
            else:
                if (px == int(PixLar/2)):
                    posicao = 'f'
                else:
                    if (px < int(PixLar/2)):
                        if (px==Pe) and (py==PixAlt):
                            posicao = 'f'
                        else:
                            posicao = 'e'
                    else:
                        if (px==Pd) and (py==PixAlt):
                            posicao = 'f'
                        else:
                            posicao = 'd'
    if posicao == '':
        resposta = ''
    else:
        if py >= (PixAlt-10):
            resposta = posicao + 'logo'
        else:
            resposta = posicao + 'a_' + str(Dobj) + '_passos'
    return resposta

net =
cv2.dnn.readNet("/home/pi/Documents/TCC2/DocsTCC2/yolov4-tiny.weigh
ts", "/home/pi/Documents/TCC2/DocsTCC2/yolov4-tiny.cfg")
model = cv2.dnn_DetectionModel(net)
model.setInputParams(size=(416, 416), scale=1/255)
class_names_sin = []
with open("/home/pi/Documents/TCC2/DocsTCC2/coco.names", "r") as f:
    class_names_sin = [cname.strip() for cname in f.readlines()]
class_names_plu = []
with open("/home/pi/Documents/TCC2/DocsTCC2/cocos.names", "r") as
f:
    class_names_plu = [cname.strip() for cname in f.readlines()]

def identificar_objetos():
    laf = {}
    lae = {}
    lad = {}
    af  = {}
    ae  = {}
    ad  = {}
    cap = cv2.VideoCapture(0)
    _, frame = cap.read()
    classes, scores, boxes = model.detect(frame, 0.1, 0.2)
    for (classid, score, box) in zip(classes, scores, boxes):
        if class_names_sin[classid[0]][0] != '_':
```

```
            loc_obj = Localizacao_Objeto(160, list(box)[0],
(list(box)[0] + list(box)[2]), (list(box)[1] + list(box)[3]))
            if 'logo' in loc_obj:
                if 'f' in loc_obj:
                    laf[classid[0]] = laf.get(classid[0], 0) + 1
                else:
                    if 'e' in loc_obj:
                        lae[classid[0]] = lae.get(classid[0], 0) + 1
                    else:
                        lad[classid[0]] = lad.get(classid[0], 0) + 1
            elif 'passo' in loc_obj:
                l_obj = ''
                for i in range(1, len(loc_obj)):
                    l_obj += loc_obj[i]
                if 'f' in loc_obj:
                    af[class_names_sin[classid[0]]] = l_obj
                else:
                    if 'e' in loc_obj:
                        ae[class_names_sin[classid[0]]] = l_obj
                    else:
                        ad[class_names_sin[classid[0]]] = l_obj
    resposta = [[laf, lae, lad], [af, ae, ad]]

    return resposta

caminho = '/home/pi/Documents/TCC2/DocsTCC2/audios/'
player1 = vlc.MediaPlayer()
player2 = vlc.MediaPlayer()
player3 = vlc.MediaPlayer()
def Informar(infor):
    inicio_frase = ['logo_a_frente_tem', 'logo_a_esquerda_tem',
'logo_a_direita_tem']
    infs = infor[0]
    for i in range(0, 3):
        inf = infs[i]
        if len(inf)!=0:
            arquivo1 = caminho + inicio_frase[i] + '.m4a'
            media1 = vlc.Media(arquivo1)
            player1.set_media(media1)
            player1.play()
            while player1.is_playing() != 1:
                aux=0
            time.sleep(2)
            for v in inf.keys():
                if inf[v] > 1:
                    arquivo2 = caminho + str(inf[v]) + '.m4a'
                    media2 = vlc.Media(arquivo2)
                    player2.set_media(media2)
```

```
                    player2.play()
                    while player2.is_playing() != 1:
                        aux=0
                    t = (0.3*player2.get_length()/1000)
                    time.sleep(t)
                    arquivo3 = caminho + class_names_plu[v] + '.m4a'
                else:
                   arquivo3 = caminho + class_names_sin[v] + '.m4a'
                media3 = vlc.Media(arquivo3)
                player3.set_media(media3)
                player3.play()
                while player3.is_playing() != 1:
                    aux=0
                time.sleep(1.2)

    inicio_frase = ['a_frente_tem', 'a_esquerda_tem',
'a_direita_tem']
    infs = infor[1]
    for i in range(0, 3):
        inf = infs[i]
        if len(inf)!=0:
            arquivo1 = caminho + inicio_frase[i] + '.m4a'
            media1 = vlc.Media(arquivo1)
            player1.set_media(media1)
            player1.play()
            while player1.is_playing() != 1:
                aux=0
            time.sleep(1)
            for v in inf.keys():
                arquivo2 = caminho + v + '.m4a'
                media2 = vlc.Media(arquivo2)
                player2.set_media(media2)
                player2.play()
                while player2.is_playing() != 1:
                    aux=0
                t = (0.5*player2.get_length()/1000)
                time.sleep(t)
                pas = inf[v]
                pa = pas[2]
                cont = 3
                while pas[cont] != '_':
                    pa += pas[cont]
                    cont = cont + 1
                int_pa = int(pa)
                if int_pa < 31:
                    arquivo3 = caminho + inf[v] + '.m4a'
                else:
```

```
                    arquivo3 = caminho + 'a_mais_de_30_passos' +
'.m4a'
                media3 = vlc.Media(arquivo3)
                player3.set_media(media3)
                player3.play()
                while player3.is_playing() != 1:
                    aux=0
                time.sleep(1.3)

while True:
   if math.fabs(angulo())>70:
       break
time.sleep(2)
print('lendo altura')
player = vlc.MediaPlayer()
while True:
   if math.fabs(angulo())<40:
       break
   caminho = '/home/pi/Documents/TCC2/DocsTCC2/audios/'
   altura = str(distancia(trig_1, echo_1))
   for i in altura:
       if i != '.':
           arquivo = caminho + i + '.m4a'
           media = vlc.Media(arquivo)
           player.set_media(media)
           player.play()
           while player.is_playing() != 1:
               aux=0
           t = (0.6*player.get_length()/1000)
           time.sleep(t)
   arquivo = caminho + 'centimetros' + '.m4a'
   media = vlc.Media(arquivo)
   player.set_media(media)
   player.play()
   time.sleep(4)

dist_min = 200
pla = vlc.MediaPlayer()
while True:
   sensor_1 = distancia(trig_1, echo_1)
   if sensor_1 <= dist_min:
       arq = caminho + 'cuidado_obstaculo_aereo' + '.m4a'
       med = vlc.Media(arq)
       pla.set_media(med)
       pla.play()
       time.sleep(6)
   informacoes = identificar_objetos()
   print(informacoes)
```

```
    Informar(informacoes)

cap.release()
gpio.cleanup()
cv2.destroyAllWindows()
```

APÊNDICE B – CÓDIGO FONTE DA SEGUNDA PARTE DO PROTÓTIPO

```
#include <Ultrasonic.h>

#define motor_direita 2
#define motor_frente 3
#define motor_esquerda 4

#define trig_direita 10
#define echo_direita 9
#define trig_frente 8
#define echo_frente 7
#define trig_esquerda 6
#define echo_esquerda 5

Ultrasonic ultrasonic_direita(trig_direita, echo_direita);
Ultrasonic ultrasonic_frente(trig_frente, echo_frente);
Ultrasonic ultrasonic_esquerda(trig_esquerda, echo_esquerda);

int dist_detec = 150;
unsigned long time_direita=0, time_frente=0, time_esquerda=0;
void setup() {
  Serial.begin(9600);
  Serial.println("Lendo dados do sensor...");
  pinMode(motor_direita, OUTPUT);
  pinMode(motor_frente, OUTPUT);
  pinMode(motor_esquerda, OUTPUT);
  digitalWrite(motor_direita, LOW);
  digitalWrite(motor_frente, LOW);
  digitalWrite(motor_esquerda, LOW);
}

float cm_direita=0, cm_frente=0, cm_esquerda=0;
void loop() {
  cm_direita =
ultrasonic_direita.convert(ultrasonic_direita.timing(),
Ultrasonic::CM);
  delay(10);
  cm_frente = ultrasonic_frente.convert(ultrasonic_frente.timing(),
Ultrasonic::CM);
  delay(10);
  cm_esquerda =
ultrasonic_esquerda.convert(ultrasonic_esquerda.timing(),
Ultrasonic::CM);
  delay(10);
  Serial.print("Sensor 1: ");
  Serial.println(cm_direita);
  Serial.print("Sensor 2: ");
```

```
  Serial.println(cm_frente);
  Serial.print("Sensor 3: ");
  Serial.println(cm_esquerda);
  Serial.println();

  if((cm_direita <= dist_detec) && (time_direita == 0)){
    digitalWrite(motor_direita, HIGH);
    time_direita = millis();
  }
  else if( (millis()-time_direita) >= 500 ){
    digitalWrite(motor_direita, LOW);
    if ( (millis()-time_direita) >= 1500 ){
      time_direita = 0;
    }
  }
  if((cm_frente <= dist_detec) && (time_frente == 0)){
    digitalWrite(motor_frente, HIGH);
    time_frente = millis();
  }
  else if( (millis()-time_frente) >= 500 ){
    digitalWrite(motor_frente, LOW);
    if ( (millis()-time_frente) >= 1500 ){
      time_frente = 0;
    }
  }
  if((cm_esquerda <= dist_detec) && (time_esquerda == 0)){
    digitalWrite(motor_esquerda, HIGH);
    time_esquerda = millis();
  }
  else if( (millis()-time_esquerda) >= 500 ){
    digitalWrite(motor_esquerda, LOW);
    if ( (millis()-time_esquerda) >= 1500 ){
      time_esquerda = 0;
    }
  }
}
```