CENTRO PAULA SOUZA FACULDADE DE TECNOLOGIA
DE SANTO ANDRÉ
Tecnologia em Mecatrônica Industrial

DIOGO ANTONIO DA SILVA
VINICIUS BRANDÃO CORDEIRO

APLICAÇÃO DO “ROS” PARA O PROJETO DE UM VEÍCULO AUTÔNOMO EM ESCALA

Santo André
2021

DIOGO ANTONIO DA SILVA

VINICIUS BRANDÃO CORDEIRO

APLICAÇÃO DO “ROS” PARA O PROJETO DE UM VEÍCULO AUTÔNOMO EM ESCALA

Trabalho de Conclusão de curso apresentado ao curso superior de tecnologia em mecatrônica industrial da FATEC Santo André como requisito parcial para obtenção do título de tecnólogo em Mecatrônica Industrial.

Orientador: Prof. Weslley Medeiros Torres

Coorientador: Caio Roberto dos Santos

Santo André

2021

DIOGO ANTONIO DA SILVA

VINICIUS BRANDÃO CORDEIRO

APLICAÇÃO DO “ROS” PARA O PROJETO DE UM VEÍCULO AUTÔNOMO EM ESCALA

Trabalho de Conclusão de curso apresentado ao curso superior de tecnologia em mecatrônica industrial da FATEC Santo André como requisito parcial para obtenção do título de tecnólogo em Mecatrônica Industrial.

Professor Orientador
Prof. Me. Weslley Medeiros Torres

Coorientador
Caio Roberto dos Santos

MEMBROS DA BANCA EXAMINADORA

Presidente da Banca
Prof. Me. Weslley Medeiros Torres
Fatec Santo André

Primeiro membro da Banca
Caio Roberto dos Santos.
Fatec Santo André

Segundo Membro da Banca
Prof. Fernando Dalbo Garup.
Fatec Santo André

Local: Fatec Santo André
Horário: 08:00
Data: 06/12/2021

SANTO ANDRÉ

2021

DEDICATÓRIA

Dedicamos esse trabalho aos nossos familiares e a todos os professores que nos ajudaram no desenvolvimento.

FICHA CATALOGRÁFICA

S586a
Silva, Diogo Antonio da
Aplicação do “ROS” para projeto de um veículo autônomo em escala / Diogo Antonio da Silva, Vinicius Brandão Cordeiro. - Santo André, 2021. – 64f: il.

Trabalho de Conclusão de Curso – FATEC Santo André.
Curso de Tecnologia em Mecatrônica Industrial, 2021.

Orientador: Prof. Weslley Medeiros Torres

1. Mecatrônica. 2. Veículos autônomos. 3. Desenvolvimento. 4. Projeto. 5. ROS. 6. Tecnologia embarcada. 7. Interface. 8. Sensores 9. Atuadores. 10. Automação. 11. Programação. I. Cordeiro, Vinicius Brandão. II. Aplicação do “ROS” para projeto de um veículo autônomo em escala.

629.892

AGRADECIMENTOS

Agradecemos a Deus por ter proporcionado e nos dado forças a todo momento para a finalização tanto do trabalho quanto do curso.

Agradecemos as nossas famílias que incentivaram e ajudaram em tudo que precisava e entenderam a nossa ausência em muitos momentos.

Agradecemos a instituição Fatec Santo André, aos professores e a todos os colegas de sala que ajudaram, tiraram nossas dúvidas e compartilharam informações.

“Algumas pessoas lêem "Guerra e Paz" e acham que é um simples romance. Outras pessoas lêem uma embalagem de chiclete e desvendam os segredos do universo...”

Lex Luthor

RESUMO

Este trabalho tem como proposta o desenvolvimento e projeto de um veículo autônomo em tamanho reduzido empregando ROS (*Robot Operating System*) para a interface entre os sensores e atuadores do veículo. O foco primário desse projeto é a aplicação do conjunto de bibliotecas oferecido pelo ROS, que é muito utilizado na área da automação fazendo diversos nós de programação, formando assim a conexão entre diferentes módulos. O protótipo empregará uma Raspberry PI que será conhecida como máquina mestre, pois receberá e irá enviar as informações para os módulos de sensores e atuadores utilizados no projeto.

Palavras-chave: ROS. Tecnologia embarcada. Veículo autônomo.

Abstract

This project has a proposal of a development and projecting one autonomous vehicle in reduced size that utilizes a tool called ROS (*Robot Operating System*). Our primary focus on this project is the application and improvement of the library set offered by ROS, that is very utilized in the automation area making several programming knot's, making the connection between different modules. Our prototype will be done using one Raspberry PI which will be known as master machine, because it will receive the information and send them to distinct modules utilized on the Project.

keywords: ROS. Embedded technology. Autonomous vehicle.

LISTA DE ILUSTRAÇÃO

LISTA DE EQUAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CMD	*Command Prompt*
CPU	*Central Processing Unit*
ECU	*Engine Control Unit*
ESP	*Electronic Stability Program*
EUA	Estados Unidos da América
GM	*General Motors Corporation*
GNU	*GNU is Not Unix*
GPL	*General Public License*
GPS	*Global Positioning System*
HDMI	*High-Definition Multimedia Interface*
IA	Inteligência Artificial
Multics	*Multiplexed Information and Computing Service*
OSRF	*Open-Source Robotics Foundation*
RGB	*Red, Green and Blue*
ROS	*Robot Operating System*
SD	*Secure Digital*
SO	Sistema Operacional
USB	*Universal Serial Bus*

LISTA DE TABELAS

SUMÁRIO

1 Introdução

Graças ao avanço na área de engenharia de *software* da engenharia automobilística, nos anos de 1950 a 1980 surgiram os primeiros protótipos e projetos veículos autônomos, os quais eram equipados com câmeras e sensores em sua estrutura possibilitando assim testes de um verdadeiro veículo que não necessitava de um condutor ou de fatores externos como guias ou outros indicativos. Com um alto custo para ser fabricado e não tendo uma total segurança contra fatores externos, sendo alguns destes fatores seres vivos, esses carros demoraram para receber autorização para circular de forma lícita e segura, sendo uma realidade no ano de 2020 com o carro Nuro R2 produzido pela empresa Nuro. (*Nuro's new delivery R2 bot gets the first driverless vehicle exemption from feds*, 2020).

Atualmente, com o uso dessas novas tecnologias, o número de veículos produzidos que empregam componentes elétricos e eletroeletrônicos têm aumentado gradativamente, o que torna mais acessíveis projetos para veículos autônomos.

O termo arquitetura na robótica móvel, pode ser aplicado a estratégia com qual se deseja realizar uma tarefa, sendo desde um comportamento baseado na dependência de sinais recebidos por sensores para adoção de uma diretiva que realizará uma tarefa específica, a meios que tomam o desenvolvimento de um conceito centralizado em decisões baseadas em quantidades probabilísticas para garantia de cumprimento de objetivos ao longo de um trajeto. Em todos os casos a arquitetura e tratada de forma a ser de nível mais alto para nível mais baixo, de modo que o sistema inteligente utilize de um grande volume de informações obtidos desde os níveis mais baixos, como sensores, atuadores e até sistemas mais complexos como câmeras e radares, para atender a missão designada pelo usuário do sistema.

Como o desenvolvimento de cada veículo autônomo tem suas dinâmicas, a definição de uma arquitetura fixa para utilização de veículos autônomos em geral se torna difícil, pela presença de diferentes comportamentos e aplicações

além de suas particularidades tais como sensores, atuadores e redes utilizadas por exemplo.

No projeto de um veículo autônomo, a comunicação entres sensores e a CPU (*Central Processing Unit*) é de extrema importância, pois é essa comunicação que determina os movimentos e ações que o carro tomará durante uma determinada situação. A coleção de *frameworks* de *software* ROS (*Robot Operating System*) foi criada visando a comunicação entre diferentes componentes elétricos e eletroeletrônicos, facilitando uma conexão entre a CPU e os sensores.

1.1 Objetivo

Os objetivos deste trabalho são pesquisar e projetar uma aplicação do ROS em um veículo autônomo em escala reduzida, que seja capaz de reconhecer obstáculos, traçar uma rota e tomar decisões em determinadas situações. servindo como aprendizado nas áreas de programação, eletroeletrônica e elétrica para os alunos da Fatec Santo André.

1.2 Metas

As principais metas desse projeto são:

- Criar uma rede de comunicação entre os módulos compostos por microcontroladores e sensores e a CPU Raspberry.
- Criar uma estrutura de decisões que devem ser tomadas em determinadas circunstâncias.
- Realizar um entendimento geográfico capaz de realizar rotas.

2 Fundamentação Teórica

Este capítulo irá abordar todo o conteúdo e conceitos necessários para o entendimento do trabalho, passando pela contextualização histórica sobre os veículos autônomos e o seu funcionamento.

A partir do término desse assunto será possível uma melhor compreensão do projeto proposto, partindo para a próxima etapa de desenvolvimento.

2.1 História dos carros autônomos

Em 1939 houve a exposição *Futurama* patrocinada pela empresa General Motors Corporation (GM), projetada pelo designer teatral Norman Melancton Bel Geddes em que foi na Feira Mundial de Nova Iorque, nos Estados Unidos da América (EUA). Este evento deu início as primeiras pesquisas na área de automação veicular, onde demonstrava como o mundo seria vinte anos depois, nesta demonstração mostraram um protótipo de sistema de rodovias automatizados com o objetivo de utilizar as estradas para corrigir possíveis falhas derivadas da condução humana, impedindo acidentes por ações que não pudessem ou não deveriam ser realizadas. (*General Motors Highways & horizons : New York World's Fair*, 2008)

Com a vinda da Segunda Guerra Mundial no mesmo ano, os esforços dos grandes fabricantes foram voltados para a produção bélica, não dando continuidade aos estudos e aos projetos de Geddes para a construção das estradas planejadas. Em 1945 com o término da Segunda Guerra Mundial, as tecnologias desenvolvidas para a utilização na guerra foram ajustadas e utilizadas para o projeto de automatizar e aumentar os recursos de navegação usados nos veículos. No ano de 1950 deu-se início a uma parceria entre as empresas *Radio Corporation of America* (RCA) e a GM, que juntas começaram a desenvolver tecnologias que automatizassem e aperfeiçoassem a condução dos carros e em 1953 realizaram um modelo em escala de um sistema rodoviário automatizado para teste e iniciaram uma produção de uma série de três carros-conceito, o primeiro carro-conceito é o Firebird que demonstrava as novas tecnologias desenvolvidas, o segundo foi o Firebird II, onde foi apresentado

como um projeto de carro tecnológico que contava com um sistema de condução automática no qual era guiado por um fio enterrado na estrada que enviava sinais para o carro e para um sistema de comunicação via rádio, porém foi apenas em 1958 que teve-se a primeira demonstração de uma condução automatizada utilizando fios elétricos enterrados no solo que guiavam o veículo por ondas eletromagnéticas.

2.2 Arquitetura de veículos autônomos

Quando citado o termo arquitetura robótica devemos ter conhecimento que a mesma não é constituída somente pelo *hardware* ou pelo *software* do robô, mas sim pela comunicação entre estes componentes. Em geral os sistemas de funcionamento dos robôs são tão complexos que tendem a ser de difícil desenvolvimento, pois integram múltiplos sensores e vários graus de liberdade precisando conciliar os sistemas em tempo real. E para implementação destes sistemas complexos temos arquiteturas robóticas com fornecimento de serviços computacionais para subsistemas e componentes. De toda forma estas arquiteturas têm tendência a serem limitadas a tarefas e ambientes específicos deixando a desejar em sua aplicabilidade. Sendo assim uma arquitetura bem adaptada para uma situação tende a não ser bem adaptada para outra situação completamente diferente.

Um conceito base para a abordagem de arquiteturas robóticas é o conceito de agente. Um agente é uma entidade autônoma, ativa e cognitiva possuindo um sistema interno de tomada de decisão, sendo capaz assim de funcionar sem necessitar de algo ou de alguém para guiá-lo.

2.2.1 Arquitetura Deliberativa

É aquela que contém um modelo simbólico do mundo onde as decisões vão ser tomadas via raciocínio lógico ou pseudológico baseado em padrões ou manipulação simbólica. Sendo assim a arquitetura deliberativa é fixa e pré-

definida, por ter essas características é sujeita a erros por ser pouco robusta e pouco flexível tendo assim difíceis adaptações a novas situações.

2.2.2 Arquitetura Reativa

Esta arquitetura opera em um nível muito baixo de abstração, sem nenhum conhecimento prévio de ambiente. Não utilizando raciocínio simbólico complexo e tendo um tempo de resposta baixo acaba conduzindo uma comunicação eficiente entre dois agentes ou entre o agente e o ambiente. No entanto, o agente não pode executar análises complexas de seus dados sensoriais não podendo assim realizar operações de alto nível cognitivo.

2.2.3 Arquitetura Híbrida

Busca o melhor das arquiteturas deliberativas e reativas, integrando ambas e criando assim uma arquitetura híbrida. A integração das duas arquiteturas explora o complemento de ambas para melhorar o sistema como todo. Onde o reativo tem preferência sobre o deliberativo por possuir resposta mais ágil às variações de ambiente deixando assim a arquitetura deliberativa presente sendo responsável pela abstração dos dados.).

2.3 Sensores presentes

Assim como o corpo humano possui diversos sistemas sensoriais que ajudam tanto na localização quanto em equilíbrio e direção, assim temos também o carro autônomo que possui sensores que auxiliam a evitar acidentes, determinar a rota e até mesmo tomar decisões.

O sensor é um dispositivo que tem como fundamentação a função de detectar e responder com eficiência algum estímulo recebido podendo ser estes por

exemplo: calor, pressão, movimento, luz entre outros. Após o mesmo receber este estímulo converte em sinal o que passa a poder ser interpretado pelos outros dispositivos. Portanto, é preciso avaliar as condições de atuação e optar pelos sensores mais adequados para aquela atividade.

2.3.1 Câmeras

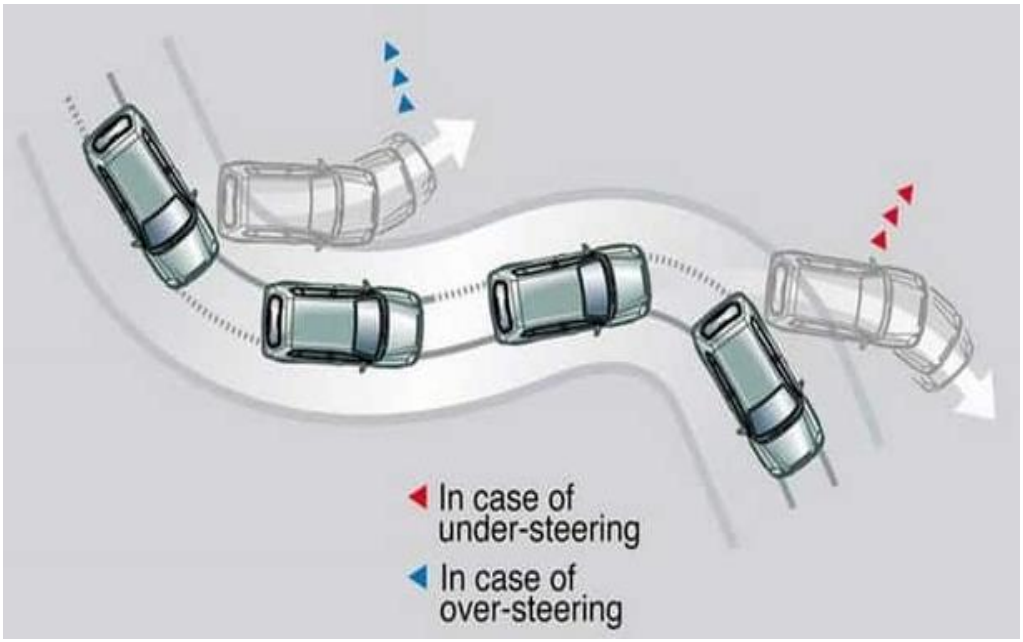
As câmeras possuem um papel muito importante, podendo ser assemelhadas aos olhos humanos, elas são responsáveis por captar a imagem e enviam para o *Engine Control Unit* (ECU), assim reconhecendo objetos, estruturas, pessoas e elementos do trânsito. Um exemplo de uso da câmera é um pedestre atravessando de maneira inesperada, a câmera irá capturar a imagem, enviar para ECU, reconhecendo como uma pessoa e assim acionando os freios para evitar um acidente.

2.3.2 Controle Eletrônico de Estabilidade

Para que o veículo seja capaz de acelerar e frear de maneira independente, é necessário que possua um controle Eletrônico de Estabilidade, O controle de estabilidade é popularmente conhecido por ESP (*Electronic Stability Program ou Programa Eletrônico de Estabilidade*), é um sistema eletrônico que atua diretamente nos freios e evitando assim que o condutor perca o controle direcional em curvas ou desvios de trajetória.

O dispositivo trabalha analisando a rotação de cada uma das rodas em relação à velocidade do veículo identificando rapidamente aquela que está perdendo aderência, acionando o freio daquela unidade corrigindo assim a trajetória do veículo evitando por assim que o mesmo rode na pista ou saía excessivamente no tangenciamento da curva, como mostra na Figura 1.

Figura 1 – Exemplificação de funcionamento do ESP.



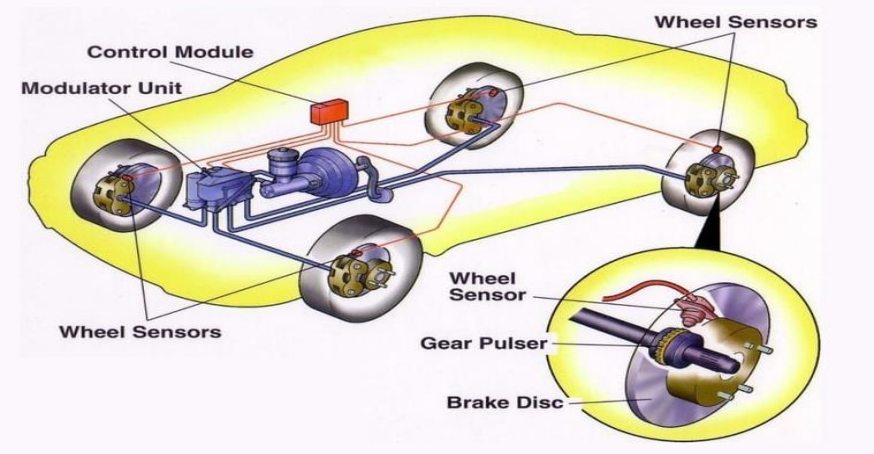
Fonte – https://images.noticiasautomotivas.com.br/img/f/esp-2-700x434.jpg

O ESP é uma tecnologia que monitora diversos aspectos da direção e do próprio carro. Alguns deles são:

- Perda de tração nas rodas
- Inclinação do carro
- Angulação dos pneus
- Velocidade do veículo
- Trajetória das curvas
- Rotação das rodas

Assim, tendo controle de estabilidade com influência sobre o sistema de direção ou mesmo agregado ao controle de condução dinâmica, oferece-se modos de direção com vários níveis de atuação do ESP, desde completa ausência até total controle da ação sobre os freios. A arquitetura ESP pode ser vista na Figura 2.

Figura 2 – Exemplificação da arquitetura do ESP.

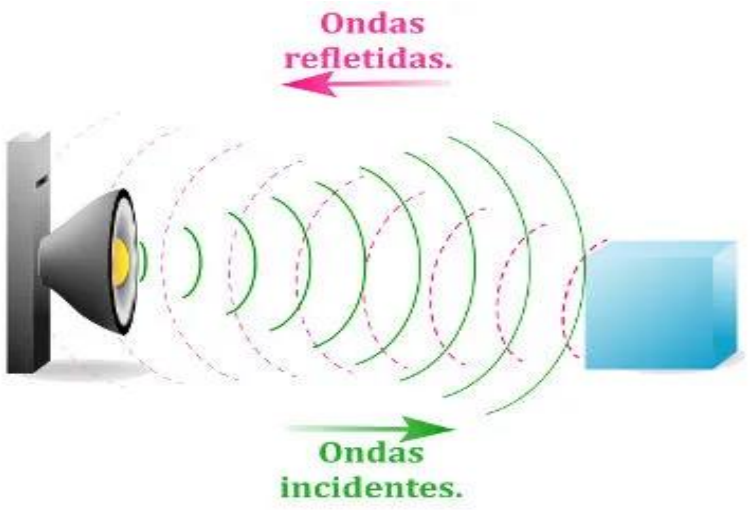


Fonte – https://images.noticiasautomotivas.com.br/img/f/TCS-Sys-700x499.jpg

2.3.3 Radar

O radar é um sensor utilizado com o objetivo de evitar colisões, por mais que as câmeras tenham a capacidade de reconhecer pessoas ou objetos, elas não são capazes de determinar a distância entre o veículo e os obstáculos, então é aí que o radar entra em ação.

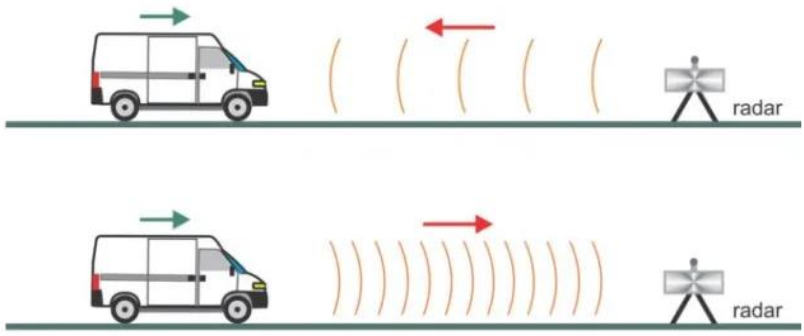
Figura 3 – Demonstração de emissão e recepção das ondas.



Fonte – https://static.mundoeducacao.uol.com.br/mundoeducacao/conteudo/sonar.jpg

Como demonstrado na Figura 3, os sensores de radar usam radar de Ondas Contínuas de Frequência Modulada para detectar de forma precisa alvos móveis ou fixos. Radares de baixa frequência podem ver muito bem objetos de alta dielétrica, incluindo carros, trens, caminhões e cargas, mesmo em condições climáticas extremas. Radares de alta frequência podem ver uma gama maior de objetos e são mais robustos do que um sensor ultrassônico. Eles utilizam frequência de rádio, onde essa frequência se propaga pelo ar, com uma velocidade de até 300 mil quilômetros por segundo, e podendo calcular a velocidade dos objetos ao seu redor, como mostrado na Figura 4.

Figura 4 – Demonstração de ondas ao atingir materiais em movimento.

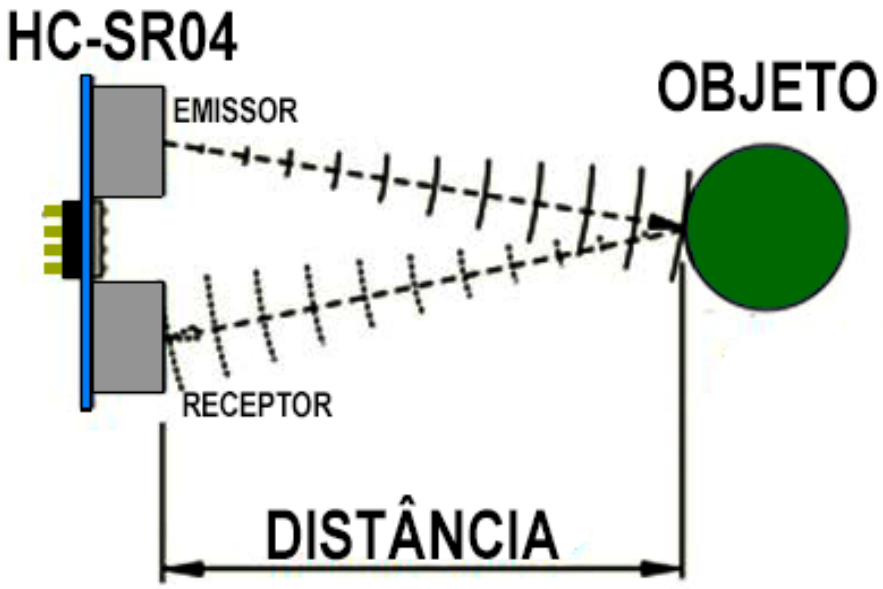


Fonte – https://i1.wp.com/www.mtitecnologia.com.br/wp-content/uploads/2017/02/imagem_release_861608-678x381.jpg?resize=640%2C360

2.3.4 Sensor Ultrassônico

Com objetivos similares ao Radar, os sensores ultrassônicos presente nos automóveis são utilizados para medir distâncias, com a diferença que são usados exclusivamente para realizar manobras em baixa velocidade o princípio de funcionamento dos sensores ultrassônicos está baseado na emissão de uma onda sonora de alta frequência, e na medição do tempo que leva para a recepção do eco, produzido quando a onda se encontra com um objeto capaz de refletir o som, como ilustrado na Figura 4.

Figura 4 – Demonstração de funcionamento do sensor ultrassônico.



Fonte – https://flaviobabos.com.br/wp-content/uploads/2021/06/sensor-ultrassonico-arduino-img2.png

Como dito funcionam através da medição do tempo de propagação do eco, isto é, o intervalo de tempo entre o impulso sonoro emitido e o eco recebido de volta. O sensor trabalha emitindo pulsos de ultrassom ciclicamente. E após o eco retornar com a presença de objeto o mesmo é convertido em forma de sinal elétrico. A detecção do eco incidente, depende de sua intensidade e da distância entre o objeto e o sensor ultrassônico. A construção do sensor faz com que o feixe ultrassônico seja emitido em forma de um cone e somente objetos dentro do raio do cone são detectados. Os objetos a serem detectados podem ser sólidos, líquidos, granulares ou pós. O material poderá ser transparente ou colorido, de qualquer formato, e com superfície polida ou fosca.

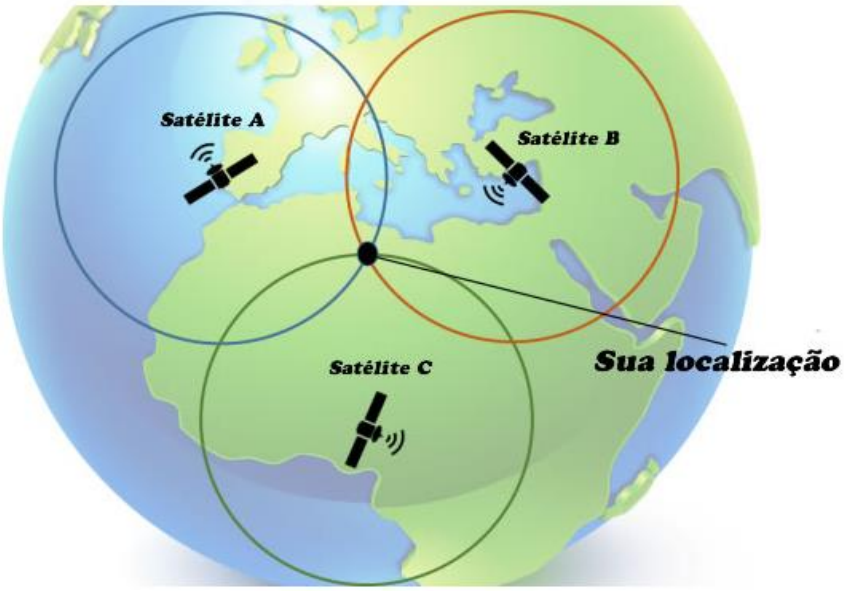
2.3.5 Global Positioning System (GPS)

O GPS é um sensor presente em muitos veículos e dispositivos móveis, sendo indispensável em um automóvel autônomo, pois é através dele que o sistema é capaz de se localizar e saber em qual direção ir. Atualmente o GPS utiliza uma

rede de 24 satélites na órbita da Terra que trocam sinais, informando a localização e fornecendo as coordenadas do determinado lugar na Terra, desde que tenhamos um receptor de sinais de GPS. Assim, aquele aparelho receptor, que está posicionado aqui na Terra, sabe exatamente onde estão os satélites.

Esses satélites estão distribuídos em setores de maneira que um receptor, posicionado em qualquer ponto da superfície terrestre, estará sempre ao alcance de pelo menos três deles (quatro ou mais para precisão maior), como mostra a Figura 5. E baseando-se em cálculos que ocorrem através de um processo chamado triangulação.

Figura 5 – Exemplificação da triangulação.



Fonte – https://www.sofisica.com.br/conteudos/curiosidades/imagens/gps2.jpg

O processo de triangulação utiliza de três satélites os quais enviam o sinal para o receptor, que calcula quanto tempo cada sinal demorou até atingir o receptor. Além de localização terrestre, o receptor GPS tem a capacidade de saber a altura do receptor em relação ao nível do mar, todavia, para isso é necessário um quarto satélite integrando assim a precisão.

Tanto os receptores quanto os satélites possuem um relógio interno que marca as horas em nanossegundos com uma alta precisão. Quando o satélite

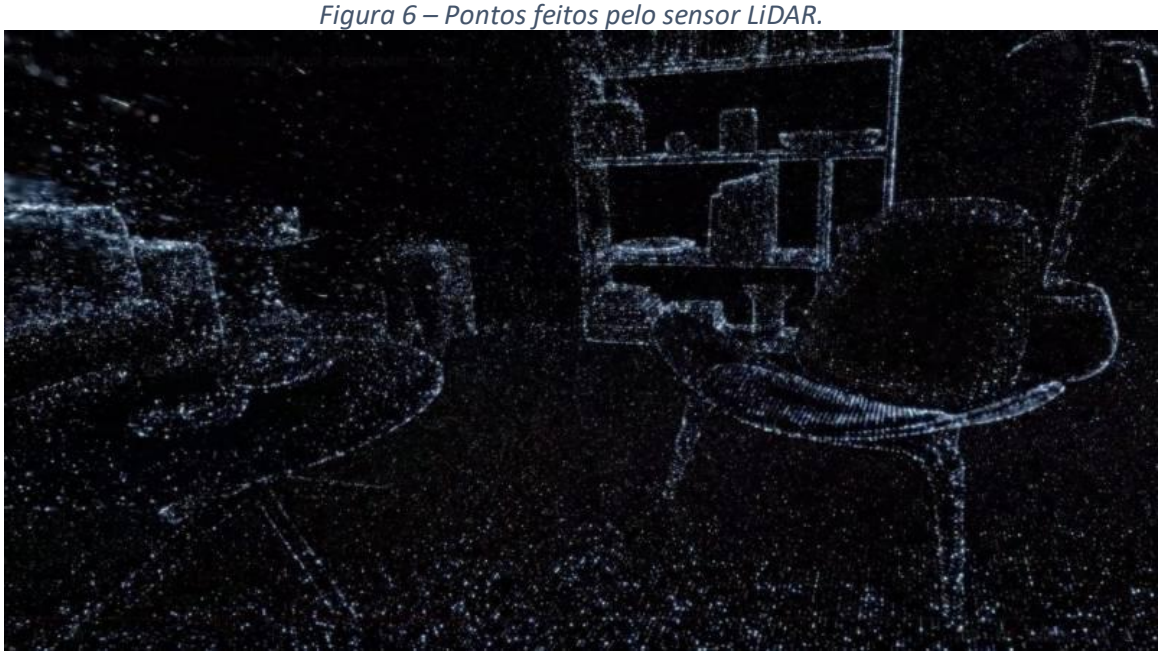
emite o sinal para o receptor em conjunto também é enviado o horário em que ele saiu do satélite.

Ao captar os sinais dos satélites, o receptor calcula a distância entre eles pelo intervalo de tempo entre o instante de envio e o instante de recebimento. Levando em conta a taxa de velocidade de propagação do sinal, possibilitando assim a identificação do local exato onde o aparelho se encontra. E para que a mesma esteja sempre atualizada, o envio desse sinal ocorre constantemente em uma velocidade de 300 mil quilômetros por segundo.

2.3.6 LiDAR

O LiDAR (Light Detection And Ranging) ou também conhecido como LADAR (*Laser Detection and Ranging*) em que utiliza um sistema de laser pulsado para poder medir propriedades da luz que é refletida. O sensor emites diversos *lasers* criando pontos conforme a Figura 6, quanto mais pontos emitidos melhor será o resultado e a qualidade do sensor. A medição é feita através do tempo de emissão do *laser* e do tempo de registro do ponto refletido pela superfície do material, fazendo com que o sensor consiga determinar a medida real do objeto.

Figura 6 – Pontos feitos pelo sensor LiDAR.



Fonte: https://www.quora.com/How-are-lidar-applications-used-to-benefit-society.

Essa tecnologia permite medir distâncias e escanear a região ao redor, formando um mapa 3D da via por onde está passando, como mostrado na Figura 7. Por possuir sua própria fonte de luz, não é afetado pela baixa luminosidade.

Figura 7 – Representação de um sensor LiDAR em um carro autônomo.

Fonte: https://medium.com/geekculture/how-lidar-fits-into-the-future-of-autonomous-driving-29fc296052bc.

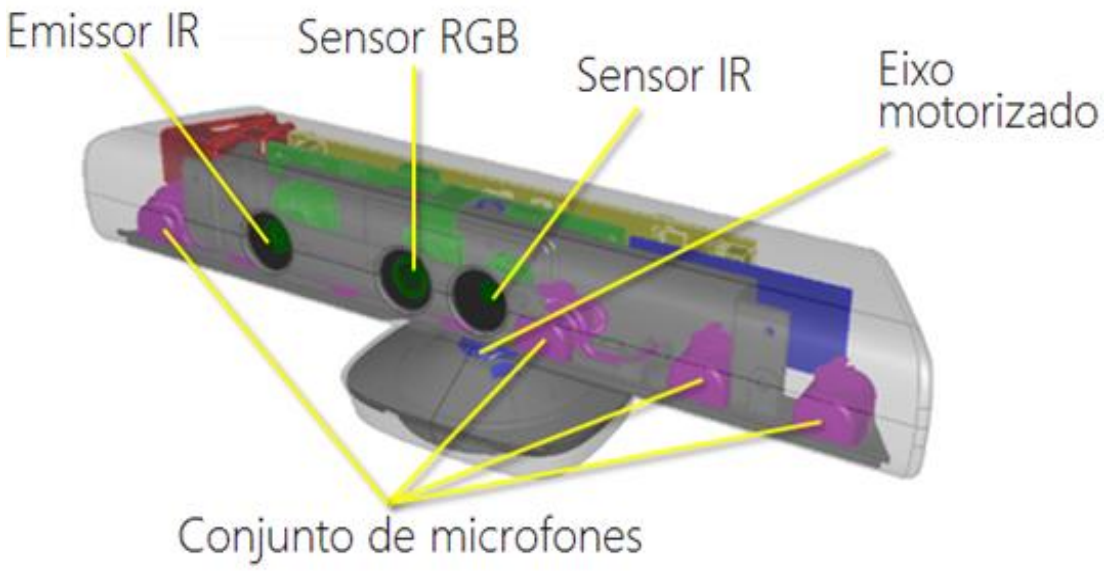
2.3.6.1 Kinect

Figura 8 – Dispositivo Kinect

Fonte: https://www.terra.com.br/gameon/o-que-e-o-kinect-5-jogos-que-usavam-o-acessorio,10ffb96b9a95d1ee977e51d8d8c1e4cb9mnvq20h

O antes denominado “Project Natal” mais conhecido como Kinect é um sensor de movimentos desenvolvido pela empresa Microsoft para os consoles Xbox360 e Xbox One, onde a tecnologia empregada ao dispositivo incorpora câmeras RGB (*Red, Green and Blue*), projetores infravermelhos e detectores os quais mapeiam a profundidade do ambiente através de cálculos estruturados de luz ou por tempo de voo, como apresentado na Figura 9. Possuindo também um conjunto de microfones que se unem ao *software* e a IA (Inteligência Artificial) da Microsoft permitindo assim a realização de reconhecimento de voz, gesto e detecção esquelética de até quatro pessoas de forma simultânea. Em suma o Kinect é um sensor de alta capacidade de captura de movimentos desenvolvido pela empresa.

Figura 9 - Representação estrutural do dispositivo Kinect



Fonte: https://www.casadocodigo.com.br/pages/sumario-kinect

2.4 Robot Operating System (ROS)

Nas palavras da própria equipe do ROS*:*

> *O Robot Operating System (ROS) é um conjunto de bibliotecas de software e ferramentas que ajudam a construir aplicativos de robô. De drivers a algoritmos de última geração e com poderosas ferramentas de desenvolvedor, o ROS tem o que você precisa para seu próximo projeto de robótica. E é tudo de código aberto.*

Teve início no ano de 2007 com o objetivo de desenvolver robôs para realizar tarefas de manipulação em ambientes humanos. Em 2008 o ROS teve sua versão inicial conhecida como Mango Tango, trazendo consigo as primeiras documentações, artigos e a vinda dos primeiros tutoriais, preparando assim um espaço para o seu lançamento em janeiro de 2010, a primeira versão oficialmente liberada sendo o ROS 1.0 virando a versão *ROS Box Turtle*, dando liberdade para a criação de mais documentação e tutoriais ensinando e comentado sobre cada funcionalidade que possuía, levando assim aos desenvolvimentos dos primeiros drone utilizando ROS, os primeiros carro autônomos rodando ROS e a adaptação do ROS para *Lego Mindstorms*.

Com a grande popularização do *software,* no ano de 2011 foi lançado o ROS *Answers,* um fórum reservado para dúvidas e respostas postadas pelos usuários que utilizavam a plataforma. Em 2012 a *Willow Garage,* responsável pelo ROS criou a *Open Source Robotics Foundation* (OSRF) e realizou no mesmo ano uma conferência de desenvolvedores ROS de todos os níveis chamada de *ROSCon*. De 2013 quando o OSRF se tornou o principal responsável pela aplicação até o ano de 2020 foram lançadas 7 novas versões do ROS, sendo a *Noetic Ninjemys* (23 de maio de 2020) a última versão do ROS 1.

Devido ao número de unidades de controle e sensores utilizados nos veículos autônomos, o uso de um *framework* como o ROS torna-se importante, pois ele implementa mecanismos de troca de mensagens entre os diversos dispositivos. A versão utilizada no projeto será o ROS Melodic Morenia, lançado em 23 de maio de 2018 com o foco na vinda do Ubuntu 18.04 lançado em 26 de abril de 2018.

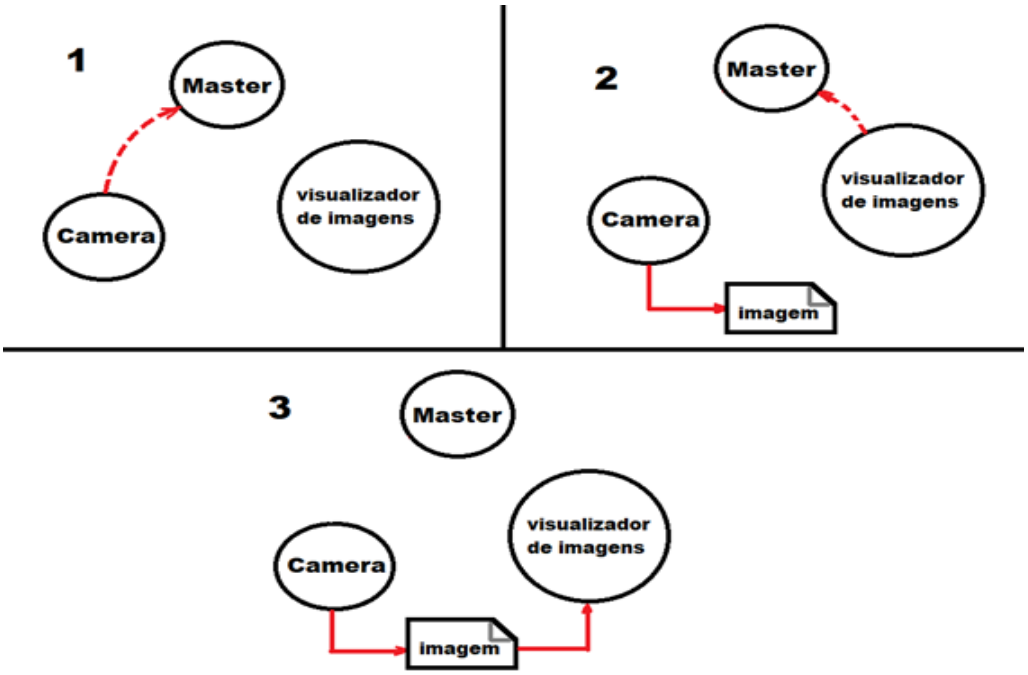
Para que o projeto possa ocorrer são necessários alguns mecanismos chamados de *service* e *master,* que faz a comunicação e permitem que os processos achem uns aos outros.

O *Master* será responsável por fornecer serviços de registros e nomes para os Nós criados e possibilita que esses Nós possam encontrar outros Nós. Além de facilitar a comunicação entre os Nós o Master funciona como um *host* que

conecta dois elementos através da topologia de rede ponto-a-ponto, permitindo a troca de mensagens diretamente.

Quando os Nós são combinados e criados formam uma espécie de grafo, onde determina o caminho em que as informações passam. No nosso projeto será utilizado uma Raspberry Pi como *master* servindo como *host* e direcionando as informações para os locais corretos, e cada sensor enviará as informações captadas exteriormente para o *master* recebendo as informações do *service* (tipo de serviço) e realizando o processamento desejado, conforme mostra a Figura 10.

Figura 10 - Representação da comunicação entre os sensores e o master

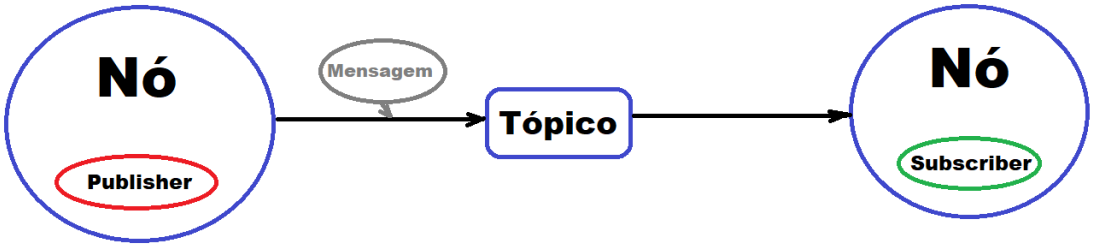


Fonte: Autor

Para que um nó se comunique com outro nó é utilizado o meio chamado no ROS de tópicos, onde são os “endereços” que os nós utilizam para a troca de mensagens, como mostra a Figura 11. Cada tópico criado possui um tipo de

mensagem específica associada, impedindo que qualquer tipo de dado seja transitado pelos tópicos.

Figura 11 – Representação da comunicação entre nós



Fonte: Autor

2.6 Raspberry Pi

Tendo início em 2006, surgiu os primeiros protótipos baseados em um microcontrolador Atmel ATmega644, utilizando de inspiração um modelo de computador antigo Acorn BBC Micro criado em 1981 e que foi utilizado para desenvolver e chegar no Raspberry Pi que é conhecido hoje.

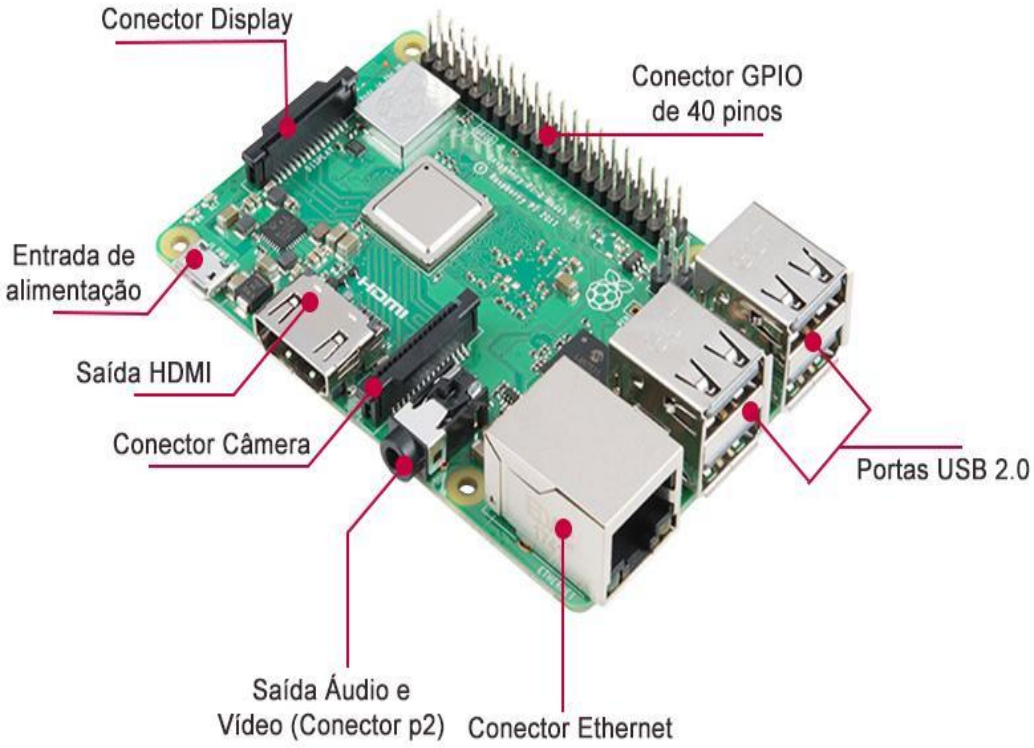
O Raspberry Pi surgiu com o objetivo de ser um computador compacto em que pudesse caber facilmente na mão, por um preço acessível e incentivar o estudo pela ciência da computação. Com o avanço dos microprocessadores e dos componentes elétricos a capacidade de processamento dos modelos lançados posteriormente se tornou cada vez maior, sendo ainda mais utilizado em projetos que exigem mais velocidade e tendo um tamanho compacto.

2.6.1 Hardware

Todos os modelos possuem a entrada de cartão micro SD (*Secure Digital*) que é utilizado para gravar e inicializar o sistema operação, entradas USB (*Universal Serial Bus*) para a conexão de dispositivos externos, entrada de alimentação, saída de imagem HDMI (*High-Definition Multimedia Interface*) e conexão à internet sem fio.

Para o projeto será utilizado o Raspberry PI 3 Model B+ lançado em 2018, tendo sua estrutura conforme a Figura 12.

Figura 12 – Componentes do Raspberry Pi 3 Model B+



Fonte: Autor

2.6.2 Especificações

Tabela 1 – Especificação do Raspberry Pi 3 Model B+

Processador	**Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz**
Memória	**1GB LPDDR2 SDRAM**
Conectividade	**2.4GHz and 5GHz IEEE 802.11.b/g/n/ac sem fio LAN, Bluetooth 4.2, BLE Gigabit Ethernet over USB 2.0 (rendimento máximo 300Mbps) 4 portas × USB 2.0**

Acesso	**Conector GPIO de 40 pinos**
Vídeo e Som	**1 × porta de exibição HDMI Porta de exibição MIPI DSI Porta da câmera MIPI CSI Saída estéreo de 4 pólos e porta de vídeo composto**
Multimidia	**H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics**
Suporte Cartão SD	**Micro SD**
Entrada de Alimentação	**5V/2.5A DC via conector micro USB 5V DC via GPIO Power over Ethernet (PoE)–enabled (requires separate PoE HAT)**
Ambiente	**Temperatura de operação, 0–50 ° C**

2.6.3 Software

A fundação Raspberry Pi disponibiliza um sistema operacional chamado de Raspbian, em que é uma versão do GNU/Linux de 32 *bit* baseada no sistema operacional Debian. Por ter uma grande liberdade de desenvolvedor o Raspberry Pi não se limita apenas ao sistema Debian, permitindo que seja instalado outros sistemas operacionais, assim como o Ubuntu que é um SO (Sistema Operacional) baseado em Linux e que será utilizado nesse projeto.

2.6.4 Unix

Em meados de 1960 Thompson e Dennis Ritchie estavam desenvolvendo um sistema operacional chamado de Multics (*Multiplexed Information and Computing Service*) que serviria com um sistema de tempo compartilhado que compasse diversos usuários, porém, por esse projeto estar à frente de seu tempo os componentes da época eram insuficientes para possibilitar a continuação, fazendo desistirem do projeto em 1969. No mesmo ano Ken Thompson começou a reescrever o Multics sem grandes ambições, chamando de Unics e utilizando

a linguagem assembly. Logo após muda o nome o rebatizando de Unix e mudando a sua linguagem para C em 1973.

> *"O que queríamos preservar não era apenas um bom ambiente para fazer programação, mas um sistema em torno do qual uma bolsa pudesse se formar", escreveu Ritchie em 1979. "Sabíamos por experiência própria que a essência da computação comunal, fornecida pelo acesso remoto, máquinas compartilhadas por tempo, não serve apenas para digitar programas em um terminal em vez de um teclado, mas para encorajar uma comunicação próxima. "*

O seu lançamento foi um grande marco para os sistemas operacionais atuando, pois o Unix era capaz de propor um sistema "multitarefas", onde a execução era tão rápida fazendo com que oferecesse a impressão de multitarefas e sendo capaz de executar dezenas de processos. Além das multitarefas deu início ao suporte a multiusuários, permitindo várias aplicações serem executadas de maneira independente e correspondente a cada usuário, recursos de rede que pudesse se comunicar com outros computadores através da rede e um sistema de arquivos eficiente. Por conta das novas propostas e inovações diversos sistemas operacionais consagrados hoje foram baseados em Unix, como GNU/Linux, Mac OS X, Solaris e BSD.

2.6.5 Linux

No ano de 1988, com 20 anos Linus Torvalds ingressou na Universidade de Helsinki no curso de ciência da computação, adquirindo conhecimento sobre a linguagem C e decidindo implementar um terminal no seu computador 80386 para poder ter acesso ao servidor da faculdade que utilizava Unix. Por usar o Minix e não gostar do *software* decidiu criar um projeto para obter um sistema que conseguisse acessar o servidor da instituição e que fosse de seu agrado.

Em 1991 Linus Torvalds começou a desenvolver por *hobby* o Linux como uma variação do sistema operacional Minix que foi criado por Andrew S. Tanenbaum baseado em Unix. nesse ano Linus publicou mensagens na Usenet solicitando sugestões e melhorias que a comunidade gostaria de ter no minix. O Nome Linux

surgiu com a mistura de seu próprio nome “Linus” com “Unix” *software* em que foi inspirado.

2.6.5.1 GNU/Linux

GNU (*GNU is Not Unix*) traduzindo significa “GNU Não é Unix” foi um projeto iniciado no ano de 1984 por Richard Stallman onde seu principal objetivo era desenvolver um sistema compatível com o Unix, porém sem usar o código do próprio Unix. Este projeto surgiu com o intuito de ser um sistema operacional de forma totalmente livre, onde o usuário possa ter total liberdade para executar, copiar, modificar e aperfeiçoar.

Em 1992, o GNU estava quase completo, porém faltava algo muito importante, o núcleo ou também chamado de “*kernel*” (componente central do SO que serve de ponte entre o processamento e os aplicativos), já estava sendo desenvolvido um núcleo pelo Richard Stallman chamado de Hurd, porém por conta da demora de desenvolvimento decidiram utilizar um núcleo compatível com a GPL (*General Public License*) do GNU e com isso foi escolhido o Linux por ser capaz de utilizar todas as ferramentas e permitindo um sistema com o código-fonte totalmente aberto e livre para os usuários.

2.6.5.2 Distribuições do Linux

Pelo Linux possuir um código-fonte aberto, com o passar dos anos diversas empresas foram criando seus próprios SO customizados inspirados em Linux e com essas versões criadas obtendo o nome de “distribuições do Linux”, essas distribuições utilizam bibliotecas e ferramentas do GNU, em que cada distribuição é composta de aplicativos mais o núcleo que é o Linux.

Cada distribuição possui tamanho, ferramentas e finalidades diferentes, cada uma variando conforme o seu objetivo de uso e cada uma com uma forma de instalação diferente. A distribuição mais famosa e a versão que será utilizado nesse projeto é o Ubuntu da empresa Canonical.

2.6.5.3 Ubuntu

Criado em 2004 por Mark Shuttleworth fundador da empresa Canonical, Mark notou o potencial no Linux por ser um *software* com o código fonte aberto e a dificuldade de usuários leigos de utilizar outras distribuições do Linux populares na época, então decidiu fazer sua própria distribuição. Com base em outra distribuição chamada Debian começou a desenvolver o Ubuntu possuindo uma interface mais fácil de entender de forma mais amigável para o usuário final que fosse simples de aprender a utilizar.

O primeiro lançamento do Ubuntu foi em 20 de outubro de 2004 com o codinome "*Warty Warthog*", esta versão possuía o versionamento 4.10 que é baseado no ano e após o mês do lançamento, enquanto o codinome é inspirado em adjetivos do nome de animais.

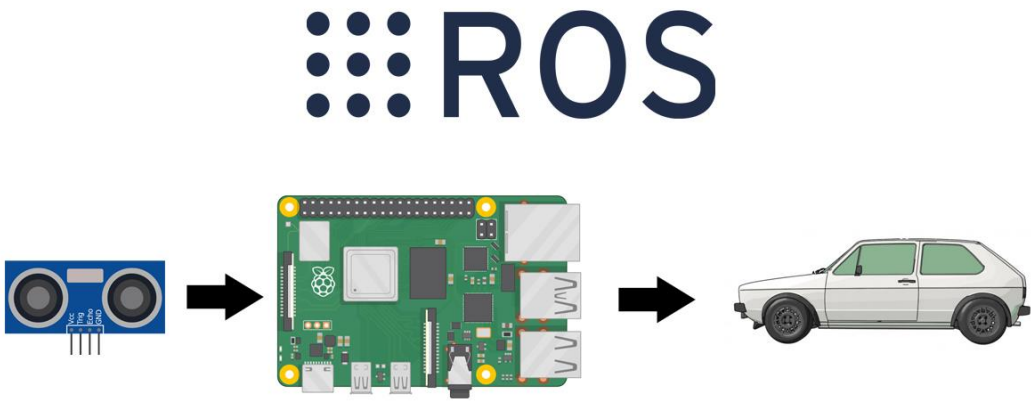
3. Metodologia

Este capítulo irá abordar as preparações, os métodos utilizados e o desenvolvimento utilizado para realizar o projeto. O projeto de aplicar o ROS em um protótipo de veículo autônomo estende diversas áreas, tais como, programação, eletrônica e eletroeletrônica, cada área pode haver mudanças com o tempo.

3.1 Metodologia do projeto

Os procedimentos utilizados no projeto tiveram como principal objetivo a coleta de informações através dos sensores, envio dessas informações pelos tópicos do ROS e a tomada de decisão pelo mestre em que nesse caso é o Raspberry Pi. O projeto pode ser interpretado seguinte estrutura mostrada pela Figura 13.

Figura 13 – Estrutura do projeto



Fonte: Autor

3.4 Arduino

Criado em 2005 pelo professor Massimo Banzi na Itália, o Arduino surgiu com o objetivo de ensinar os alunos conceitos de eletrônica e de programação, incentivando o desenvolvimento de novos projetos na área de robótica, porém não encontrava placas para utilizar em que tivesse um baixo custo ou com esquemas simplificados para os seus alunos. Por esses motivos Banzi desenvolveu e criou uma placa de baixo custo que fosse facilmente programável e com grande funcionalidade.

O Arduino é uma placa de prototipagem eletrônica e de código aberto, possibilitando o desenvolvimento de diversos projetos na área da robótica. O Arduino é desenvolvido por meio da linguagem C/C++ e seu *hardware* é composto por um microcontrolador Atmel AVR de 8 bits, algumas entradas digitais e analógicas e uma interface serial ou USB, como visto na Figura 14.

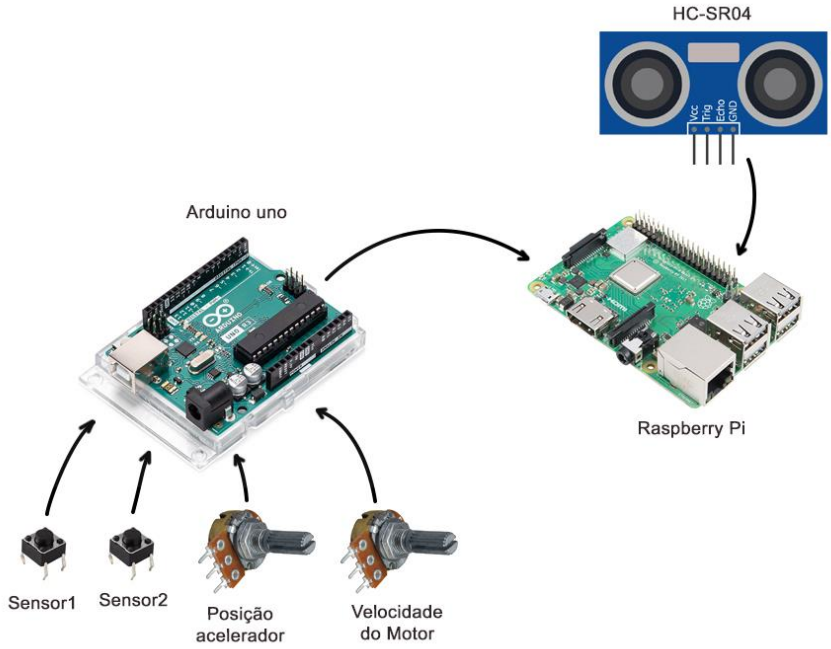
Sendo muito conhecido o Arduino se destaca pela liberdade de criação, possibilidade de desenvolver algo novo e ter uma *Interface* de fácil entendimento e por possuir uma estrutura compacta e de fácil instalação, podendo transportar facilmente para outros lugares.

Figura 14 – Placa Arduino

Fonte: https://store-usa.arduino.cc/products/arduino-uno-rev3

3.4.1 Arquitetura de Hardware

Figura 15 – Arquitetura projetada do projeto



Fonte: Autor

Como ilustrado na Figura 15, para o recebimento das informações dos sensores presentes será utilizado o Raspberry Pi 3 Model B que será o *master* na comunicação do ROS e o responsável pelas decisões a serem tomadas de acordo com as informações recebidas. Em conjunto com o Raspberry Pi será utilizado um Arduino Uno por possuir uma fácil prototipagem e maior familiaridade com os sensores utilizados.

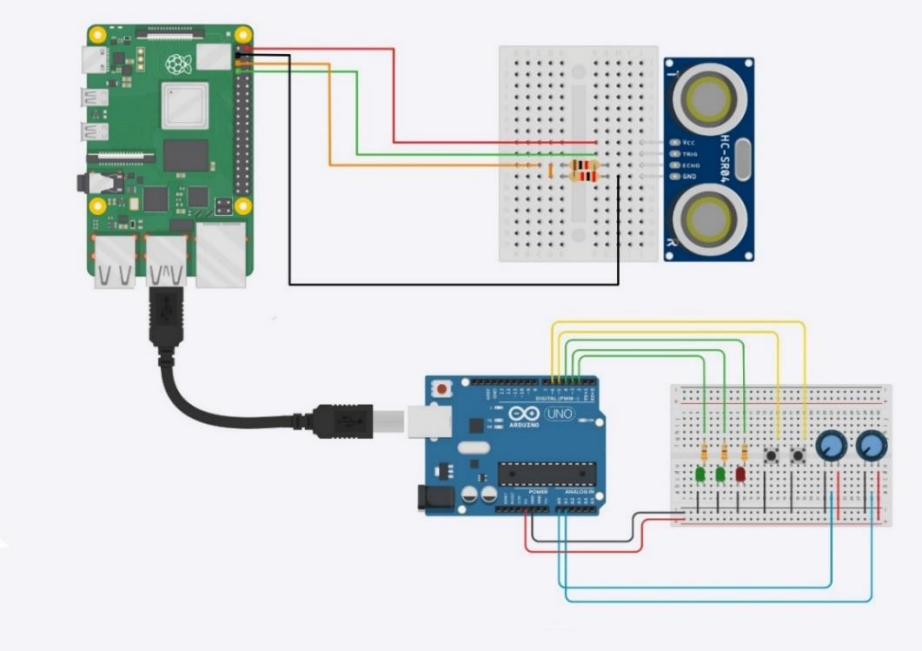
Na arquitetura de *hardware* o Arduino receberá informações de 4 sensores, sendo eles dois botões que simularão 2 sensores e 2 potenciômetros em que simularão a posição do acelerador e a velocidade do motor respectivamente, por serem os materiais acessíveis para o grupo. As informações desses sensores irão ser publicadas do Arduino por um nó e sendo recebidas pelo Raspberry Pi. O Raspberry Pi recebera as informações do sensor ultrassônico publicando essas informações e recebendo as informações de todos os sensores em um único nó que guardará essas informações e fará um processamento tomando uma decisão.

Para a montagem do projeto será utilizado:

- 2 Protoboards;
- 1 Sensor ultrassônico HC-SR04;
- 23 jumpers;
- 1 resistor de 1k Ohms;
- 1 resistor de 2k Ohms;
- 1 resistor de 560 Ohms;
- 2 resistores de 470 Ohms;
- 2 LEDs verdes e 1 vermelho;
- 2 botões;
- 2 potenciômetros;
- 1 Arduino Uno;
- 1 Raspberry Pi 3 Model B+.

3.4.2 Circuito elétrico

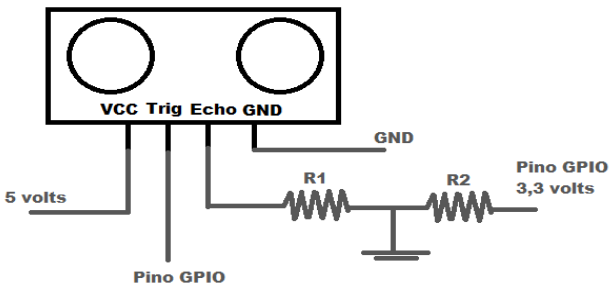
Figura 16 – Circuito elétrico do projeto



Fonte: Autor

Para a montagem do circuito elétrico do projeto, foi separado em duas partes, onde a primeira parte o Raspberry Pi que receberá as informações do sensor ultrassônico HC-SR04 diretamente, como demonstrado na Figura 16. Por esse sensor receber e retornar um sinal 5 volts é necessário utilizar resistores para reduzir a tensão do sinal de retorno, já que a placa do Raspberry Pi só pode receber um sinal de 3,3 volts. É possível ver a conexão do sensor na Figura 17.

Figura 17 – Circuito elétrico do sensor HC-SR04



Fonte: Autor

Ao adicionar um valor de 1k Ohms para o resistor R1 resta descobrir o valor do resistor R2 para que o valor de 5 volts da entrada saia com 3,3 volts, obtendo a seguinte fórmula:

$$VX = (\frac{R2}{R1+R2}) \times 5V$$

$$\frac{VX}{5V} = \left(\frac{R2}{R1+R2}\right)$$

Considerando:

$$R1 = 1k\ ohms$$

$$VX = 3.3\ V$$

Substituindo:

$$\frac{3.3}{5} = \frac{R2}{1K+R2}$$

$$0.66 = \frac{R2}{1K+R2}$$

$$0.66\ \times (1000 + R2) = R2$$

$$660 + (0.66\ \times\ R2) = R2$$

$$660 = R2 - 0.66R2$$

$$660 =\ 0.34R2$$

$$R2 = \frac{660}{0.34}$$

$$R2 = 1941.17\Omega \qquad (1)$$

Portanto ao utilizar um resistor de 1000 Ohms é necessário utilizar outro de 1941.17 Ohms. Seguindo este cálculo, foi usado o resistor mais próximo que o grupo tinha, sendo assim foi escolhido um resistor de 2000 Ohms.

Na segunda parte, o Arduino Uno será o mediador entre os outros sensores e o Raspberry Pi e ao acionar os botões, os LEDs equivalentes irão ligar demostrando exteriormente que a ação foi executada.

Para utilizar os LEDs é necessário utilizar resistores para não prejudicar a placa Arduino. O valor desse resistor é descrito pela seguinte equação:

$$R = \frac{(Valimentação - Vled)}{I} \qquad (2)$$

Onde:

- R: Resistor;
- Valimentação: Tensão de alimentação;
- Vled: Tensão do led;
- I: Corrente do led em amperes;

Levando em consideração as seguintes informações segundo a Figura 18

Figura 18 – tensão e corrente dos LEDs

Cor do led	Faixa de tensão	Corrente máxima
Vermelho	1,8 V - 2,0 V	20 mA
Amarelo	1,8 V - 2,0 V	20 mA
Laranja	1,8 V - 2,0 V	20 mA
Verde	2,0 V - 2,5 V	20 mA
Azul	2,5 V - 3,0 V	20 mA
Branco	2,5 V - 3,0 V	20 mA

Fonte: https://aprendendoeletrica.com/como-calcular-o-resistor-para-um-led/

Calculando o valor do resistor para o LED vermelho:

$$Rvermelho = \frac{12 - 2}{0{,}02} = 500\Omega \qquad (3)$$

Sendo assim o resistor comercial mais próximo de 500 Ohms o resistor de 560 Ohms.

Calculando o valor do resistor para o LED verde:

$$Rverde = \frac{12 - 2{,}5}{0{,}02} = 475\Omega \qquad (4)$$

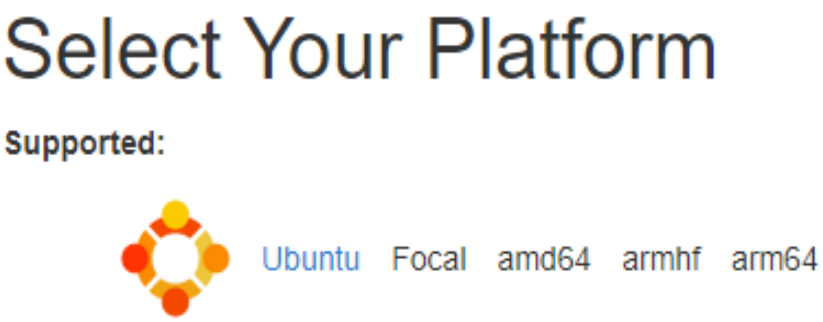
Sendo assim o resistor comercial mais próximo do valor de 475 Ohms é o resistor de 470 Ohms.

3.5 Instalação do ROS

Nesta fase do processo é necessário a instalação do ROS na máquina, para isto será utilizado o Raspberry PI 3 Model B+ como anteriormente dito como a máquina de instalação. Após a instalação do sistema operacional Ubuntu no minicomputador foi possível a iniciação da preparação do sistema para recebimento dos pacotes.

Dentro da máquina virtual foi acessado o site "http://wiki.ros.org/melodic/Installation/Ubuntu", um site qual apresenta uma gama de informações vasta sobre o Ros e suas variáveis utilizadas em diferentes sistemas operacionais. Como o selecionado para este trabalho foi o sistema Ubuntu foi procurado pela versão disponibilizada para o mesmo, como mostra na Figura 19. Após isto foi iniciado o "Terminal", um executável do próprio sistema sendo o qual recebe comandos diretamente via programação, similar ao visto "CMD" (Prompt de Comando).

Figura 19 – Apresentação da seleção de plataforma

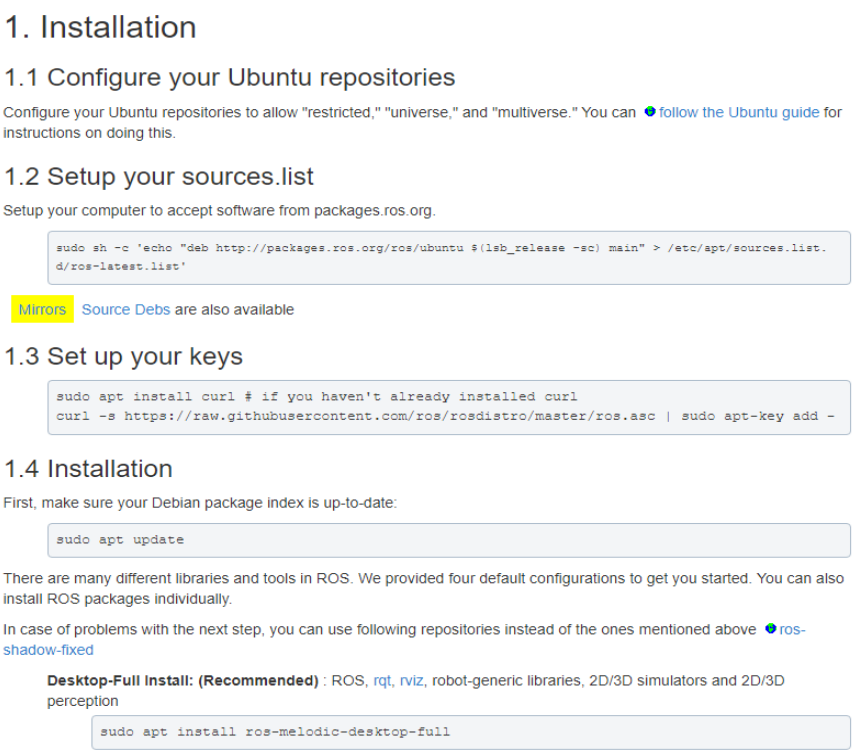


Fonte: http://wiki.ros.org/noetic/Installation

Pelo "Terminal" aptget pode-se copiar e adicionar os pacotes de recursos disponibilizados no site, executando os mesmos no terminal e configurando assim nossa "Source.list". Após adicionar os pacotes de recursos é necessário adicionar pacotes de chaves ao mesmo, posteriormente é necessária uma atualização dos dados do terminal sendo assim necessário a utilização da sintaxe *"sudo apt update",* tendo assim um terminal com lista de programas e repositórios atualizada permitindo agora de fato a inicialização da instalação de fato.

Como mostra a Figura 20, neste mesmo site é possível observar a disponibilidade de três tipos de instalação a utilizada neste trabalho foi a chamada "*Desktop Full*" pois abrange mais funções e recursos a serem utilizados, para iniciar a instalação da mesma é necessário a introdução da sintaxe *"sudo apt install ros-noetic-desktop-full"* ao terminal. Após total instalação dos pacotes, é necessário a inclusão de uma nova função chamada "*setup.bash"*, para possibilitar a utilização de algumas das funções as quais utilizaremos neste trabalho.

Figura 20 – Apresentação do layout guia de instalação



Fonte: http://wiki.ros.org/melodic/Installation/Ubuntu

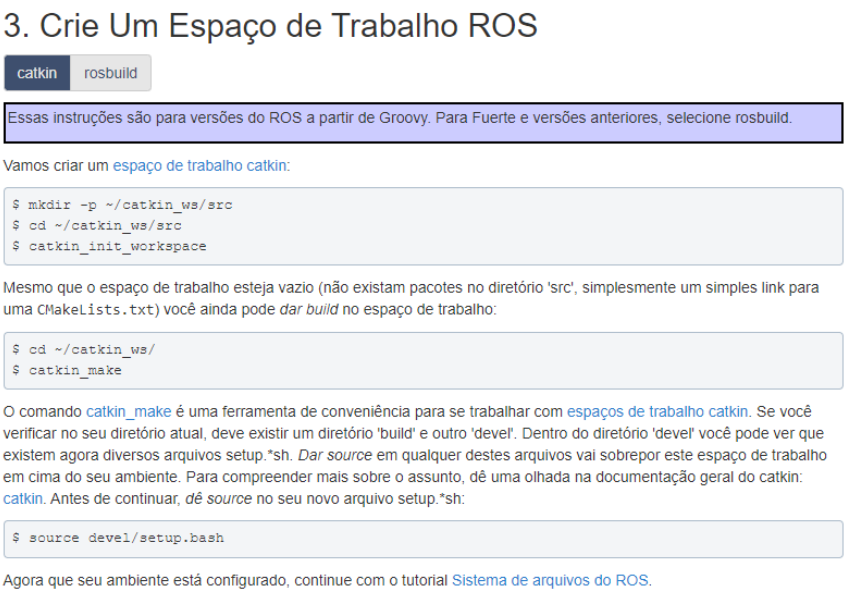
Após a introdução do Bash, é executado o mesmo via terminal de forma manual introduzindo a sintaxe *"sudo gedit ~/.bashrc"* e adicionando os pacotes de recurso do "*setup.bash"* ao final do eco do bash. Adicionando assim os comandos ao terminal possibilitando a execução do Roscore e podendo assim verificar o funcionamento do mesmo através das sintaxes "*rosrun roscpp_tutorials talker" e "rosrun roscpp_tutorials listener",* tendo assim o talker como publicador e o listener verificando essas mesmas publicações enviadas em tempo real.

3.5.1 Preparação do Ambiente ROS

Para realizar um novo projeto e começar a desenvolver é necessário preparar e configurar um novo ambiente do ROS, este ambiente é composto em três principais pastas, sendo elas a pasta “src” que é responsável por ter todos os arquivos fontes dos pacotes criados, a pasta “*build*” que terá os arquivos que serão utilizados para fazerem as compilações e a pasta “*devel*” onde haverá diversos *scripts,* sendo o mais importante e o mais utilizado o “setup.bash” que é necessário utilizar todas vezes que é aberto um novo terminal para poder utilizar os pacotes instalados para o ROS.

Para a criação do Ambiente de trabalho desse projeto foi utilizado o guia visto no site oficial do ROS, como mostra a Figura 21.

Figura 21 – Guia de criação de um espaço de Trabalho ROS



Fonte: http://wiki.ros.org/pt_BR/ROS/Tutorials/InstallingandConfiguringROSEnvironment

3.6 Arquitetura de Software

A arquitetura de *software* consiste nos arquivos utilizados durante o desenvolvimento do projeto e da estrutura de comunicação utilizado pelo *framework* ROS para transitar as informações dos sensores.

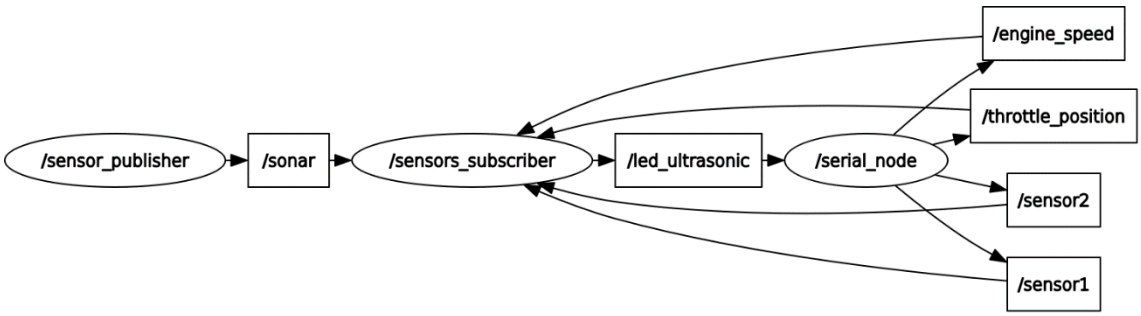
Para a estrutura de comunicação é necessário que uma máquina tenha o ROS instalado e rodando o comando ROSCORE, está máquina será conhecida como o mestre e responsável por inicializar todos os serviços necessários para que os nós ROS possam se comunicar entre si. É possível ter diversos nós ativos simultaneamente e em diversas máquinas da mesma rede. Cada nó criado pode receber ou enviar mensagens através de tópicos. Os tópicos são endereços que servem para direcionar o caminho entre o nó que está publicando e o nó que está recebendo a informação, impedindo que as informações sejam dispersadas entre os nós aleatoriamente.

A Figura 22 e Figura 23 mostram como foi feita a comunicação entre os nós.

Figura 22 – Estrutura de comunicação

Fonte: Autor

Figura 23 – Estrutura de comunicação com os tópicos detalhados



Fonte: Autor

Para realizar essa arquitetura foi desenvolvido três arquivos, no qual cada arquivo é um nó. O primeiro arquivo é o “sonar_publisher.py”, desenvolvido em linguagem python e responsável pelo nó “/sensor_publisher” que publica as informações capturadas pelo sensor ultrassônico através do tópico “/sonar”. O segundo arquivo é o “arduino_publisher.ino” no qual foi desenvolvido em

linguagem C e utilizando o próprio *software* de desenvolvimento do Arduino e compilado na placa Arduino Uno, esse arquivo é responsável pelo nó criado “/seria_node” que recebe as informações diretamente dos demais sensores e publica nos tópicos “/engine_speed”, “/throttle_position”, “/sensor1” e “sensor2”. Além de publicar as informações dos sensores, esse arquivo também possui um *subscriber* que recebe informações do tópico “/led_ultrasonic”. Para iniciar qualquer nó que esteja conectado por meio serial é necessário utilizar o comando “rosrun rosserial_python serial_node.py /dev/ttyUSB0”, onde “/dev/ttyUSB0” é a porta serial utilizado no projeto.

O terceiro arquivo foi denominado “sensors_subscriber.py”, desenvolvido em linguagem python, esse arquivo recebe as informações de todos os nós que estão publicado os dados obtidos dos sensores e armazenando em variáveis que serão utilizadas para determinar as ações a serem tomadas, após o processamento de decisão, as informações serão imprimidas na tela. Foi colocado um tópico de publicação chamado “/led_ultrasonic” que enviará um sinal para o Arduino que acenderá um led caso o sensor ultrassônico detecte algum objeto próximo.

4. Resultados e Análises

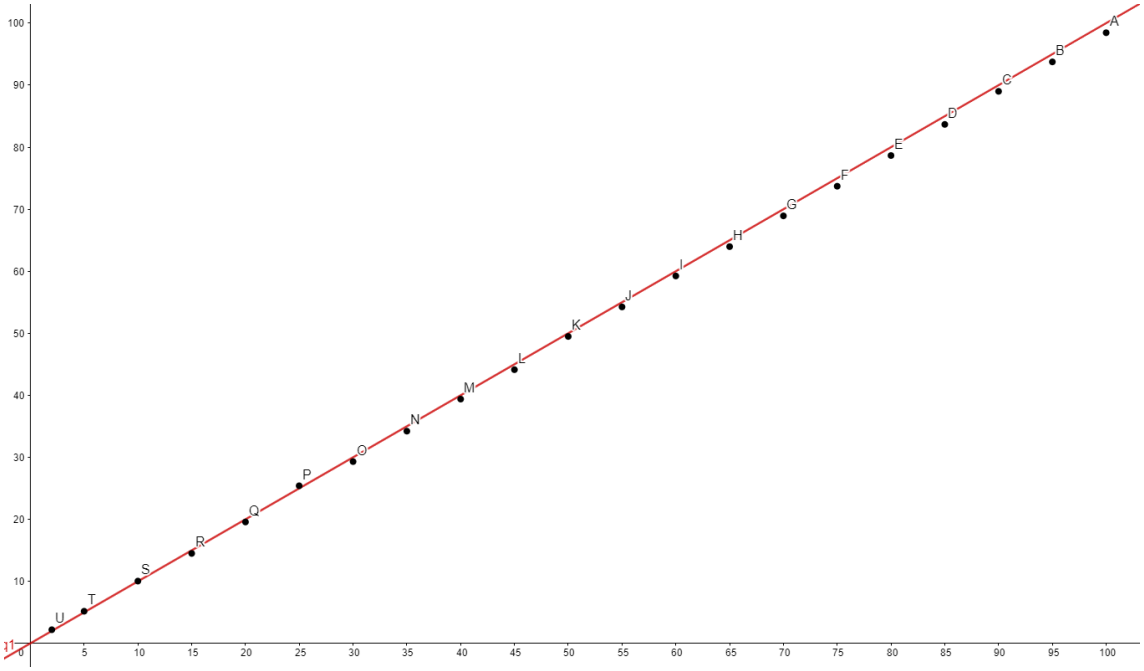
Os testes foram realizados com o propósito de medir as resoluções e capacidades máximas dos sensores utilizados, a velocidade de resposta do nó.

4.1 Sensor ultrassônico HC- SR04

De com os testes realizados e as especificações, o sensor possui uma resolução de 0.3 cm, trabalhando a partir de 2 cm de distância, podendo chegar a uma distância de 200 cm e tendo um ângulo de eficácia de 15º graus.

Para o primeiro teste foi feito um gráfico de medida real por medida capturada, visto na Figura 24, onde é possível ver a divergência entre o ideal e o averiguado.

Figura 24 – Resultado da medição do sensor HC-SR04



Fonte: Autor

Esses valores utilizados foram as médias dos valores obtido em diversas repetições do mesmo teste.

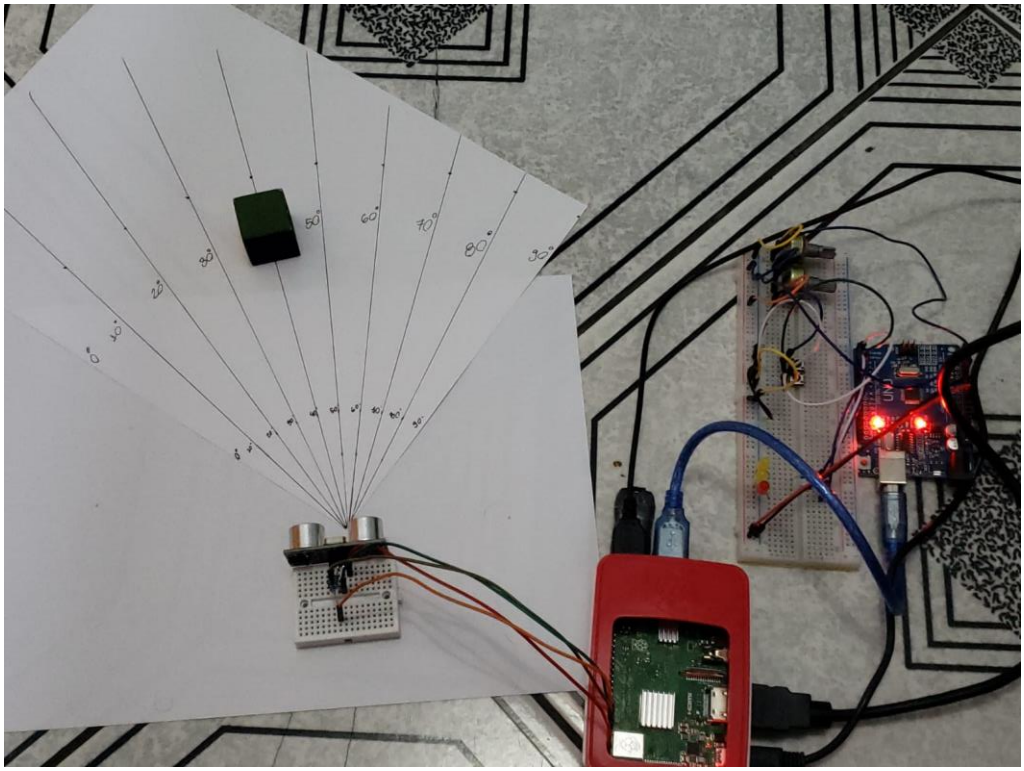
Na Tabela 2 é possível notar através dos testes que a partir da distância de 60 cm o erro começa a diminuir e conforme mais próximo está o objeto esse erro diminui mais.

Tabela 2 – Medições do Sensor HC-SR04

Medida real (cm)	Média das medições (cm)	Erro médio (cm)
100	98,45	1,55
95	93,75	1,25
90	89	1
85	83,67	1,33
80	78,65	1,35
75	73,7	1,3
70	68,92	1,08
65	63,96	1,04
60	59,23	0,77
55	54,24	0,76
50	49,47	0,53
45	44,12	0,88
40	39,37	0,63
35	34,21	0,79
30	29,3	0,7
25	24,55	0,45
20	19,56	0,44
15	14,49	0,51
10	10,03	0,03
5	5,17	0,17
2	2,19	0,19

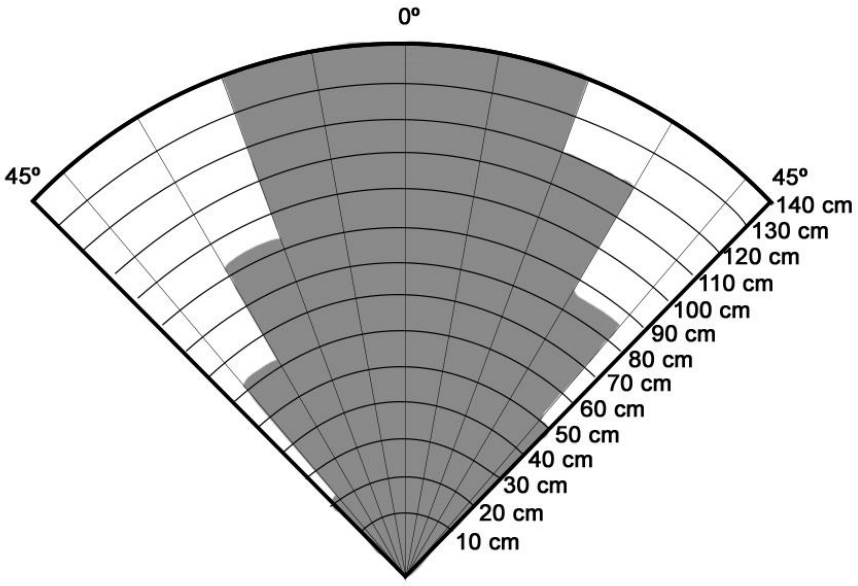
Para o segundo teste foi feito uma demarcação diante do sensor onde foram utilizados ângulos de 0 graus a 90 graus com intervalos de 10 graus entre cada um, como mostra na Figura 25. Esse teste possui a finalidade de verificar a tolerância de ângulo de medição.

Figura 25 – Teste de tolerância de ângulo



Fonte: Autor

Figura 26 – Resultado do teste de tolerância de ângulo do sensor HC-SR04



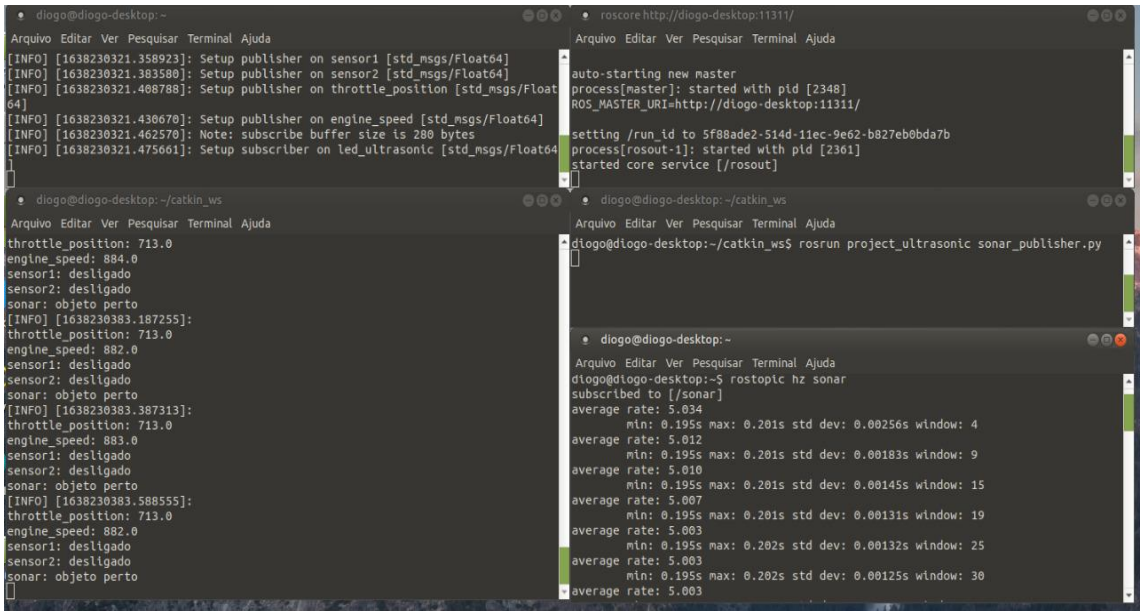
Fonte: Autor

Na Figura acima pode-se notar que o lado direito obteve melhores resultado quando feito os testes de tolerância de ângulo, isso acontece pois o transmissor do sensor HC-SR04 se localiza do lado direito e o receptor do lado esquerdo, tendo assim uma diferença de alcança entre eles.

4.2 Resultados durante a execução do software

Durante a execução é possível ver a frequência em que o resultado é emitido. Esses testes foram realizados iniciando o ROS e iniciando cada nó manualmente em terminais separados.

Figura 27 – Execução do projeto

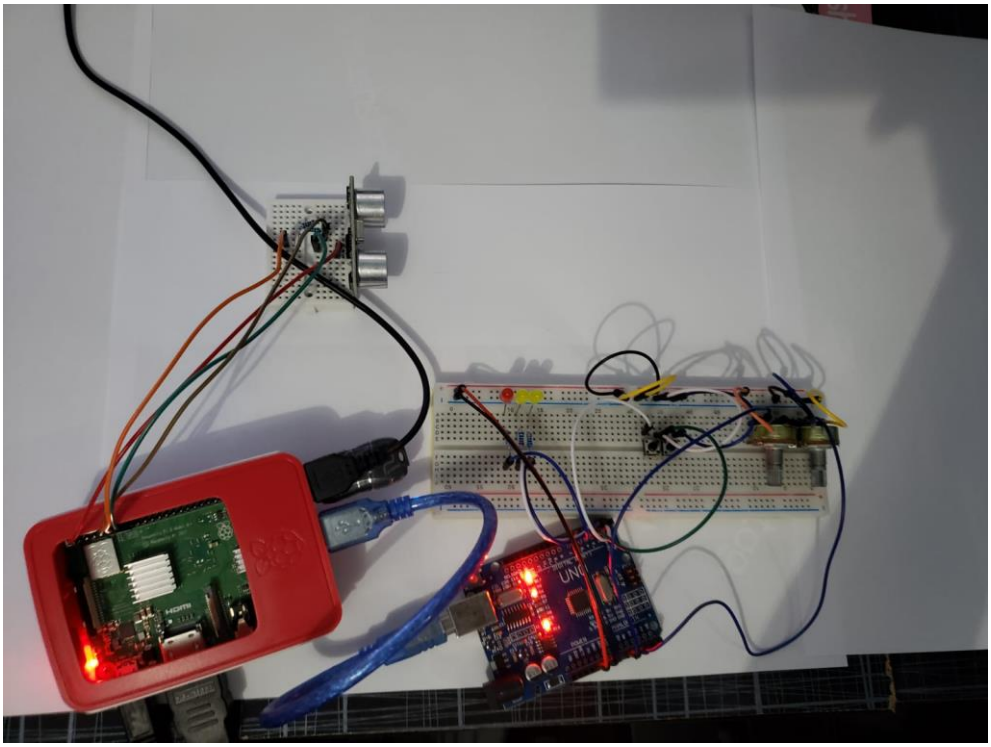


Fonte: Autor

No terminal que está no canto inferior direito é possível notar a frequência da resposta, o tempo mínimo de 0.195 segundos e o tempo máximo de reposta de 0.201 segundos, dando uma frequência de 5Hz.

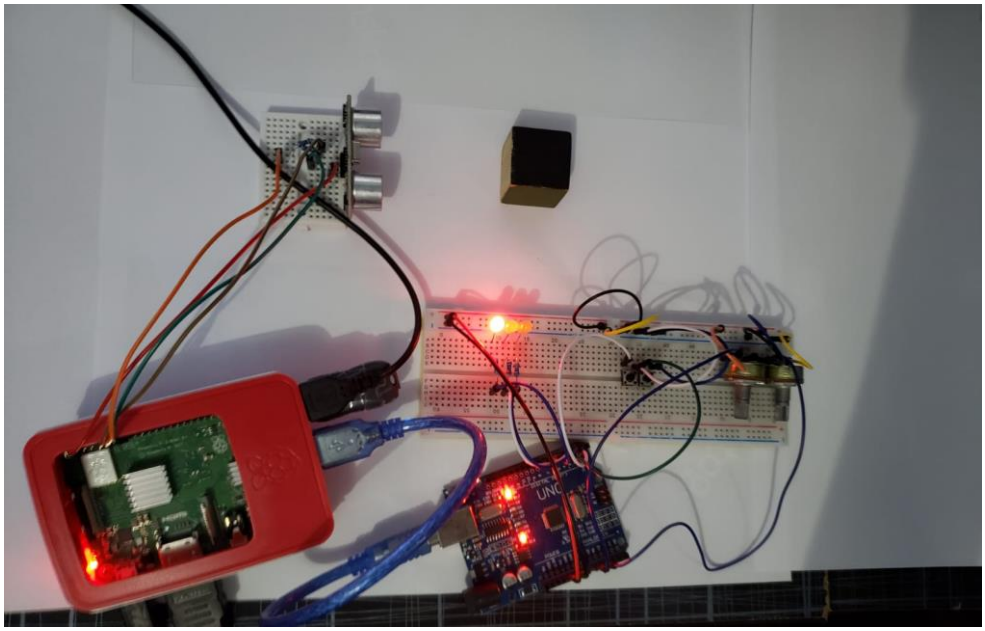
Com o *software* em execução é possível também observar os LEDs ligando quando ocorresse alguma ação dos sensores, como o exemplo mostrado na Figura 27 e Figura 28.

Figura 28 – Teste sem obstáculo



Fonte: Autor

Figura 29 – Teste com obstáculo



Fonte: Autor

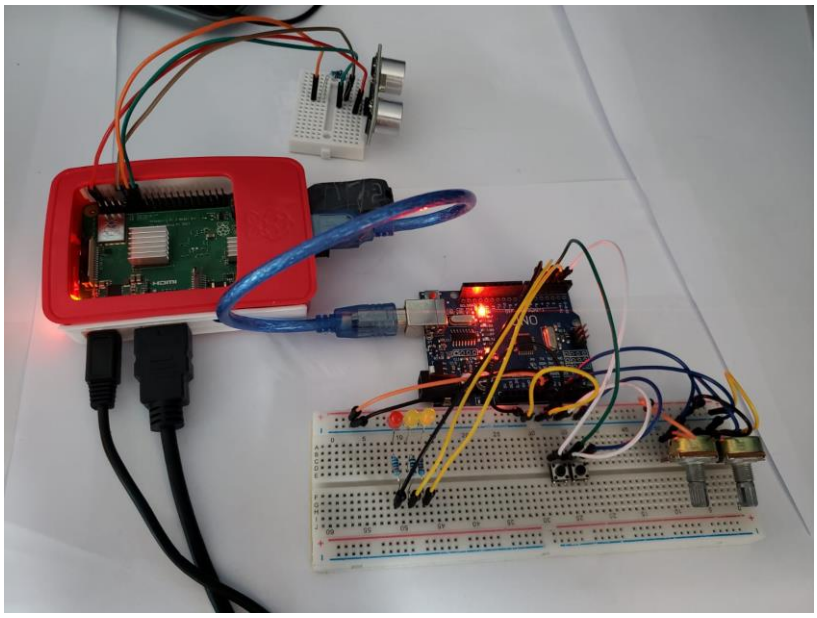
5. Conclusão

Com a proposta inicial de aplicar o ROS em um projeto de veículo autônomo em escala, foi possível entender sobre o funcionamento do *framework* ROS, a sua história, como é feito a comunicação entre seus nós e o desenvolvimento utilizando a ferramenta.

Durante todo o desenvolvimento foi utilizado ferramentas, *hardwares* e sensores que estavam à disposição do grupo, foi utilizado a linguagem python para a parte de programação pela familiaridade dos integrantes com a linguagem de programação e para a escolha da versão do ROS foi feito um estudo sobre a capacidade de processamento da placa Raspberry Pi e foi verificado as versões que possuem atualmente atualizações e suporte. Na parte de composição foi feito a comunicação dos sensores de duas maneiras, diretamente com a placa Raspberry Pi e tendo o Arduino Uno como Intermediador, mostrando um pouco sobre a capacidade do *framework* em fazer a troca de informações com elementos externos.

Foi feito um repositório *online* contendo todos os códigos que foram elaborados referente ao trabalho e todos os registros obtidos durante o desenvolvimento.

Figura 30 – projeto



Fonte: https://github.com/diogo-antonio57/TCC_Aplicacao_ROS_Veiculo_Autonomo

5.1 Propostas futuras

Para o aprimoramento deste projeto com base nos resultados obtidos são:

- Criação de um protótipo de veículo autônomo em escala
  Com um protótipo é possível explorar mais da ferramenta e do projeto desenvolvido.
- Implementação dos sensores, Lidar, Câmera, Radar e GPS
  Buscando explorar e tornar o mais próximo possível de um veículo autônomo é necessário que sejam implementados os sensores presentes em veículo autônomo real.
- Inserir um módulo *Wi-fi* para poder acessar as informações do veículo pelo celular;
  Hoje em dia ter tudo controlado na palma da mão tem se tornado mais realidade.

6 Referências

Primeiro projeto de carro autônomo data de 1920. Disponível em: <https://summitmobilidade.estadao.com.br/carros-autonomos/primeiro-projeto-de-carro-autonomo-data-de-1920/> Acesso em: 09 de maio, 2021

Carnegie Mellon University. **From 0-70 in 30**. Disponível em: < https://www.cmu.edu/homepage/environment/2014/fall/from-0-70-in-30.shtml> Acesso em: 09 de maio, 2021

Geddes, Norman Bel. **General Motors Highways & horizons : New York World's Fair**. Disponível em:<https://archive.org/details/generalmotorshig00geddrich/mode/2up> Acesso em: 09 de maio, 2021

Korosec, Kirsten. **Nuro's new delivery R2 bot gets the first driverless vehicle exemption from feds**. Disponível em: < https://techcrunch.com/2020/02/06/nuros-new-delivery-r2-bot-gets-the-first-driverless-vehicle-exemption-from-feds/> Acesso em: 09 de maio, 2021

RD80s Staff. **NavLab: The Self-Driving Car of the '80s**. Disponível em: < https://www.cmu.edu/homepage/environment/2014/fall/from-0-70-in-30.shtml> Acesso em: 09 de maio, 2021

Santino, Renato. **Este é o primeiro carro autônomo autorizado a circular sem supervisão humana**. Disponível em: <https://olhardigital.com.br/2020/02/07/carros-e-tecnologia/este-e-o-primeiro-carro-autonomo-autorizado-a-circular-sem-supervisao-humana/> Acesso em: 09 de maio, 2021

John J. **UGV History 101: A Brief History of Unmanned Ground Vehicle (UGV) Development Efforts**. Disponível em: <https://apps.dtic.mil/sti/citations/ADA422845> Acesso em: 09 de maio, 2021

Felipe, Luiz. **Desenvolvimento de Sistemas de Navegação Autônoma por GNSS**. Disponível em: <https://www.teses.usp.br/teses/disponiveis/3/3138/tde-19072011-

162537/publico/Dissertacao_Felipe_Goncalves.pdfl> Acesso em: 15 de maio, 2021

Oliveira, Luciane. **APLICAÇÃO DE TÉCNICAS DE APRENDIZADO PARA O CONTROLE INTELIGENTE DE VEÍCULOS AUTÔNOMOS**. Disponível em: <http://osorio.wait4.org/oldsite/alunos/tcc/lfortes-tc.pdf> Acesso em: 18 de maio, 2021

Amadeu, Tiago. **ARQUITETURA DE HARDWARE E SOFTWARE PARA SUPERVISÃO E CONTROLE DE UM CARRO AUTÔNOMO**. Disponível em: <https://www.ppgee.ufmg.br/defesas/195M.PDF> Acesso em: 23 de maio, 2021

Koubaa, Anis. **Services**. Disponível em: <http://wiki.ros.org/Services> Acesso em: 1 de junho, 2021

Wu, Yanqing. **Master**. Disponível em: <http://wiki.ros.org/Master> Acesso em: 1 de junho, 2021

Benoni, Alyson. **ROSRemote: Utilizando ROS para acesso remoto a robôs**. Disponível em: <https://repositorio.unifei.edu.br/xmlui/bitstream/handle/123456789/1199/dissertacao_pereira_2018.pdf?sequence=1&isAllowed=y> Acesso em: 1 de junho, 2021

ROS team. **ROS.** Disponível em: <https://www.ros.org/> Acesso em: 1 de junho, 2021

Acas, Guilherme; Baratera, Lucca. **ROBÔ GUIADO MAPEADOR DE AMBIENTES INTERNOS**. Disponível em: <http://fatecsantoandre.edu.br/arquivos/TCC/162-Mecatronica/162-TCC0011.pdf> Acesso em: 1 de junho, 2021

ROS team. **History**. Disponível em: <https://www.ros.org/history/> Acesso em: 1 de junho, 2021

ROS Tutorials and Turtles. Disponível em: <https://www.ros.org/news/2009/12/ros-tutorials-and-turtles.html> Acesso em: 1 de junho, 2021

Welcome to ros.org. Disponível em: <https://www.ros.org/news/2009/08/welcome-to-ros-org.html> Acesso em: 1 de junho, 2021

ROS 0.4 Release. Disponível em: <https://www.ros.org/news/2009/02/ros-04-release.html> Acesso em: 1 de junho, 2021

Robots Using ROS: Marvin autonomous car (Austin Robot Technology/UT Austin). Disponível em: <https://www.ros.org/news/2010/03/robots-using-ros-marvin-autonomous-car.html> Acesso em: 2 de junho, 2021

Robots Using ROS: Penn Quadrotors. Disponível em: <https://www.ros.org/news/2010/05/robots-using-ros-penn-quadrotors.html> Acesso em: 2 de junho, 2021

Robots Using ROS: Lego NXT. Disponível em: <https://www.ros.org/news/2010/08/robots-using-ros-lego-nxt.html> Acesso em: 2 de junho, 2021

Robot Operating System. Disponível em: <https://en.wikipedia.org/wiki/Robot_Operating_System> Acesso em: 2 de junho, 2021

Alexis, Marlon; Andrés, William**. Trabajo de fin de carrera presentado como requisito para la obtención del título de Ingeniero Electrónico**. Disponível em: < https://repositorio.usfq.edu.ec/bitstream/23000/9292/1/134633%20-%20124116.pdf> Acesso em: 2 de junho, 2021

Conti, Henrique; Bernardi, Élder; Mario, João**. SISTEMA ROBÓTICO BASEADO NUMA ARQUITETURA DE IOT, UTILIZANDO**

ROS, SOA E IBM WATSON.1. Disponível em: <https://painel.passofundo.ifsul.edu.br/uploads/arq/20190221151237937899162.pdf> Acesso em: 2 de junho, 2021

CPE Tecnologia**. LiDAR: saiba mais sobre essa tecnologia!**. Disponível em: < https://blog.cpetecnologia.com.br/lidar-saiba-mais-sobre-essa-tecnologia/> Acesso em: 16 de agosto, 2021

Luisa, Ana**. Sensores Lidar - Entenda o que são e como funcionam!**. Disponível em: < https://mundoconectado.com.br/artigos/v/15382/sensores-lidar-entenda-o-que-sao-e-como-funcionam> Acesso em: 16 de agosto, 2021

Raspiberrypi**. Raspberry Pi 3 Model B+**. Disponível em: < https://s3-sa-east-1.amazonaws.com/robocore-lojavirtual/735/rpi-brief.pdf> Acesso em: 20 de setembro, 2021

TechMob**. O que é Raspberry Pi e para que serve?**. Disponível em: <https://techmob.com.br/o-que-e-raspberry-pi-e-para-que-serve/> Acesso em: 20 de setembro, 2021

Guse, Rosana**. O que é Raspberry Pi?** Disponível em: <https://www.filipeflop.com/blog/o-que-e-raspberry-pi/> Acesso em: 20 de setembro, 2021

Ciriaco, Douglas**. O que é Raspberry Pi?**. Disponível em: <https://canaltech.com.br/hardware/o-que-e-raspberry-pi/> Acesso em: 20 de setembro, 2021

Campos, Felipe**. História da Raspberry Pi**. Disponível em: <https://blog.smartkits.com.br/historia-da-raspberry/> Acesso em: 25 de setembro, 2021

Augusto, Cassio**. A História do UNIX(Sistema Operacional)**. Disponível em: <http://ninjadolinux.com.br/a-historia-do-unix/> Acesso em: 05 de outubro, 2021

Ciriaco, Douglas**. Unix: o pai de todos os sistemas operacionais**. Disponível em: <https://www.tecmundo.com.br/macos/10556-unix-o-pai-de-todos-os-sistemas-operacionais.htm> Acesso em: 05 de outubro, 2021

Delony, David**. The History of Unix: From Bell Labs to the iPhone**. Disponível em: <https://www.techopedia.com/2/29273/software/the-history-of-unix-from-bell-labs-to-the-iphone> Acesso em: 05 de outubro, 2021

Ames, Iowa**. UNIX Introduction**. Disponível em: <https://www.hpc.iastate.edu/guides/unix-introduction> Acesso em: 05 de outubro, 2021

Ritchie, Dennis**. The Evolution of the Unix Time-sharing System***. Disponível em: <https://www.bell-labs.com/usr/dmr/www/hist.pdf> Acesso em: 05 de outubro, 2021

Thomaz, José**. O que é Unix e qual a sua importância?**. Disponível em: <https://www.luby.com.br/infraestrutura/o-que-e-unix/> Acesso em: 05 de outubro, 2021

Delfino, Pedro**. Entenda O Que É O Projeto GNU E Porque Ele É Tão Importante Para O Sucesso Do Linux**. Disponível em: <https://e-tinet.com/linux/projeto-gnu/> Acesso em: 18 de outubro, 2021

*Free Software Foundation***. O Sistema Operacional GNU**. Disponível em: <https://www.gnu.org/home.pt-br.html> Acesso em: 18 de outubro, 2021

*Free Software Foundation***. Linux e o Sistema GNU**. Disponível em: <https://www.gnu.org/gnu/linux-and-gnu.pt-br.html> Acesso em: 18 de outubro, 2021

Equipe do Instalador Debian**. O que é GNU/Linux?**. Disponível em: <https://www.debian.org/releases/stable/s390x/index.pt.html> Acesso em: 20 de outubro, 2021

Escola de Informática Aplicada (EIA)**. DISTRIBUIÇÕES LINUX**. Disponível em: <https://guialinux.uniriotec.br/distribuicoes-linux/ > Acesso em: 20 de outubro, 2021

ESCOLA, Equipe Brasil**. História do Linux**. Disponível em: <https://brasilescola.uol.com.br/informatica/historia-do-linux.htm > Acesso em: 22 de outubro, 2021

Alecrim, Emerson**. O que é Linux e qual a sua história?**. Disponível em: <https://www.infowester.com/historia_linux.php> Acesso em: 22 de outubro, 2021

Wikilivros**. Manual do Ubuntu/Uma breve história do Ubuntu**. Disponível em: <https://pt.wikibooks.org/wiki/Manual_do_Ubuntu/Uma_breve_hist%C3%B3ria_do_Ubuntu> Acesso em: 22 de outubro, 2021

Simioni, Dionatan**. A história do Ubuntu, a distribuição Linux mais popular**. Disponível em: <https://www.hostgator.com.br/blog/a-historia-do-ubuntu-a-distribuicao-linux-mais-popular/> Acesso em: 22 de outubro, 2021

Equipe Tech Start**. A história do Ubuntu**. Disponível em: <https://techstart.xyz/linux/historia-do-ubuntu-linux/> Acesso em: 22 de outubro, 2021

Oliveira, Ricardo**. Como agem os controles de estabilidade e de tração**. Disponível em: <https://www.noticiasautomotivas.com.br/como-agem-os-controles-de-estabilidade-e-de-tracao/> Acesso em: 30 de outubro, 2021

Sense sensors & instruments**. Sensores Ultrassônicos**. Disponível em: <https://www.sense.com.br/arquivos/produtos/arq1/Sensores_Ultrass%C3%B4nicos_Sense_Folheto_Rev_%20J.pdf> Acesso em: 30 de outubro, 2021

Virtuous Tecnologia da Informação**. GPS - O que é, como funciona**. Disponível em: <https://www.sofisica.com.br/conteudos/curiosidades/gps.php> Acesso em: 30 de outubro, 2021

Banner do Brasil**. Sensores Radar**. Disponível em: <https://www.bannerengineering.com/br/pt/products/sensors/radar-sensors.html#all> Acesso em: 30 de outubro, 2021

Escola do Mecânico**. Controle Eletrônico de Estabilidade: tudo que você precisa saber**. Disponível em: <https://escoladomecanico.com.br/controle-eletronico-de-estabilidade/> Acesso em: 30 de outubro, 2021

APÊNDICE A – EXEMPLO DE ADIÇÃO DE UM SENSOR NO ARDUINO UNO

```
#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Float64.h>
#include <std_msgs/Empty.h>

ros::NodeHandle slave_arduino;  // instancia o node Handle

std_msgs::Float64 mensagem;  // instacia as variaveis com o tipo de mensagem

ros::Publisher arduino_btn("sensor", &mensagem);  // inicia um publicador com o nome do topico de sensor

int btn = 2;
int led = 3;

// funcao chamada ao receber um subscriber
void aviso(const std_msgs::Float64& msg_sub){
  float estado = msg_sub.data;  // recebe os dados da mensagem

  // acao a ser executada
  if(estado == 1.0){
    digitalWrite(led, HIGH);
    }
    else{
      digitalWrite(led, LOW);
    }
  }

ros::Subscriber<std_msgs::Float64> sub("status", &aviso);  // cria um assinante do topico status e chama a fucao aviso

void setup() {
  // put your setup code here, to run once:

  slave_arduino.initNode();  // inicia o identificador do no ROS

  // anuncia todos os topicos que estao sendo publicados e assina todos os topicos que deseja ouvir
  slave_arduino.advertise(arduino_btn);
  slave_arduino.subscribe(sub);

  // declara os pinos
  pinMode(2, INPUT_PULLUP);
  pinMode(3, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:

  // le as informacoes dos sensores e guardam nas mensagens que serao postadas
  if (digitalRead(btn) == LOW){
    mensagem.data = 1;
  }
  else{
    mensagem.data = 0;
  }

  // publica a mensagen
  arduino_btn.publish( &mensagem );

  slave_arduino.spinOnce();  // todos os callbacks de comunicacao ROS sao tratados

  delay(200);
}
```

APÊNDICE B – EXEMPLO DE ADIÇÃO DE UM SENSOR NO RASPBERRY PI

```python
#!/usr/bin/env python2

import rospy
from std_msgs.msg import Float64, String
import RPi.GPIO as gpio

import sys
import signal

def signal_handler(signal, frame): # ctrl + c -> sai do programa
        sys.exit(0)
signal.signal(signal.SIGINT, signal_handler)

gpio.setmode(gpio.BCM)  # Configura o GPIO do Raspberry

# determina os pinos no gpio
led = 27 # GPIO 27 - 7 pino do lado esquerdo
btn = 17 # GPIO 17 - 6 pino do lado esquerdo

gpio.setup(led, gpio.OUT)  # sinal de saida
gpio.setup(btn, gpio.IN, gpio.PUD_UP)  # sinal de entrada

class Class_Name:
    def __init__(self):  # funcao construtora

        # subscreve no topico sensor e ao receber algo chama a funcao self.funcao
        self.var_sub = rospy.Subscriber('/sensor', Float64, self.funcao)
        self.var_pub = rospy.Publisher('/status', Float64, queue_size=5)  # cria a publicacao status

    def funcao(self, msg):
        self.mensagem = float(msg.data)  # receber a informacao do topico
        rospy.loginfo(str(self.mensagem))  # informa na tela a mensagem recebida

        # processamento com a mensagem recebida
        if self.mensagem == 1:
            gpio.output(led, gpio.HIGH)
        else:
            gpio.output(led, gpio.LOW)

        #processamento com o sensor no raspberry
        mensagem_pub = Float64()  # instancia a variavel mensagem_pub para o tipo Float64
        if gpio.input(btn) == gpio.LOW:  # verificar o estado do botao
            mensagem_pub.data = 1  # atribui valor na variavel mensagem_pub
        else:
            mensagem_pub.data = 0
        self.var_pub.publish(mensagem_pub)  # publica a variavel mensagem_pub no topico status

if __name__ == '__main__':
    try:
        rospy.init_node('add_sensor_py')  # cria o no sensor_subscriber
        sensors_subscriber()  # chama a classe
        rospy.spin()  # impede que o no seja encerrado sozinho
    except rospy.ROSInterruptException:
        gpio.cleanup()  # limpa o gpio
    except (KeyboardInterrupt, SystemExit):
        gpio.cleanup()
```