Projeto de Graduação

DEE DEPARTAMENTO DE ENGENHARIA ELÉTRICA

16/07/2019

BIAROBOT: ROBÔ AUTÔNOMO DE INFORMAÇÕES

Luiz Antonio Martins

PUC RIO

Projeto de Graduação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

BIAROBOT: ROBÔ AUTÔNOMO DE INFORMAÇÕES

Aluno: Luiz Antonio Martins

Orientador: Wouter Caarls

Trabalho apresentado como requisito parcial à conclusão do curso de Engenharia de Controle e Automação na Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil.

Agradecimentos

À minha família que esteve presente em todos os momentos;

Aos amigos que sempre me apoiaram;

Aos professores que me passaram o conhecimento necessário para minha formação.

Resumo

Este projeto consiste em construir um robô capaz de se movimentar sem atingir nenhum obstáculo, sem nenhum tipo de controle remoto, e também capaz de interagir com qualquer usuário que se aproxime dele. Para isso serão usados motores, para sua movimentação; sensores, para garantir que ele não colida com nada; e, por fim, microcontroladores embarcados, que serão vitais para o processamento de dados obtidos e disponibilizados. Além disso, outros dispositivos como microfones, alto falantes e uma tela serão utilizados para garantir a melhor experiência para o usuário.

Palavras-chave: Robô, Autônomo, Microcontrolador, ROS, Inteligência Artificial

BIAROBOT : AUTONOMOUS INFORMATION ROBOT

Abstract

This project consists in building a robot capable of moving itself avoiding any obstacles, without any kind of remote control. Also capable of interacting with every user that approaches it. For this purpose, will be used motors for it's movement; sensors, to garantee it doesn't colide and, last but not least, the microcontrollers that will be vital to data processing. In addition, other devices such as microphones, speakers and a screen will be used to ensure the best experience for the user.

Keywords: Robot, Autonomous, Microcontroller, ROS, Artificial Intelligence

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

a Robô Autônomo

Robôs autônomos é um tipo de robô que realiza atividades e cumprem objetivos sem navegação ou intervenção humana. Eles são estruturados de acordo com o nível de autonomia desejado, dependendo da função que precisam cumprir. Basicamente, os robôs autônomos precisam captar dados do ambiente que estão, trabalhar sem a interferência humana, se deslocar entre pontos sem navegação humana e substituir o trabalho humano em situações de perigo. Conceitos variam de pequenos insetos a máquinas de robôs humanoides altamente sofisticados com inteligência social e consciência de seu ambiente. Na Indústria 4.0, os robôs autônomos possuem também a capacidade de "aprender sozinho" (consegue ganhar novas habilidades sem ajuda externa) e reformular sua estratégia de acordo com o ambiente em que estão.[14]

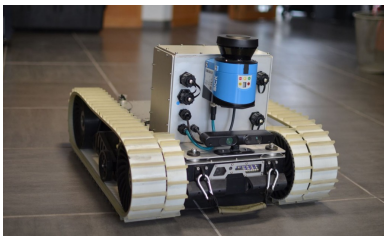


Figura 1: Robô autônomo de inspeção

1 Robôs Autônomos consolidados

Atualmente, já existem diversos robôs autônomos que já estão consolidados no mercado com inúmeras e das mais diversas aplicações diferentes. Alguns deles serão listados abaixo:

AMY Como uma boa secretária, Amy toma notas de encontros, prepara relatórios, marca e desmarca reuniões, fala idiomas, busca os visitantes na portaria e os acompanha até a sala de conferências. Ela ainda monitora o ar-condicionado, sabe se a impressora está funcionando e manda fazer cópias. Pode até mudar de voz, se o patrão preferir um secretário. Equipado com um sistema que evita colisões – detecta objetos com até 20cm de distância - que contém câmera 3D, raios infravermelhos e ondas ultrassônicas. Outro modelo dispõe de mais sensores como de presença, de digital, microfones para comandos de voz.[1]

Figura 2: Amy

Bossanova Robô do Walmart para escanear as prateleiras, ajudando funcionários na hora da reposição e também clientes na hora de achar o produto necessário. Ele consegue detectar objetos nos corredores, como por exemplo, caixas, carrinhos de compras e até mesmo pessoas.[4]

Figura 3: Bossanova

Kiva É o típico robô para movimentação de cargas em armazéns, conhecidos pela sua eficiência. Por conta de seu tamanho, é extremamente fácil percorrer e evita congestionamentos pelos armazéns, ao encontrar o volume desejado se acopla a ele e utiliza o menor caminho até o destino, onde está um funcionário para embalar e despachar o produto. Quando sua bateria está acabando, automaticamente procura um ponto de recarga. Eles também reorganizam o ambiente, colocando os itens mais requisitados em áreas próximas. [10]

Figura 4: Kiva

KnightScope Basicamente, um segurança para áreas grandes e abertas. Podem chegar a 5km/h e tem 1,60 de altura. Tem visão de calor, pode transmitir áudio e vídeo para uma central, interage com humanos,

pode reproduzir mensagens gravadas anteriormente. Ainda é capaz de gravar até 300 placas de carros em cada uma de suas 4 câmeras. Sua própria lista negra pode ser criada e um aviso chegará assim que a placa marcada aparecer. Também protege sua rede de pessoas e softwares indesejados. Pode até identificar suspeitos de um possível futuro crime na área baseado em informações de dispositivos móveis. Utilizam tecnologia similar aos carros autônomos para conseguir caminhar por locais com uma quantidade considerável de pessoas. Também possuem sensores de distancia e alarmes graduais de distancia, caso uma pessoa se aproxime demais.[11]

Figura 5: KnightScope

BeneBot Lançado em 2017 o Benebot 3 teve introduzido em si toda a tecnologia de núcleo do robô. Possui programa de integração por voz. Considera-se usá-lo em áreas de serviço público, grandes e de fatores ambientais complexos. Seu conjunto de microfones é especialmente otimizado para diminuição de ruídos, identificação de campo próximo, compreensão semântica e beamforming direcional. Também usa um programa de conversa de domínio aberto – para interação não comercial, como em shoppings e comunicação com clientes – além da conversa vertical – mais usada na parte de negócios. Benebot é capaz de iniciar uma conversa com um cliente que não iniciou uma conversa e vai restringindo sua busca baseado nas respostas da pessoa. Por exemplo, no shopping, nenhuma iniciativa dos clientes, o robô vai tomar a iniciativa de perguntar ao cliente se deseja ir às compras ou jantar, se o cliente responder fazer compras, o robô tomará a iniciativa de informar as roupas, bolsas, cosméticos e outras categorias, e ainda perguntar ao cliente no que está interessado. Sua visão computacional permite identificar rostos de pessoas VIPs, líderes e convidados e ao mesmo tempo irá reconhecer atributos – sexo, idade, expressão, entre outros – e vai determinar a interação com o sujeito, o que sugerir e onde o levar, por exemplo. Também usa machine learning para identificar objetos para se posicionar melhor ao saber onde fica o balcão, elevador, estacionamento etc. Detecta obstáculos em até 25m (em raio) e pode, simultaneamente, mapear e se localizar num shopping, banco e outros locais de uso comercial. Para isso foi usado radares a laser e câmeras de sentido reais, para reconhecimento, detecção de obstáculos em 2D e estereoscópio 3D (que garante que paredes, pilastras sejam vistas e identifica o caminhar e multidões – objetos dinâmicos – para aumentar sua segurança em publico).[3]

Figura 6: Benebot

Care O Bot Robô que auxilia as pessoas em tarefas domésticas distintas que já está em sua 4ª geração. Pode ter um, dois ou nenhum braço. Possui junta esférica em seu pescoço e quadril, possibilitando um giro 360° na cabeça e torso. Cada robô pode ser configurado para uma ampla gama de aplicações: um centro de informações móvel em museus, lojas e aeroportos, para serviços de coleta e entrega em residências e escritórios, para aplicações de segurança, entre outros - Care-O-bot 4 é um ajudante humano seguro e acessível em todos os momentos. Nessa geração, o Care O Bot está mais sociável. Porque se acredita que a interação social é fundamental para sua aceitação. Ele usa um monitor integrado em sua cabeça e baseado na situação atual pode exibir diferentes atmosferas. Care-O-bot 3 era um mordomo mais reservado e cauteloso, seu sucessor é tão cortês, amigável e afável como um cavalheiro.[5]

Figura 7: Care-O-Bot

b BIA Robot

1 Objetivo

O objetivo desse trabalho, além de criar um robô que consiga se movimentar e interagir autonomamente, será documentar e explicar todo esse processo. Desde seus componentes até as ideias que basearam a construção, confecção e programação. Tendo como base as ideias demonstradas acima.

A BIA já possui um website(`https://www.facebook.com/falecomBIA/`) e outro objetivo do projeto consiste em integrar o robô desenvolvido ao site já existente, criando assim uma interação entre os alunos na biblioteca

com a BIA.

Segue um esquema que explica, sucintamente, o funcionamento da BIA e seu objetivo:

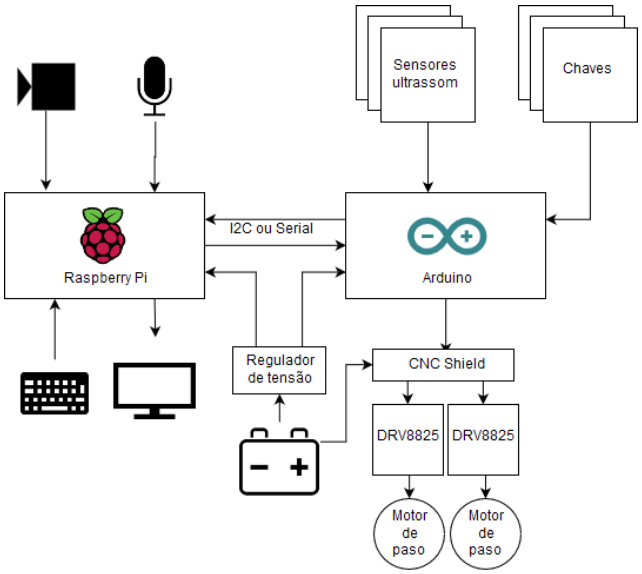


Figura 8: Esquema

2 Visão Geral

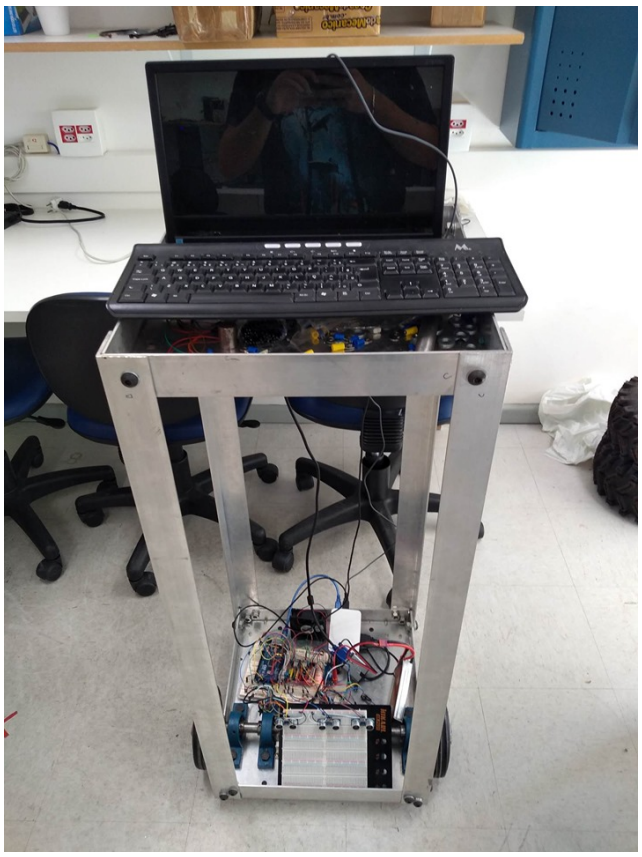


Figura 9: BIA

Assim está a BIA, com todos os componentes que serão apresentados e explicados no relatório do projeto. Todas as peças mecânicas, eletrônicas e toda a parte de computação e programação estarão detalhadas abaixo. A BIA é capaz de andar livremente evitando qualquer tipo de obstáculos e parando quando se aproxima muito de um obstáculo, podendo esse obstáculo ser uma pessoa ou não. O controle do projeto será feito de forma inteligente, mais precisamente um controle Fuzzy que será alimentado com os sensores presentes e alimentará os motores.

2 Parte física

a Componentes do projeto

1 Motores

Foram utilizados dois motores de passo da marca Akiyama. Motores de passo são controlados por uma série de campos eletromagnéticos que são ativados e desativados eletronicamente. Fazendo com que ele tenha uma precisão de angulação e posição, por isso ele foi escolhido para essa aplicação. A velocidade do motor de passo é definida pela frequência com que esses pulsos são enviados e o número de voltas do eixo é definida pela quantidade de pulsos. O dispositivo que faz o controle desses pulsos elétricos que são enviados para o motor, é chamado driver, nesse caso o driver será nosso shield do Arduíno.

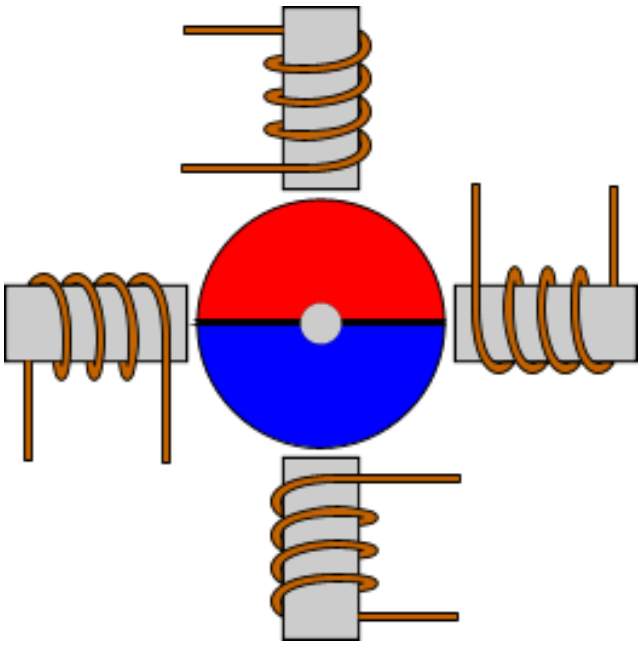


Figura 10: Motor de passo

O motor de passo da Akiyama utiliza corrente de 2,1 A(4.2V) e possui as dimensões exatas para o funcionamento perfeito do sistema. Será usado o de 6 fios, também possuem motores com 4 e 8 fios. Motores de 4 fios são ligados somente em bipolar série, motores de 6 fios podem ser ligados em bipolar série ou unipolar e motores de 8 fios podem ser ligados em bipolar série, bipolar paralelo ou unipolar.

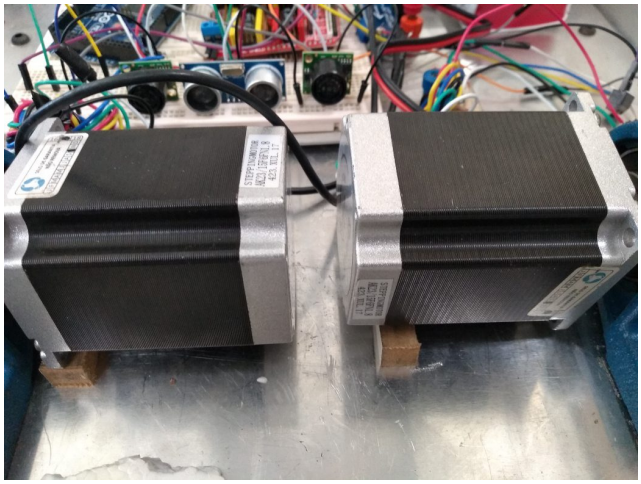


Figura 11: Motores

Os motores estão conectados diretamente aos seus respectivos eixos que tem uma roda na outra ponta, fazendo com o robô tenha sua movimentação feita por meio de rodas que serão alimentadas pelos eixos ligados aos motores. Pode se observar na foto que foi preciso colocar 2 mancais improvisados em cada motor. Os mancais foram necessários para fixá-los e não deixar nenhuma folga entre os eixos e os motores, e também para redução de ruídos.[13]

2 Microcontroladores

Para controlar o robô precisa-se de dois microcontroladores, o Arduíno e o Raspberry Pi, cada um na sua função.

O Raspberry Pi será usado para o processamento de dados e também é onde está instalado o ROS (Robotic Operating System) - uma coleção de ferramentas e bibliotecas que visam simplificar a tarefa de criar um comportamento de robô complexo e robusto. O microcontrolador será responsável por analisar os dados dos motores e dos sensores, assim como também será responsável pela interface de comunicação.

Figura 12: Raspberry

O Arduíno, por outro lado, será utilizado para as partes mecânicas do robô, uma das mais importantes é a movimentação das rodas, pela atuação dos motores. Para isso será usado um shield específico para essa função, que será explicado e especificado a seguir. O Arduíno teve preferência por ser usado porque a corrente que o Raspberry gera é mais baixa e é mais usado para processamento de dados porque conta com mais memória RAM e possui entradas USB e de vídeo que possibilitam ligar o controlador como fosse um computador portátil. Entretanto, consome mais bateria, em comparação com o Arduíno.

Outro ponto favorável ao Arduíno é o fato de não usar sistema operacional senso assim "real time", ou seja, o tempo de resposta é mais confiável, sendo vital para os pulsos que serão enviados aos motores.

Figura 13: Arduíno

3 Driver de controle

O driver de controle dos motores usado foi um shield CNC próprio para motores de passo e Arduíno. Ele tem suporte para até 4 motores diferentes.

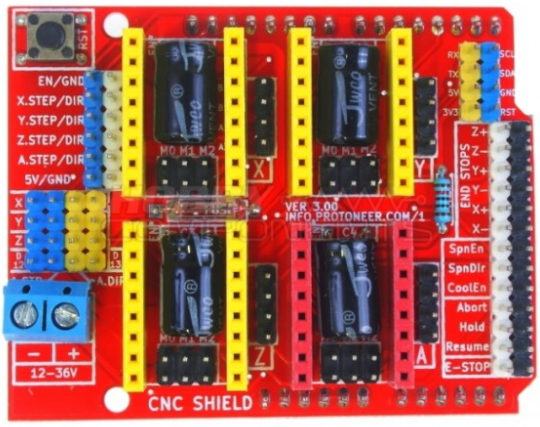


Figura 14: Driver dos motores

Esse driver nos permitiu controlar os dois motores presentes no robô independentemente um do outro. O shield precisa ser alimentado por 12V, os motores precisam ser conectados às entradas do Arduíno, juntamente com

o Enable e o GND e após o shield conectado ao microcontrolador, conecta-se os motores aos seus respectivos locais. Nesse caso serão utilizados os motores X e Y.

Além do shield foram usados drivers (DRV8825), um para cada motor para que consiga transmitir os dados - corrente e voltagem, na frequência certa - necessários para o funcionamento. Que conta com um dissipador de calor, que é de extrema importância, já que o robô estará fechado.[8]

A corrente dos motores precisou ser limitada e isso afetou levemente sua movimentação. Mas sem as limitações na corrente era inviável para o projeto devido ao superaquecimento do driver e a perda de passos do motor.

4 Sensores

Os sensores usados no projeto serão do tipo ultrassônico e são do próprio Arduíno, com código HC-SR04, seu datasheet consta no site da Elec Freaks. O princípio do sensor de ultrassom, que também pode ser chamado de sonar, consiste em um emissor e um receptor. O emissor emite um pequeno pulso sonoro de alta frequência que se propagará na velocidade do som no meio em questão, no nosso caso ar, na CNTP seria de 340 m/s. Quando este pulso atingir um objeto, um sinal de eco será refletido para o receptor. A distância entre o sensor e o objeto pode então ser calculada caso saibamos o tempo entre a emissão e a recepção do sinal, além da velocidade do som no meio em questão.

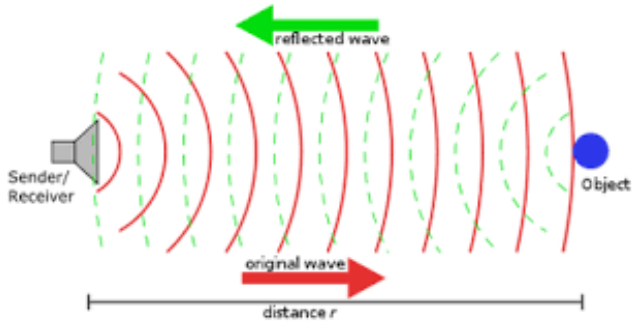


Figura 15: Tipo do Sensor

O sensor do Arduíno, por sua vez, tem Vcc fixo em 5V DC e trabalha numa corrente de 15mA.Também será alimentado pelos 5V do Arduíno. Sua amplitude de percepção de objetos vai de 2cm até 400cm (4m), podendo dar diferenças de até 3mm. Sua frequência é de 40Hz, sendo duas vezes mais rápido que outro sensor.[2]

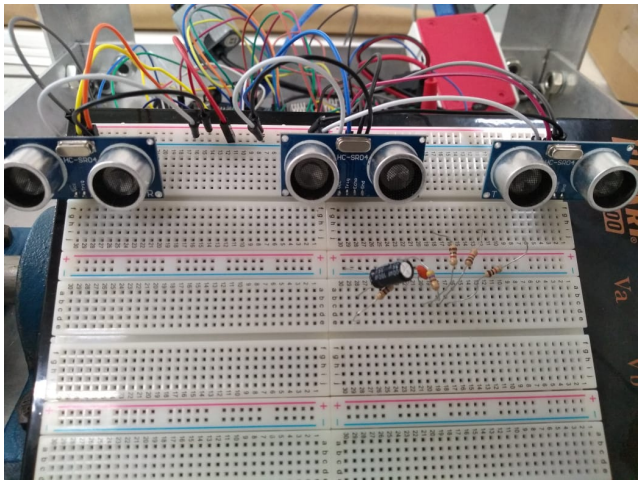


Figura 16: Sensores

Para a utilização desses sensores há duas opções: A primeira seria fazer o algoritmo de funcionamento do zero, à mão e, a segunda usar a biblioteca Ultrasonic[17] do Arduíno, que possui funções próprias, não sendo necessário fazer o funcionamento dos emissores e receptores, como na primeira opção, e retornando diretamente a distância ao passar os pinos do emissor e receptor. A segunda opção foi a adotada e pode ser visto que a biblioteca é incorporada no código presente nos apêndices.

5 Bateria

A bateria escolhida para alimentar o robô e seus componentes é uma de 11.1V de Lítio Polímero com capacidade de 4500mAh com 3 células. Levando em consideração que os motores usam 2.1A cada, a autonomia da bateria seria de pouco mais de 1h, sem levar em consideração outros dispositivos que também usam a bateria como fonte.

Figura 17: Bateria de Lítio Polímero 4500 mAh

Lembrando que o Raspberry Pi não pode ser alimentado com uma fonte de 12V. Logo, foi preciso de um pequeno circuito, chamado de conversor buck [6], que quando tem 12V em sua entrada disponibiliza apenas 5V em sua saída, podendo aumentar ou não a corrente fornecida. Então o microcontrolador em questão deve ser ligado na saída desse pequeno circuito. Conversores buck são compostos de indutores, diodos, capacitores e resistores, e ainda pode contar transistores e/ou MOSFET's.

6 Estrutura

A estrutura foi confeccionada e criada na faculdade. Todos os cálculos foram feitos para que a estrutura fique firme suficiente para conseguir fazer curvas, acelerar e frear sem que caia ou desmonte. As laterais ainda estão abertas devido a frequência que mudanças precisam ser feitas em alguma conexão da parte eletrônica e computacional do projeto porém, uma vez este pronto serão fechadas, o que dará mais estabilidade e firmeza.

O material utilizado foi o alumínio, que é um material leve e resistente, de forma que a estrutura possa ser carregada sem problemas. A estrutura utiliza:

- 4 vigas em L de 1m

- 8 vigas em L de 40cm
- 2 placas quadradas de 39.6cm
- parafusos
- arruelas
- porcas

Os últimos três itens são apenas para fixação das vigas e placas. Os itens que compõem a estrutura não estavam previamente furados para a utilização dos parafusos. A carcaça foi construída em conjunto com a professora Karla Figueiredo, do LIRA, e os alunos Guilherme Eduardo e Alexandre Mulina, que deixaram o projeto para seguir outros rumos.[9]

7 Periféricos

Para que os microcontroladores - mas especificamente o Raspberry - sejam controlados é necessário mouse, teclado e uma tela, que será a mesma para a integração do robô com os usuários.

Todos os componentes listados acima serão alimentados pela bateria, de forma que a BiaRobot fique totalmente livre de fios para fora de sua estrutura e consiga transitar por qualquer ambiente sem nenhum problema.

Figura 18: Periféricos

b Montagem dos componentes

Todos os componentes citados acima foram colocados e montados na parte interna da estrutura. Os motores estão presos aos eixos que estão conectados a suas respectivas rodas e fixos na fundação da estrutura; para a conexão dos fixos do Arduíno para o driver e do driver para os motores precisou ser feito por meio de uma protoboard - placa de ensaio -, que também está na fundação da estrutura. Essa placa está sendo

alimentada pela bateria já citada. Ambos os microcontroladores estão posicionados, apenas apoiados, também na fundação. Do Raspberry Pi sai os fios dos periféricos que estão na parte superior da BIA.

Um dos empecilhos da montagem é o espaço reduzido juntamente com o excesso de fios a serem ligados. O que resultou em um pouco de ruído visual e também pode acarretar em alguns curtos circuitos e/ou mal contato em algum fio. O espaço reduzido também atrapalhou na hora de acomodar alguns componentes, que tiveram que ficar suspensos ou com fios fazendo uma espécie de cotovelo, o que pode ser um problema a longo prazo.

Juntamente com o espaço reduzido para colocar os componentes, o assoalho da BIA é uma chapa de metal, o que pode ser um problema caso um fio se solte ou alguma parte metálica do driver ou de um dos microcontroladores entre em contato com o fundo.

Uma outra dificuldade da montagem foi a escolha das posições dos sensores. Uma vez que a abertura de sensoriamento deles não é alta, por volta de 30º. E precisa estar na mesma altura que o obstáculo está. Como a altura dos obstáculos é variada, precisaríamos de inúmeros sensores em diversas alturas diferentes. O protótipo possui apenas sensores na parte da frente, nas suas extremidades e no centro, na parte de baixo. Porque dessa forma ele consegue perceber obstáculos como pés de mesas, cadeiras e humanos, mochilas, caixas, paredes etc. Porém, se houver algum tipo de bancada ou mesa com pés que não estejam na direção dos sensores, infelizmente, não serão detectados.

Vale ressaltar que toda a montagem do robô é apenas temporária, uma vez que é apenas um protótipo e precisa de melhorias em quase todos os quesitos físicos.

c Parte Mecânica

1 Estabilidade

Como a BIA é uma estrutura consideravelmente alta, aproximadamente 1m de altura, sua velocidade não pode ser demasiadamente alta, porque ao frear bruscamente a estrutura pode balançar, tomar e vir a cair. A velocidade também precisa ser ajustada para que suas curvas e movimentação sejam suaves. Os componentes foram colocados em seu assoalho porque diminui a altura do centro de massa, colocando o mais próximo de seus apoios, rodas, afim de dar mais estabilidade.

Também precisa se levar em conta a força centrífuga e centrípeta dos componentes internos e externos - essas forças são contrárias, sendo a centrífuga que puxa o corpo para fora da trajetória e a centrípeta a que o mantém. Conforme a 2ª lei de Newton (F = ma), quanto maior a aceleração sobre uma mesma massa, maior a força que atua sobre esse corpo. Consequentemente, para reduzir a força que atua sobre os componentes e estrutura, gostaríamos de trabalhar com acelerações reduzidas, tanto para movimentação reta quanto para as curvas.

Para aumentar a estabilidade da BIA projeta-se fechar suas laterais. Ao realizar esse procedimento esconderia todo o "trabalho sujo" - os componentes internos e fios - e iria dar mais sustentação ao projeto. Se isso não

for feito, estuda-se colocar travessas diagonais em suas laterais, o que também daria mais estabilidade e sustentabilidade ao robô.

2 Tração

Para garantir que a rotação do motor passe para a roda não foi necessário nenhum tipo de ajuste na rotação e/ou no torque do motor, podendo ser feita a ligação motor-roda diretamente por meio do eixo. Eixo este que foi usinado para acomodar o eixo do motor sem folgas. Ao longo desse eixo até as rodas foram colocados mancais para sua fixação.

Como as rodas são independentes entre si, são necessários 2 eixos e são conforme descritos acima.

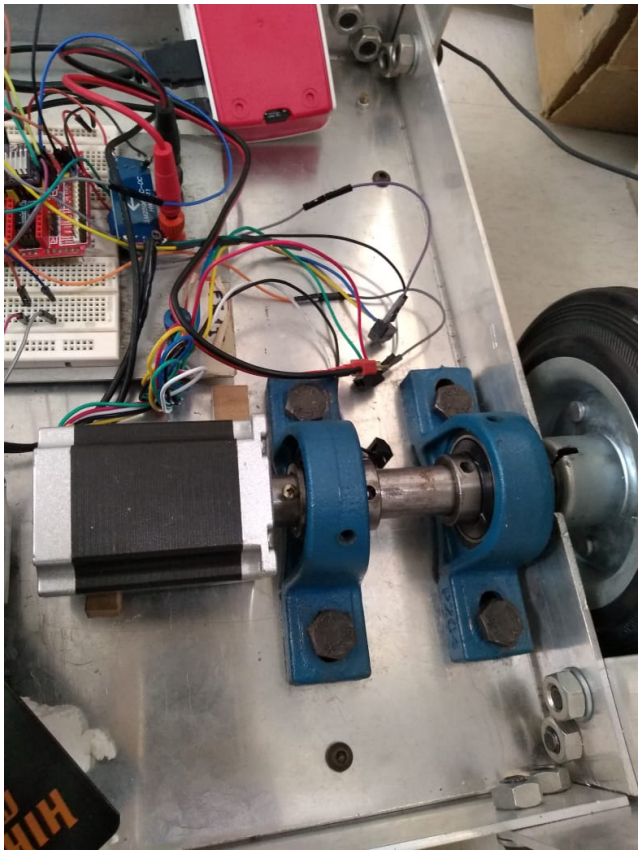


Figura 19: Eixo da roda esquerda

As rodas que estão no eixo são de borracha, que garantem um atrito considerável com o piso e consegue realizar a movimentação da BIA. Também há, na parte de trás do robô, duas outras rodas menores que servem somente para apoio, elas podem se mover em qualquer direção, facilitando para as rodas frontais de realizarem alguma curva.

3 Parte Eletrônica e Computacional

a Eletrônica

Com respeito a eletrônica, para interligar todos os componentes elétricos do projeto foi foi usado uma protoboard. E para fixar os sensores foi preciso outra placa similar, localizada na frente da BIA. Nessa protoboard, foi fixado o terra e o V+ da bateria. Por causa do tamanho dos fios disponíveis, algumas pontes foram feitas na protoboard, já que alguns componentes ficaram afastados de seus destinos. Os motores foram ligados usando também usando a placa, embora os fios pudessem ser conectados sem o uso da mesma.

Além da placa de ensaio, para garantir os 5V de alimentação para o Raspberry, foi usado um pequeno circuito que possui em sua entrada 12V e em sua saída 5V. Esse circuito foi ligado diretamente na bateria, na sua entrada, e na saída foi ligada no microcontrolador. O Arduíno não precisa de alimentação externa, somente a ligação na USB com o Raspberry é suficiente para mantê-lo ligado.

O terra do Arduíno foi conectado no terra da placa, oriundo da bateria. Algumas ligações posteriores de 5V foram necessárias e estas foram feitas usando o pino de 5V presente no Arduíno. Essas ligações de 5v citadas acima foram:

- Enable do driver;
- Alimentação dos três sensores.

Nenhuma porta analógica foi usada, uma vez que os sensores e o driver usam somente sinais digitais, são do tipo binário(somente 1 ou 0) e sinais analógicos dependem de quantos bits estão disponíveis(2^x,onde x é o número de bits).

b Computacional

Para esse projeto, além das partes físicas já evidenciadas anteriormente, a parte computacional também está presente e será explicada a seguir:

1 Arduíno

O Arduíno será responsável por transferir os comandos que serão enviados pelo ROS/Raspberry para os motores, em uma via e na via - seguindo o caminho contrário - enviará os dados do motor e dos sensores para análise e controle. Para programá-lo dessa forma foi usado o próprio programa do desenvolvedor, que utiliza a linguagem C++. Algumas bibliotecas precisaram ser adicionadas para o pleno funcionamento da BIA.

Somente pinos digitais foram utilizados. E suas utilizações serão descritas na lista a seguir:

- Pino 0: Não conectado;

- Pino 1: Não conectado;
- Pino 2: X Step, ligação no driver;
- Pino 3: Y Step, ligação no driver;
- Pino 4: Emissor de um dos sensores;
- Pino 5: X Dir, ligação no driver;
- Pino 6: Y Dir, ligação no driver;
- Pino 7: Receptor de um dos sensores;
- Pino 8: Enable do driver;
- Pino 9: Receptor de um dos sensores;
- Pino 10: Emissor de um dos sensores;
- Pino 11: Receptor de um dos sensores;
- Pino 12: Emissor de um dos sensores;
- Pino 13:

Os pinos 0 e 1 não foram usados porque são o Rx e Tx, respectivamente. Os pinos 2,3,5,6 e 8 foram conectados para mandar sinais para o driver dos motores inclusive o enable e os pinos 4,7,8,9,10,11 são responsáveis por mandar e receber os sinais que virão a ser as leituras dos sensores.

Os dados que serão transmitidos para o ROS serão enviados em um array de 4 posições com a velocidade sendo o primeiro elemento do array e as distâncias serão alocadas nas três posições seguintes.

2 Raspberry

No Raspberry, será escrito o código em Python que irá pegar os dados dos sensores e do motor. Após adquirir esses dados, irá analisar esses dados e controlar a movimentação da BIA baseado nesses números, usando uma lógica Fuzzy - também chamada de lógica nebulosa - que será descrita nos próximos tópicos. Depois de analisados os dados dos sensores, a velocidade será alterada, se necessário, e repassada de volta para o Arduíno que seguirá com a informação para o driver dos motores.

O código presente do Raspberry também conta com o ROS para publicar a resultante dos motores e subscrever os sensores. Além dos nós do ROS também tem a biblioteca para controle que será explicada na próxima seção quando o controle estiver em evidência.

3 ROS

ROS é a sigla para Robotic Operating System, e é um conjunto de bibliotecas e ferramentas para ajudar na construção de aplicações robóticas. O ROS foi utilizado para facilitar a integração entre os microcontroladores. O Raspberry continha o ROS por meio do ubuntu e com isso todas as informações que o Arduíno passa para o Raspberry, o ROS tem acesso. Mas porque usar o ROS?

Ele tem um design modular, ou seja, permite que escolha quais partes prontas são uteis e quais partes a implementação é necessária. A comunidade do ROS conta com usuários de todo o mundo e é muito ativa, o wiki tem cerca de 30 edições por dia e o ROS Answers possui 5700 usuários. Além do wiki e Answers, também oferece acesso a drivers de hardware, recursos de robôs genéricos, ferramentas de desenvolvimento, bibliotecas externas úteis etc. A principal filosofia do ROS é o desenvolvimento compartilhado de componentes comuns. Basicamente, o ROS integra os cientistas mundo afora, cada um contribuindo da melhor forma.

A principal ferramenta do ROS será utilizada no projeto é a infraestrutura de comunicação. Ele oferece uma interface de transmissão de mensagens. Essa interface - chamada de middleware - dispõe de recursos que serão úteis na comunicação entre os controladores. O sistema de mensagens, que já foi testado,economiza tempo e força a implementação de interfaces claras entre os nós do sistema. Os dados podem ser capturados sem que o código se altere, porque o sistema é assíncrono e anônimo. Essas virtudes do ROS podem reduzir o esforço de desenvolvimento e flexibilizar o sistema. Entretanto, se desejado, também tem o recurso de usar respostas síncronas no sistema.

Outra qualidade do ROS é que ele pode se comunicar em linguagens diferentes, por exemplo, dois nós - um em Python e outro em C++, podem se comunicar sem problemas. E, embora contenha todos esses recursos, o ROS não é pesado para se instalar, tanto que está instalado no Raspberry Pi e pode adquirir pacotes separadamente. Além de tudo isso é "open source"(que tem como tradução para o português, código aberto), isto é, é livre para uso.

O ROS usa o conceito de nós e tópicos. Um nó é um arquivo executável que se comunica com a plataforma e com outros nós, eles podem ser "publishers" - que publicam, transmitem algum dado continuamente; também chamado de "talker" -, "subscriber" - que subscreve esses dados; que também são chamados de "listener". Nós também podem fornecer ou usar algum serviço. Tópicos são, analogamente, como se fossem variáveis globais, que os nós podem enviar ou ler dados. Os tópicos são a forma que os nós conversam entre si. Nesse projeto, serão usados dois nós e um tópico. O tópico será onde a velocidade e as distâncias serão transmitidas. E os nós serão onde a transmissão dos dados será iniciada e também a movimentação.[15]

4 Controle

Para controlar a movimentação do robô será usado agentes inteligentes, os quais usam um sistema baseado na lógica nebulosa como base de conhecimento. Esses sistemas tem demonstrado serem efetivos para problemas de controle. [7] [18]. A teoria de agentes inteligentes é flexível para ambientes dinâmicos, complexos e/ou desconhecidos. O agente atua segundo um conjunto de regras formuladas a partir das crenças e base de conhecimento. As crenças são definidas pelos valores dos sensores e sua velocidade atual.

As abordagens mais comuns para o problema de controle de robô tendem a não ser suficientemente rápidas a mudanças no ambiente. A principal dificuldade está no fato de que o ambiente não é previamente conhecido, tornando difícil avaliar, em um dado tempo, qual a melhor ação a ser tomada, uma vez os estímulos que o robô recebe no presente referem-se a avaliações passadas. Para o sistema aqui apresentado, ou seja, ambiente com mudanças rápidas e desconhecido, não é possível obter flexibilidade suficiente a partir de instruções explicitas - é impossível programar todas as possíveis contingências.

A inteligência do agente é modelada por sistema nebuloso sem memória de percurso. O processamento quando feito nesse molde é direto, um estado de entrada é processado e produz um estado de saída. O estado de entrada diz respeito ao ambiente do agente e o de saída pelos efeitos que o agente causa no ambiente. Os agente são mais indicados para ambientes complexos e/ou com informações faltando. Um agente pode ter inúmeras classificações, que não serão explicadas aqui, mas ele é dito inteligente quando possui capacidade de aprender.

O próximo passo é modelar o controlador, começando pelas variáveis linguísticas, que são os sensores presentes no robô; esquerda, meio e direita, que assumem valor da distância até obstáculo e 0 quando não consegue captar nenhum. A saída do sistema nebuloso são as velocidades angulares a serem aplicadas em cada motor(esquerdo e direito). Podendo ser velocidades para frente e para trás.[12]

As variáveis descritas acima foram divididas em conjuntos nebulosos, o que também é chamado de Fuzzificação.

- Sensores: Muito-Longe, Longe, Perto, e Muito-Perto - 4 conjuntos;
- Velocidade: Negativo-Alto, Negativo-Médio, Zero, Positivo-Médio e Positivo-Alto - 5 conjuntos;

Definidas as divisões dos conjuntos nebulosos das variáveis, os gráficos ficaram da seguinte maneira:

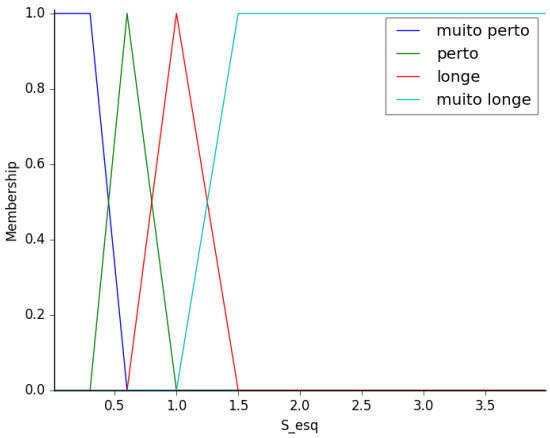


Figura 20: Pertinência dos sensores

Como os três sensores são iguais as pertinências são iguais para os 3 então os gráficos são os mesmos e apenas um deles está presente.

Situação parecida acontece com as pertinências dos motores, que são espelhados, e apenas uma será mostrada abaixo:

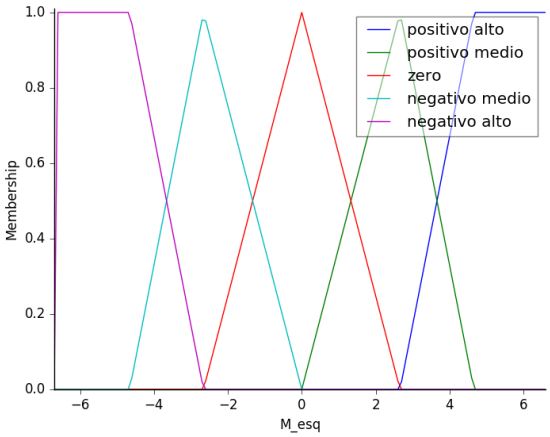


Figura 21: Pertinência dos motores

As funções dos motores tendo as pertinências dos sensores como entrada são espelhadas, de forma que o motor da esquerda e da direita funcionam da mesma maneira, espelhados, para ter um movimento de desvio suave e curvas não abruptas. Evitando acelerações altas que poderiam resultar em trancos no motor e na estrutura.

Podemos perceber que as pertinências dos motores estão mais próximas de 0, isso se dá porque quanto mais perto de 0 mais crítico deve ser a redução de velocidade e controle. Por isso as funções foram ajustadas para adquirir mais valores quando o controle precisa ser feito com mais precisão e exatidão. E quanto mais longe for ficando o obstáculo, sua velocidade pode ser constante e alta para sua movimentação ficar fluida e direta.

a Regras

O conjunto de regras possui dois movimentos: regras de movimento em linha reta, são as que levam o robô a se movimentar em linha reta, para frente ou para trás, ou seja, atribuem o mesmo valor de potência (em módulo e sentido) a ambos os motores; regras de desvio, são as responsáveis por executar as manobras de desvio de obstáculos.[18]

No final das regras, o resultado será obtido a partir dos centros de massa obtidos nas saídas quando são afetadas pelas regras e entradas, chamadas de antecedentes. Os esses cálculos serão todos feitos com a biblioteca descrita acima.

As regras foram definidas e serão mostradas na tabela abaixo:

Entradas			Saídas	
Sensor Esq	Sensor Dir	Sensor Mid	Motor Esq	Motor Dir
MP	MP	MP	0	0
MP	MP	P	0	0
MP	MP	L	0	0
MP	MP	ML	0	0
MP	P	MP	PM	NM
MP	P	P	PM	NM
MP	P	L	PM	NM
MP	P	ML	PM	NM
MP	L	MP	PA	0
MP	L	P	PM	0
MP	L	L	PA	PM
MP	L	ML	PA	PM
MP	ML	MP	PA	PM
MP	ML	P	PA	PM
MP	ML	L	PA	PM
MP	ML	ML	PA	PM
P	MP	MP	NM	PM
P	MP	P	NM	PM
P	MP	L	NM	PM
P	MP	ML	NM	PM
P	P	MP	0	0
P	P	P	0	0
P	P	L	PM	PM
P	P	ML	PM	PM

Entradas			Saídas	
P	L	MP	PM	NM
P	L	P	PM	NM
P	L	L	PA	PM
P	L	ML	PA	PM
P	ML	MP	PM	NM
P	ML	P	PM	NM
P	ML	L	PA	PM
P	ML	ML	PA	PM
L	MP	MP	0	PA
L	MP	P	0	PM
L	MP	L	PM	PA
L	MP	ML	PM	PA
L	P	MP	NM	PM
L	P	P	NM	PM
L	P	L	PM	PA
L	P	ML	PM	PA
L	L	MP	PM	PM
L	L	P	PM	PM
L	L	L	PA	PA
L	L	ML	PA	PA
L	ML	MP	PM	PM
L	ML	P	PM	PM
L	ML	L	PA	PA
L	ML	ML	PA	PA
ML	MP	MP	PM	PA
ML	MP	P	PM	PA
ML	MP	L	PM	PA
ML	MP	ML	PM	PA
ML	P	MP	NM	PM
ML	P	P	NM	PM
ML	P	L	PM	PA
ML	P	ML	PM	PA
ML	L	MP	PM	PM
ML	L	P	PM	PM
ML	L	L	PA	PA
ML	L	ML	PA	PA

Entradas			Saídas	
ML	ML	MP	PM	PM
ML	ML	P	PM	PM
ML	ML	L	PA	PA
ML	ML	ML	PA	PA

Tabela 1: Tabela de Regras

Na tabela, temos as possíveis entradas de todos os sensores, em variáveis nebulosas, e para cada conjunto de entradas, as respectivas saídas dos motores, também em variáveis nebulosas. Que serão convertidas em números absolutos pelo compilador. As siglas da tabela para os sensores são:

- MP = Muito perto
- P = Perto
- L = Longe
- ML = Muito longe

E as siglas para os motores são:

- PA = Positivo alto
- PM = Positivo médio
- 0 = Zero
- NM = Negativo médio
- NA = Negativo alto

Essas são todas as regras possíveis para o controle entretanto, não é necessário adicionar todas as regras na programação porque algumas delas são equivalentes e não precisam ser adicionadas para não usar memória demais podendo gerar lentidão durante o processo de controle. Das 64 regras existentes apenas X foram usadas e forem suficientes para o controle pleno do robô.

b Programação

Para programar tal controle foi utilizado uma biblioteca existente[16], que torna a programação do controle mais simples. Entretanto, pela presença de bastantes regras, fica trabalhoso colocar uma por uma no programa. A biblioteca já contém as regras, antecedentes - entradas - , consequentes - saídas - e também recebe valores de entradas e simula as saídas, mostrando o resultado em um gráfico - que mostra o centro de massa citado na seção anterior. Algumas imagens das simulações virão a seguir:

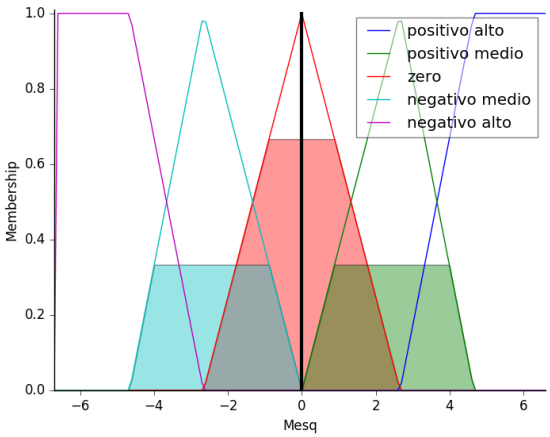


Figura 22: Todos os sensores a 0.5m

Como podemos ver na figura, a velocidade do motor da esquerda será aproximadamente 0 - linha preta vertical no gráfico - e como todos os sensores estão na mesma distância, o motor da direita terá a mesma velocidade. A próxima figura será com outra distância lida.

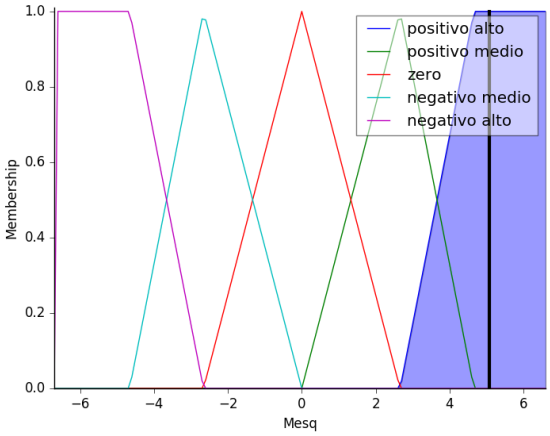


Figura 23: Todos os sensores a 1m

Já neste gráfico vimos que a velocidade do motor da esquerda é aproximadamente 5 rad/s que resultará em aproximadamente 0.375 m/s, uma velocidade razoável para a distância dada. Novamente os motores tem a mesma velocidade visto que os sensores estão a mesma distância.

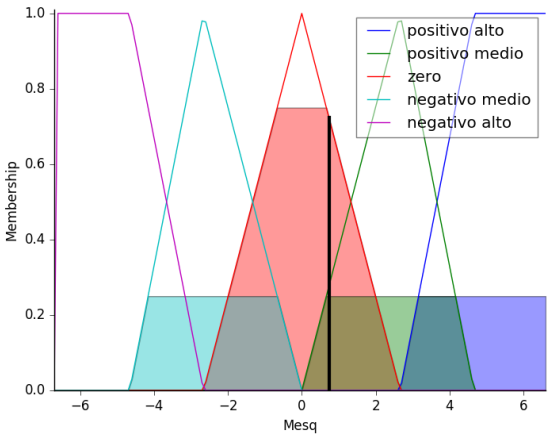
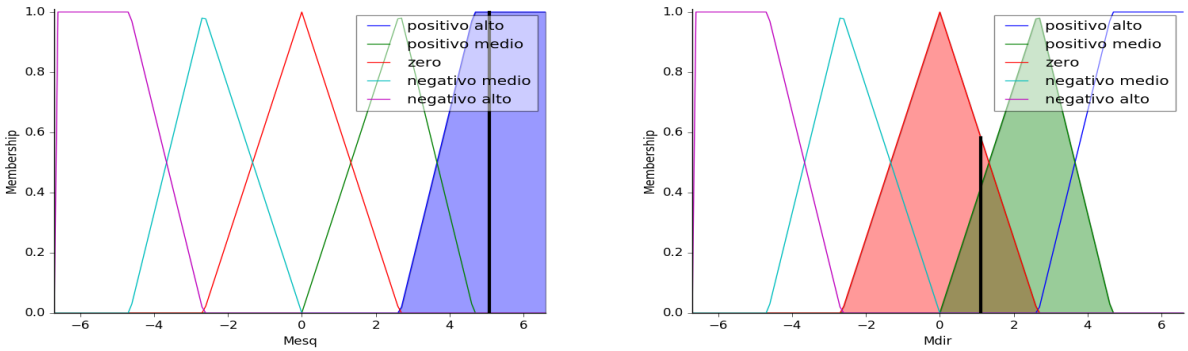


Figura 24: Todos os sensores a 0.7m

Agora que a distância não é nem perto, nem longe segundo as pertinências, a velocidade é corrigida para aproximadamente 0.7 rad/s - 0.05 m/s, reduzindo bastante a velocidade quando um objeto se aproxima. Os motores ainda possuem a mesma velocidade porque não precisa desviar de nenhum obstáculo.

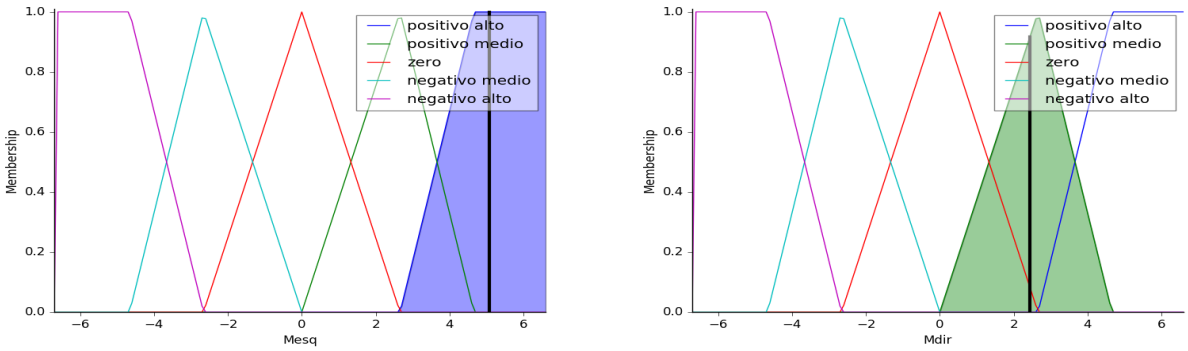


(a) Motor esquerdo

(b) Motor direito

Figura 25: Sensores: Esq - 0.3; Cen - 0.3; Dir - 1

Desta vez, dois gráficos foram adicionados porque os sensores das pontas possuem valores diferentes, o que vai fazer com que os motores também tenham velocidades diferentes. O sensor da esquerda observou um valor mais baixo, ou seja, mais perto. Portanto o motor da esquerda terá uma velocidade maior do que o direito para conseguir desviar do objeto. Velocidade da esquerda é por volta de 5 rad/s e a da direita é aproximadamente 1 rad/s.



(a) Motor esquerdo

(b) Motor direito

Figura 26: Sensores: Esq - 0.3; Cen - 1; Dir - 1

Agora ao perceber que o centro também está a uma distância considerável, maior que a anterior, ele aumenta a velocidade do motor direito, indo para pouco maior que 2 rad/s.

Para os cálculos de rad/s para m/s é necessário usar o raio da roda que, no caso, é de 7.5cm (0.075m). A fórmula que foi usada foi :$v = w * r$. Onde w é a velocidade angular, em rad/s, r é o raio proposto acima, em m, e v é a velocidade linear, em m/s.

5 Desafios

Em um projeto grande como esse, certamente, há inúmeros desafios. Alguns deles serão explicados abaixo, suas origens e quais foram os métodos escolhidos para deixá-los para trás.

a Eletrônicos

Um dos desafios da eletrônica do projeto foi a não fixação dos fios. O que pode levar em mal contato ou alguma conexão(ou até mesmo uma desconexão) indesejada devido à movimentação do robô. Os fios não foram soldados porque o projeto ainda é um protótipo e precisa de ajustes a todo momento portanto, os fios precisam ser conectados e desconectados a qualquer instante.

Outro desafio da eletrônica veio a partir do driver dos motores. Nos testes e durante a programação foi analisado que os drivers funcionam melhor quando a velocidade que lhes é passada é alta. A velocidade é passada ao motor em forma de pulsos e quanto menor o intervalo entre esses pulsos melhor o funcionamento do driver. Quando os pulsos estavam espaçados demais, o que gera uma velocidade menor, o motor não respondia de acordo com o esperado devido ao driver. Esse funcionamento inesperado do driver fazia ele esquentar demais mesmo com os dissipadores de calor acoplados a ele e com a plena ventilação da estrutura.

Eletronicamente falando, outro problema apareceu. Havia uma corrente flutuante que atrapalhava a leitura dos sensores e para contornar esse problema foi necessário um pequeno atraso, que foi implementado no código do Arduíno, e assim as leituras dos sensores foi estabilizada. A corrente flutuante fazia com que o resultado dos sensores não fosse constante como deveria e assim impossibilitava o controle da BIA.

b Computacionais

A parte computacional é de extrema importância nesse projeto, logo seus desafios devem ser contornados para que tudo ocorra como planejado. E estes desafios apareceram nos dois principais dispositivos utilizados para a área computacional do protótipo.

A interface do ROS no Raspberry é de utilidade quase imensurável e muito facilitadora porém, o ROS é um sistema operacional que necessita de outro sistema operacional como base, no caso, o Ubuntu. Por isso em computadores com pouca memória e/ou espaço em disco pode acarretar em lentidão para processar algo. E o programa do Arduíno também pode demorar um pouco para processar se o código for longo ou apresentar muitas variáveis e funções. Então, a comunicação pós estabelecimento é sólida e confiável mas sua inicialização não é das mais velozes.

Outro desafio foi a utilização de duas linguagens diferentes ao mesmo tempo(O Arduíno utiliza a linguagem C++ e o Raspberry utiliza Python). E ambos programas precisam estar funcionando para que o projeto atinja

todo seu potencial porque eles se comunicam a todo instante e se um deles não estiver funcionado o robô como um todo também não funcionará para seu objetivo.

O principal desafio computacional foi a configuração e modelagem do controle Fuzzy. Mesmo a modelagem e parte da configuração já estando pronta no LIRA (Laboratório de Inteligência e Robótica aplicadas), a grande maioria das configurações e funções de controle precisaram ser refeitas, testadas e simuladas. O ajuste fino das funções de pertinência para cada variável e para isso foi usado o sckit-fuzzy.

Embora já havia modelagem pronta, com a biblioteca usada a visualização dos resultados foi de forma mais natural, com gráficos de fácil interpretação. Outro ponto que foi negativo para o material do LIRA, foi a linguagem de programação utilizada, era necessário em Python e estava escrito em C.

c Mecânicos

Os principais desafios, com foco na mecânica, foram a estabilidade e a conexão das rodas com os motores. A estabilidade do robô está diretamente ligada a distribuição de sua massa, da geometria de sua carcaça e também de seu movimento, em geral. Manter o robô balanceado foi o principal desafio juntamento com a conexão dos motores com os eixos.

Inicialmente, a fixação do conjunto motor-eixo-roda também foi um problema. Havia folga nos eixos das rodas. Para corrigir esse pequeno defeito, foram utilizados parafusos nas cabeças dos eixos, logo após as rodas para fixá-las. O problema com os motores foi devido a falta dos mancais, o que deixava eles soltos porque estavam presos somente nos eixos e, por vezes, os motores giravam e não seu eixo devido a resistência maior no eixo do que no motor em si.

Para o desafio da estabilidade do robô, foi definida uma velocidade máxima de meio metro por segundo(0.5m/s). Para chegar nessa velocidade foi necessário fazer algumas operações matemáticas com os valores de pulsos gerados, frequência do controlador(16MHz), instruções internas do timer,características do motor, quantidade de micropassos por pulso e por fim, raio da roda. A velocidade no código é definida pela quantidade de chamadas da função por pulso. Juntamente com as outras variáveis do código consegue-se converter para radianos por segundo e enfim para metros por segundo.

Para complementar a fixação dos motores foi preciso imprimir 2 mancais na impressora 3D presente no laboratório. Com os mancais impressos, eles foram fixados na fundação do robô - foi feito 3 furos na base - e os motores foram parafusados nos mancais, deixando os motores fixos na base. Essa fixação melhorou a movimentação do robô já que inibiu o deslizamento entre os motores e eixos que, eventualmente, poderiam ocorrer.

Para chegar em números reais no controle e facilitar o entendimento para qualquer um foi preciso fazer contas e transformar variáveis como frequência dos passos do motor em m/s. Para isso, foi preciso pegar o timer e a frequência do Arduíno, 200 instruções por chamada do timer e 16000000 instruções por segundo, respectivamente. Após isso, quantas chamadas da funções do motor por pulso. Por outro lado, configuramos o driver para dar 32

passos por pulso e a cada pulso o motor gira 1.8º. Com esses dados temos a velocidade em rad/s - w . Depois disso usamos que V=w*r - r é conhecido - então, achamos a velocidade em m/s.

6 Resultados

Nesse trabalho foi implementado somente a movimentação do robô com o desvio de objetos incluído. A parte de interação com o usuário precisará ser feita depois, com a movimentação totalmente funcional.

Com tudo configurado anteriormente, tanto no aspecto físico quanto computacional, os resultados foram os esperados. Entretanto, é difícil demonstrar esses resultados em texto sem vídeos para comprovação, portanto, uma forma de quantificar e qualificar os resultados são vídeos e eles podem ser encontrados em : `https://www.youtube.com/playlist?list=PLyoCJm-9wZgkff85akDGcO3hGTEcIy-EM`. Como pode ser visto, o robô comportou conforme esperado e seu controle foi como previsto nas simulações.

Além do vídeo, o registro da leitura dos motores e os resultados dos motores foram plotados com ajuda do Matlab, afim de conseguir visualizar se os resultado estão condizentes. Para cada motor foi gerado um gráfico com os 3 sensores sendo os eixos e o valor do motor numa escala de cores, em 4D.

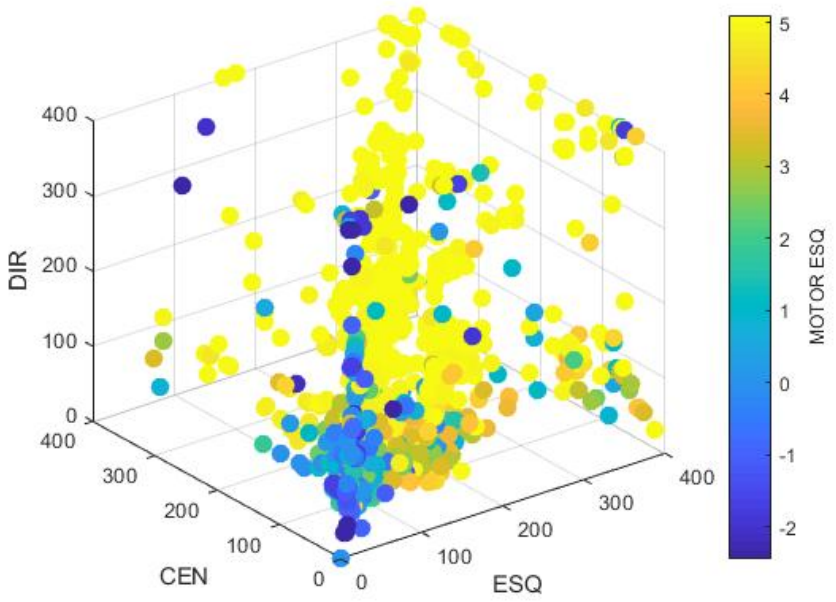


Figura 27: Resultado motor esquerdo

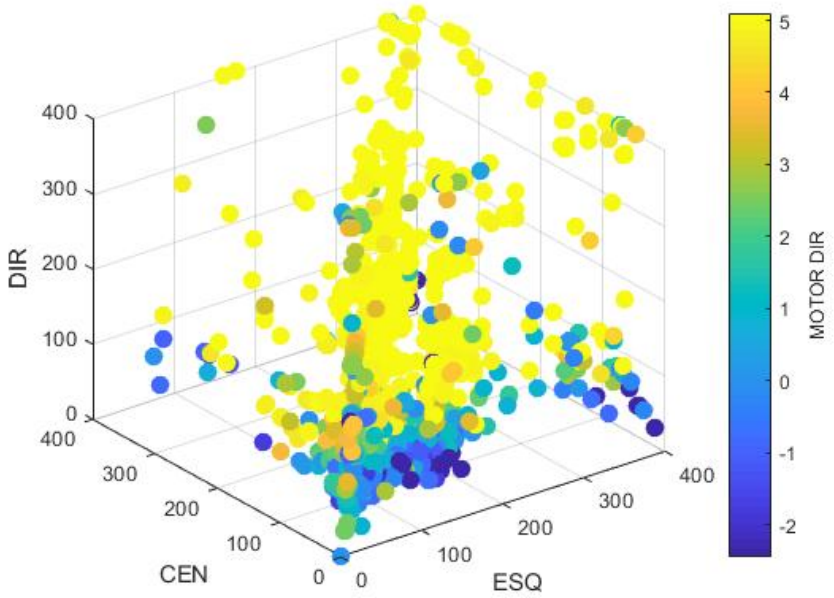


Figura 28: Resultado motor direito

Ademais os gráficos em 4 dimensões das leituras, também foram feitos gráficos separados para cada sensor e cada motor, uma vez que gráficos em 4D podem ser de difícil interpretação. Ao usar uma variável comum para todos os dados, o tempo, conseguimos visualizar o resultado. Para melhor percepção, os gráficos estão todos separados porém, a ideia inicial era colocar as entradas, os sensores, todas juntas e as saídas também juntas. Entretanto a visualização dos resultados ficou poluída e não conseguia ter uma conclusão sobre o resultado.

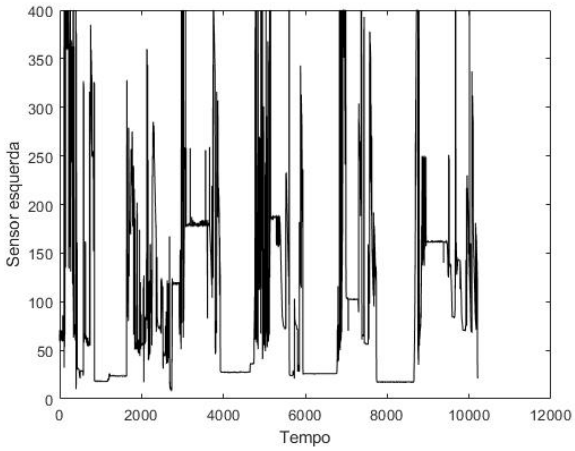


Figura 29: Resultado sensor esquerdo no tempo

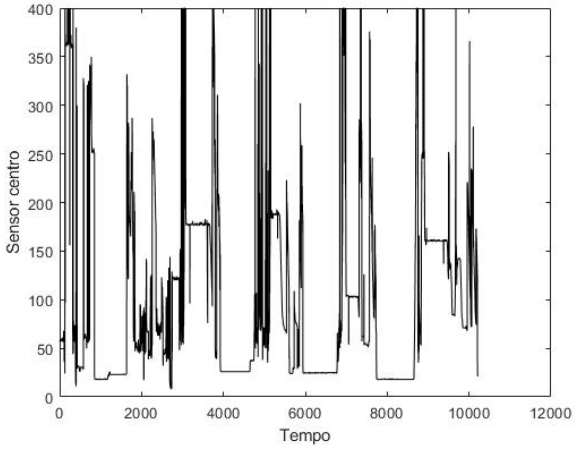


Figura 30: Resultado sensor esquerdo no tempo

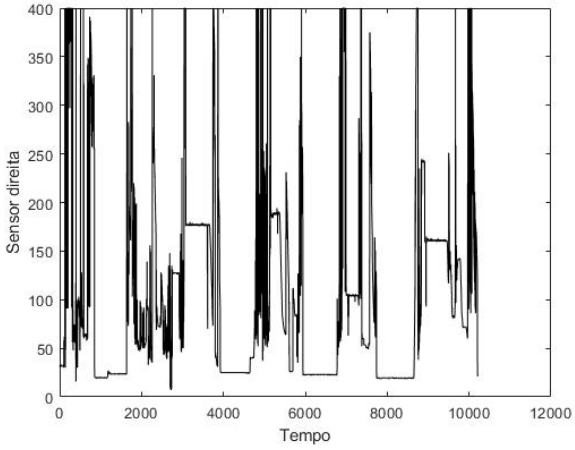


Figura 31: Resultado sensor direito no tempo

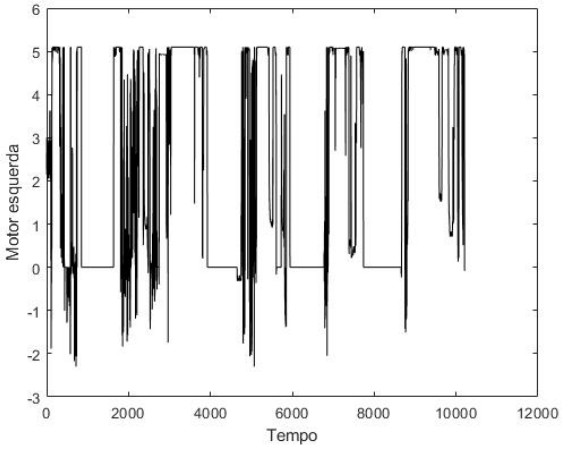


Figura 32: Resultado motor esquerdo no tempo

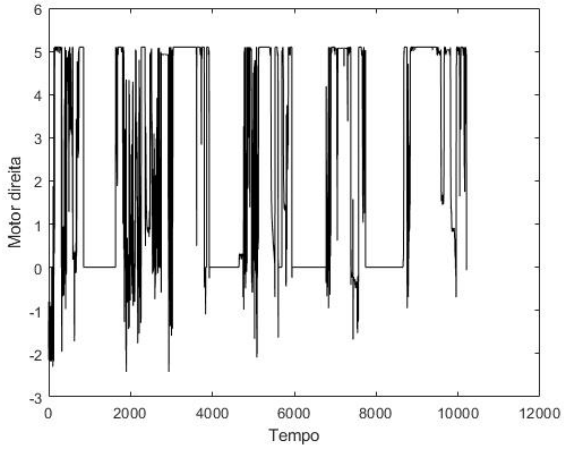


Figura 33: Resultado motor direito no tempo

Embora os resultados tenham sido satisfatórios, como visto acima, foi observado que o driver, com a corrente limitada, não gera torque suficiente para movimentar o robô, que em alguns momentos precisa de um empurrãozinho, literalmente. Esse problema já está em vias de ser resolvido com um driver novo, já encomendado pelo professor orientador.

7 Conclusão

a Futuras Melhorias

Apesar do projeto ter se mostrado adequado ao objetivo proposto, diversas melhorias podem ser implementadas ao mesmo.

b Possíveis melhorias

- Número maior de sensores e em locais mais espalhados;

  Para melhorar o controle do robô, sensores mais no alto para conseguir detectar mais obstáculos e, também nas laterais e na traseira.

- Mais espaço interno para organização dos componentes;

  Uma prateleira ou duas ajudariam na montagem.

- Melhora no algoritmo de controle(por exemplo, uma variável de memória);

- Bateria com maior autonomia;

  A bateria atual dá autonomia de pouco mais de 1h.

- Melhoria do driver do motor.

  Já encomendado pelo professor orientador. A melhoria no driver é essencial para o projeto conseguir uma maior fluidez na sua movimentação.

Toda documentação do projeto, assim como os arquivos com códigos gerados, gráficos, imagens estão em: `https://drive.google.com/drive/folders/19z9C7I5_UaTPycxzj_I0-2oCVjnnJJwa`.

Appendices

a Código Python

```
#!/usr/bin/python

import numpy as np
import skfuzzy as fuzz
import time
import rospy
import math
import time
from std_msgs.msg import Int32MultiArray
from std_msgs.msg import Int32
from skfuzzy import control as ctrl

#duvidas?: https://github.com/scikit-fuzzy/scikit-fuzzy

intm_left = 0
intm_right = 0
S_Esq = 0
S_Fre = 0
S_Dir = 0

def callback(msg):
        global intm_left, intm_right
        V_Esq = msg.data[0]
        V_Dir = msg.data[1]
        S_Esq = msg.data[2]
        S_Fre = msg.data[3]
        S_Dir = msg.data[4]
        global S_Dir
        global S_Esq
        global S_Fre
        if V_Esq != 0:
                V_Esquerda =20/V_Esq
        else:
```

```
            V_Esquerda = 0
        if V_Dir != 0:
            V_Direita = 20/V_Dir
        else:
            V_Direita = 0
        robot.input['Sesq'] = S_Esq/100.
        robot.input['Scen'] = S_Fre/100.
        robot.input['Sdir'] = S_Dir/100.
        robot.compute()
        intm_left = robot.output['Mesq']
        intm_right = robot.output['Mdir']


# New Antecedent/Consequent objects hold universe variables and membership
# functions
Sesq = ctrl.Antecedent(np.arange(0.01, 4.0, 0.01),'Sesq')
Scen = ctrl.Antecedent(np.arange(0.01, 4.0, 0.01),'Scen')
Sdir = ctrl.Antecedent(np.arange(0.01, 4.0, 0.01),'Sdir')

Mesq = ctrl.Consequent(np.arange(-8, 8, .1), 'Mesq')
Mdir = ctrl.Consequent(np.arange(-8, 8, .1), 'Mdir')

# Custom membership functions can be built interactively with a familiar,
# Pythonic API
Sesq['muito_perto'] = fuzz.trapmf(Sesq.universe, [0.01,0.01, 0.3, 0.6])
Sesq['perto'] = fuzz.trimf(Sesq.universe, [0.3, 0.6, 1])
Sesq['longe'] = fuzz.trimf(Sesq.universe, [0.6, 1, 1.5])
Sesq['muito_longe'] = fuzz.trapmf(Sesq.universe, [1, 1.5, 4,4])
Sdir['muito_perto'] = fuzz.trapmf(Sdir.universe, [0.01,0.01, 0.3, 0.6])
Sdir['perto'] = fuzz.trimf(Sdir.universe, [0.3, 0.6, 1])
Sdir['longe'] = fuzz.trimf(Sdir.universe, [0.6, 1, 1.5])
Sdir['muito_longe'] = fuzz.trapmf(Sdir.universe, [1, 1.5, 4,4])
Scen['muito_perto'] = fuzz.trapmf(Scen.universe, [0.01,0.1, 0.3, 0.6])
Scen['perto'] = fuzz.trimf(Scen.universe, [0.3, 0.6, 1])
Scen['longe'] = fuzz.trimf(Scen.universe, [0.6, 1, 1.5])
Scen['muito_longe'] = fuzz.trapmf(Scen.universe, [1, 1.5, 4,4])
```

```
Mesq['positivo_alto'] = fuzz.trapmf(Mesq.universe, [2.66, 4.66, 6.66,6.66])
Mesq['positivo_medio'] = fuzz.trimf(Mesq.universe, [0, 2.66, 4.66])
Mesq['zero'] = fuzz.trimf(Mesq.universe, [-2.66, 0, 2.66])
Mesq['negativo_medio'] = fuzz.trimf(Mesq.universe, [-4.66, -2.66, 0])
Mesq['negativo_alto'] = fuzz.trapmf(Mesq.universe, [-6.66,-6.66, -4.66, -2.66])
Mdir['positivo_alto'] = fuzz.trapmf(Mdir.universe, [2.66, 4.66, 6.66, 6.66])
Mdir['positivo_medio'] = fuzz.trimf(Mdir.universe, [0, 2.66, 4.66])
Mdir['zero'] = fuzz.trimf(Mdir.universe, [-2.66, 0, 2.66])
Mdir['negativo_medio'] = fuzz.trimf(Mdir.universe, [-4.66, -2.66, 0])
Mdir['negativo_alto'] = fuzz.trapmf(Mdir.universe, [-6.66, -6.66, -4.66, -2.66])


rule1 = ctrl.Rule(Sesq['muito_perto'] & Sdir['muito_perto'], (Mdir['zero'], Mesq['zero']))

rule2 = ctrl.Rule(Sesq['muito_perto'] & Sdir['perto'], (Mdir['negativo_medio'],
Mesq['positivo_medio']))

rule3 = ctrl.Rule(Sesq['perto'] & Sdir['muito_perto'], (Mdir['positivo_medio'],
Mesq['negativo_medio']))

#regra 2 e 3 sao o grupo 1 da tabela de regras.

rule4 = ctrl.Rule(Sesq['muito_perto'] & Sdir['longe'] & Scen['perto'], (Mdir['zero'],
Mesq['positivo_medio']))

rule5 = ctrl.Rule(Sesq['muito_perto'] & Sdir['longe'] & Scen['muito_perto'],
(Mdir['negativo_medio'], Mesq['positivo_alto']))

rule6 = ctrl.Rule(Sesq['muito_perto'] & Sdir['longe'], (Mdir['positivo_medio'],
Mesq['positivo_alto']))

rule7 = ctrl.Rule(Sesq['longe'] & Sdir['muito_perto'] & Scen['perto'],
(Mdir['positivo_medio'], Mesq['zero']))

rule8 = ctrl.Rule(Sesq['longe'] & Sdir['muito_perto'] & Scen['muito_perto'],
(Mdir['positivo_alto'], Mesq['negativo_medio']))
```

```
rule9 = ctrl.Rule(Sesq['longe'] & Sdir['muito_perto'], (Mdir['positivo_alto'],
Mesq['positivo_medio']))

# regra 4,5,6,7,8 e 9 sao o grupo 2 da tabela de regras.

rule10 = ctrl.Rule(Sesq['muito_perto'] & Sdir['muito_longe'],
(Mdir['positivo_medio'], Mesq['positivo_alto']))

rule11 = ctrl.Rule(Sesq['muito_longe'] & Sdir['muito_perto'],
(Mdir['positivo_alto'], Mesq['positivo_medio']))

#regra 10 e 11 sao o grupo 3 da tabela de regras.

rule12 = ctrl.Rule(Sesq['perto'] & Sdir['perto'] & Scen['perto'],
(Mdir['zero'], Mesq['zero']))

rule13 = ctrl.Rule(Sesq['perto'] & Sdir['perto'] & Scen['longe'],
(Mdir['positivo_medio'], Mesq['positivo_medio']))

rule14 = ctrl.Rule(Sesq['perto'] & Sdir['longe'] & Scen['perto'],
(Mdir['zero'], Mesq['positivo_medio']))

rule15 = ctrl.Rule(Sesq['perto'] & Sdir['longe'] & Scen['longe'],
(Mdir['positivo_medio'], Mesq['positivo_alto']))

rule16 = ctrl.Rule(Sesq['longe'] & Sdir['perto'] & Scen['perto'],
(Mdir['positivo_medio'], Mesq['zero']))

rule17 = ctrl.Rule(Sesq['longe'] & Sdir['perto'] & Scen['longe'],
(Mdir['positivo_alto'], Mesq['positivo_medio']))

#regra 14,15,16 e 17 sao o grupo 4 da tabela de regras.

rule18 = ctrl.Rule(Sesq['perto'] & Sdir['muito_longe'] & Scen['perto'],
(Mdir['zero'], Mesq['positivo_medio']))

rule19 = ctrl.Rule(Sesq['perto'] & Sdir['muito_longe'] & Scen['longe'],
```

```
(Mdir['positivo_medio'], Mesq['positivo_alto']))

rule20 = ctrl.Rule(Sesq['muito_longe'] & Sdir['perto'] & Scen['perto'],
(Mdir['positivo_medio'], Mesq['zero']))

rule21 = ctrl.Rule(Sesq['muito_longe'] & Sdir['perto'] & Scen['longe'],
(Mdir['positivo_alto'], Mesq['positivo_medio']))

#regra 18,19,20 e 21 sao o grupo 5 da tabela de regras.

rule22 = ctrl.Rule(Sesq['longe'] & Sdir['longe'], (Mdir['positivo_alto'],
Mesq['positivo_alto']))

rule23 = ctrl.Rule(Sesq['longe'] & Sdir['longe'] & Scen['muito_perto'],
(Mdir['positivo_medio'], Mesq['positivo_medio']))

rule24 = ctrl.Rule(Sesq['muito_longe'] & Sdir['muito_longe'],
(Mdir['positivo_alto'], Mesq['positivo_alto']))

rule25 = ctrl.Rule(Sesq['muito_longe'] & Sdir['muito_longe'] & Scen['muito_perto'],
(Mdir['positivo_medio'], Mesq['positivo_medio']))

rule26 = ctrl.Rule(Sesq['longe'] & Sdir['muito_longe'], (Mdir['positivo_alto'],
Mesq['positivo_alto']))

rule27 = ctrl.Rule(Sesq['longe'] & Sdir['muito_longe'] & Scen['muito_perto'],
(Mdir['positivo_medio'], Mesq['positivo_medio']))

rule28 = ctrl.Rule(Sesq['muito_longe'] & Sdir['longe'], (Mdir['positivo_alto'],
Mesq['positivo_alto']))

rule29 = ctrl.Rule(Sesq['muito_longe'] & Sdir['longe'] & Scen['muito_perto'],
(Mdir['positivo_medio'], Mesq['positivo_medio']))

#regra 26,27,28 e 29 sao o grupo 6 da tabela de regras.

robot_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8,
```

```
rule9,rule10, rule11, rule12, rule13, rule14, rule15, rule16, rule17, rule18, rule19,
rule20,rule21,rule22, rule23, rule24, rule25, rule26, rule27, rule28, rule29])
robot = ctrl.ControlSystemSimulation(robot_ctrl)

if __name__=='__main__':

        outfile  =  open("out.txt", 'a')
        rospy.init_node('chatter')
        rospy.Subscriber('chatter',Int32MultiArray, callback)
        pub = rospy.Publisher('feedback', Int32MultiArray, queue_size=1)
        rate = rospy.Rate(10)

        feedback = Int32MultiArray()
        feedback.data = [int(0), int(0)]

        while not rospy.is_shutdown():
                feedback.data[0] = intm_right
                feedback.data[1] = intm_left
                print(feedback.data)
                outfile.write('%d ' % S_Esq)
                outfile.write('%d ' % S_Fre)
                outfile.write('%d ' % S_Dir)
                outfile.write(' '.join([str(i) for i in feedback.data]))
                outfile.write('\n')
                pub.publish(feedback)
                rate.sleep()

        outfile.close()
```

b Código C

```
#define EN        8

//Direction pin
#define X_DIR     5
#define Y_DIR     6

//Step pin
#define X_STP     2
#define Y_STP     3

//Define os pinos para o trigger e echo – meio
#define pino_trigger_m 11
#define pino_echo_m 12

//Define os pinos para o trigger e echo – direita
#define pino_trigger_d 4
#define pino_echo_d 7

//Define os pinos para o trigger e echo – esquerda
#define pino_trigger_e 9
#define pino_echo_e 10

#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Int32MultiArray.h>

#include <Ultrasonic.h>

int v1=0;// velocidade do motor da direita   v maxima 20 / v minima 200 em modulo
int v2=0;// velocidade do motor da esquerda
long cmE, cmD, cmC;

//Inicializa o sensor nos pinos definidos acima
Ultrasonic ultrasonicm(pino_trigger_m, pino_echo_m);
Ultrasonic ultrasonicd(pino_trigger_d, pino_echo_d);
```

```
Ultrasonic ultrasonice(pino_trigger_e, pino_echo_e);
//DRV8825

void MessageFB(const std_msgs::Int32MultiArray& velo)
{
  v1 = velo.data[0];
  v2 = velo.data[1];
}

ros::NodeHandle  nh;

std_msgs::Int32MultiArray str_msg;
std_msgs::Int32MultiArray msg_fb;
int32_t str_data[5];
ros::Publisher chatter("chatter", &str_msg);
ros::Subscriber<std_msgs::Int32MultiArray> sub("feedback", &MessageFB);


void setup(){

  pinMode(X_DIR, OUTPUT);
  pinMode(Y_DIR, OUTPUT);
  pinMode(X_STP, OUTPUT);
  pinMode(Y_STP, OUTPUT);
  pinMode(EN, OUTPUT);

  digitalWrite(EN, LOW);  // HIGH desabilita os motores.
  digitalWrite(X_DIR, HIGH);
  digitalWrite(X_DIR, HIGH);
  digitalWrite(X_STP, LOW);
  digitalWrite(Y_STP, LOW);


  cli();

 TCCR3A=0;
 TCCR3B=0;
```

```
  TCNT3=0;

  OCR3A=400;
  TCCR3B|=(1<<WGM32);
  TCCR3B|=(1<<CS30);
  TIMSK3|=(1<<OCIE3A);

  sei();

  str_msg.data_length=5;
  str_msg.data = str_data;

  nh.initNode();

  nh.advertise(chatter);
  nh.subscribe(sub);

}

void loop(){

  le_sensores();

  float vel1=0, vel2=0;

  if (v1 != 0)
    vel1 = 80/v1;

  if (v2 != 0)
    vel2 = 80/v2;

  str_msg.data[0] = vel1;
  str_msg.data[1] = vel2;
  str_msg.data[2] = cmE;
  str_msg.data[3] = cmC;
  str_msg.data[4] = cmD;
```

```
  chatter.publish( &str_msg );
  nh.spinOnce();
  delay(100);

}

ISR(TIMER3_COMPA_vect)
{
 static int high1=0;
 static int d1=0;
 static int high2=0;
 static int d2=0;

 d1++;
 d2++;
// motor direita
 if (v1 && d1 >= abs(v1))
 {
   digitalWrite(Y_DIR, v1 > 0);

   if (high1)
   {
    digitalWrite(Y_STP, LOW);
   }
   else
   {
    digitalWrite(Y_STP, HIGH);
  }

  high1 = !high1;
  d1 = 0;
 }

//motor esquerda
if (v2 && d2 >= abs(v2))
 {
   digitalWrite(X_DIR, v2 > 0);
```

```
    if (high2)
    {
     digitalWrite(X_STP, LOW);
    }
    else
    {
     digitalWrite(X_STP, HIGH);
   }

   high2 = !high2;
   d2 = 0;
 }
//if (v1==0) && (v2==0)
// {
//  digitalWrite(En, HIGH): // enabla a placa se os dois sao 0
// }
}

void le_sensores(){

  cmC = ultrasonicm.Ranging(CM);
  cmD = ultrasonicd.Ranging(CM);
  cmE = ultrasonice.Ranging(CM);
  // dist 0 o pulso nao voltou, nao ha obstaculo

  if (cmC ==0)
  {
    cmC = 400;
  }

  if (cmD ==0)
  {
    cmD = 400;
  }

  if (cmE ==0)
```

```
    {
      cmE = 400;
    }
}
```

References

[1] . *AMY*. URL: http://www.amyrobotics.com/indexen. (acessado: 12.06.2019).

[2] Arduino. *Sensor Ultrassom*. URL: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. (acessado: 07.07.2019).

[3] . *Benebot*. URL: https://www.popsci.com/ces-2015-ecovacs-robotics-showcases-benebot-replacement-shopping-assistants-video/. (acessado: 10.06.2019).

[4] . *Bossanova*. URL: https://www.bossanova.com/. (acessado: 12.06.2019).

[5] *CareOBot*. URL: https://www.care-o-bot.de/en/care-o-bot-4.html. (acessado: 12.06.2019).

[6] *Conversor*. URL: https://pt.wikipedia.org/wiki/Conversor_buck. (acessado: 25.07.2019).

[7] Earl Cox. *The Fuzzy Systems Handbook,2nd edition*. Morgan Kaufmann, 1998. ISBN: 978-0121944551.

[8] *Driver controle*. URL: https://www.openimpulse.com/blog/wp-content/uploads/wpsc/downloadables/Arduino-CNC-Shield.pdf. (acessado: 07.07.2019).

[9] Karla Figueiredo. *BIA*. URL: https://www.dropbox.com/home/BIARobot/documentacao?select=Montagem+do+rob%C3%B4+BIA.docx. (acessado: 15.06.2019).

[10] . *Kiva*. URL: https://exame.abril.com.br/tecnologia/kiva-os-robos-espertos-que-a-amazon-comprou/. (acessado: 12.06.2019).

[11] . *KnightScope*. URL: https://www.knightscope.com/knightscope-k5/. (acessado: 12.06.2019).

[12] Ivo Lima Brasil Jr. Marley Velasco Marco Aurelio Pacheco. “Mobile Robot Control Using Fuzzy Logic”. In: ().

[13] NeoMotion. *Motores de Passo*. URL: https://neomotion.com.br/wp-content/uploads/2019/03/Cat%C3%A1logo-Datasheet-dos-motores-de-passo-R05.pdf. (acessado: 01.07.2019).

[14] . *Robo*. URL: https://pedrogarcia12av1.wordpress.com/about/robos-autonomos/. (acessado: 10.06.2019).

[15] *ROS*. URL: https://www.ros.org/. (acessado: 23.06.2019).

[16] *Scikit Fuzzy*. URL: https://github.com/scikit-fuzzy/scikit-fuzzy. (acessado: 10.07.2019).

[17] *Ultrasonic*. URL: https://www.arduinolibraries.info/libraries/ultrasonic. (acessado: 18.07.2019).

[18] Li Xin Wang. *Adaptative Fuzzy Systems and Control*. Prentice Hall, 1994. ISBN: 978-0130996312.