UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE PROCESSAMENTO DE ENERGIA ELÉTRICA
CURSO ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Thiago Rodrigues Garcia

CONSTRUÇÃO DE UM ROBÔ MÓVEL E CONTROLE DE TRAJETÓRIA UTILIZANDO CONTROLADORES FUZZY E PID COM ODOMETRIA

Santa Maria, RS, Brasil

2019

Thiago Rodrigues Garcia

CONSTRUÇÃO DE UM ROBÔ MÓVEL E CONTROLE DE TRAJETÓRIA UTILIZANDO CONTROLADORES FUZZY E PID COM ODOMETRIA

Trabalho de Conclusão apresentado ao Curso de Engenharia de Controle e Automação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Controle e Automação**

Orientador: Prof. Dr. Daniel Fernando Telo Gamarra

Santa Maria, RS

2019

Thiago Rodrigues Garcia

CONSTRUÇÃO DE UM ROBÔ MÓVEL E CONTROLE DE TRAJETÓRIA UTILIZANDO CONTROLADORES FUZZY E PID COM ODOMETRIA

Trabalho de Conclusão apresentado ao Curso de Engenharia de Controle e Automação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Controle e Automação**

Aprovado em 05 de julho de 2019

_____

Daniel Fernando Telo Gamarra. Dr
(Presidente/Orientador**)**

_____

Fábio Ecke Bisogno. Dr (UFSM, RS)

_____

Hamiltom Confortin Sartori. Dr (UFSM, RS)

Santa Maria, RS

2019

DEDICATÓRIA

Dedico esse trabalho ao meu irmão Christian e minha mãe Gleda (*in memorian*) que sempre farão parte da minha vida mesmo não estando mais entre nós.

AGRADECIMENTOS

Primeiramente aos meus pais pelo apoio e incentivo incondicional.

Ao orientador prof. Dr. Daniel Fernando Tello Gamarra que desde inicio sempre me incentivou, e orientou com muita competência e paciência.

A minha namorada Tatiana Borba, pelo total apoio e carinho nos momentos de estudos.

Ao Núcleo de Pesquisa e Desenvolvimento em Engenharia Elétrica (NUPEDEE) por fornecer total apoio para o desenvolvimento das atividades acadêmicas.

Ao grupo de robótica GARRA, em especial aos integrantes Rodrigo Mattos e Douglas Reis, pelo apoio no desenvolver do trabalho.

A todos que direta ou indiretamente fizeram parte de minha formação, o meu muito obrigado.

RESUMO

CONSTRUÇÃO DE UM ROBÔ MÓVEL E CONTROLE DE TRAJETÓRIA UTILIZANDO CONTROLADOR FUZZY E PID COM ODOMETRIA

AUTOR: Thiago Rodrigues Garcia

ORIENTADOR: Daniel Fernando Telo Gamarra

O presente trabalho consistiu em projetar e fabricar um robô móvel com o objetivo de obter uma plataforma para desenvolver atividades de ensino, pesquisa e extensão. Para que essas atividades possam ser realizadas, o robô móvel foi projetado para suportar diferentes tipos de sensores e atuadores normalmente utilizados nos projetos com robótica móvel. O robô construído foi utilizado para implementar e testar dois controladores de trajetória. Um controlador baseado em lógica *Fuzzy* e outro PID (Proporcional, Integral, Derivativo). Também foi implementada a odometria no robô móvel para que fosse possível estimar a sua posição e orientação, para isso, utilizou-se o método *Dead-Reckoning*. Os controladores, a estimação da posição e orientação e o sistema foram simulados em MatLab e Python. São descritos os resultados obtidos em simulação e com a plataforma robótica.

ABSTRACT

CONSTRUCTION OF A MOBILE ROBOT AND TRAJECTORY CONTROL USING FUZZY AND PID CONTROLLERS WITH ODOMETRY

AUTHOR: Thiago Rodrigues Garcia
ADVISOR: Daniel Fernando Telo Gamarra

The present work consisted of designing and manufacturing a mobile robot with the objective of obtaining a platform used for teaching, research, university and community activities. In order for these activities to carry, the mobile robot is designed to support different types of sensors and actuators normally used in mobile robotic. The built robot was used to implement and test two trajectory controllers. A controller based on Fuzzy logic and another based on PID (Proportional, Integral, Derivative). It was also implemented the odometry in the mobile robot, so it was possible to estimate its position and orientation, for that, the Dead-Reckoning method was used. The controllers, estimation of position and orientation in the system were simulated in MatLab and Python. The simulation results and the robotic platform are described.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Em (BEZERRA, 2004) a diversidade e relevância de muitas tarefas requerem que o robô possa executar ações como seguir uma trajetória determinada e conhecer sua localização e, para isso, o mesmo apresenta uma arquitetura híbrida utilizando odometria e processamento digital de imagens usando marcos naturais. O trabalho apresenta uma abordagem baseada em lógica *fuzzy* e odometria para resolver as referidas ações que são básicas para qualquer sistema que utilize robôs moveis.

O controle da trajetória do robô móvel utilizara um sistema baseado em lógica *fuzzy*, esta abordagem foi explorada anteriormente nos trabalhos de (OMRANE, 2016) que utilizou somente um controlador *fuzzy* para o controle da trajetória e para o desvio de obstáculos. Para isso, o mesmo utilizou recursos de simulação do software MatLab® chamado SIMIAN. Neste ambiente computacional foi possível tanto descrever trajetórias quanto desviar de obstáculos. (XIONG, 2010) desenvolveram um método para veículos seguidores de caminho com 2 controladores *fuzzy*.

Um dos métodos mais comuns para estimar posição e orientação de robôs é o *Dead-Reckoning*. E dentro da literatura há diversos trabalhos que utilizam *Dead-Reckoning* como auxilio para desenvolver técnicas, métodos e algoritmos para controle de trajetórias de robôs móveis. Muitas pesquisas com o intuito de aperfeiçoar a localização e orientação de robôs têm sido exploradas, podemos relatar o trabalho de (PARK, 1998) que apresenta um sistema de navegação para um robô móvel com rodas de tração diferencial que utiliza o método *Dead-Reckonig*. O mesmo é composto por um giroscópio e encoders acoplados nas rodas, e para aumentar a confiabilidade e diminuir o esforço computacional é aplicado um Filtro de Kalman alternativo na fusão dos dados dos sensores.

Em (FU,2013) , é apresentada uma rede de sensores sem fio baseada na localização calculada pelo método *Dead-Reckoning*. Com base nisso, é aplicado um Filtro de Kalmam para reduzir as incertezas do sistema. O autor também demonstra que é possível aplicar mais de uma vez o método *Dead-Reckoning* (Backward) para obter posições mais precisas.

Algumas aplicações também unem mais de uma técnica de navegação sendo que os sistemas *fuzzy* são capazes de auxiliar em técnicas mais complexas como no trabalho proposto por FARIA et al. Neste, um robô móvel é constituído por 7 sonares que, através de um sistema *fuzzy*, buscou classificar melhor as suas leituras para desvios de obstáculos e elaboração de

trajetórias. As leituras das distâncias dos sonares eram classificadas de acordo com as regras *fuzzy* e o resultado deste sistema gerou um grau de incerteza que posteriormente foi incorporado a um algoritmo de aprendizado por reforço *R'-Learning*.

Em (MOREIRA, 2018), é implementado um sistema de controle PID em um servomotor, aplicado a um robô móvel. Este sistema de controle é utilizado no TFC-KIT(The Freescale Cup), um protótipo de robô autônamo. Nesse sistema, o protótipo é capaz de seguir uma trajetória utilizando delimitada por duas linhas pretas em um chão branco usando sensor óptico (câmera). O controle PID é calculado através de dados recebidos pela câmera em que é possível medir a posição das rodas do robô móvel em relação as linhas pretas e, desta forma, atuar nos servos motores das rodas para que o mesmo se possa seguir a sua trajetória de referência. Bons resultados foram obtidos utilizando esta arquitetura e estratégia de controle.

(DIAS, 2009) utiliza uma estratégia de controle em um robô móvel com tração diferencial para controlar o seguimento de trajetória que implementa um controlador adaptativo e robusto, porém, também implementa outros dois controladores PID, um para cada roda. O intuito deste controlador é compensar as perturbações no modelo cinemático devido aos transitórios existentes com os controladores PID. Com isto, é possível anular as incertezas paramétricas, dinâmicas não modeladas e disturbios uniformemente limitados. (DIAS, 2009) também destaca que durante o transitório do sistema em malha fechada com PID, o modelo cinemático não é válido devido a dinâmica do próprio controlador PID. É importante destacar que essas características se tornam mais acentuadas e expressiva quando se deseja deslocamentos maiores velocidades ou em transporte de cargas.

No trabalho de (RIBEIRO, 2011), o autor utiliza um sistema de contro PI para controlar um robô móvel e, também monitora sua trajetória através de um sistema supervisório em tempo real desenvolvido na plataforma Lazarus IDE. A unidade de processamento utilizada foi o PIC32. Fisicamente, o robô móvel é constituido por três motrores de corrente contínua espaçados em 120º entre si na base. Sendo assim, a estratégia de controle considera apenas as características dinâmicas das malhas de controle e, também, é utilizado o modelo cinemático de sua respectiva base. O trabalho conta, também, com um grande diferencial que é a implementação de um sistema supervisório em tempo real. O objetivo do autor ao projetar esse sistema foi torná-lo responsável pelos modos operacionais, adquirir os dados, apresentar um relógio de tempo real e do esforço computacional além de ilustrar os movimentos do robô em um ambiente de navegação específico.

Uma visão mais aprofundada acerca de sistemas de controle pode ser observada em (NASCIMENTO, 2009). O autor, além de apresentar o modelo cinemático e dinâmico em

espaço de estados em sua dissertação, também, direciona em seu trabalho o controle multivariavel de trajetória de um robô móvel. Mais de um sensor é utilizado em sua plataforma robótica, o que é muito importante, pois, com suas respectivas informações e juntamente com o planejamento de execução da trajetória, concentra todo o controle em uma equação para facilitar o processamento final. A sua arquitetura física é composta por três rodas omnidirecionais defasadas em 120º entre si. Rodas omnidirecionais possuem rolamentos ao longo de sua superficie de contato que permitem que a roda deslize, também, na direção lateral e, assim, aumentao um grau de liberdade do robô.

A partir disso, este trabalho tem como objetivo simular e implementar um controlador baseado em controlador baseado em lógica *fuzzy* e outro baseado em PID utilizando dados de odometria calculados através do método *Dead-Reckoning*. Os testes experimentais foram realizados em uma plataforma robótica que foi construída em nosso laboratório descrevendo uma determinada trajetória. O trabalho esta dividido em 6 seções: sendo a primeira uma breve Introdução; a segunda seção apresenta o embasamento teórico onde é abordado o modelo cinemático do robô, a lógica *fuzzy*, controle PID e o método *Dead-Reckoning*; a terceira seção descreve o projeto e fabricação do robô; a quarta seção aborda as metodologias adotadas; a quinta seção mostra os resultados obtidos; a sexta seção apresenta as conclusões.

Descrição Do Problema

Será construído um robô móvel terrestre de tração diferencial para que o mesmo possa servir tanto em trabalhos de pesquisa quanto em ensino, auxiliando na prática de técnicas de controle, integração sensorial, inteligência artificial, etc. Posteriormente são implementados dois controladores, um baseado em lógica fuzzy e outro em PID, ambos para fazer o robô seguir uma trajetória pré-definida.

Objetivos

Este trabalho tem como objetivos gerais criar uma plataforma de robótica móvel para que possa desenvolver e aplicar técnicas e conceitos teóricos de áreas como a robótica, sistemas de controle e inteligência artificial. Estas aplicações servirão para atender o ensino, pesquisa e extensão.Este trabalho também tem como objetivo implementar um controlador PID (Proporcional Integral Derivativo) e um controlador *Fuzzy*, ambos utilizando odometria.

Objetivos específicos

Os objetivos deste trabalho são:

- Projetar e construir o robô móvel;

- Implementar e simular no software de simulação SimIam a odometria usando o método *Dead-Reckoning*;

- Implementar e simular no software de simulação SimIam a odometria com o controlador PID;

- Implementar em linguagem Python a odometria com o controlador PID, integrá-los com a plataforma Arduino e Driver MD49;

- Implementar o controlador *Fuzzy* utilizando toolbox do software MatLab®;

- Implementar e simular no software de simulação SimIam a odometria com o controlador *Fuzzy*;

- Realizar testes experimentais;

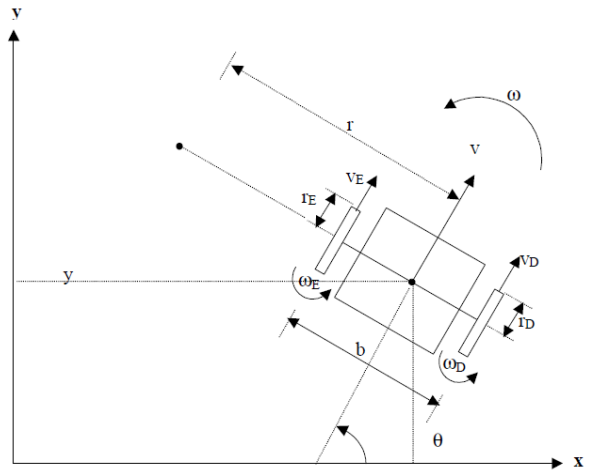
2. EMBASAMENTO TEÓRICO

A seção de Embasamento Teórico descreve as teorias, cálculos e métodos necessários para a realização deste trabalho. Desta forma, é possível ter um melhor entendimento do problema para traçar as metodologias mais simples e adequadas.

2.1. Modelo Cinemático

O modelo cinemático do robô móvel utilizado neste trabalho pode ser observado na Figura 1. O robô possui duas rodas independentes de tração diferencial e uma roda de movimento livre, denominada roda Castor, para manter o equilíbrio da estrutura.

Figura 1 . Modelo cinemático do robô. Vista superior



Fonte: https://www.dca.ufrn.br/~pablo/FTP/sist_rob/cap02.pdf

Onde:

(x, y) = Posição do referencial fixo no robô em relação ao referencial fixo no espaço de trabalho.

θ = Ângulo de orientação do robô em relação ao referencial fixo no espaço de trabalho.

b = Comprimento do eixo.

r = Raio de giro do robô.

$r_d(r_e)$ = Raio da roda direita (e esquerda).

ω = Velocidade angular do robô.

$\omega_d(\omega_e)$ = Velocidade angular da roda direita (e esquerda).

v = Velocidade linear do robô.

$v_d(v_e)$ = Velocidade linear da borda da roda direita (e esquerda).

A velocidade linear e a velocidade angular do robô podem ser escritas em termos da velocidade angular das rodas, do raio e do comprimento do eixo do robô (distância entre as rodas). Desta forma, é possível encontrar a relação entre a velocidade linear v e angular ω do robô conforme mostra a equação 1.

$$\begin{bmatrix} \omega \\ v \end{bmatrix} = \begin{bmatrix} (r_d/2)(r_e/2) \\ (r_d/b) - (r_e/b) \end{bmatrix} * \begin{bmatrix} \omega_d \\ \omega_e \end{bmatrix} \quad (1)$$

As coordenadas (x, y) e θ dão a posição do robô no plano cartesiano. É relevante observar que, caso as velocidades das rodas forem iguais (v_d=v_e), o robô percorrerá uma linha reta. Se as velocidades forem diferentes e a velocidade da roda direita for maior que a da roda esquerda (v_d>v_e), o robô descreverá uma trajetória circular no sentido anti-horário. De outro modo, quando a velocidade da roda esquerda for maior que a da roda direita (v_e>v_d), o robô percorrerá uma trajetória circular no sentido horário.

2.2 Método Dead-Reckoning

Um dos problemas primordiais na robótica móvel consiste em determinar a localização atual do robô. Com essa noção é possível deslocar-se para um próximo ponto e com isso descrever a trajetória desejada.

O método Dead-Reckoning consiste em calcular-se o deslocamento linear e angular do robô a partir dos deslocamentos das rodas obtidos por encoders incrementais.

Pode-se encontrar a distância percorrida pelo ponto central (equação 2) do robô a partir da média das distâncias entre as duas rodas.

$$d_{centro} = \frac{d_{esquerda} + d_{direita}}{2} \quad (2)$$

Onde $d_{esquerda}$ e $d_{direita}$ correspondem, respectivamente, a distância percorrida pela roda esquerda e roda direita. Por sua vez, a distância percorrida por cada roda é dada pela relação

$$d = 2.\pi.R\frac{\Delta tick}{N} \quad (3)$$

onde

N = Número de pulsos por volta dos encoders.

Δtick = Diferença da quantidade de pulsos atuais e a quantidade de pulsos da última medição.

Determina-se a variação do ângulo de orientação do robô, denominado phi (ø), com base nas distâncias percorridas por cada roda.

$$ø = \frac{d_{direita} - d_{esquerda}}{b} \quad (4)$$

sendo que b é distância entre as rodas (comprimento do eixo).

Logo o ângulo de orientação atual do robô fica:

$$ɵ' = ɵ + ø \quad (5)$$

Por fim, a posição atual do robô pode ser obtida pela relação a seguir.

$$x' = x + d_{centro}\cos(ɵ) \quad (6)$$
$$y' = y + d_{centro}\sin(ɵ) \quad (7)$$

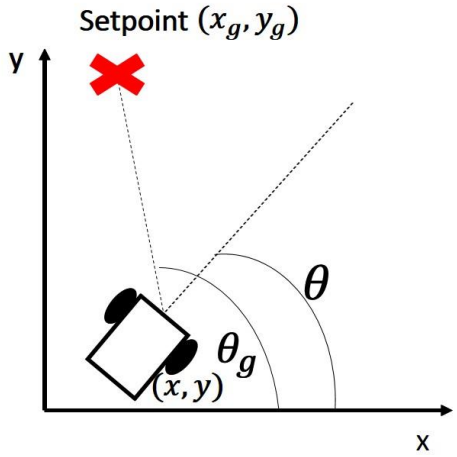
Evidencia-se que esse método consiste em contar-se a quantidade de pulsos obtidos a partir dos encoders de cada roda em um certo intervalo de tempo. Logo o cálculo da posição atual depende da posição passada, conduzindo ao acúmulo de erros. Sendo esse método não indicado para longas distâncias. A odometria não é o único método para a localização de robôs é possível também utilizar marcadores visuais como no trabalho de (BATISTA, 2016).

2.3 Controlador PID

O objetivo de implementar um controlador PID neste trabalho se deve a sua simplicidade visto que o controle é feito com base no seu modelo cinemático. A variável manipulada, neste caso, é a velocidade angular w. O robô também possui uma velocidade linear que é mantida constante. A velocidade angular w é convertida em rotação para as suas rodas para que o robô possa muda a sua orientação, representado por θ, a ação de controle visa

compensar a diferença entre os o ângulo θ do robô e o ângulo θ_g do Setpoint. A figura 2 ilustra as posições e orientações do robô e do Setpoint.

Figura 2 . Localização do robô e do Setpoint



Fonte: Autor

No programa escrito em python, são implementadas equações e variáveis do contorlador. Adotando como x_g e y_g são os pontos de referência para o robô, onde $x\ e\ y$ são a localização atual do robô e θ como respectivo ângulo de orientação, temos:

$$e_x = x_g - x \quad (8)$$

$$e_y = y_g - y \quad (9)$$

$$\theta_g = arctg(e_y, e_x) \quad (10)$$

$$gama = \theta_g - \theta \quad (11)$$

Sendo que e_x e e_y são o erro entre a componente x da posição do objetivo com a posição estimada e o erro da componente y da posição do objetivo com a posição estimada, respectivamente.

O erro do ângulo objetivo com o estimado é representado pela variável $gama$.

$$w = (K_P * e_P) + (K_I * e_I) + (K_D * e_D) \quad (12)$$

onde:

$$e_P = gama \quad (13)$$

$$e_I = prev_{eI} + (gama * dt) \quad (14)$$

$$e_D = (e_P - prev_{eP})/dt \quad (15)$$

e_P, e_I e e_D são as parcelas do controlador PID relativas aos erros proporcional, integral e derivativo, repectivamente. Estas parcelas são constantemente atualizadas a cada iteração. Sendo as constantes K_P, K_I e K_D os ganhos PID, estes valores são fixos a cada iteração e

somente é alterado pelo o usuário para mudar o comportamento do controlador. É necessário armazenar os valores de e_P *e* e_I (do erro proporcional e integral, respectivamente) para ser computados na próxima iteração, para isso, estes valores são armazenados nas variáveis $prev_{eP}$ e $prev_{eI}$. O erro relativo a parcela derivativa é a própria diferença entre o erro da parcela proporcional atual e anterior ($e_P - prev_{eP}$) dividos pela fração de tempo dt. Esta, é o tempo que se passou desde a última iteração. Para calular o dt, é necessário armazenar o tempo atual, chamado de $current\ time$, e subtrair do tempo armazenado na última iteração, sendo que a variável que armazena este tempo é a $prev\ time$.

$$current\ time = time.clock() \quad (16)$$

$$dt = current\ time - prev\ time \quad (17)$$

$$prev\ time = current\ time \quad (18)$$

$$prev_{eP} = e_P \quad (19)$$

$$prev_{eI} = e_I \quad (20)$$

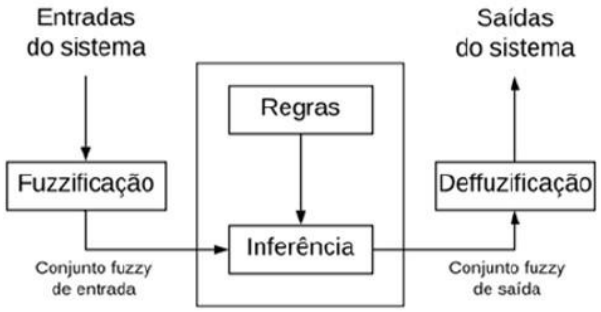
2.4 Sistema Fuzzy

Lógica *fuzzy* consiste num sistema que difere dos sistemas lógicos tradicionais por considerar valores aproximados, ou seja, a lógica *fuzzy* admite determinados graus de pertinências ao invés de exatos. Ou seja, Na lógica *fuzzy*, o valor verdade de uma proposição pode ser um subconjunto *fuzzy* de qualquer conjunto parcialmente ordenado, ao contrário dos sistemas lógicos binários, onde o valor verdade só pode assumir dois valores: verdadeiro (1) ou falso (O) (GOMIDE, 1994). Ainda, de acordo o mesmo, a lógica *fuzzy* constitui a base para o desenvolvimento de métodos e algoritmos de modelagem e controle de processos, permitindo a redução da complexidade de projeto e implementação, tornando-se a solução para problemas de controle até então intratáveis por técnicas clássicas.

Normalmente um controlador *fuzzy* é composto pelas seguintes etapas: fuzzificação, conjunto de regras, inferência e defuzzificação. Na fuzzificação é onde as entradas do sistema são modeladas, ou seja, onde são atribuídos graus de pertinência para as mesmas (BELLUCI, 2009). O conjunto de regras é base de regras onde se caracteriza a estratégia de controle e suas metas. O processo de Inferência processa os dados *fuzzy* de entrada, junto com as regras, de modo a inferir ações de controle *fuzzy*, aplicando o operador de implicação e as regras de

inferência . Na defuzzyficação ocorrerá a conversão da saída de controle *fuzzy* em um número real. Ou seja, é a determinação da ação de controle que melhor represente a decisão *fuzzy*. A Figura 3 ilustra, em forma de diagrama, um sistema *fuzzy*.

Figura 3 . Sistema Fuzzy



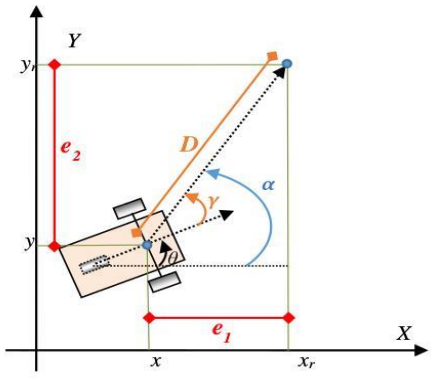
Fonte: (COELHO, 2018)

2.5 Controlador Fuzzy

Um controlador *fuzzy* foi projetado para o robô mover-se de um ponto a outro. O controlador *fuzzy* tem como objetivo minimizar a diferença de ângulo entre a orientação do robô e a orientação do próximo ponto (ângulo ɣ) e a distância entre as duas retas. As regras *fuzzy* são representadas por duas entradas e duas saídas. O modelo básico do controlador conhecido na literatura e as regras *fuzzy* foram fundamentadas no trabalho de (MOTTA, 2017).

O controlador recebe a distância (D) e a diferença de ângulo (ɣ) entre o robô e o ponto de destino e retorna à velocidade linear e à velocidade angular necessárias para minimizar, respectivamente, a distância e a diferença de ângulo. Na figura abaixo pode-se observar o gráfico representando o centro de massa do robô móvel (x,y) e o ponto que deseja-se alcançar (x_r,y_r). Onde $\gamma = \alpha - \theta$. As regras fuzzy abordadas fundamentam-se totalmente na Figura 4 que apresenta as variáveis do sistema.

Figura 4 . Distância entre o centro de massa do robô e o ponto destino



Fonte: (PIMENTEL, 2013)

Da figura podemos concluir que se ɣ for positivo, ou seja, α > θ, o robô precisará virar para à esquerda para aumentar θ, minimizando a diferença entre os ângulos e vice-versa. Se o

ângulo ɣ for nulo, o robô deverá percorrer uma reta. A denominação das variáveis *fuzzy* e as regras *fuzzy* são explicitadas na Tabela 1 e Tabela 2 respectivamente:

Tabela 1. Regras Fuzzy

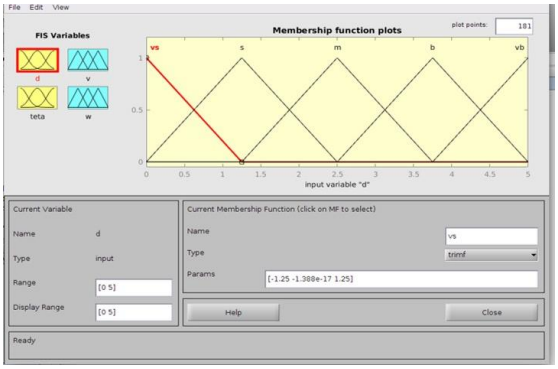
			ɣ				
		Ação	**NB**	**NS**	**Z**	**PS**	**PB**
D	**VS**	v	VS	VS	VS	VS	VS
		ω	RF	RS	Z	LS	LF
	S	v	S	S	S	S	S
		ω	RF	RS	Z	LS	LF
	M	v	M	M	M	M	M
		ω	RF	RS	Z	LS	LF
	B	v	B	B	B	B	B
		ω	RF	RS	Z	LS	LF
	VB	v	VB	VB	VB	VB	VB
		ω	RF	RS	Z	LS	LF

Tabela 2. Variáveis Fuzzy

Variável	Denominação	Descrição
ɣ	NB	Negative Big
	NS	Negative Small
	ZE	Zero
	PS	Positive Small
	PB	Positive Big
D e v	VS	Very Small
	S	Small
	M	Medium
	B	Big
	VB	Very big
ω	LF	Left Fast
	LS	Left Small
	Z	Zero
	RS	Right Slow
	RF	Right Fast

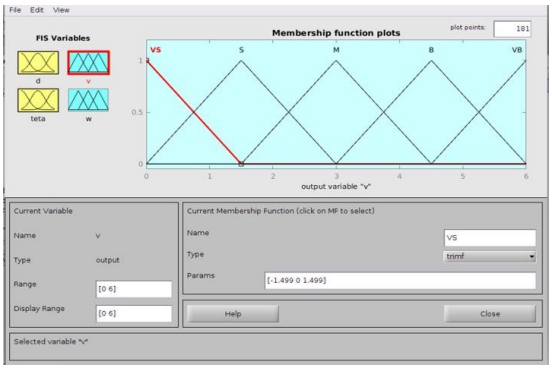
Após definir as regras e variáveis fuzzy, foi utilizado a toolbox *Fuzzy* do software MatLab® para projetar e gerar um arquivo que, posteriormente, será utilizado para gerar as ações de controle. A Figura 5 e a Figura 6 mostram o ambiente de programação da toolbox.

Figura 5 . Toolbox Fuzzy



Fonte: Autor

igura 6 . Toolbox Fuzzy



Fonte: Autor

A partir da velocidade máxima que cada roda pode assumir, é possível calcular a velocidade angular (ω) e linear (v) do robô, desse modo vel_e *e* vel_d são respectivamente a velocidade da roda esquerda e da roda direita, da seguinte forma:

$$v = \frac{r}{2} * (vel_d + vel_e) \omega = \frac{r}{b} * (vel_d - vel_e) \quad (21)$$

Modificações foram feitas na proposta original do controlador com a finalidade de adaptação do controlador para o robô. O robô projetado para os testes possui dimensões maiores e motores mais robustos, sendo necessária a alteração do intervalo de valores, bem como as relações das funções de pertinências, de todas as variáveis fuzzy. A tabela 3 apresenta os intervalos de valores que as variáveis de entrada e saída do sistema Fuzzy podem assumir e são descritos abaixo.

Tabela 3. Intervalo de valores das variáveis *fuzzy* do controlador com D*ead-Reckoning*.

Distância (D)	0 - 6 (metros)
Diferença de ângulo (ɣ)	-3.14 - 3.14
Velocidade linear do robô (v)	0 - 7.6
Velocidade ângular do robô (ω)	-15.6 - 15.6

3. PROJETO E FABRICAÇÃO DO ROBÔ

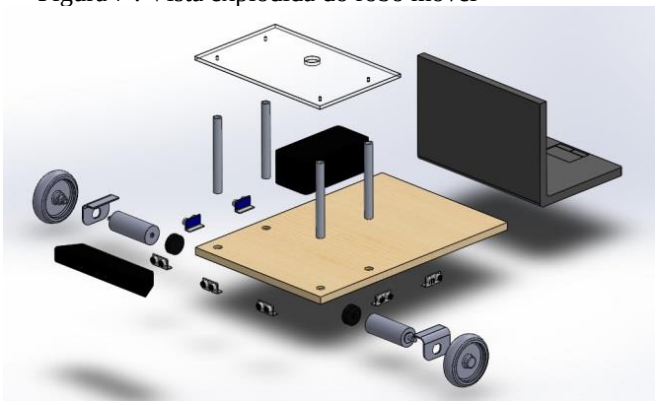
Esta seção irá descrever como o robô foi projetado e fabricado de acordo com as necessidades e disponibilidade de recursos. Também é explicado com detalhes os elementos que utilizados e os que podem ser utilizados em outros trabalhos.

O robô foi construído pelo grupo de extensão de robótica da universidade chamado de GARRA (Grupo de Automação e Robótica Aplicada). A metodologia de construção seguida para a construção do robô envolveu três componentes principais: projeto mecânico, equipamentos eletricos e equipamentos eletrônico.

3.1 Projeto Mecânico

Mecanicamente, o robô foi projetado para oferecer suporte a diversos tipos de sensores e componentes no estudo da robótica móvel. A base é de madeira MDF com 18 mm de espessura projetada para suportar o peso dos sensores e arquiteturas de sistemas que envolvam um notebook como mostra a Figura 7.

Figura 7 . Vista explodida do robô móvel



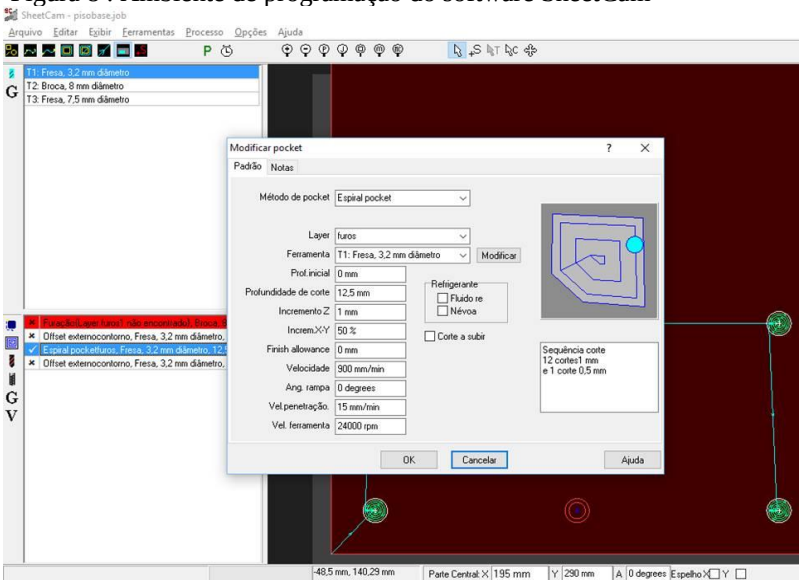
Fonte: Autor

Conforme a Figura 7, há ainda um pavimento superior de acrílico suportado por 4 hastes cilíndricas de alumínio no qual serve para anexar outros sensores como kinect, sensor laser e câmera. Este segundo pavimento é removível para facilitar a manutenção e conexões entre os componentes.

A fabricação dos furos e rebaixos da base e os componentes que suportam os motores foram realizados em uma fresadora CNC para ter maior precisão e produtividade. A fresadora utiliza o sistema CAD/CAM para executar os programas CNC. Um programa CNC é uma lista

de instruções codificadas que descrevem como a peça será usinada, sendo que cada linha do programa é chamada de bloco, e esses blocos são executados sequencialmente (GONÇALVES, 2007). O sistema CAD/CAM permite interpretar a geometria de uma peça criada em software CAD e gerar programas CNC. O software CAM utilizado é SheetCAM. A Figura 8 ilustra o ambiente do software SheetCAM e a configuração para a realização do processo de rebaixos na base do robô, respectivamente.

Figura 8 . Ambiente de programação do software SheetCam



Fonte: Autor

3.2 Equipamentos Eletrônicos

O robô é constituído de equipamentos elétricos e eletrônicos suficientes para realização de tarefas que envolvam programação e conceitos de robótica móvel, porém, ele foi projetado para suportar outros tipos de sensores como, por exemplo, sensor laser, kinect e giroscópio. Os quais o grupo de robótica já possui e se encontram em fase de estudo. A seguir, são descritos os componentes que podem ser utilizados no robô.

3.2.1 Driver

Para drenar a corrente para os motores e também protegê-los contra curto-circuito é utilizado o driver MD49, conforme mostra a Figura **9** . Driver MD49 O driver também serve para ler os dados dos encoders, fornecendo pulsos para determinar a sua velocidade de rotação

e sentido de giro acessando os registradores via software. Possui alimentação de 24V e realiza o controle dos motores de corrente continua utilizados pelo robô para seu deslocamento.

Figura 9 . Driver MD49

Fonte: Autor

3.2.2 Sensor Ultrassônico

O robô móvel foi projetado para suportar sensores ultrassônicos. Os sensores são muito utilizados em robótica móvel (Figura 10) devido a sua praticidade e facilidade de programação. Muito útil para ler informações do ambiente, evitar colisões e desviar de obstáculos. Seu princípo de funcionamento se baseia na emissão e recepção de uma onda sonora (não audível para o ser humando). A onda ao bater em algum objeto ela será refletida pelo mesmo e, ao recebê-la e cronometrando o tempo a partir da emissão, é possível calcular a distância levando em conta a velociadade som. Embora não tenham sido utilizados nesse trabalho, os sensores de ultrassom foram montados no robô e utilizados em outro trabalho de conclusão de curso.

Figura 10 . Sensor ultrassônico

Fonte: Autor

3.2.3 Kinect

Um dos sensores que está cada vez ganhando mais espaço no campo da pesquisa de visão computacional é o Kinect (Figura 11). Primeiramente desenvolvido para jogos de video game com o intuito que os usuários pudessem ter mais interação com os jogos, em pouco tempo

começou a ser utilizado em algoritmos de que envolviam visão computacional devido aos elementos que compõem o sensor. Além de retronar imagens de muito boa qualidade, o Kinect também retorna informações de profundidade. (REIS, 2019) utilizou, em seu trabalho, o Kinect para rechonecer objetos utilizando algoritmos de inteligência artificial e aplicou os dados na navegação do robô móvel.

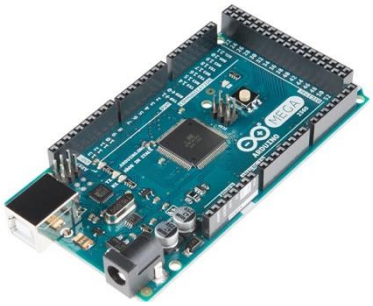
Figura 11 . Sensor Kinect

Fonte: Autor

3.2.4 Arduino

O elemento responsável por abrigar toda a programação, ler os dados dos sensores e controlar os demais dispositivos do robô é o microcontrolador Arduino Mega 2560 (Figura 12), conforme Figura 12. Este foi escolhido para suprir os trabalhos iniciais do robô devido aos seus 54 pinos de entrada e saída e, também seu clock de 16 MHz. O microcontrolador Arduino será encarregado de executar as tarefas de baixo nível como o controle dos drivers dos motores de corrente continua, as leituras dos encoders do driver MD49 ao microcontrolador, e a leitura dos sensores de ultrassom, entre outras tarefas de baixo nível.

Figura 12 . Arduino Mega 2560



Fonte: Autor

A programação é via o ambiente de programação IDE (*Integrated Development Environment*) característica do microcontrolador Arduino. A interface baseada de C++ e se caracteriza por ser muito intuitiva. Para programá-la é necessário interconectar o Arduino com

o desktop ou notebook via porta USB. O programa desenvolvido no robô permite ligar e desligar os motores, programar as velocidades de rotação dos motores, fazer as leituras dos encoders, executar trajetórias simples do robô, ler os sensores de ultrassom.

3.2.5 Regulador de tensão

Como o driver MD49 possui uma alimentação de entrada de 24V, é necessário utilizar duas baterias em série, totalizando 29,6V. Essa tensão poderá danificar o driver, então, para solucionar este problema, é utilizado um regulador de tensão comercial onde é possível ajustar e visualizar o valor da tensão de saída. O regulador utilizado é mostradao na figura 13.

Figura 13 . Regulador de tensão

Fonte: Autor

3.3 Equipamentos Elétricos

3.3.1 Baterias

Para alimentar o driver e, consequentemente, os motores são utilizadas duas baterias série. Devido ao robô já possuir uma das baterias recebida por doação e, também, pela seu tamanho e peso reduzidos, as baterias escolhidas para esta aplicação são as de LiPo (Polímero de Lítio) (figura 14) com 14,8V cada uma. Sua composição química possui sais de Lítio retidos em polímeros sólidos nos eletrólitos permitindo que tenha diferentes formatos e altas taxas de descarga. Estas materias são muito comuns em aplicaçãos com drones.

Figura 14 . Bateria de LiPo

Fonte: Autor

3.3.2 Motor C.C com encoder

O robô possui 2 motores de corrente contínua (CC), capazes de suportar toda a estrutura e executar trajetórias pré-programadas. Cada motor CC conta com uma caixa de redução na qual possui uma relação de 49:1, o que permite um torque de 16 kg/cm. A tensão de alimentação é 24V CC com uma corrente nominal de 2,1 A, 122 rpm e uma potência de 34,7W. Os seus respectivos encoders são do tipo incremental para leituras de distância e posição. Possuem resolução de 341.2 PPR, ou seja, para cada revolução em seu eixo, ele gera 341,2 pulsos elétricos. A Figura 15 mostra somente o encoder e a Figura 16 mostra o motor com encoder.

Figura 15 . Encoder

Fonte: Autor

Figura 16 . Motor CC com encoder acoplado

Fonte: Autor

3.3.3 Botão de emergência

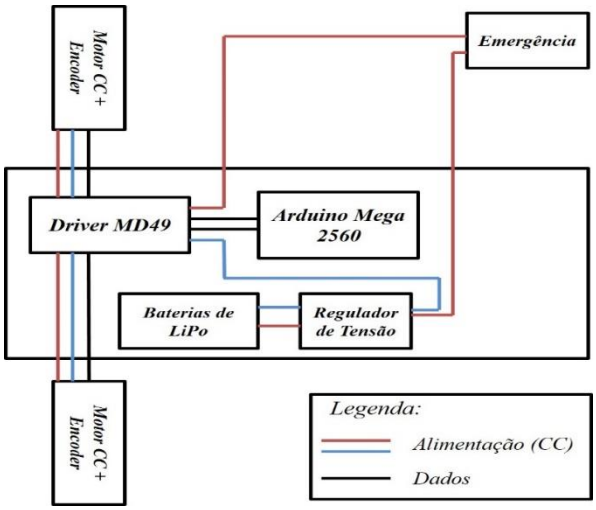
Pensando na segurança nas realizações do ensaios com o robô móvel, tanto instalaçoes do laboratório, do próprio robô e também do(s) usuário(s), foi projetado um botão de emergência para interromper a alimentação dos motores e, assim, parar o robô caso algo não ocorra conforme o planejado. O botão de emergência é mostrado na figura 17.

Figura 17 . Botão de emergência

Fonte: Autor

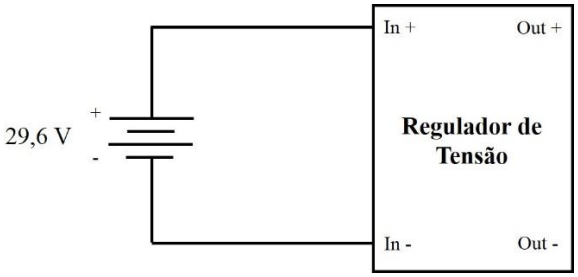
Para melhor ilustrar as conexões dos componentes entre si, a figura 18 mostra um diagrama simplificado de como os mesmos estão interconectados e a figura 19 ilustra as conexões da bateria com o regulador de tensão.

Figura 18 . Diagrama de conexões



Fonte: Autor

Figura 19 . Diagrama de conexões das baterias com regulador de tensão

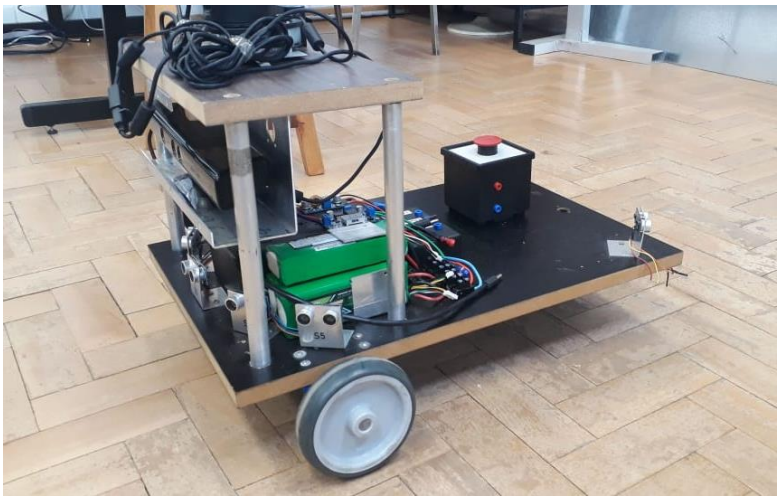


Fonte: Autor

O diagrama da figura 19 ilustra como as baterias estão conectadas com o regulador de tensão. As duas baterias estão em série e cada uma possui uma tensão de 14,8 V, totalizando 29,6 V, desta forma, é necessário ajustar a tensão para 24 V (tensão de operação do driver).

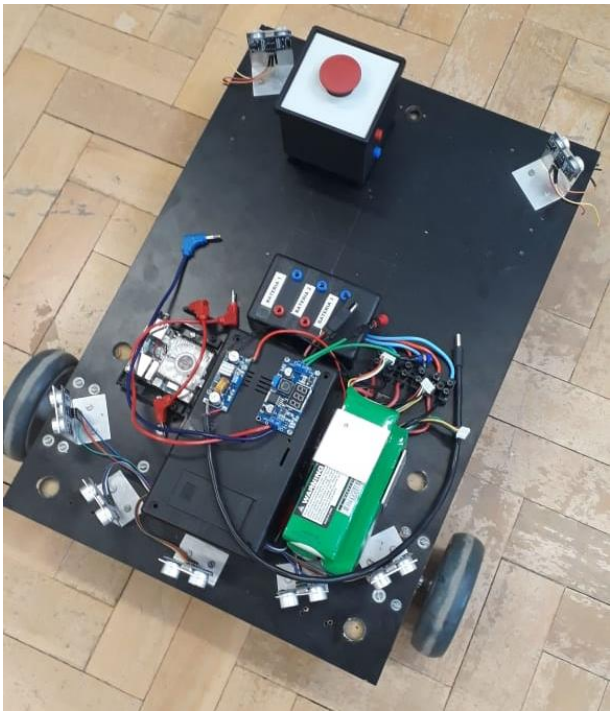
Por fim, a figura 20 e a figura 21 mostram o robô já montado.

Figura 20 . Robô móvel construído.



Fonte: Autor

Figura 21 . Robô móvel visto de cima sem o suporte do Kinect



Fonte: Autor

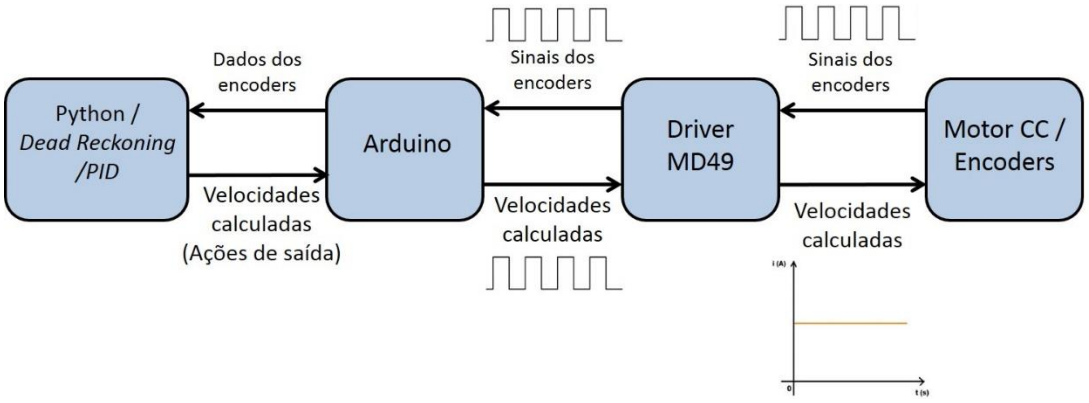
4. METODOLOGIA

A seção de metodologia irá descrever as estratégias adotadas para que os experimentos pudessem ser realizados. Consta os componentes utilizados e como os mesmos se comunicam e a forma como cada um é programado para realizar a sua respectiva função. Os códigos utilizados estão disponíveis nos anexos nas seções finais deste trabalho.

4.1 Controlador PID

Conforme ilustra a figura 22, a arquitetura do sistema para o controlador PID é mais simplificada devido ao fato de não precisar utilizar o módulo de controle *fuzzy* do software MatLab®. Isso se deve devido à implementação do controlador PID no próprio programa escrito em Python. As equações do controlador PID doram descritas na Seção 2.5 e o código completo se encontra no "Apêndice A". A relação entre os componentes, ou seja, a plataforma Arduino/Driver/Motor CC/ Encoder permanecem as mesmas conforme foi descrita na seção anterior. A ação de controle calculada é convertida para cada roda e envida via USB para o Arduino.

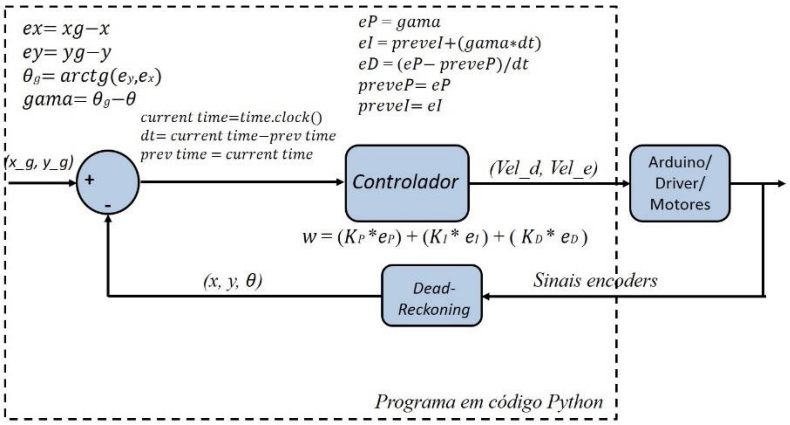
Figura 22 . Arquitetura do sistema para controlador PID



Fonte: Autor

A figura 23, ilustra mais detalhadamente o controlador contido no código escrito em Python juntamente com as equações descritas na seção anterior.

Figura 23 . Diagrama do controlador PID

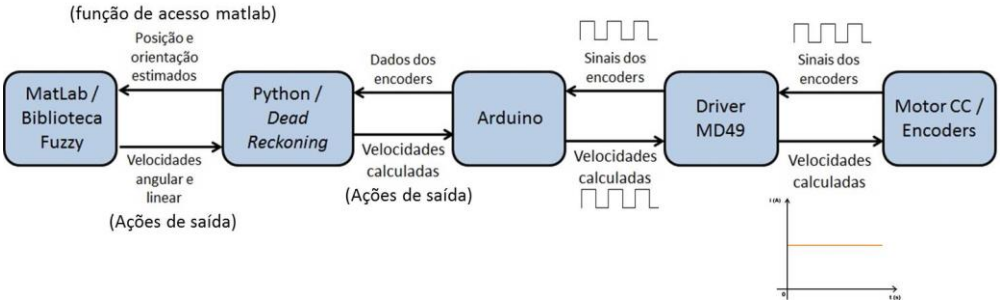


Fonte: Autor

4.2 Controlador Fuzzy

Um panorama geral da arquitetura do sistema proposto é apresentado na Figura 24. Devido a facilidade de implementação, depuração e visualização do controlador *fuzzy*, utilizou-se o programa MatLab® junto com o toolbox de lógica *fuzzy*. O MatLab® permite a exportação do controlador em um formato específico, sendo disponibilizado pela desenvolvedora uma biblioteca para Python que permite a leitura e manipulação desses arquivos. O sinal dos encoders, acoplados nas rodas, é capturado pelo *driver MD*-49, o qual envia essas informações para o microcontrolador Arduino. O mesmo envia essas informações através da Serial para o script em Python, o qual é responsável pela correlação de todo o sistema e realiza o cálculo de orientação e posição por meio do método *Dead-Reckoning*. Essa informação é enviada para o controlador *fuzzy*, obtendo a ação de controle, a qual é enviada via serial para o microcontrolador.

Figura 24 . Arquitetura do sistema



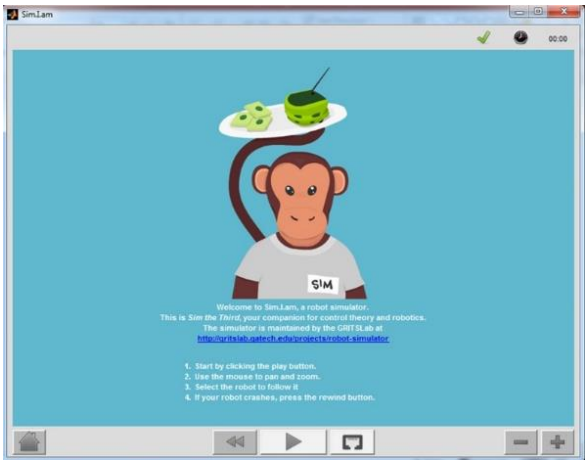
Fonte: Autor

4.3 Software de Simulação SimIam

Devido a grande complexidade da implementação do controle em hardware, primeiramente utilizou-se uma ferramenta de simulação para validar tanto os controladores *fuzzy* e PID quanto o método de estimação da posição e orientação. O ambiente de simulação utilizado denomina-se Sim.I.am, desenvolvido por Georgia Robotics and InTelligent Systems Laboratory, o qual consiste em um simulador de robôs móveis em ambiente Matlab®. O emprego do simulador facilita a implementação e o design de controladores e algoritmos para robótica móvel.

Nesse ambiente de simulação foi executado o método *Dead-Reckoning* para a obtenção da posição e orientação do robô junto com o controlador *fuzzy* e, após, com o controlador PID. A utilização do simulador é muito apropriada, pois além do controlador também ser implementando em Matlab®, a semelhança da sintaxe entre Python e Matlab® permite uma fácil migração tanto do código do método *Dead-Reckoning* quanto dos controladores. Como o simulador possui código seu aberto, a implementação dos controladores e do método é feita acrescentado ou alterando scripts. A figura 25 mostra a interface do simulador.

Figura 25 . Interface do simulador Sim.I.am



Fonte: Software Sim.I.am

4.4 Trajetória

O objetivo consiste no robô desenvolver uma trajetória dada pela curva algébrica denominada Lemniscata de Bernoulli. A curva pode ser descrita pela seguinte equação cartesiana:

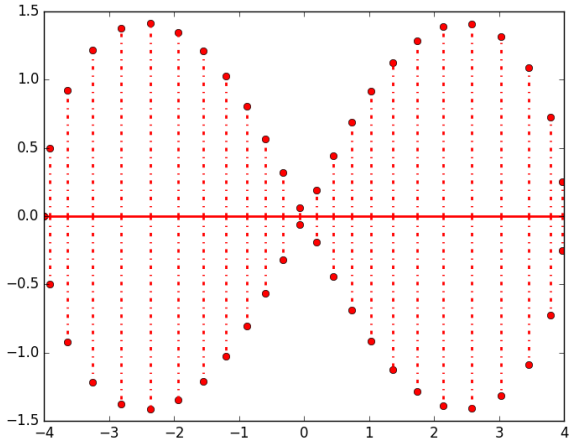
$$(x^2 + y^2)^2 = 2a^2(x^2 - y^2) \quad (22)$$

Parametrizando a curva para representar a equação em termos de um único parâmetro, chega-se a seguinte equação:

$$x = \frac{4\cos(t)}{(1+(\sin(t))^2)}\, y = \frac{4\cos(t)\sin(t)}{(1+(\sin(t))^2)} \quad (23)$$

Plotando a curva discreta para 50 pontos de x e y com t entre -π e +π obtém-se o seguinte gráfico. A Figura 26 mostra a curva da Lemniscata.

Figura 26 . Leminiscata Discreta



Fonte: Autor

Dessa forma são gerados os pontos desejados para robô percorrer a fim de descrever a trajetória da Leminiscata.

5. RESULTADOS

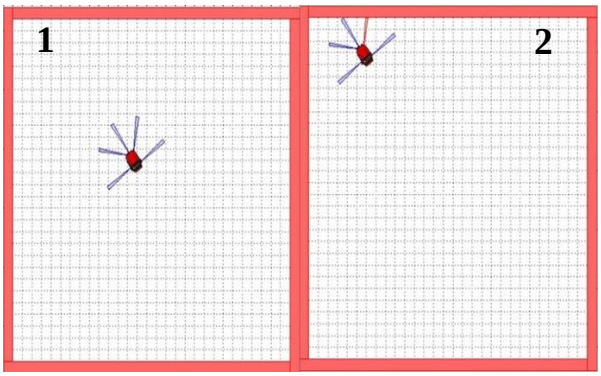
Este capítulo, pode-se agrupar em duas partes, a primeira apresenta resultados em simulação do robô utilizando os controladores Fuzzy e PID, tanto de ponto a ponto quanto descrevendo a trajetória Lemniscata. A segunda parte mostra os resultados experimentais utilizando os controladores Fuzzy e PID seguindo a mesma metodologia.

5.1 Resultados em Simulação

5.1.1 Controlador PID ponto a ponto

A figura 27 mostra a simulação do robô móvel utilizando controlador PID ponto a ponto onde a posição inicial é o ponto (0,0) e o ponto final é (-1, 1.3). A imagem é captura da animação realizada pelo simulador e, para uma melhor compreenção, foi numerada nos cantos superiores a sequência de movimento do robô móvel.

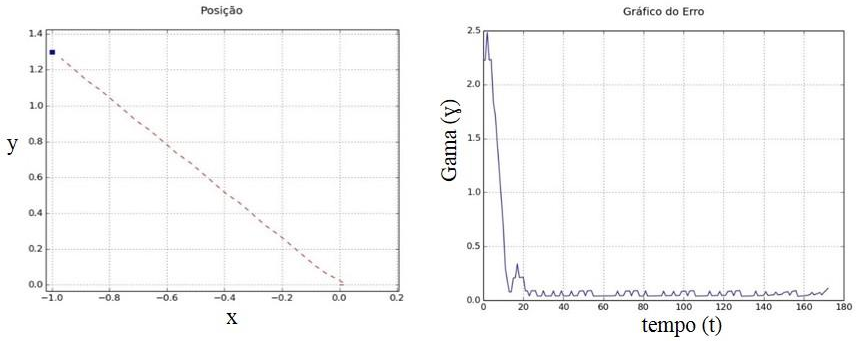
Figura 27 . Simulação da posição (0,0) para a posição (-1, 1.3)



Fonte: Autor

A figura 28 (a) mostra o gráfico da trajetória percorrida pelo robô e figura 28 (b) erro de orientação ao longo do tempo na simulação. No gráfico da posição, os eixos x e y representam a posição do robô. No gráfico do erro o eixo y representa o erro de orientação do robô e o eixo x representa o tempo decorrido.

Figura 28 . Dados de posição e erro da simulação



(a) (b)

Fonte: Autor

Tabela 4. Ganhos PID utilizados na simulação

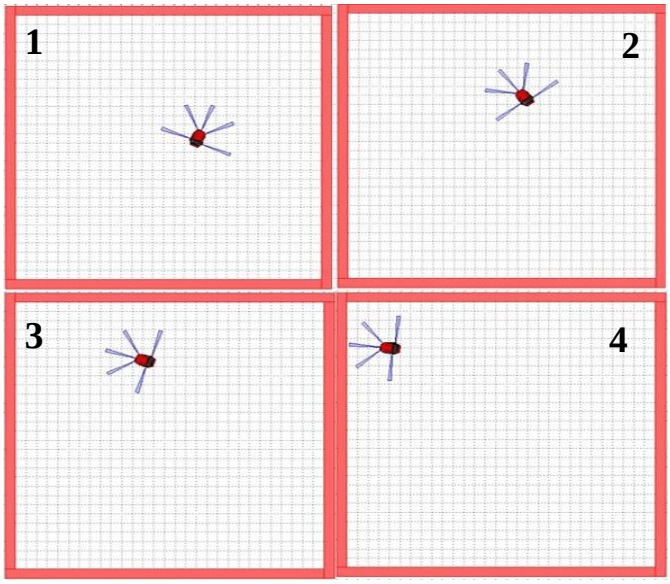
Constantes	Ganhos
K_P	2
K_I	0
K_D	0,5

Conforme mostra as figuras acima, o controle PID de ponto a ponto apresenta, em simulação, um erro muito pequeno quase que totalmente anulado e um pequeno. Essa simulação é mostrada em um vídeo que pode ser visualizado na íntegra acessando o link: https://www.youtube.com/watch?v=wHaNd8IXm8w . A tabela 4 apresenta os ganhos utilizados na simulação.

5.1.2 Controlador Fuzzy ponto a ponto

A figura 29 mostra a simulação do robô móvel utilizando controlador *Fuzzy* ponto a ponto onde a posição inicial é o ponto (0,0) e o ponto final é (-1.2, 1). A imagem é composta por algumas capturas realizada pelo simulador e, para uma melhor compreenção, foi numerada nos cantos superiores a sequência de movimento do robô móvel.

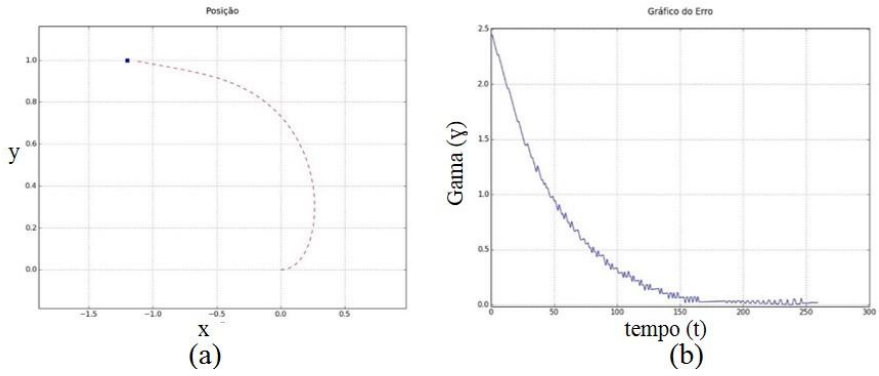
Figura 29 . Simulação da posição (0,0) para posição (-1.2, 1)



Fonte: Autor

A figura 30 (a) mostra o gráfico da trajetória percorrida pelo robô e a figura 30 (b) o erro de orientação ao longo do tempo na simulação. No gráfico da posição, os eixos x e y representam o deslocamento do robô. No gráfico da figura 30 (b) o eixo y representa o erro de orientação do robô e o eixo x representa o tempo decorrido.

Figura 30 . Dados de posição e erro da simulação



Fonte: Autor

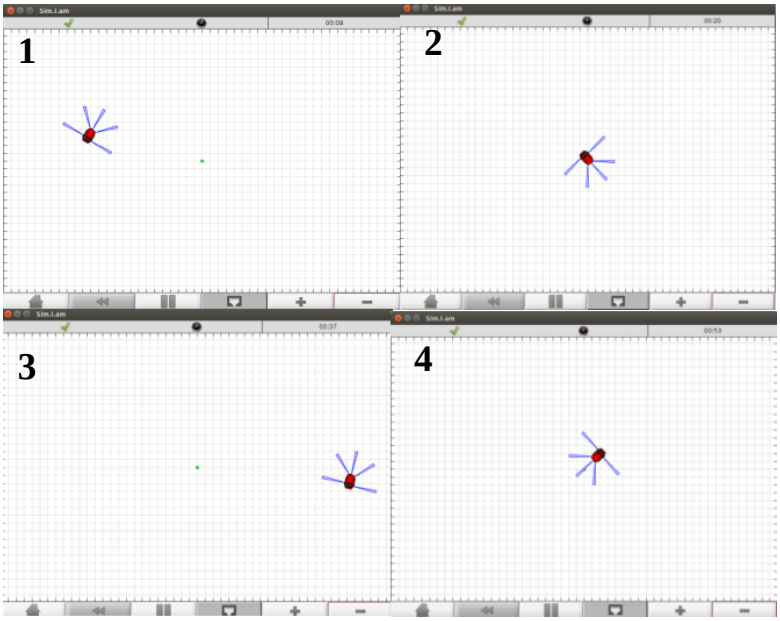
Neste teste, é possível evidenciar um tempo maior para anular o erro de diferença entre os ângulos de setpoint e o ângulo estimado. Isto é devido a diferença de erro ser maior no início do teste em relação ao teste anterior, ou seja, possui uma ação de controle mais lenta. O vídeo desta simulação pode ser acessado através do seguinte link: https://www.youtube.com/watch?v=Mf3Hku8obMQ

5.1.3 Controlador Fuzzy Lemniscata

A figura 31 mostra a simulação do robô móvel utilizando controlador *Fuzzy* percorrendo uma trajetória pré-definida. A trajetória adotada é a Lemniscata. A imagem é composta por

algumas capturas realizada pelo simulador e, para uma melhor compreenção, foi numerada nos cantos superiores a sequência de movimento do robô móvel.

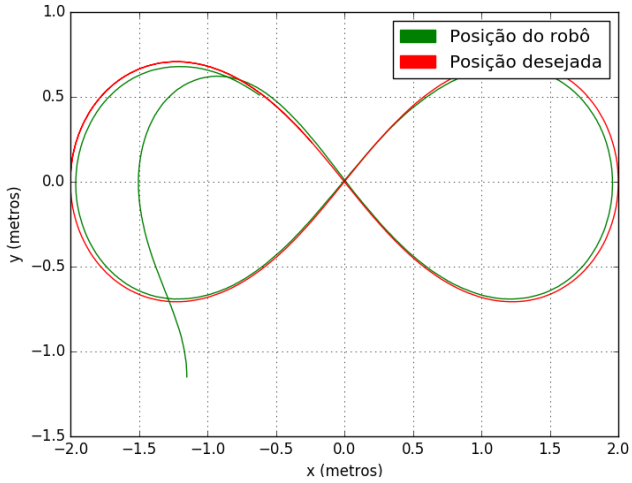
Figura 31 . Robô simulando trajetória no Simian



Fonte: Autor

A figura 32 mostra o gráfico da trajetória percorrida pelo robô juntamente com o a trajetória de referência. No gráfico, os eixos x e y representam o deslocamento do robô.

Figura 32 . Movimento nos x e y em simulação



Fonte: Autor

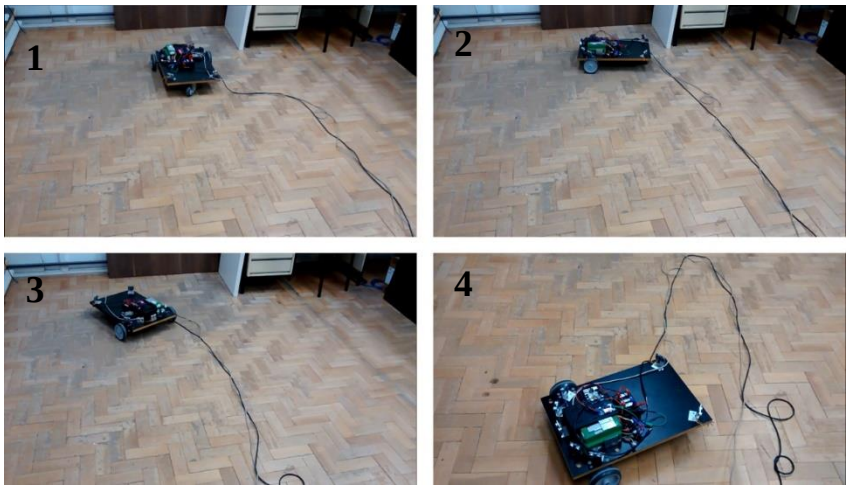
Na simulação da Lemniscata, é possível perceber que o controlador é satisfatório apesar de apresenta um pequeno erro. É importante salientar que a própria estimação da posição acumula erro, desta forma o erro apresentado não recai somente sobre o controlador.

5.2 Resultados Experimentais

5.2.1 Controlador PID ponto a ponto

A figura 33 mostra o teste experimental do robô móvel utilizando controlador PID ponto a ponto onde a posição inicial é o ponto (0,0) e o ponto final é (-1.3, 1.4). A imagem é composta por algumas capturas do vídeo realizado e, para uma melhor compreenção, foi numerada nos cantos superiores a sequência de movimento do robô móvel.

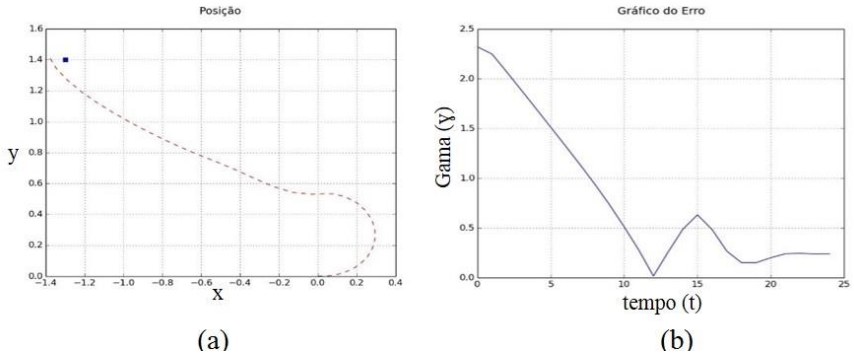
Figura 33 . Teste experimental da posição (0,0) para a posição (-1.3, 1.4)



Fonte: Autor

A figura 34 (a) mostra o gráfico da trajetória percorrida pelo robô e a figura 34 (b) o erro de orientação ao longo do tempo no experimento. No gráfico da posição, os eixos x e y representam o deslocamento do robô. Na figura 34 (b) o eixo y representa a o erro de orientação do robô e o erro ponto desejado e o eixo x representa o tempo decorrido.

Figura 34 . Dados de posição e erro do teste experimental



Fonte: Autor

É possível notar um erro levemente maior no experimento prático do que na simulação. Ambientes de simulação dificilmente conseguem transmitir as imperfeições mecânicas dos

solos e erros nas medições. Desta forma, como já é esperado, haverá um erro um maior que nas simulações. Com um erro inicial maior, nota-se que há um tempo maior para ação de controle diminuir o erro em relação a simulação, que já se esperava devido ao erro acumulado pelo método, pequenos deslizamentos das rodas etc. No link a seguir, o ensaio poderá ser visualizado na íntegra. https://www.youtube.com/watch?v=osFtqld7x9c . A tabela 5 apresenta os ganhos utilizados nos testes experimentais.

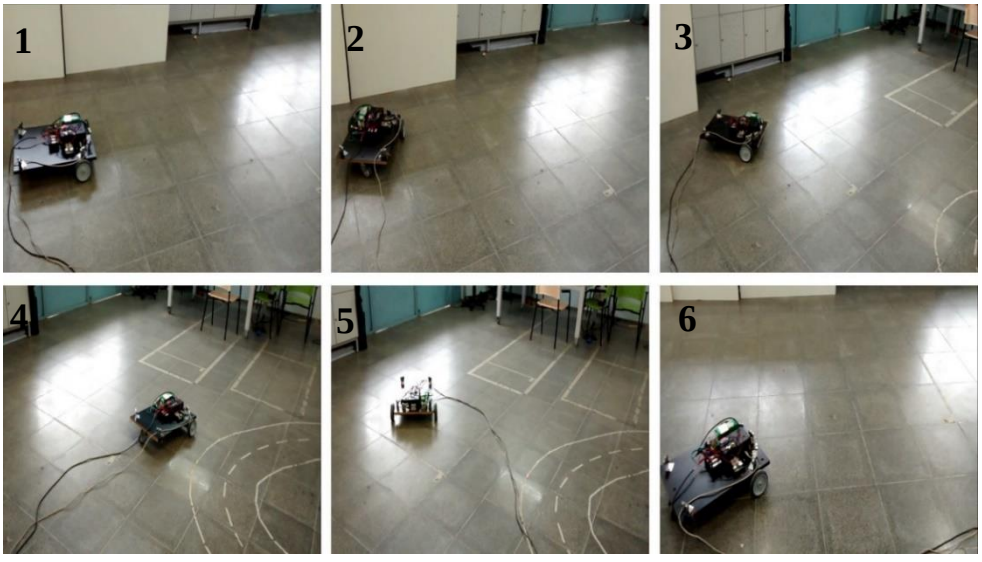
Tabela 5. Ganhos PID utilizados nos testes experiemtais

Constantes	Ganhos
K_P	20
K_I	1,5
K_D	0,5

5.2.2 Controlador PID Lemniscata

A figura 35 mostra o teste experimental do robô móvel utilizando controlador PID percorrendo uma trajetória pré-definida. A trajetória adotada é a Lemniscata. A imagem é composta por algumas capturas do vídeo realizado e, para uma melhor compreenção, foi numerada nos cantos superiores a sequência de movimento do robô móvel.

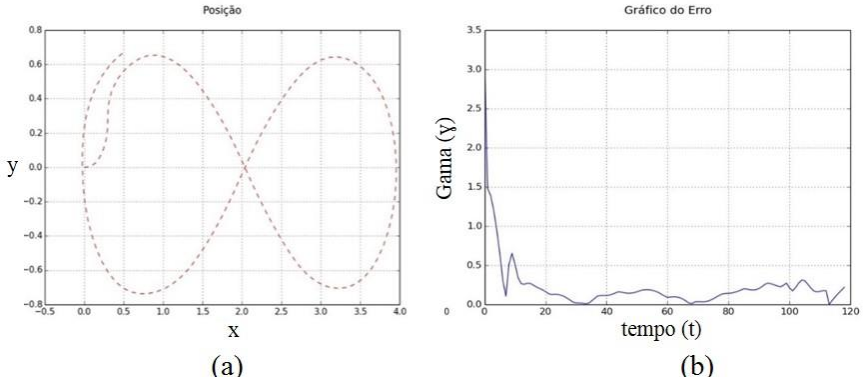
Figura 35 . Teste experimental descrevendo a Lemniscata



Fonte: Autor

A figura 36 (a) mostra o gráfico da trajetória percorrida pelo robô e a figura 36 (b) o erro de orientação ao longo do tempo no experimento. No gráfico da posição, os eixos x e y representam o deslocamento do robô. Na figura 36 (b) o eixo y representa o erro de orientação do robô e o eixo x representa o tempo decorrido.

Figura 36 . Dados de posição e erro do teste experimental



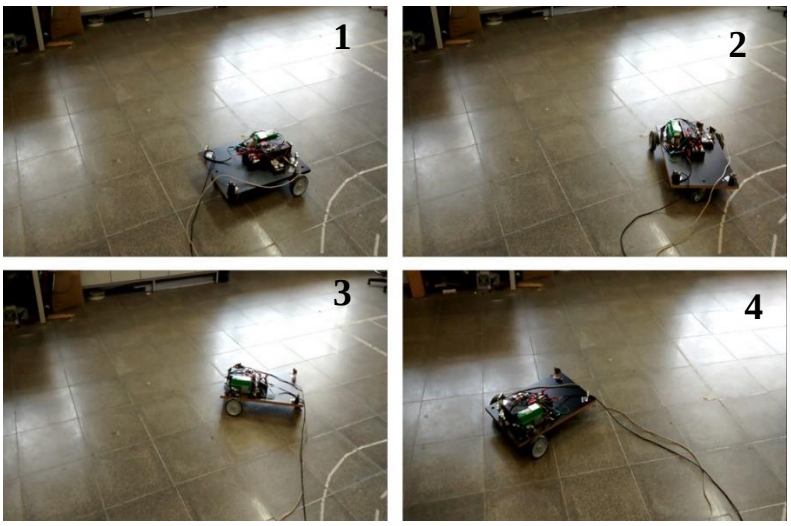
Fonte: Autor

Pode-se notar que robô descreveu com êxito a trajetória pré-estabelecida ainda que com um pequeno erro a ação de controle foi considerada satisfatória. Devido as dimensões da trajetória, o erro pôde ser desconsiderado. Para circuitos maiores, o método de estimação não é indicado.

5.2.3 Controlador Fuzzy ponto a ponto

A figura 37 é composta por algumas capturas realizada pelo vídeo realizado e, para uma melhor compreenção, foi numerada nos cantos superiores a sequência de movimento do robô móvel.

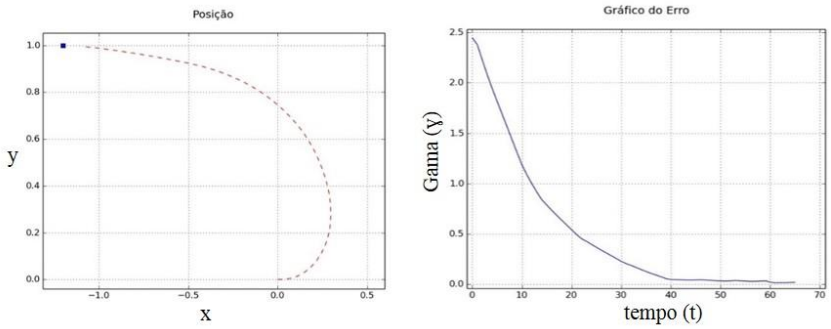
Figura 37 . Teste experimental da posição (0,0) para a posição (-1.2, 1)



Fonte: Autor

A figura 38 (a) mostra o gráfico da trajetória percorrida pelo robô e a figura 38 (b) o erro de orientação ao longo do tempo no experimento. No gráfico da posição, os eixos x e y representam o deslocamento do robô. Na figura 38 (b) o eixo y representa a o erro de orientação do robô e o erro ponto desejado e o eixo x representa o tempo decorrido.

Figura 38 . Dados de posição e erro do teste experimental



(a) (b)

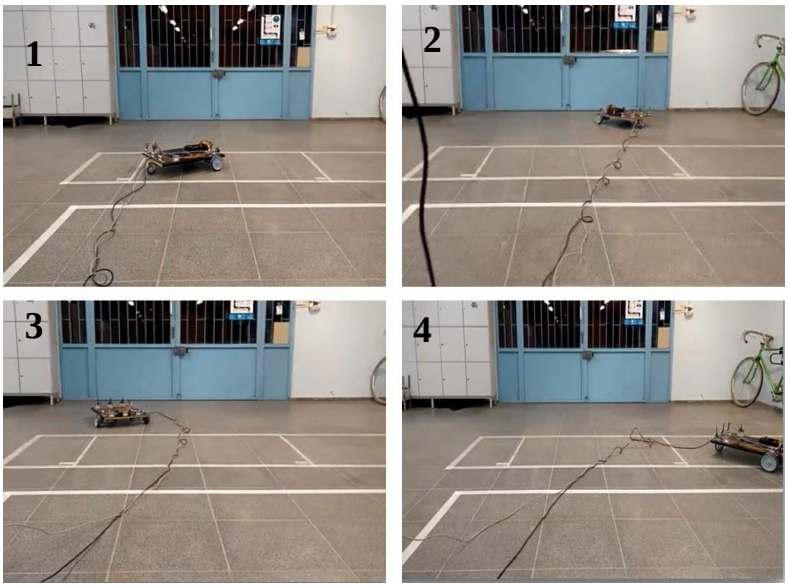
Fonte: Autor

Como já esperado, devido ao grande erro inicial e ação de controle mais lenta que o controlador PID, nota-se que, apesar do robô atingir o objetivo, um overshoot maior que na simulação, aumentando o tempo de para diminuição do erro. A simulação deste experimento pode ser visualizada no link: https://www.youtube.com/watch?v=66b8W4-KoZ8

5.2.4 Controlador Fuzzy Lemniscata

A figura 39 mostra o teste experimental do robô móvel utilizando controlador *Fuzzy* percorrendo uma trajetória pré-definida. A trajetória adotada é a Lemniscata. A imagem é composta por algumas capturas do vídeo realizado e, para uma melhor compreenção, foi numerada nos cantos superiores a sequência de movimento do robô móvel.

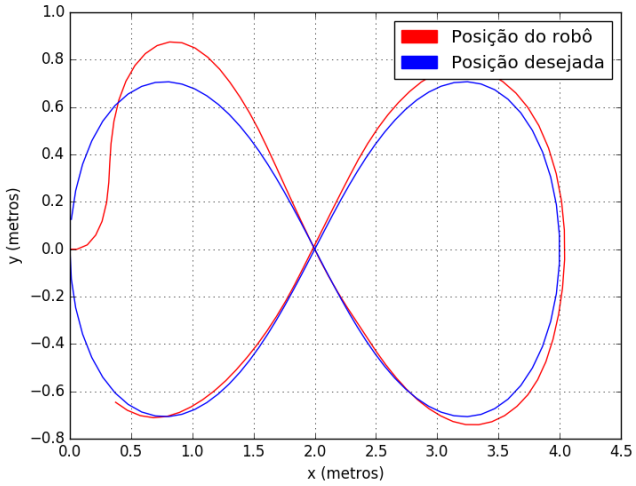
Figura 39 . Teste experimental descrevendo a Lemniscata



Fonte: Autor

A figura 40 mostra o gráfico da trajetória percorrida pelo robô e sua trajetória de referência. No gráfico da posição, os eixos x e y representam o deslocamento do robô.

Figura 40 . Movimento nos eixos x e y



Fonte: Autor

Mesmo com uma ação de controle mais lenta, é possível notar que o mesmo segue a referência e que o erro maior se encontra no inicio do percurso, ou seja, onde há maior diferença de erro dos ângulos de setpoint e estimado. É possível notar também erros onde há pontos de inflexão da função da trajetória. Desta forma pode ser considerado satisfatório o controlador fuzzy. O vídeo do teste experimental pode ser visualizado na íntegra através o link a seguir: https://www.youtube.com/watch?v=d14RfijCwCk

6. CONCLUSÕES

Neste trabalho foi apresentada a construção de um robô móvel e um sistema de localização de robôs móveis utilizando o método de *Dead-Reckoning* para estimar a posição e orientação de um robô construído inteiramente no laboratório do grupo GARRA bem como dois controladores, um *fuzzy* e outro PID, para controlar trajetórias pré-estabelecidas no trabalho.

Com os controladores elaborados, atingiu-se o objetivo de que o robô possa seguir uma trajetória de um ponto a outro em simulação e em testes experimentais, descrevendo uma trajetória da Leminiscata. A utilização da lógica *fuzzy* é conveniente, uma vez que direciona o foco para a resolução do problema, facilitando e otimizando a implementação do controlador, pois dispensa a necessidade de custosas modelagens matemáticas. O controlador PID teve um rendimento excelente e com erro pequeno. O controlador PID teve grande vantagem sobre o controlador *fuzzy* e gerou um menor esforço computacional. Levando em conta o melhor desempenho do controlador PID, este poderia ser a melhor opção numa aplicação que envolvesse desvio de obstáculos onde é necessário um controlador com ação de controle rápida. É importante salientar que a técnica de estimação de posição e orientação utilizada para este trabalho é indicada para trajetórias pequenas, como foi o caso, devido ao erro intríseco acumulado da odometria.

Com a construção do robô móvel, será possível o desenvolvimento de outros trabalhos como, por exemplo, aplicação de novos controladores, introdução de novos sensores, implementação de algoritmos para desvio de obstáculos, localização e mapeamento. Desta forma será muito útil para fácil de visualizar o desempenho de controladores. Outro ganho é a sua utilização em aulas práticas para o ensino, facilitando uma melhor avaliação discente nas disciplinas.

O robô foi utilizado nas disciplinas de Projeto Integrador, TCC, em aulas práticas e artigos foram aprovados nos eventos: Congresso Brasileiro de Ensino em Engenharia (COBENGE), Congresso Regional de Iniciação Científica e Tecnológica em Engenharia (CRICTE), Jornada Acadêmica Integrada (JAI), Workshop-Escola de Informática Teórica (WEIT) e Congresso Brasileiro de Sistemas Fuzzy (CBSF).

APÊNDICE A – Código do calculo da odometria e controle Fuzzy “odometria.py”

```
import time
import serial
from numpy import *
import matplotlib.pyplot as plt
import urllib2
import base64
import matplotlib.patches as mpatches
from math import *

ser = serial.Serial('/dev/ttyACM0',9600,timeout=1)
#ser = serial.Serial('/dev/ttyACM1',9600,timeout=1)
time.sleep(1.8)
global l
l = 0
x_g = []
y_g = []
locx = []
locy = []
ticks_direita_anterior = 0.0
ticks_esquerda_anterior = 0.0
theta = 0.0
x = 0
y = 0
encod1 = 0.0
encod2 = 0.0
d_esquerda_total = 0.0
d_direita_total = 0.0

eP = 0
eI = 0
eD = 0
prev_eP = 0
prev_eI = 0
prev_time = 0.0
dt = 0.0

i = linspace(-pi, pi, 100)



for k in range(100):
    xg = 2*cos(i[k]) / (1 + (sin(i[k])) ** 2) + 2
```

```
    yg = 2*cos(i[k]) * sin(i[k]) / (1 + (sin(i[k])) ** 2)
    x_g.append(xg)
    y_g.append(yg)
def odometria():
    global ticks_esquerda_anterior
    global ticks_direita_anterior
    global x
    global y
    global theta
    global d_esquerda_total
    global locx
    global locy
    global d_direita_total

    serialread()
    # Implementacao odometria
    R = 0.0625
    L = 0.5

    # raio das rodas (diametro)
    N = 980.0
    # ticks por revolucao
    B = 0.53
    # comprimento do eixo

    ticks_direita = encod2
    ticks_esquerda = encod1

    d_esquerda = ((2*pi*R)*(ticks_esquerda - ticks_esquerda_anterior))/N
    d_direita = ((2*pi*R)*(ticks_direita - ticks_direita_anterior))/N

    d_centro = (d_esquerda + d_direita)/2
    phi = (d_direita - d_esquerda)/B
    theta_dt = phi
    x_dt = d_centro*cos(theta)
    y_dt = d_centro*sin(theta)
    x_novo = x + x_dt
    y_novo = y + y_dt
    theta_novo = theta + theta_dt

    x = x_novo
    y = y_novo
    theta = theta_novo
```

```
        ticks_direita_anterior = ticks_direita
        ticks_esquerda_anterior = ticks_esquerda
        d_esquerda_total +=  d_esquerda
        d_direita_total += d_direita

    def serialread():
        global encod1
        global encod2

        PARAM_ASCII = str(chr(105))
        ser.write(PARAM_ASCII)
        time.sleep(0.1)

        enc1 = ser.readline()
        enc1 = list(enc1)
        for i in range(len(enc1)):
            if (enc1[i] == '\r'):
                if i == 1:
                    encod1 = ((int(enc1[0])))
                if i == 2:
                    encod1 =((int(enc1[0]))*10)+((int(enc1[1])))
                if i == 3:
                    encod1 = ((int(enc1[0]))*10**2)+((int(enc1[1]))*10)+((int(enc1[2])))
                if i == 4:
                    encod1 =((int(enc1[0]))*10**3)+((int(enc1[1]))*10**2)+((int(enc1[2]))*10**1)+((int(enc1[3])))
                if i == 5:
                    encod1 =
((int(enc1[0]))*10**4)+((int(enc1[1]))*10**3)+((int(enc1[2]))*10**2)+((int(enc1[3]))*10**1)+((int(enc1[4])))

        time.sleep(0.1)
        PARAM_ASCII = str(chr(106))
        ser.write(PARAM_ASCII)
        time.sleep(0.1)
        #0.5, 0.5, 0.2
        enc2 = ser.readline()
        enc2 = list(enc2)
        for i in range(len(enc2)):
            if (enc2[i] == '\r'):
                if i == 1:
                    encod2 = ((int(enc2[0])))
                if i == 2:
                    encod2 =((int(enc2[0]))*10)+((int(enc2[1])))
                if i == 3:
```

```
            encod2 = ((int(enc2[0]))*10**2)+((int(enc2[1]))*10)+((int(enc2[2])))
        if i == 4:
            encod2 =((int(enc2[0]))*10**3)+((int(enc2[1]))*10**2)+((int(enc2[2]))*10**1)+((int(enc2[3])))
        if i == 5:
            encod2 =
((int(enc2[0]))*10**4)+((int(enc2[1]))*10**3)+((int(enc2[2]))*10**2)+((int(enc2[3]))*10**1)+((int(enc2[4])))

    while(1):

        if (l >= 100):
            l = 0
        else:
            l = l + 1

        red_patch = mpatches.Patch(color='red', label=u'desire postion')
        blue_patch = mpatches.Patch(color='blue', label=u'robot position (dead reckoning)')
        plt.legend(handles=[red_patch, blue_patch])
        plt.scatter(x, y,c ='blue') #alterando para a leminiscata
        plt.scatter(x_g[l-1], y_g[l-1] , c ='red')
        #plt.pause(0.05)
        plt.pause(0.05)
        odometria()

        ex = x_g[l-1] - x
        ey = y_g[l-1] - y
        #ex = x_g - x #alterando para a leminiscata
        #ey = y_g - y #alterando para a leminiscata

        teta_g = atan2(ey,ex)
        gama = (teta_g-theta)
        gama = atan2(sin(gama) , cos(gama))

        gama = float(gama)
        ek = gama

        distancia = ((x - x_g[l-1] )**(2) + (y - y_g[l-1])**(2))**(0.5)
        distancia = float(distancia)

        # ------------------ Controle PID -------------------------------------------
        if (distancia > 0.1):

            #kP = 25
            #kP = 20
```

```
kP = 20
kI = 1.5
kD = 0.5

current_time = time.clock()

dt = current_time - prev_time
eP = gama
eI = prev_eI + (gama*dt)
eD = (eP - prev_eP)/dt

prev_time = current_time
prev_eP = eP
prev_eI = eI

w = (kP * eP) + (kI * eI) + (kD * eD)
v = 8 / ( abs( w ) + 1 )**0.5

if (w) > 8.0 :
   w = sign(w) * 8.0
#if (v) > 5.0 :
   #v = sign(v) * 5.0
if (v) > 3.1 :
   v = sign(v) * 3.1
if v < 0.0:
   v = 0

R = 0.0625
L = 0.5

vel_r = ((2 * v + w * L) / (2 * R))
vel_r = (0.5*vel_r) + 128

vel_l = ((2 * v - w * L) / (2 * R))
vel_l = (0.5*vel_l) + 128

# direita
PARAM_ASCII = str(chr(100))
ser.write(PARAM_ASCII)
vel_r = str(vel_r)
ser.write(vel_r)

# esquerda
```

```
        PARAM_ASCII = str(chr(101))
        ser.write(PARAM_ASCII)
        vel_l = str(vel_l)
        ser.write(vel_l)

    else:
        # Comando para parar
        # direita
        PARAM_ASCII = str(chr(102))
        ser.write(PARAM_ASCII)

        # esquerda
        PARAM_ASCII = str(chr(102))
        ser.write(PARAM_ASCII)

    #ftemp = open("t", "a")
    ft = open("theta", "a")
    fx = open("cordx","a")
    fy = open("cordy","a")
    fx_g = open("cordx_g","a")
    fy_g = open("cordy_g","a")
    fek = open("ek","a")

    a = str(x)
    fx.write(a)
    fx.write(" ")

    b = str(y)
    fy.write(b)
    fy.write(" ")
    c = str(theta)
    ft.write(c)
    ft.write(" ")
##    d = str(t)
##    ftemp.write(d)
##    ftemp.write(" ")

    i = str(ek)
    if i != 0.0:
      fek.write(i)
      fek.write(" ")
    h = str(x_g)
    fx_g.write(h)
```

```
    fx_g.write(" ")
    f = str(y_g)
    fy_g.write(f)
    fy_g.write(" ")



ser.close()
```

APÊNDICE B – Código para acionamento das rodas “odometria.ino”

```
#define CMD        (byte)0x00          // MD49 command address of 0
#define GET_SPEED1       0x21
#define GET_ENC1         0x23
#define GET_ENC2         0X24
#define SET_SPEED1       0x31
#define SET_SPEED2       0x32
#define ENC_RESET        0x35
uint32_t encoder = 0;
byte enc1a, enc1b, enc1c, enc1d = 0;
byte speed1a = 0;
byte speed1b = 0;
String stringOne = "Hello String";
// reads the value of encoder 1 into an unsigned 32 bit int
int a, b, c, d;
unsigned int integerValue = 0; // Max value is 65535
char incomingByte;
int sensors[8] = {0};
static char outstr[15];
char inChar;

float encoder1 = 0;
float encoder2 = 0;
int vel_e = 128;
int vel_d = 128;


void setup()
{
  Serial.begin(9600);
  Serial1.begin(9600);
}

void loop()
{
 if(Serial.available() > 0)
     {
       int roda = Serial.read();
       if (roda == 102){
        Serial1.write(CMD);
        Serial1.write(SET_SPEED1);
        Serial1.write(128);
```

```
 Serial1.write(CMD);
 Serial1.write(SET_SPEED2);
 Serial1.write(128);

}
if(roda == 101) // roda esquerda
{
 vel_e =  Serial.parseInt();
 Serial1.write(CMD);
 Serial1.write(SET_SPEED1);
 Serial1.write(vel_e);
}
if(roda == 100) // roda direita
{
 vel_d = Serial.parseInt();
 Serial1.write(CMD);
 Serial1.write(SET_SPEED2);
 Serial1.write(vel_d);

}
if(roda == 105)
{
  Serial1.write(CMD);
  Serial1.write(GET_ENC1);                  // Recieve encoder 1 value
  delay(50);
 if (Serial1.available() > 3)
 {
   enc1a = Serial1.read();
   enc1b = Serial1.read();
   enc1c = Serial1.read();
   enc1d = Serial1.read();
  }
   int count = sizeof(enc1a);
   char* chars;
   encoder = (((uint32_t)enc1a << 24) +
  ((uint32_t)enc1b << 16) +
  ((uint32_t)enc1c << 8) +
  ((uint32_t)enc1d << 0));
   Serial.println(encoder,DEC);
  delay(300);
  }

if(roda==106)
```

```
    {

     Serial1.write(CMD);
     Serial1.write(GET_ENC2);                // Recieve encoder 1 value
     delay(50);
    if (Serial1.available() > 3)
    {
      enc1a = Serial1.read();
      enc1b = Serial1.read();
      enc1c = Serial1.read();
      enc1d = Serial1.read();
     }
      int count = sizeof(enc1a);
      char* chars;
      encoder = (((uint32_t)enc1a << 24) +
     ((uint32_t)enc1b << 16) +
     ((uint32_t)enc1c << 8) +
     ((uint32_t)enc1d << 0));
      Serial.println(encoder,DEC);
     delay(300);
    }
  }
}
```

APÊNDICE C – Código da simulação PID “go_to_goalPID.m”

```
classdef GoToGoal < simiam.controller.Controller
%% GOTOGOAL steers the robot towards a goal with a constant velocity using PID
% Copyright (C) 2013, Georgia Tech Research Corporation
% see the LICENSE file included with this software

  properties
    %% PROPERTIES

    % memory banks
    E_k
    e_k_1

    % gains
    Kp
    Ki
    Kd

    % plot support
    p
  end

  properties (Constant)
    % I/O
    inputs = struct('x_g', 0, 'y_g', 0, 'v', 0);
    outputs = struct('v', 0, 'w', 0);
  end

  methods
  %% METHODS

    function obj = GoToGoal()
      %% GOTOGOAL Constructor
      obj = obj@simiam.controller.Controller('go_to_goal');

      % initialize memory banks
      obj.Kp = 2;
      obj.Ki = 0;
      obj.Kd = -10;

      % errors
```

```
    obj.E_k = 0;
    obj.e_k_1 = 0;

    % plot support
    obj.p = [];
end

function outputs = execute(obj, robot, state_estimate, inputs, dt)
%% EXECUTE Computes the left and right wheel speeds for go-to-goal.

    % Retrieve the (relative) goal location

    x_g = -1;
    y_g = 1.3;

    % Get estimate of current pose
    [x, y, theta] = state_estimate.unpack();

    % Compute the v,w that will get you to the goal
    v = inputs.v;

    %% START CODE BLOCK %%
    %salvando arquivos em txt para gerar grafico
    dadosx = fopen('cordx.txt','a');
    fprintf(dadosx,'%f ',x);
    fclose(dadosx);
    dadosy = fopen('cordy.txt','a');
    fprintf(dadosy,'%f ',y);
    fclose(dadosy);

    dadosx_g = fopen('cordx_g.txt','a');
    fprintf(dadosx_g,'%f ',x_g);
    fclose(dadosx_g);
    dadosy_g = fopen('cordy_g.txt','a');
    fprintf(dadosy_g,'%f ',y_g);
    fclose(dadosy_g);

    % 1. Calculate the heading (angle) to the goal.

    % distance between goal and robot in x-direction
    u_x = x_g - x;

    % distance between goal and robot in y-direction
```

```
u_y = y_g - y;

% angle from robot to goal. Hint: use ATAN2, u_x, u_y here.
theta_g = atan2(u_y, u_x);

% 2. Calculate the heading error.

% error between the goal angle and robot's angle
% Hint: Use ATAN2 to make sure this stays in [-pi,pi].
alpha = theta_g - theta;
e_k = atan2(sin(alpha),cos(alpha));
dadoserro = fopen('ek.txt','a');
fprintf(dadoserro,'%f ',e_k);
fclose(dadoserro);

% 3. Calculate PID for the steering angle

% error for the proportional term
e_P = e_k;



% error for the integral term. Hint: Approximate the integral using
% the accumulated error, obj.E_k, and the error for
% this time step, e_k.
e_I = e_k + obj.E_k;

% error for the derivative term. Hint: Approximate the derivative
% using the previous error, obj.e_k_1, and the
% error for this time step, e_k.
e_D = e_k - obj.e_k_1;

%% END CODE BLOCK %%

w = obj.Kp*e_P + obj.Ki*e_I + obj.Kd*e_D;

% 4. Save errors for next time step
obj.E_k = e_I;
obj.e_k_1 = e_k;

% plot
%obj.p.plot_2d_ref(dt, atan2(sin(theta),cos(theta)), theta_g, 'r');

outputs = obj.outputs;  % make a copy of the output struct
```

```
            outputs.v = v;
            outputs.w = w;
        end

    end

end
```

APÊNDICE D – Código da simulação Fuzzy “go_to_goalfuzzy.m”

```
classdef GoToGoal < simiam.controller.Controller
%% GOTOGOAL steers the robot towards a goal with a constant velocity using PID
%
% Properties:
%   none
%
% Methods:
%   execute - Computes the left and right wheel speeds for go-to-goal.

% Copyright (C) 2013, Georgia Tech Research Corporation
% see the LICENSE file included with this software

  properties
    %% PROPERTIES

    % memory banks
    E_k
    e_k_1

    % gains
    Kp
    Ki
    Kd
    i
   t
    % plot support
    p
    grafx
    grafy

  end

  properties (Constant)
    % I/O
    inputs = struct('x_g', 0, 'y_g', 0, 'v', 0);
    outputs = struct('v', 0, 'w', 0);
  end

  methods
  %% METHODS
```

```
function obj = GoToGoal()
   %% GOTOGOAL Constructor
   obj = obj@simiam.controller.Controller('go_to_goal');

   % initialize memory banks
   obj.Kp = 5;
   obj.Ki = 1;
   obj.Kd = 0;
   obj.i = 0;
   obj.t = linspace(-pi,pi,600);
   % errors
   obj.E_k = 0;
   obj.e_k_1 = 0;

   % plot support
   obj.p = [];
end

function outputs = execute(obj, robot, state_estimate, inputs, dt)
%% EXECUTE Computes the left and right wheel speeds for go-to-goal.
%   [v, w] = execute(obj, robot, x_g, y_g, v) will compute the
%   necessary linear and angular speeds that will steer the robot
%   to the goal location (x_g, y_g) with a constant linear velocity
%   of v.
%
%   See also controller/execute

   %fis2 = readfis('/home/rodrigo/Ã?rea de Trabalho/Controle Fuzzy real/final.fis');
   %fis2 = readfis('C:/Users/Thiago/Desktop/fwdcodigos/final.fis');
   fis2 = readfis('E:/codigos python/fwdcodigos/Controle Fuzzy real(1)/Controle Fuzzy real/final.fis');
   % Retrieve the (relative) goal location
    %x_g = inputs.x_g;
    %y_g = inputs.y_g;

   %x_g = 4*cos(dt+0.8)/(1+(sin(dt+0.8))^2);
   %y_g = 4*cos(dt+0.8)*sin(dt+0.8)/(1+(sin(dt+0.8))^2);

   %x_g = (2*cos(dt+0.8)*(2*cos(2*dt+0.8))^0.5)
   %y_g = (2*sin(dt+0.8)*(2*cos(2*dt+0.8))^0.5)


   %fprintf('ti: (%0.3f)\n', obj.i);
```

```
        %obj.i = obj.i + 1

%       if obj.i > 599
%               obj.i = 1;
%       else
%               obj.i = obj.i + 1
%       end


        %x_g = 1*cos(obj.t(obj.i))
        %y_g = 1*sin(obj.t(obj.i))
      %x_g = 4*cos(obj.t(obj.i))/(1+(sin(obj.t(obj.i)))^2); % para
      %testar ponto a ponto
        %y_g = 4*cos(obj.t(obj.i))*sin(obj.t(obj.i))/(1+(sin(obj.t(obj.i)))^2) % para
      %testar ponto a ponto

        x_g = -1.2
        y_g = -1
        %xg_a=x_g; xg_a=y_g;

        % Get estimate of current pose
        [x, y, theta] = state_estimate.unpack();
        if obj.i>0
           grafx(obj.i) = x;
           grafy(obj.i) = y;
        end
        %salvando arquivos em txt para gerar grafico
        dadosx = fopen('cordx.txt','a');
        fprintf(dadosx,'%f ',x);
        fclose(dadosx);
        dadosy = fopen('cordy.txt','a');
        fprintf(dadosy,'%f ',y);
        fclose(dadosy);

        dadosx_g = fopen('cordx_g.txt','a');
        fprintf(dadosx_g,'%f ',x_g);
        fclose(dadosx_g);
        dadosy_g = fopen('cordy_g.txt','a');
        fprintf(dadosy_g,'%f ',y_g);
        fclose(dadosy_g);
        % dados sobre o erro está salvo logo abaixo

        % Compute the v,w that will get you to the goal
```

```
%v = inputs.v;

%% START CODE BLOCK %%

% 1. Calculate the heading (angle) to the goal.
d1 = ((x)^2+(y)^2)^(0.5);
d2 = ((x_g)^2+(y_g)^2)^(0.5)
% distance between goal and robot in x-direction
u_x = x_g - x;

% distance between goal and robot in y-direction
u_y = y_g - y;

d = ((u_y)^2 + (u_x)^2)^(0.5)

% angle from robot to goal. Hint: use ATAN2, u_x, u_y here.
theta_g = atan2(u_y,u_x);

% 2. Calculate the heading error.

% error between the goal angle and robot's angle
% Hint: Use ATAN2 to make sure this stays in [-pi,pi].
alpha = theta_g - theta;
e_k = atan2(sin(alpha),cos(alpha));

dadoserro = fopen('ek.txt','a');
fprintf(dadoserro,'%f ',e_k);
fclose(dadoserro);

% 3. Calculate PID for the steering angle

% error for the proportional term
e_P = e_k;

% error for the integral term. Hint: Approximate the integral using
% the accumulated error, obj.E_k, and the error for
% this time step, e_k.
e_I = obj.E_k + e_k * dt;

% error for the derivative term. Hint: Approximate the derivative
% using the previous error, obj.e_k_1, and the
% error for this time step, e_k.
e_D = (e_k - obj.e_k_1) * dt;
```

```
%            e_D = obj.e_k_1 - e_k;


        %% END CODE BLOCK %%


        %w = evalfis(e_k,fis1);%
        %v = evalfis(d,fis);

       output = evalfis([d e_k], fis2);
        v = output(1);
        w = output(2);

       %fprintf('distancia: (%0.3f)\n', d);
        % 4. Save errors for next time step

        % plot
        %obj.p.plot_2d_ref(dt, atan2(sin(theta),cos(theta)), theta_g, 'r');
        grid on
        %obj.p.plot_2d_ref(dt,x,y);
        %obj.p.plotarloc(x,y);
        %plot(x_g ,y_g,'r');
        %obj.p.plot_2d_ref(dt,theta_g,theta,'r');
        fprintf('x_g:(%0.3f)\n', x_g)
        fprintf('y_g: (%0.3f)\n', y_g)
        outputs = obj.outputs;  % make a copy of the output struct
        outputs.v = v;
        outputs.w = w;
    end

  end

end
```

APÊNDICE E – Código com odometria e PID “odometriaPID.py”

```
import time
import serial
from numpy import *
import matplotlib.pyplot as plt
import urllib2
import base64
import matplotlib.patches as mpatches
from math import *

ser = serial.Serial('/dev/ttyACM0',9600,timeout=1)
#ser = serial.Serial('/dev/ttyACM1',9600,timeout=1)
time.sleep(1.8)
global l
l = 0
x_g = []
y_g = []
locx = []
locy = []
ticks_direita_anterior = 0.0
ticks_esquerda_anterior = 0.0
theta = 0.0
x = 0
y = 0
encod1 = 0.0
encod2 = 0.0
d_esquerda_total = 0.0
d_direita_total = 0.0

eP = 0
eI = 0
eD = 0
prev_eP = 0
prev_eI = 0
prev_time = 0.0
dt = 0.0

i = linspace(-pi, pi, 100)

#x_g = -1.3 #alterando para a leminiscata
#y_g = -1.4 #alterando para a leminiscata
```

```
# """ #alterando para a leminiscata
for k in range(100):
    xg = 2*cos(i[k]) / (1 + (sin(i[k])) ** 2) + 2
    yg = 2*cos(i[k]) * sin(i[k]) / (1 + (sin(i[k])) ** 2)
    x_g.append(xg)
    y_g.append(yg)
#""" #alterando para a leminiscata
def odometria():
    global ticks_esquerda_anterior
    global ticks_direita_anterior
    global x
    global y
    global theta
    global d_esquerda_total
    global locx
    global locy
    global d_direita_total

    serialread()
    # Implementacao odometria
    R = 0.0625
    L = 0.5

    # raio das rodas (diametro)
    N = 980.0
    # ticks por revolucao
    B = 0.53
    # comprimento do eixo

    ticks_direita = encod2
    ticks_esquerda = encod1

    d_esquerda = ((2*pi*R)*(ticks_esquerda - ticks_esquerda_anterior))/N
    d_direita = ((2*pi*R)*(ticks_direita - ticks_direita_anterior))/N

    d_centro = (d_esquerda + d_direita)/2
    phi = (d_direita - d_esquerda)/B
    theta_dt = phi
    x_dt = d_centro*cos(theta)
    y_dt = d_centro*sin(theta)
    x_novo = x + x_dt
    y_novo = y + y_dt
```

```
        theta_novo = theta + theta_dt

        x = x_novo
        y = y_novo
        theta = theta_novo
        ticks_direita_anterior = ticks_direita
        ticks_esquerda_anterior = ticks_esquerda
        d_esquerda_total +=  d_esquerda
        d_direita_total += d_direita

    def serialread():
        global encod1
        global encod2

        PARAM_ASCII = str(chr(105))
        ser.write(PARAM_ASCII)
        time.sleep(0.1)

        enc1 = ser.readline()
        enc1 = list(enc1)
        for i in range(len(enc1)):
            if (enc1[i] == '\r'):
                if i == 1:
                    encod1 = ((int(enc1[0])))
                if i == 2:
                    encod1 =((int(enc1[0]))*10)+((int(enc1[1])))
                if i == 3:
                    encod1 = ((int(enc1[0]))*10**2)+((int(enc1[1]))*10)+((int(enc1[2])))
                if i == 4:
                    encod1 =((int(enc1[0]))*10**3)+((int(enc1[1]))*10**2)+((int(enc1[2]))*10**1)+((int(enc1[3])))
                if i == 5:
                    encod1 =
((int(enc1[0]))*10**4)+((int(enc1[1]))*10**3)+((int(enc1[2]))*10**2)+((int(enc1[3]))*10**1)+((int(enc1[4])))

        time.sleep(0.1)
        PARAM_ASCII = str(chr(106))
        ser.write(PARAM_ASCII)
        time.sleep(0.1)
        #0.5, 0.5, 0.2
        enc2 = ser.readline()
        enc2 = list(enc2)
        for i in range(len(enc2)):
            if (enc2[i] == '\r'):
```

```
            if i == 1:
                encod2 = ((int(enc2[0])))
            if i == 2:
                encod2 =((int(enc2[0]))*10)+((int(enc2[1])))
            if i == 3:
                encod2 = ((int(enc2[0]))*10**2)+((int(enc2[1]))*10)+((int(enc2[2])))
            if i == 4:
                encod2 =((int(enc2[0]))*10**3)+((int(enc2[1]))*10**2)+((int(enc2[2]))*10**1)+((int(enc2[3])))
            if i == 5:
                encod2 =
((int(enc2[0]))*10**4)+((int(enc2[1]))*10**3)+((int(enc2[2]))*10**2)+((int(enc2[3]))*10**1)+((int(enc2[4])))

    while(1):

        if (l >= 100):
            l = 0
        else:
            l = l + 1

        red_patch = mpatches.Patch(color='red', label=u'desire postion')
        blue_patch = mpatches.Patch(color='blue', label=u'robot position (dead reckoning)')
        plt.legend(handles=[red_patch, blue_patch])
        plt.scatter(x, y,c ='blue') #alterando para a leminiscata
        plt.scatter(x_g[l-1], y_g[l-1] , c ='red')
        #plt.scatter(x_g, y_g , c ='red') #alterando para a leminiscata
        #plt.pause(0.05)
        plt.pause(0.05)
        odometria()

        ex = x_g[l-1] - x
        ey = y_g[l-1] - y
        #ex = x_g - x #alterando para a leminiscata
        #ey = y_g - y #alterando para a leminiscata

        teta_g = atan2(ey,ex)
        gama = (teta_g-theta)
        gama = atan2(sin(gama) , cos(gama))

        gama = float(gama)
        ek = gama

        distancia = ((x - x_g[l-1] )**(2) + (y - y_g[l-1])**(2))**(0.5)
        #distancia = ((x - x_g )**(2) + (y - y_g)**(2))**(0.5) #alterando para a leminiscata
```

```
distancia = float(distancia)

# ------------------ Controle PID ---------------------------------------------
if (distancia > 0.1):

    #kP = 25
    #kP = 20
    kP = 20
    #kI = 0.2
    #kI = 1.5
    kI = 1.5

    #kD = 10
    #kD = 0.5
    kD = 0.5


    current_time = time.clock()

    dt = current_time - prev_time
    eP = gama
    eI = prev_eI + (gama*dt)
    eD = (eP - prev_eP)/dt

    prev_time = current_time
    prev_eP = eP
    prev_eI = eI

    w = (kP * eP) + (kI * eI) + (kD * eD)
    v = 8 / ( abs( w ) + 1 )**0.5

    if (w) > 8.0 :
        w = sign(w) * 8.0
    #if (v) > 5.0 :
        #v = sign(v) * 5.0
    if (v) > 3.1 :
        v = sign(v) * 3.1
    if v < 0.0:
        v = 0

    R = 0.0625
    L = 0.5
```

```
    vel_r = ((2 * v + w * L) / (2 * R))
    vel_r = (0.5*vel_r) + 128

    vel_l = ((2 * v - w * L) / (2 * R))
    vel_l = (0.5*vel_l) + 128

    # direita
    PARAM_ASCII = str(chr(100))
    ser.write(PARAM_ASCII)
    vel_r = str(vel_r)
    ser.write(vel_r)

    # esquerda
    PARAM_ASCII = str(chr(101))
    ser.write(PARAM_ASCII)
    vel_l = str(vel_l)
    ser.write(vel_l)

else:
    # Comando para parar
    # direita
    PARAM_ASCII = str(chr(102))
    ser.write(PARAM_ASCII)

    # esquerda
    PARAM_ASCII = str(chr(102))
    ser.write(PARAM_ASCII)

#ftemp = open("t", "a")
ft = open("theta", "a")
fx = open("cordx","a")
fy = open("cordy","a")
fx_g = open("cordx_g","a")
fy_g = open("cordy_g","a")
fek = open("ek","a")

a = str(x)
fx.write(a)
fx.write(" ")

b = str(y)
fy.write(b)
fy.write(" ")
```

```
    c = str(theta)
    ft.write(c)
    ft.write(" ")
##    d = str(t)
##    ftemp.write(d)
##    ftemp.write(" ")

    i = str(ek)
    if i != 0.0:
      fek.write(i)
      fek.write(" ")
    h = str(x_g)
    fx_g.write(h)
    fx_g.write(" ")
    f = str(y_g)
    fy_g.write(f)
    fy_g.write(" ")



ser.close()
```

REFERÊNCIAS

BEZERRA, C. G. Localização de um robô móvel usando odometria e marcos naturais. MS thesis. Universidade Federal do Rio Grande do Norte, 2004.

LAGES, W. F. Estimação de Posição e Orientação. Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Engenharia Elétrica ELE00070 Tópicos Especiais em Controle e Automação I, 2008.

OMRANE, H ; MASMOUDI, S. M. ;MASMOUDI, M. Fuzzy Logic Based Control for Autonomous Mobile Robot Navigation, Computational Intelligence and Neuroscience, Volume 2016, Article ID 9548482, 10 pages, 2016

XIONG, B ; QU, S. R. Intelligent vehicle's path tracking based on fuzzy control, Journal of Transportation Systems Engineering and Information, vol. 10, no. 2, pp. 70–75, 2010.

PARK, K ; CHUNG, H ; LEE, J. G. Dead Reckoning Navigation for Autonomous Mobile Robots. 3rd IFAC Symposium on Intelligent Autonomous Vehicles. 1998, Madrid, Spain

FU, G ; ZHANG, J ; CHEN, W ; PENG, F ; YANG, P ; CHEN, C. Precise Localization of Mobile Robots via Odometry and Wireless Sensor Network. International Journal of Advanced Robotic Systems, Vol 10, 203:2013

BATISTA, L.G.B.; ALVES, R.S.A.; MARTINS, G.S.M.; GAMARRA, D.F.T.; CUADROS, M.A.D.L. Localização de Robôs Móveis Usando Landmarks Artificiais, Internantional Conference on Industry and Applications (INDUSCON), Curitiba, 2016.

GOMIDE, F. A. C; GUDWIN, R. R. Modelagem, controle, sistemas e lógica *fuzzy*. SBA Controle & Automação. Vol. 4, nº 3, setembro-outubro. Universidade Estadual de Campinas, 1994.

BELLUCI, D. P. Sistemas Baseados em Regras Fuzzy e Aplicações. Dissertação de Mestrado em Matemática Aplicada. Universidade Federal do ABC, Santo André, SP, 2009.

MOTTA, D.R.V.; SALAROLLI, P. F.; ALMEIDA, G.M.; GAMARRA, D.F.T.; CUADROS, M.A.D.L. Comparative Trajectory Controllers: Fuzzy, Fixed Gains and Backstepping for Robots with Differential Traction Using Image Processing, Simposium Brasileiro de Automação Inteligente (SBAI), Porto Alegre, 2017.

GARCIA, T. R. ; SCHIRMER, R. D. ; JESUS, J. C. ; SCHUSTER, B. ; SEVERO, M. B. ; RICHARDS, D. R. ; CONRAD, C. C. ; SILVA, R. M. ; GUERRA, R. S. ; GAMARRA, D. F.T. . Construção de um Robô Móvel para Ensino das Disciplinas dos Cursos de Engenharia. In: Congresso Brasileiro de Educação em Engenharia, 2017, Joinville. Congresso Brasileiro de Educação em Engenharia (XLV COBENGE), 2017.

NASCIMENTO, E. J. ; CUADROS, M. A. S. L. ; GAMARRA, D. F.T. . Utilização dos algoritmos K-Means e Fuzzy C-Means para Controle de um robô móvel através de um Sensor Kinect. In: WEIT 2017 - IV Workshop-Escola de Informática Teórica, 2017, santa Maria. WEIT 2017 - IV Workshop-Escola de Informática Teórica, 2017.

MOREIRA, A. F. ; MIRANDA, P. H. A; FERNANDES, A.S; BATISTA, A. V. A; SILVA J. B. Controle De Trajetoria De Um Robô Móvel Baseado Na Arquitetura Ackermann. Congresso Brasileiro de Automática (CBA), 2018, João Pessoa, Paraíba.

DIAS, S. M; MARTINS, A. M; ARAÚJO A. M; ALSINA, P. J. Controlador Adaptativo Robusto Para Um Robô Móvel Com Acionamento Diferencial. Simpósio Brasileiro De Automação Inteligente, Brasília, DF, 2009 (IXSBAI).

RIBEIRO, T. T; SANTOS, J. T. ; CONCEIÇÃO A. G. S. ; COSTA, A. L. ; Sistema Microprocessado Para Controle Em Tempo Real De Robôs Móveis Ominidrecionais. Simpósio Brasileiro de Automação Inteligente. São João del-Rei – MG. 2011 (XSBAI).

NASCIMENTO, T. P. Controle De Trajetória De Robôs Móveis Omni-Direcionais: Uma Abordagem Multivariável. Dissertação de Mestrado, Universidade Federal da Bahia, Salvador, 2009.

COELHO, M. H; FERRI, V. P; VALERIANO, E; FRIGO, L. B; POZZEBON, E. Application of Fuzzy Control in Embedded Sensing Systems for Beekeeping Monitoring. XLIV Conferência Latino-Americana de Informática, Universidade Presbiteriana Mackenzie, São Paulo, 2018.

PIMENTEL, L. S. S; PANCERI, J. A. C; PANCERI, R. C; SOUZA, M. A. Q. L; ALMEIDA, G. M; PEREIRA, R. P. A; Construção de um Robô Móvel como Plataforma Didática para Ensino de Microcontroladores. XLI Congresso Brasileiro de Educação em Engenharia (COBENGE), Gramado, RS, 2013.