UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

TELMO FRIESEN

CLASSIFICAÇÃO DE OBJETOS METÁLICOS E MEDIDA DE SUA RESPECTIVA POSIÇÃO ANGULAR

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2013

TELMO FRIESEN

CLASSIFICAÇÃO DE OBJETOS METÁLICOS E MEDIDA DE SUA RESPECTIVA POSIÇÃO ANGULAR

Trabalho de Conclusão de Curso apresentado ao Departamento Acadêmico de Informática como requisito parcial para obtenção do grau de Engenheiro no Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Gustavo Benvenutti Borba

CURITIBA

2013

AGRADECIMENTOS

Agradeço primeiramente à Deus pela inspiração para realização deste trabalho. Agradeço a meus pais pelo amor e dedicação ao me dar a oportunidade de cursar Engenharia de Computação. E finalmente agradeço ao professor Gustavo Benvenutti Borba pela orientação do trabalho.

RESUMO

FRIESEN, Telmo. Classificação de objetos metálicos e medida de sua respectiva posição angular. 88 f. Trabalho de Conclusão de Curso – Bacharelado em Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

TODO: ARRUMAR CASO NÃO SEJA FEITA A IMPLEMENTAÇÃO EM C++ Técnicas de visão computacional podem ser ferramentas úteis para a automação de linhas de produção em tarefas como, por exemplo, a fabricação de componentes eletrônicos, a inspeção do acabamento em objetos metálicos, a produção de circuitos impressos, entre outros. Muitas vezes, existem etapas da produção em indústrias metalúrgicas, nas quais se necessita identificar objetos metálicos e sua respectiva orientação angular, seja para análise, separação ou mesmo somente para classificação desses objetos. O objetivo deste trabalho é desenvolver um sistema capaz de classificar objetos metálicos e medir seu respectivo ângulo de rotação com relação ao eixo x do plano cartesiano, sendo os objetos previamente conhecidos pelo sistema utilizando-se técnicas de aprendizagem de máquina. O desenvolvimento do projeto é dividido em quatro etapas. A primeira etapa consiste no estudo de técnicas de segmentação de imagem e implementação no software MATLAB. A segunda etapa consiste no estudo de técnicas de descrição de imagens e também implementação no MATLAB. Na terceira etapa é implementado no MATLAB um método de classificação. Finalmente, as técnicas das etapas anteriores que apresentarem os melhores resultados são implementadas utilizando-se linguagens como C, C++ ou Java, com o objetivo de construir um protótipo funcional a ser utilizado no ambiente de chão de fábrica. O sistema é dividido em três módulos: segmentação, descrição e classificação dos objetos. Após a aquisição da imagem do objeto a ser classificado o módulo de segmentação seleciona a área de interesse da imagem. No módulo de descrição são extraídas características da área de interesse da imagem, as quais são fornecidas ao terceiro módulo do sistema que classifica o objeto e mede a sua respectiva orientação angular. Portanto, o resultado do trabalho é um sistema capaz de classificar objetos utilizando-se algoritmos de aprendizagem de máquina, medir o ângulo desses objetos e implementar esta solução em uma plataforma que possa ser utilizada em ambientes industriais.

Palavras-chave: Processamento de Imagens, Classificação de Objetos, Aprendizagem de Máquina, Rede Neural, Análise de Componentes Principais, Descritores de Fourier

ABSTRACT

FRIESEN, Telmo. Classification of metallic objects and measure of their respective angular position. 88 f. Trabalho de Conclusão de Curso – Bacharelado em Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

TODO: TRADUZIR C++ This document presents a complete description of the reconstruction of a robotic platform, in addition to the development and test of two autonomous navigation algorithms for it, aiming to analyze and compare the behavior of the ED-FCM approach to robotic navigation (Event-Driven Fuzzy Cognitive Maps) against a classic Fuzzy logic based navigation. In this document, all steps of the reconstruction of the robotic platform, including all needed documentation for any future use, are described. Additionally, all theoretical knowledge necessary to understand both navigation algorithms is provided, together with all description of the implementation, test and comparison of the algorithms.

Keywords: Image Processing, Object Classification, Machine Learning, Neural Network, Principal Components Analysis, Fourier Descriptors

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

LISTA DE SÍMBOLOS

SUMÁRIO

1 INTRODUÇÃO

O problema do controle de navegação de robôs móveis autônomos apresenta um grande desafio, devido ao fato de o ambiente ser dinâmico (geralmente), haver sensoriamento sujeito a ruídos e exigências de controle e tomada de decisão em tempo real (FRACASSO; COSTA, 2005). Um sistema de navegação deve garantir que o robô móvel atinja satisfatoriamente o destino de sua trajetória, enviando ao robô comandos necessários para sua locomoção, de maneira precisa e suave, ao mesmo tempo em que permite reações rápidas às mudanças de ambiente para evitar colisões (FRACASSO; COSTA, 2005).

Na robótica móvel, existem dois paradigmas principais que guiam os projetos de diversas arquiteturas de sistemas de navegação: o reativo e o deliberativo. O paradigma reativo procura reproduzir a reação imediata dos animais aos estímulos do ambiente. Geralmente, arquiteturas reativas são empregadas como uma camada de nível inferior na navegação de robôs móveis, pois apresentam a vantagem de resposta em tempo real uma vez que mapeiam a leitura dos sensores, diretamente, em ações. Arquiteturas deliberativas, por outro lado, intercalam o processo da tomada de decisão, desde a percepção até a ação, com uma etapa de planejamento que pode demandar grande tempo computacional, impedindo a atuação do robô em tempo real. Atualmente, são definidas também arquiteturas híbridas, conjugando ambos os paradigmas (FRACASSO; COSTA, 2005).

Sensores de distância são comumente utilizados na construção de robôs móveis autônomos. Esses sensores são capazes de medir a distância do robô em relação a um obstáculo e funcionam através de princípios físicos diversos, como reflexão de ondas de ultrassom e raios infravermelhos. A medição efetuada por esses sensores é frequentemente utilizada como a entrada dos sistemas de navegação e é comum que esses sensores, em versões de tamanho reduzido, sejam encontrados em robôs móveis de pequeno porte. Basicamente, um robô móvel autônomo é formado por: um sistema de locomoção, por exemplo, um chassi com duas rodas de tração e uma terceira roda guia; um sistema de potência capaz de fornecer corrente aos motores das rodas de tração; alimentação por bateria; sensores de distância; *encoders* para auxílio à odometria; um sistema microcontrolado capaz de processar os sinais dos sensores e *encoders*, controlar a

potência dos motores e comunicar-se através de um meio de transmissão para programação do robô.

Existem metodologias de inteligência artificial que podem ser empregadas em robótica móvel, como a lógica *fuzzy*, redes neurais ou uma combinação de ambas denominada "neuro-*fuzzy*". O presente trabalho é um estudo comparativo entre dois algoritmos de navegação, que foram implementados para desempenhar o comportamento reativo de um robô móvel autônomo, o qual deverá desviar de obstáculos. Um dos algoritmos consiste de um mecanismo de inferências utilizando conjuntos *fuzzy*, com tomada de decisão baseada diretamente em regras de inferência, enquanto que o outro é um algoritmo baseado em ED-FCM (*Event-Driven Fuzzy Cognitive Maps*), que é considerado por alguns autores como um sistema neuro-*fuzzy*, o que pode ser lido no artigo de (MENDONÇA; ARRUDA; NEVES, 2011). Para realizar a comparação desses algoritmos, é necessária uma plataforma robótica real, obtida pela reconstrução do robô "Bellator", cordialmente cedido à equipe pelo professor Heitor Silvério Lopes, do Departamento de Eletrônica da Universidade Tecnológica Federal do Paraná (DAELN/UTFPR).

O Bellator é um robô móvel que estava em desenvolvimento em um trabalho anterior (com propósito distinto dos objetivos deste projeto). Este robô teve de ser reconstruído e adaptado às necessidades deste projeto, levando em conta a possibilidade de utilizá-lo em projetos futuros. O robô Bellator foi equipado com sensores de distância em quantidade suficiente para alimentar a entrada do sistema de navegação, encoders para a realização de odometria, baterias para o fornecimento de energia, e dispositivos de *hardware* adequados para o controle do robô. Os algoritmos de navegação foram "embarcados" no mesmo, evitando o uso de um PC convencional para controlá-lo, ou necessidade de comunicação sem fio. Desse modo o robô pode operar de forma totalmente autônoma, sendo capaz de se locomover e tomar decisões sem necessidade de comunicação com um sistema externo de controle.

A lógica *fuzzy* é uma abordagem estudada no sétimo período, na disciplina de Sistemas Inteligentes, do curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná. Este trabalho então propôs a reconstrução e adequação do robô Bellator, e da implementação de dois algoritmos de controle de navegação autônoma para o mesmo: um controlador *fuzzy* e um controlador baseado no algoritmo ED-FCM, como proposto por MENDONÇA, ARRUDA e NEVES (2011). O algoritmo ED-FCM possui uma proposta inovadora, cujos conceitos podem ser aplicados na robótica móvel, porém também em controladores industriais (MENDONÇA; ARRUDA; NEVES, 2011). O presente trabalho também apresenta uma avaliação qualitativa dos controladores desenvolvidos, permitindo uma discussão sobre a adequação das abordagens *fuzzy* e ED-FCM na tarefa de controlar o comportamento reativo de um

robô móvel real.

Através deste trabalho, a equipe integrou vários assuntos condizentes com a área da Engenharia de Computação. A reconstrução do robô levou em conta conhecimentos de *hardware* e eletrônica, programação de baixo nível, experimentos e testes em bancada de laboratório de eletrônica. A programação dos algoritmos avaliou a capacidade dos integrantes de colocar em prática conceitos previamente conhecidos, como a lógica *fuzzy*, e assimilarem novos conceitos, como o ED-FCM. O resultado final foi obtido ao unir e integrar o *hardware* ao *software* de controle do robô, o que representou um grande desafio e uma experiência enriquecedora como trabalho de graduação.

1.1 MOTIVAÇÃO

A pesquisa do estado da arte revelou a existência de trabalhos que apresentam novos métodos para navegação autônoma através do uso de lógica *fuzzy*, entretanto, são poucos os trabalhos propondo a comparação entre os métodos já existentes. Com o intuito de enriquecer essa área de pesquisa, a equipe optou por desenvolver este projeto. Para realizar uma comparação prática desses algoritmos, que levasse em conta condições reais de ruídos e imperfeições dos dados provenientes dos sensores, foi indispensável a utilização de uma plataforma robótica real, o que agregou a necessidade de conhecimentos de *hardware* à equipe, porém forneceu resultados mais significativos do que se fosse utilizada uma simulação computacional. A plataforma foi obtida através da reconstrução e adaptação do "Bellator", o qual fora usado em um projeto da disciplina de Oficina de Integração 3 (MARIN et al., 2010), mas em um escopo diferente. Teve-se em vista a possibilidade de adquirir uma plataforma comercial, porém o custo elevado de plataformas robóticas móveis disponíveis no mercado levou a equipe à optar por reconstruir o Bellator. Uma das plataformas comerciais foi a X80, cujo preço é de 2795 dólares (DOCTOR-ROBOT, 2012). A possibilidade de disponibilizar essa plataforma reconstruída e documentada para trabalhos acadêmicos futuros também foi um fator decisivo nessa escolha.

Os algoritmos escolhidos para implementação foram o algoritmo de navegação *fuzzy*, baseado em inferências sobre conjuntos *fuzzy*, e o algoritmo de navegação ED-FCM, tendo em vista a oportunidade de analisar o comportamento de uma nova abordagem de navegação robótica em relação à uma abordagem já conhecida e estudada ao longo do curso de graduação, ambas utilizando conceitos de lógica *fuzzy* de maneiras distintas.

1.2 OBJETIVOS

Os objetivos do presente trabalho foram a reconstrução e adequação da plataforma robótica Bellator, a descrição do *hardware* e *software* do mesmo, o estudo, projeto e a implementação dos algoritmos de navegação, a execução de testes de comparação do comportamento dos algoritmos e a análise dos resultados provenientes desses testes.

O robô deve ser reconstruído e adequado, equipado com sensores de distância e *encoders* acoplados às rodas, e deve ser alimentado através de baterias que forneçam adequadamente energia ao sistema microcontrolado, sensores e motores; deve ser capaz de ajustar a velocidade de cada roda de maneira independente, deve processar os sinais dos sensores de distância e dos *encoders* e implementar rotinas para disponibilizar esses dados.

Os algoritmos de navegação devem ser executados em um *hardware* independente acoplado à plataforma, visando minimizar atrasos de comunicação e tornar o robô autônomo. Esse *hardware* deve ser capaz de ler os dados dos sensores do robô, como fonte de dados para os algoritmos, e enviar comandos de movimentação. Cada teste dos algoritmos deve ser realizado em um ambiente igual para ambos algoritmos, sendo que o objetivo de cada algoritmo é guiar o robô no deslocamento, desviando-o de possíveis obstáculos, tendo como entradas os valores de leitura dos sensores de distância e dos encoders e como saída, a velocidade e direção do robô. Os experimentos devem ser registrados em gravação de vídeo e a análise deve ser baseada nesses resultados.

1.3 METODOLOGIA

A metodologia deste trabalho foi dividida nas seguintes etapas: reconstrução e adequação do robô Bellator, estudo, projeto, implementação, testes dos algoritmos de navegação e análise dos resultados. A equipe optou por essa divisão porque o escopo do projeto foi abrangente, envolvendo desenvolvimento de *hardware* e *software*. Desse modo, a reconstrução e adequação do robô foram pré-requisitos para o projeto e implementação dos algoritmos, sendo estes, pré-requisitos para realização dos testes que, finalmente, foram pré-requisitos para a análise do resultado final.

A reconstrução e adequação consistiram na avaliação do estado do robô, o projeto e implementação do *hardware* e *software* necessários para o funcionamento adequado do mesmo. Durante a avaliação, foram testados os sensores de distância, os encoders das rodas, baterias, motores e demais componentes disponíveis. A elaboração do *hardware* consistiu na confecção

de uma placa de roteamento para alimentar os sensores e encoders, tratar os sinais dos enconders amplificando-os, rotear os sinais dos sensores de distância e encoders aos respectivos pinos de entrada do microcontrolador e rotear os sinais de controle para os motores (utilizando *drivers* de potência contendo pontes H). O *software* consistiu na adequação do código dispobilizado no projeto Bellator (MARIN et al., 2010), no qual as rotinas de leitura dos sensores, comunicação de dados, recepção de comandos de controle do robô e protocolo de comunicação foram atualizadas de acordo com as necessidades desse projeto. O *hardware* acoplado à plataforma foi configurado para comunicar-se com o microcontrolador e um *software* foi implementado para executar os algoritmos de navegação.

O estudo dos algoritmos de navegação consistiu na revisão bibliográfica da Lógica *Fuzzy* e do ED-FCM. Os projetos dos algoritmos tomaram como base esse estudo, e a implementação consistiu no desenvolvimento dos códigos em linguagem C++, compilados para execução no *hardware* acoplado, e os testes consistiram na realização de diversos experimentos nos quais o robô navegou guiando-se exclusivamente pelos algoritmos implementados. Esses experimentos foram registrados em gravação de vídeo e a análise consistiu na interpretação qualitativa desses resultados, evidenciando as vantagens e desvantagens de cada algoritmo.

1.4 ESTRUTURA DO DOCUMENTO

Esta monografia está dividida em quatro capítulos: o primeiro corresponde à introdução, na qual são apresentadas a motivação, os objetivos e a metodologia empregada. O segundo capítulo contém a fundamentação teórica do projeto, em que são apresentados o estado inicial e o estado final do robô após a reconstrução e adequação, a explicação da Lógica *Fuzzy* e da abordagem ED-FCM. O terceiro capítulo é o desenvolvimento do trabalho, no qual são descritas as atividades realizadas pela equipe, incluindo a reconstrução e adequação do robô, o projeto e a implementação dos algoritmos, o desenvolvimento dos testes em campo e análise dos resultados. O quarto capítulo contém a conclusão do projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresentará a fundamentação teórica, que consiste na descrição do estado inicial robô Bellator, com uma visão geral do projeto no qual este robô estivera em uso, a descrição do material entregue à equipe, a especificação do robô após a reconstrução e adequação, a apresentação dos estudos de Lógica Fuzzy e ED-FCM (*Event-Driven Fuzzy Cognitive Maps*).

2.1 ESTADO INICIAL DO PROJETO

Esta seção visa descrever com quais recursos a equipe iniciou a execução do trabalho, ou seja, a situação do robô e seus componentes de *hardware*, o principal e mais importante recurso desse projeto, os componentes de *software* e a documentação de ambos, da forma como foram entregues à equipe.

2.1.1 Visão Geral

O robô Bellator, disponibilizado à equipe para realização deste trabalho, já havia sido utilizado anteriormente em um projeto de Oficina de Integração 3, que visou a implementação da eletrônica embarcada que permitisse que a plataforma robótica pudesse ser controlada remotamente por *joystick* (MARIN et al., 2010). O trabalho desempenhado na disciplina consistiu no projeto e implementação de uma plataforma robótica dividida em três camadas: baixo nível, alto nível e supervisória. A camada de baixo nível era responsável por controlar os motores do robô e receber as leituras dos sensores. A camada de alto nível tinha como responsabilidades comunicar-se com a camada de baixo nível via conexão serial, fazer obtenção de vídeo através de uma *webcam* e comunicar-se com a camada supervisória através de uma conexão sem fio (para transmissão dos dados de vídeo captados pela *webcam*). Finalmente, a camada supervisória era responsável por receber os dados de vídeo, mostrar na tela para o usuário, e permitir o controle do robô remotamente através de um *joystick*. O diagrama esquemático da figura 1 ilustra a configuração do projeto de oficinas, ponto de partida para a reconstrução do robô Bellator.

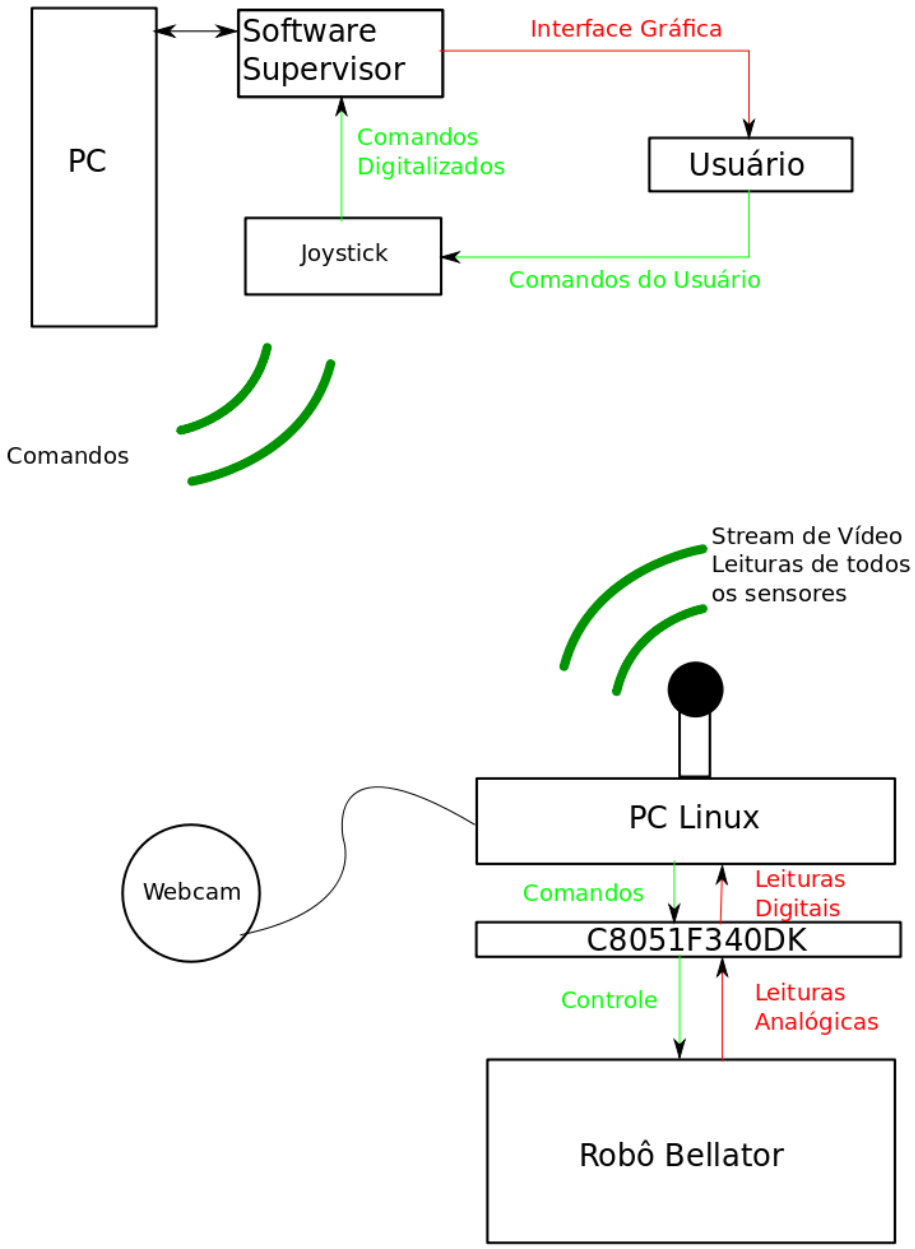

Figura 1: Diagrama do projeto de oficina 3.

Fonte: (MARIN et al., 2010)

A partir da figura 1, o funcionamento do projeto Bellator pode ser explicado: a camada de baixo nível é composta pelo robô Bellator, equipado com dois motores elétricos Bosch FPG 12V, cinco sensores de distância "2Y0A02F98" da Sharp, uma bateria Unybatt 12V-7,2 Ampère hora, duas pontes H LN298, dois *encoders* ópticos HEDS-9700 e uma placa microcontrolada C8051F340, que é capaz de ler e converter leituras de tensão analógicas dos sensores bem como

produzir sinais de controle para os motores do robô. Esta placa é conectada à camada de alto nível, composta por um PC Embarcado VIA EPIA ME6000 Mini-ITX com sistema operacional Linux, através de uma conexão serial. Utilizando-se de um protocolo de comunicação, esse PC embarcado envia comandos de movimentação para a camada de baixo nível (conexão serial) e recebe as leituras dos sensores obtidas pela camada de baixo nível. O PC embarcado também comunica-se com a camada supervisória para receber comandos de movimentação do usuário e enviar as leituras dos sensores para o mesmo (comunicação sem fio). Além disso, o PC embarcado envia à camada supervisória um fluxo(*stream*) de vídeo gerado por uma *webcam* Genius iLook 316. Finalmente, o *software* supervisor remoto, o qual é executado em um PC com máquina virtual Java, fornece as informações recebidas da camada de alto nível para o usuário, permitindo-o tomar decisões sobre a locomoção do robô. O *software* também recebe comandos de movimentação do usuário, gerados em um *joystick* do videogame Sony Playstation 2, enviando-os para a camada de alto nível pela mesma conexão. Ao receber os comandos de movimentação, a camada de alto-nível repassa para a camada de baixo nível, responsável pela efetivação dos comandos, alterando os PWMs enviados aos motores de acordo com os comandos recebidos, controlando suas velocidades.

Os componentes de *hardware* do robô, o *software* de controle supervisório e a camada de baixo nível, ou seja, o *software* da placa C8051F340, utilizados no projeto de oficina estão documentados em detalhes em (MARIN et al., 2010). A seguir, será descrito como o robô foi recebido pela equipe e quais componentes foram aproveitados.

2.1.2 Recebimento do Robô

O robô Bellator foi entregue à equipe em Abril de 2011, em uma caixa, desmontado, juntamente com toda a documentação (MARIN et al., 2010) disponível em mídia digital. A caixa continha os seguintes itens:

- Chassi do robô Bellator com dois motores Bosch FPG12V e pontes H acopladas;
- Um par de *encoders* ópticos HEDS-9700 acoplados ao eixo de cada roda;
- Cinco sensores de distância “2Y0A02F98” da Sharp;
- Duas baterias Unybatt 12V-7,2 Ampére hora;
- Uma placa microcontrolada C8051F340;
- Uma placa de roteamento, produzida no projeto Bellator (MARIN et al., 2010);

- Um PC Embarcado VIA EPIA ME60000 Mini-ITX.

O chassi do robô e os componentes acoplados foram a base da plataforma robótica utilizada pela equipe e foram essenciais para a execução desse trabalho. Os sensores de distância foram utilizados na localização de obstáculos, fornecendo dados de entrada aos algoritmos de navegação para tomada de decisão e determinação de ações de controle. Os *encoders* foram utilizados para fornecer uma realimentação odométrica para ajustar a velocidade das rodas. A placa microcontrolada foi utilizada para realizar o controle de baixo nível do robô, que foram: acionamento dos motores, conversão analógica para digital das leituras dos sensores, contagem dos pulsos dos *encoders* e comunicação serial com o *hardware* acoplado. As baterias foram utilizadas para alimentar os sensores, *encoders*, os motores e a placa microcontrolada.

Alguns itens mencionados na seção 2.1.1, referentes ao projeto de oficinas, não foram recebidos ou não foram utilizados nesse trabalho. A *webcam* e *joystick*, por exemplo, não foram entregues pois não foram necessários. O *joystick* não foi necessário porque esse trabalho se trata de um sistema de navegação autônomo, que descarta a necessidade de um controle remoto, e a *webcam* não foi necessária porque esse trabalho não abordou a navegação através de imagem de vídeo. A placa de roteamento entregue foi utilizada nos testes dos componentes, visto que foi necessária para testar o funcionamento do robô. Essa placa foi reprojetada e reconstruída. O PC embarcado foi entregue destituído de qualquer documentação e, além disso, como o novo objetivo do robô não implicou a necessidade de comunicação sem fio ou implementação de stream de vídeo, motivo principal para a utilização desse PC no projeto Bellator (MARIN et al., 2010), a equipe optou por descartar esse recurso e utilizar outra placa, descrita em detalhes na seção 2.2.4. O processo de reconstrução e adaptação da plataforma robótica é descrito em detalhes no capítulo 3.

2.1.3 Considerações

A possibilidade de reconstruir e adequar o robô Bellator e o recebimento desse material consistiram uma importante etapa nesse projeto. A plataforma Bellator foi uma opção de recurso ao apoio do estudo qualitativo proposto, mas precisou ser reconstruída e adequada às necessidades do projeto. Essa reconstrução mostrou-se viável porque o material estava disponível. A especificação do robô após a reconstrução é descrita na seção 2.2.

2.2 ESPECIFICAÇÕES DO ROBÔ

Esta seção visa descrever a composição de *hardware* do robô Bellator após a reconstrução e adequação do mesmo. Essa decrição inclui a apresentação dos sensores infravermelhos, enconders ópticos, placa microcontrolada C8051F340DK, placa TS-7260, placa de roteamento e a apresentação do produto final, ou seja, o robô montado.

2.2.1 Sensor IR 2Y0A02F98

O sensor 2Y0A02F98 é um sensor analógico infravermelho e mede distâncias no intervalo de 20 a 150 centímetros (SHARPCORPORATION, 2006), sendo que os valores de tensão de resposta do sensor seguem a curva mostrada na figura 2. Como o valor de leitura é analógico, foi necessário que a placa C8051F340DK fosse programada para converter essa leitura em digital. O código de conversão está de acordo com o projeto Bellator (MARIN et al., 2010) e também efetua a transferência desses dados por comunicação serial.

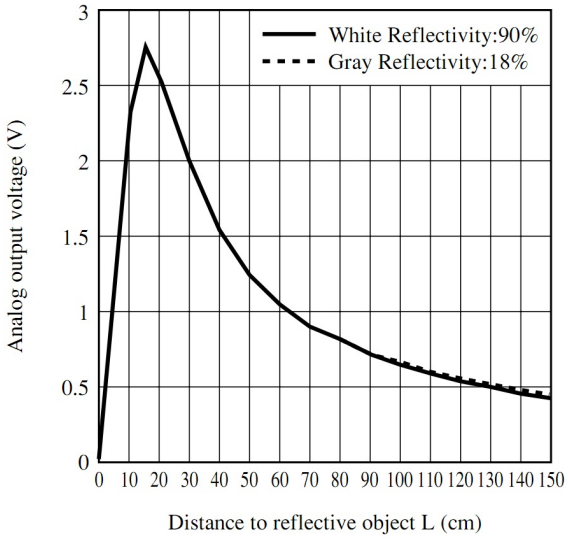

Figura 2: Curva de resposta do sensor de distância.
Fonte: (SHARPCORPORATION, 2006)

O modelo é pouco influenciado pelas cores dos objetos refletidos, isso é devido ao método de medição baseado em triangulação (SHARPCORPORATION, 2006). O sensor possui uma tensão de alimentação recomendada na faixa de 4,5 a 5,5 V (SILICONLABS, 2006). Como as baterias disponíveis eram de 12 V, foi necessária utilização de um regulador de tensão. O cálculo dos valores dos resistores foram baseados na equação 1 e o diagrama esquemático do

regulador é mostrado na figura 3. Esse circuito compõe uma das partes da placa de roteamento, conforme mostrado na seção 2.2.5.

$$V_{OUT} = 1,25V(1 + R_2/R_1) \quad (1)$$

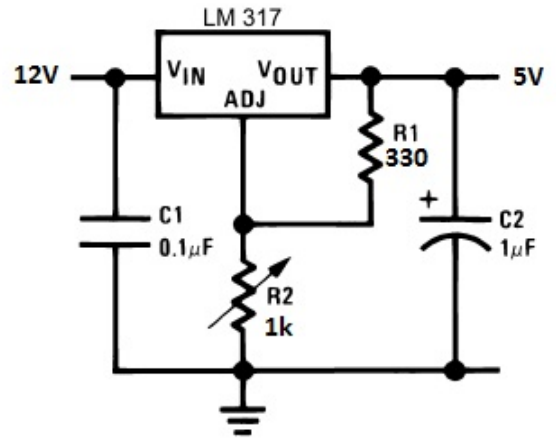

Figura 3: Diagrama esquemático do regulador de tensão dos sensores de distância.
Fonte: (SEMICONDUCTOR, 2012)

O modelo em questão é adequado ao projeto pois apresenta dimensões compatíveis com o chassi do robô e, como a finalidade desses sensores é auxiliar a navegação do robô em ambientes fechados, a faixa de resposta é suficiente para detecção de objetos. Contudo, há a possibilidade de, em projetos futuros, acrescentar outros tipos de sensores mais precisos voltados à medição de distâncias menores.

2.2.2 Encoder Óptico HEDS-9700

O *encoder* óptico HEDS-9700 é um circuito capaz de gerar uma onda quadrada à medida que um *encoder* de quadratura é rotacionado (AGILENTTECHNOLOGIES, 2002). A curva de resposta desse sensor é ilustrada na figura 4. Nessa figura são ilustrados dois canais A e B, havendo um defasamento de ϕ entre os sinais. Nesse projeto, foi utilizado o sinal de apenas um dos canais, pois a informação desejada era simplesmente a contagem de pulsos gerada por cada encoder. O *encoder* de quadratura utilizado na plataforma robótica está fixado no eixo de cada roda e apresenta 1800 pulsos por volta. A placa C8051F340DK foi programada para contar esses pulsos e fornecer uma medida odométrica para realimentação da velocidade. Essa programação é descrita na seção 3.1.4 e permite ajustar a velocidade das rodas de forma

independente. Como a placa de roteamento do projeto de oficinas (MARIN et al., 2010) não foi projetada para tratar o sinal deste *encoder*, foi projetada uma nova versão dessa placa, descrita na seção 2.2.5.

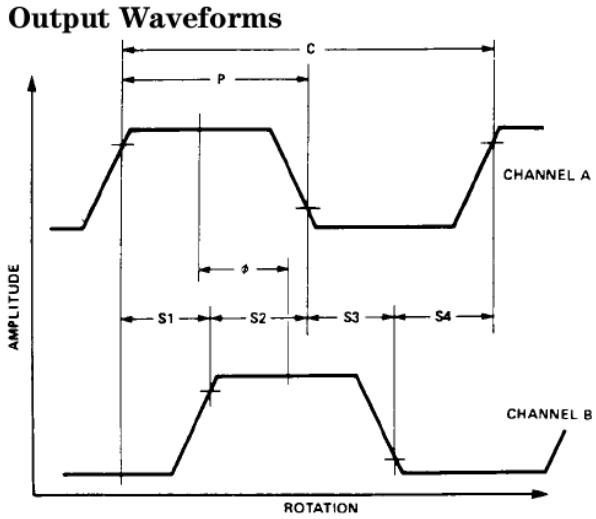

Figura 4: Formas de onda de saída do encoder óptico.
Fonte: (AGILENTTECHNOLOGIES, 2002)

2.2.3 C8051F340DK

O C8051F340 é uma unidade microcontroladora (MCU) equipada com um núcleo da família 8051 e vários dispositivos periféricos dispostos em uma placa de circuito impresso. As especificações dessa unidade foram retiradas do relatório do projeto Bellator (MARIN et al., 2010). O diagrama em blocos do kit é apresentado na figura 5. A C8051F340 possui as seguintes características (SILICONLABS, 2006):

- Conversor ADC 10 bits de até 200 ksps (amostras por segundo);
- Dois comparadores;
- *Brown-out Reset* e *Power-on Reset*;
- Tensão de Referência interna;
- Porta USB 2.0;
- Duas *interfaces* seriais (UART) e uma *interface* SPI;
- Fonte de Alimentação de 2.7 até 5.25V regulada internamente;

- Micro-processador 8051 de até 48 MIPS;
- 4352 Bytes de memória RAM;
- 40 Portas de E/S;
- 4 temporizadores de 16 bits;
- Seleção de *clock* interno de alta ou baixa velocidade ou *clock* externo.

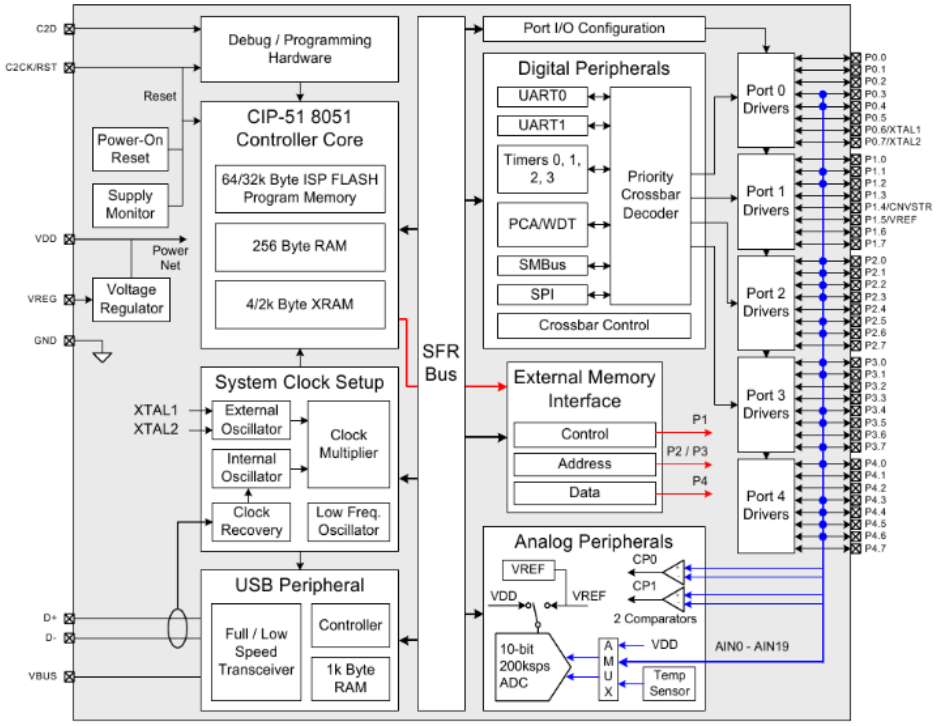

Figura 5: Diagrama em blocos do kit C8051F340DK.
Fonte: (SILICONLABS, 2006)

2.2.4 Placa TS-7260

A placa TS-7260 é um sistema embarcado equipado com um processador ARM e sistema operacional Linux. O sistema possui periféricos para realização de comunicação serial, ethernet, usb, entre outros. A lista a seguir descreve os componentes relevantes para o projeto. Os dados, bem como a figura 6, foram retirados do *datasheet* (ARM, 2009).

- Processador ARM9 de 200MHz baseado no Cirrus EP9302
- 32MB de memória NAND Flash

- 32MB de memória SDRAM
- Consumo menor que 1 Watt mesmo em capacidade máxima
- Porta Ethernet 10/100
- Duas portas USB 2.0
- Entrada de 4.5 a 20 Volts
- Dimensões: 9.7 cm por 11.5cm

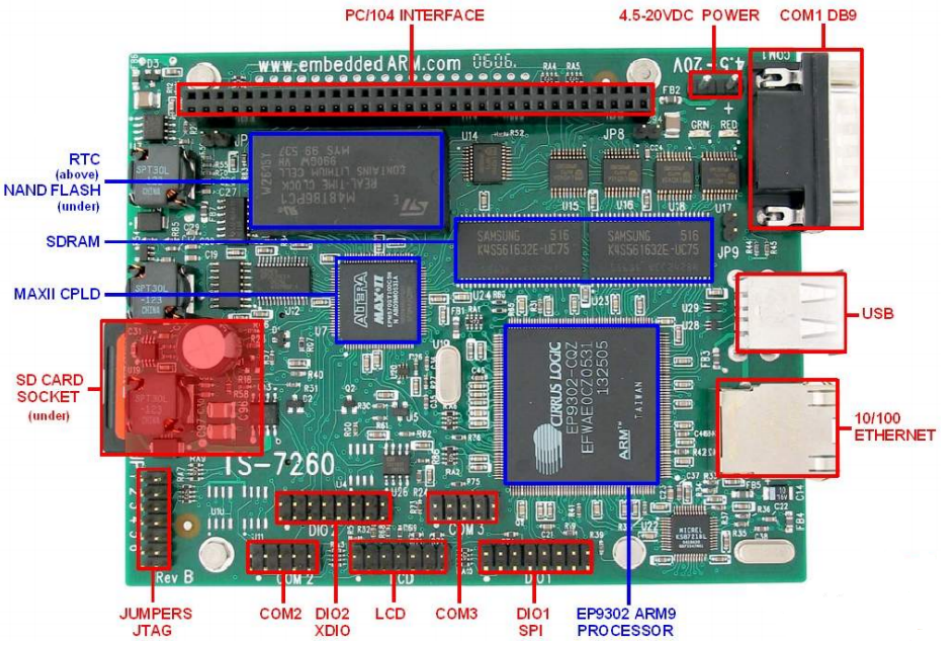

Figura 6: Foto da placa TS-7260.

Fonte: (ARM, 2012)

Mais informações sobre o sistema, bem como a documentação completa, estão disponíveis no manual da placa, disponível nas referências bibliográficas (ARM, 2012).

2.2.5 Placa de Roteamento

A placa de roteamento é um componente desenvolvido com base na placa disponível do projeto Bellator (MARIN et al., 2010). Como a placa de roteamento recebida com o Bellator não tratava os sinais dos *encoders* e estava construída de forma rudimentar, em uma placa universal, ela foi reprojetada e reconstruída pela equipe. As especificações da placa de roteamento são:

- Dimensões: 5 x 9 cm;
- Trilhas de cobre de aproximadamente 1 mm;
- Regulador de tensão LM317T: Entrada até 40V, Saída 5,05V;
- *Buffer* para PWM: 74LS244;
- Conectores para cabos *flat*, PWM, Sensores e *Encoders*.

O projeto da placa, métodos de desenvolvimento, requisitos, entre outros serão descritos de forma detalhada na seção 3.1.6.

2.2.6 Robô Bellator

O robô Bellator, como mencionado na seção 2.1, foi reconstruído e adaptado ao projeto. O robô consiste de um chassi de 40 cm de largura por 50 cm de comprimento, duas rodas de tração e uma roda guia, conforme ilustrado na figura 7. As rodas de tração estão nas laterais da parte dianteira do robô e possuem diâmetro de 20 cm e largura de 4 cm. Ambas possuem o *encoder* óptico HEDS-9700, descrito na seção 2.2.2, fixados nos respectivos eixos. A roda guia está no centro da parte traseira do robô e possui diâmetro de aproximadamente 6 centímetros e espessura de 2 centímetros. Todas as rodas são da marca Schioppa e chassi do robô permanece a 3 cm da superfície do solo. Com o objetivo de auxiliar a navegação do robô e fazer varreduras do ambiente, foram fixados cinco sensores do modelo 2Y0A02F98, descrito na seção 2.2.1, dispostos uniformemente nas laterais do chassi, conforme ilustra a figura 7. Os outros componentes do robô estão listados a seguir:

- 2 Motores Bosch FPG 12V;
- 2 Baterias Unybatt 12V-7,2 Ampére-hora;
- Duas pontes H L 298.

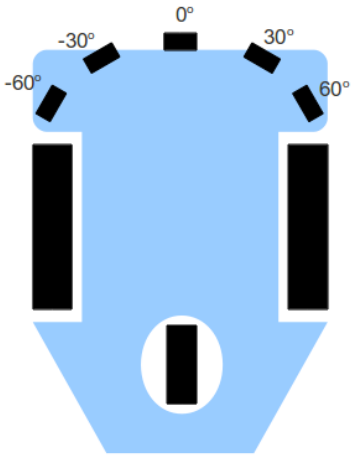

Figura 7: Disposição dos sensores infravermelhos e rodas.

Fonte: Autoria própria

A plataforma transporta as placas do sistema microcontrolado C8051F340DK, descrito na seção 2.2.3, o sistema embarcado TS-7260, descrito na seção 2.2.4 e a placa de roteamento, descrita na seção 2.2.5. O robô montado pode ser visualizado na figura 8, que ilustra a disposição dos sensores infravermelhos, rodas, baterias e placas no chassi.

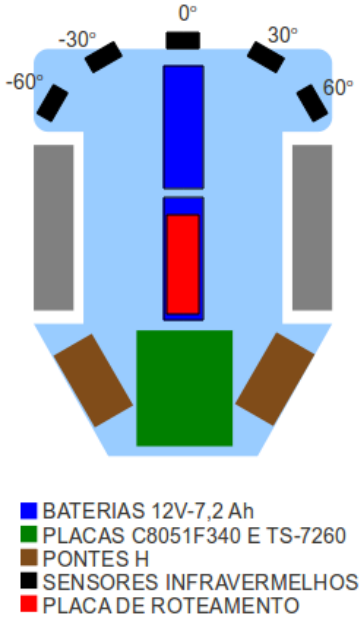

Figura 8: Robô Bellator montado.

Fonte: Autoria própria

2.3 *PULSE WIDTH MODULATION* (PWM)

PWM é a abreviação de *Pulse Width Modulation* ou Modulação por Largura de Pulso e pode ser aplicado no controle de potência de motores DC. Em (ELETRONICAORG, 2012), o circuito da figura 9, que é formado por um interruptor de ação muito rápida e uma carga que deve ser controlada, foi utilizado para explicar o princípio de funcionamento dessa tecnologia.

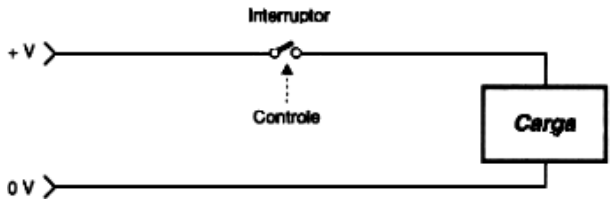

Figura 9: Interruptor de ação muito rápida e o controle de potência através de PWM.
Fonte: (ELETRONICAORG, 2012)

Nessa figura, quando o interruptor é aberto, não há corrente na carga e a potência aplicada é nula e, no instante em que o interruptor é fechado, a carga recebe a tensão total da fonte e a potência aplicada é máxima. Controlando a abertura e fechamento da chave, pode-se determinar um nível de potência intermediário aplicado à carga. Desse modo, se a chave permanecer aberta por um instante de tempo t_1 e fechada por um instante de tempo t_2, sendo t_1 igual a t_2, então a potência média aplicada será de 50%, conforme ilustra a figura 10.

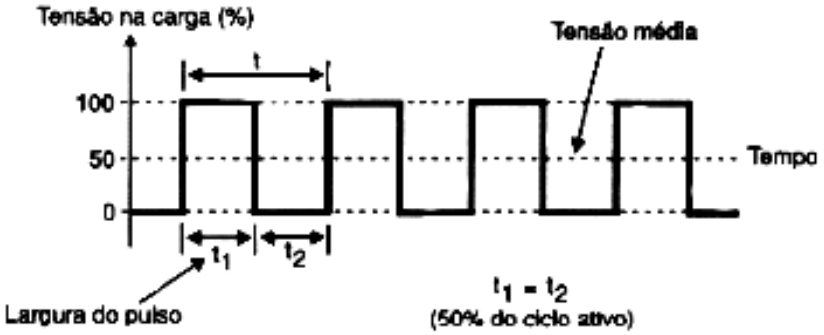

Figura 10: Situação na qual a potência média aplicada à carga é 50% da potência máxima.
Fonte: (ELETRONICAORG, 2012)

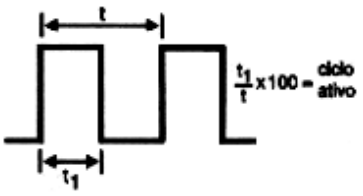

Figura 11: Ilustração do ciclo ativo do PWM.
Fonte: (ELETRONICAORG, 2012)

A soma dos intervalos t_1 e t_2 define o período t do sinal de PWM e variando-se o intervalo t_1, tempo pelo qual a chave permanece fechada, define-se a largura de pulso do ciclo ativo, ilustrado na figura 11. A equação 2 determina o ciclo ativo do PWM, que é utilizada para controlar a potência média aplicada a uma carga. Assim, quando o ciclo ativo do sinal for variado, conforme a figura 12, modifica-se a potência média aplicada à carga.

$$d = 100 \times \frac{t_1}{t} \tag{2}$$

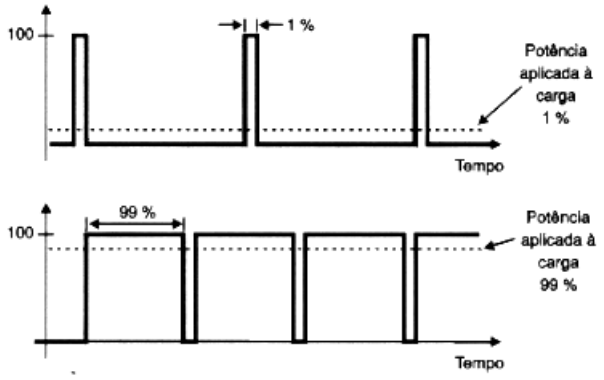

Figura 12: Variação do ciclo ativo do PWM e controle da potência.
Fonte: (ELETRONICAORG, 2012)

2.4 LÓGICA FUZZY

Esta seção descreve os conceitos fundamentais utilizados pela equipe para o entendimento e implementação do algoritmo de navegação *fuzzy*, descrito em detalhes na seção 3.2.1.

2.4.1 Conjuntos *Fuzzy*

A teoria de conjuntos *fuzzy* foi elaborada inicialmente por Lofti Zadeh (ZADEH, 1965), visando explorar a possibilidade de criar um novo critério de afiliação à conjuntos. Na teoria clássica de conjuntos, um elemento pode apenas pertencer ou não a um conjunto, sendo impossível um nível de pertinência parcial. Já em um conjunto *fuzzy*, isto torna-se possível. Um conjunto *fuzzy* pode ser definido por um conjunto de pares ordenados com o elemento e sua pertinência ao conjunto *fuzzy*. Seja F um conjunto *fuzzy* e X um conjunto de objetos arbitrários, tem-se:

$$F = \{(x, f(x)), x \in X\}, f(x) \in [0,1] \tag{3}$$

Assim sendo, considere um conjunto "A" simples que contenha tudo o que tem sabor doce. Neste conjunto uma barra de chocolate é doce da mesma forma que cana de açucar, visto que ambos pertencem ao conjunto, ou seja: (barra de chocolate) $\in$ A e (cana de açucar) $\in$ A. Em um conjunto *fuzzy* B que contemple tudo o que tem sabor doce, torna-se possível atribuir um nível de afiliação ao conjunto através de uma função de pertinência f permitindo dizer que, por exemplo, a cana de açucar é doce com nível de pertinência 1, enquanto que a barra de chocolate é doce com nível de pertinência 0.8, ou seja:

((cana de açucar),f(cana de açucar)) $\in$ B, f(cana de açucar) = 1
((barra de chocolate),f(barra de chocolate)) $\in$ B, f(barra de chocolate) = 0.8

Estas definições são mais próximas à forma como a cognição e intuição humana funcionam, frequentemente utilizando palavras como "mais", "muito", "pouco", entre outras, para definir graus de pertinência a conjuntos de uma forma subjetiva.

2.4.2 Variável Linguística

Uma aplicação direta de conjuntos *fuzzy* é a definição de variáveis linguísticas (PEDRYCZ; GOMIDE, 2007). Considerando que uma variável x pode assumir um valor qualquer dentro de um dado conjunto A, pode-se definir uma variável linguística como uma variável cujo conjunto A de valores possíveis é um conjunto de termos linguísticos, tais como: alto, baixo, curto, longo, entre outros. Pode-se estender este conceito associando cada termo linguístico possível de uma variável linguística a um conjunto *fuzzy*.

Para entender esta definição, considere a variável linguística Temperatura (T) composta pelos termos linguísticos frio, morno e quente:

$$T = \{frio, morno, quente\} \tag{4}$$

Considere também os seguintes conjuntos *fuzzy*:

$$\begin{aligned} F &= \{(t, f(t)), t \in \mathbb{R}\} \\ M &= \{(t, m(t)), t \in \mathbb{R}\} \\ Q &= \{(t, q(t)), t \in \mathbb{R}\} \end{aligned} \tag{5}$$

sendo *f(t), m(t) e q(t)* funções de pertinência, respectivamente, aos conjuntos *fuzzy F, M e Q*, com valores de pertinência pertencentes ao intervalo [0,1] e *t* uma variável representando a temperatura em um material qualquer. Considere ainda a associação dos conjuntos *fuzzy F, M e Q* aos termos linguísticos frio, morno e quente, respectivamente. Neste cenário, um dado valor para a variável *t* pode ser traduzido em um valor equivalente para a variável linguística *T*, dependendo apenas da definição das funções de pertinência *f(t), m(t) e q(t)*.

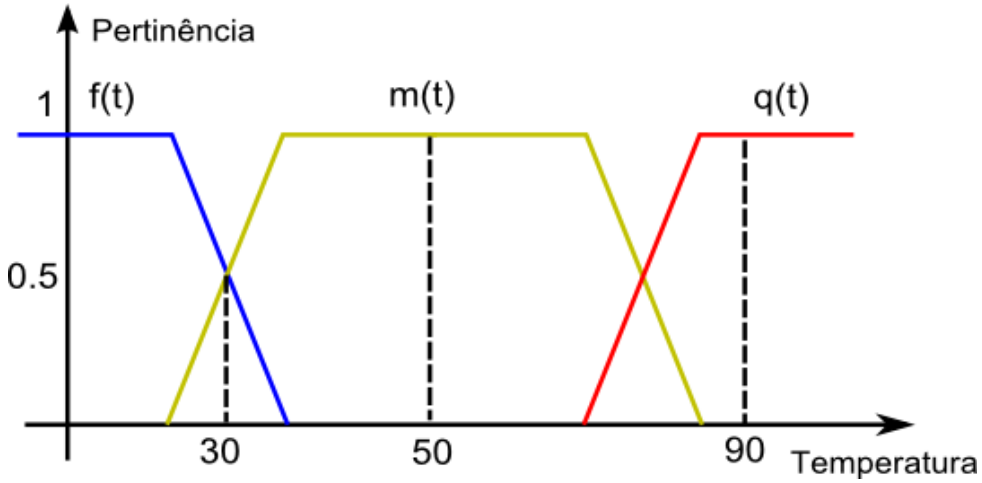

Figura 13: Funções de Pertinência f(t), m(t), q(t).

Se considerarmos, por exemplo, a definição gráfica para as funções *f(t), m(t) e q(t)* dada na figura 13, e três valores de temperatura, *t1 = 30, t2 = 50 e t3 = 90*, temos que os valores correspondentes de temperatura para a variável linguística T são T1 = (0.5 frio, 0.5 morno, 0.0 quente), T2 = (0.0 frio, 1.0 morno, 0.0 quente) e T3 = (0.0 frio, 0.0 morno, 1.0 quente), respectivamente. Ou seja, informalmente, a temperatura t1 representa que o material está "meio frio" e "meio morno" , a temperatura t2 indica que está "morno" e a temperatura t3, por sua vez, "quente" . Este processo de conversão para uma variável linguística é comumente chamado de "fuzzificação".

2.4.3 Controle *Fuzzy*

Após definidas as variáveis linguísticas, conjuntos *fuzzy* e suas funções de pertinência, descritas na seção 2.4.2, pode-se construir um controle *fuzzy* baseado em um conjunto de regras de inferência.

Regras de inferência sobre conjuntos *fuzzy* podem ser categorizadas como uma generalização do *modus ponens* binário. Em lógica binária, dada a regra "se X então Y", onde X e Y são variáveis binárias, a partir do momento que a premissa, representada pela variável X, assume valor lógico verdadeiro, a conclusão, dada por Y, é verdadeira também. Em lógica *fuzzy*, o mesmo raciocínio é valido, porém X e Y são variáveis linguísticas, e as regras de inferência são definidas a partir dos valores que estas variáveis linguísticas podem assumir, permitindo inclusive ativação parcial de regras de inferência. Estendendo o exemplo das temperaturas, apresentado na seção 2.4.2, e supondo que seja necessário controlar a velocidade de um *cooler* de processador, de acordo com a temperatura em que este se encontra, pode-se utilizar um sistema de controle *fuzzy*. As regras de inferência para este controle podem ser, por exemplo:

Se *morno* então *médio*
Se *quente* então *forte*

Sendo "médio" e "forte" valores possíveis da variável linguística Velocidade (V), que controla a velocidade do *cooler*, com "médio" correspondendo a função de pertinência vm e "forte" correspondendo a vf, apresentados na figura 14:

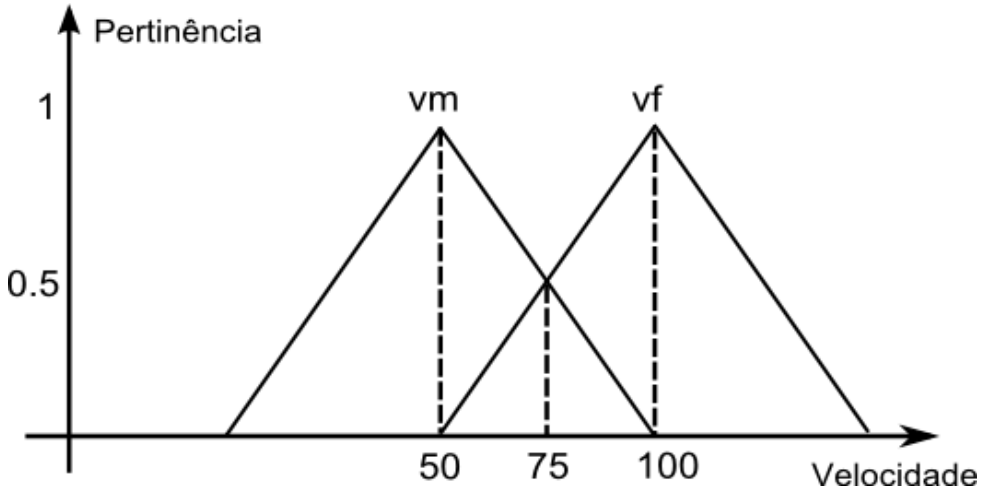

Figura 14: Funções de Pertinência vm e vf.

De acordo com estas regras de inferência, se a temperatura *fuzzificada* for inteiramente fria, nenhuma regra será ativada e a velocidade do *cooler* será nula. Porém, se a temperatura

for maior que a mínima necessária para começar a ser classificada como morna, haverá ativação integral ou parcial de uma ou ambas as regras de inferência. Neste caso, é necessário determinar o grau de ativação de cada uma das regras e produzir uma saída que contemple estes graus de ativação, que é o processo inverso à *fuzzificação*, a *defuzzificação*. Um destes métodos é a média do máximo, que consiste da média ponderada dos máximos de cada valor *fuzzy* de saída, com os pesos correspondendo às ativações das regras de inferência. Considerando novamente o exemplo do controle de velocidade de um *cooler* e considerando que, em um determinado momento, a temperatura está "meio morno" e "meio quente" , ou seja, 0.5 de pertinência à classe "morno" e a classe "quente" , ambas as regras serão ativadas igualmente e a velocidade do *cooler* será:

$$v = \frac{(0.5 * 50 + 0.5 * 100)}{1} = 75 \tag{6}$$

2.4.4 Considerações

O conceito de imprecisão introduzido pela Lógica Fuzzy permite a modelagem de sistemas com problemas de decisão cujas variáveis são dinâmicas. O problema de navegação robótica é altamente dinâmico, pois as decisões são tomadas sob a influência de vários sensores simultaneamente, todos passíveis de ruído, e a abordagem Fuzzy é uma opção plausível para tratá-lo.

2.5 *EVENT-DRIVEN FUZZY CONGNITIVE MAPS* (ED-FCM)

Esta seção tem como objetivos: explicar o que são mapas cognitivos *fuzzy*, também conhecidos como FCM (*Fuzzy Cognitive Maps*), abordando o conceito, a estrutura, as propriedades e vantagens desse modelo; apresentar os passos para contrução de um FCM; apresentar exemplos, que descrevem o uso dessa abordagem em situações reais. Além disso, visa a apresentação do ED-FCM (*Event-Driven Fuzzy Cognitive Maps*), que é uma modificação do FCM para suportar a atualização dos pesos dinamicamente.

O modelo FCM é abordado na tese de doutorado de (MENDONÇA, 2011). Mapas cognitivos são diagramas que representam ligações entre palavras, idéias, tarefas ou outros itens ligados a um conceito central, dispostos radialmente, intuitivamente e de acordo com a importância de cada conceito. Crenças ou afirmações a respeito de um domínio de conhecimento limitado são expressas por palavras ou expressões linguísticas interligadas por relações de causa e efeito, que possibilitam prever as consequências que essa organização implica ao universo representado. O mapa cognitivo *fuzzy* é gerado quando se agrega à essa estrutura a incerteza característica da lógica Fuzzy.

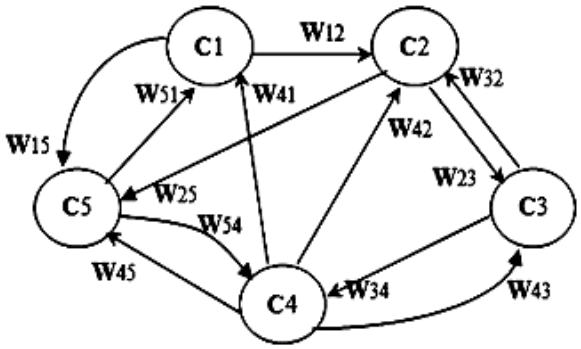

Figura 15: Exemplo de um FCM (grafo).
Fonte: (STYLIOS; GROUMPOS, 2000)

A estrutura de um FCM é um grafo direcionado, como o exemplo da figura 15, em que os valores numéricos são variáveis ou conjuntos *fuzzy*, os "nós" são conceitos linguísticos, representados por conjuntos *fuzzy* e cada "nó" é associado a outros nós através de conexões (relacionamentos), a cada qual está associado um peso numérico, que representa a variável *fuzzy* relacionada ao nível de causalidade entre os conceitos. De acordo com (MENDONÇA; ARRUDA; NEVES, 2011), um FCM suporta diversos tipos de conceitos e relacionamentos:

- Conceito de nível: Esse conceito pode pode ser representado por um valor absoluto;
- Conceito de variação: Esse tipo de conceito representa a variação de um valor no tempo;
- Conceitos de entradas: Esses conceitos recebem um valor de entrada e podem interagir com outros conceitos;
- Conceitos de saída ou de decisão: Esses conceitos representam o resultado das inferências do FCM e não interagem com outros conceitos;
- Relações causais: Essas conexões representam as relações de causa e efeito entre os conceitos e são calculadas através da matriz de pesos (matriz W, ver exemplo na equação 7);
- Declarações condicionais: Esses elementos são as relações causais expressas na forma de regras *se-então* e são atualizadas temporalmente.

Na figura 15, os conceitos (C1 a C5) podem ser atualizados através da interação com outros conceitos por meio das relações causais ($w_{i,j}$) e com seu próprio valor. A matriz na equação 7

representa o peso das relações causais entre os conceitos e podem ser atualizados através da equação 8. Esta descreve a evolução do FCM, na qual *j* é o contador das iterações, *n* é o número de nós do grafo, W_{ji} é o peso do arco que conecta o conceito C_j ao conceito C_i, A_i e $A_i^{anterior}$ são o valor do conceito C_i na iteração atual e anterior, respectivamente, e a função f (9) é uma função do tipo sigmóide.

$$w_{i,j} = \begin{pmatrix} 0 & w_{12} & 0 & 0 & w_{15} \\ 0 & 0 & w_{23} & 0 & w_{25} \\ 0 & w_{32} & 0 & w_{34} & 0 \\ w_{41} & 0 & w_{43} & 0 & w_{45} \\ w_{51} & 0 & 0 & w_{54} & 0 \end{pmatrix} \tag{7}$$

$$A_i = f(\sum_{\substack{j=1 \\ j \neq i}}^{n} A_j \times W_{ji}) + A_i^{anterior} \tag{8}$$

$$f(x) = \frac{1}{1 + e^{-\lambda x}} \tag{9}$$

Em (KOSKO, 1986), são apresentados os seguintes passos para construção de um FCM clássico:

- Passo 1 - Identificação dos conceitos e das suas interconexões ou relações determinando a natureza (positiva, negativa ou neutra) das relações causais entre conceitos;
- Passo 2 - Aquisição de dados iniciais, através de ponderação de opinião de especialistas e ou análise do sistema de equações, quando se conhece o modelo matemático;
- Passo 3 - Apresentação dos dados referentes à opinião dos diversos especialistas a um sistema lógico *fuzzy* que tem como saída os valores dos pesos do FCM;
- Passo 4 - Tratamento da informação, adaptação e ou otimização do FCM inicialmente proposto, ajustando suas respostas às saídas desejadas;
- Passo 5 - Validação do FCM ajustado nas condições de operação do sistema ou processo modelado.

Um FCM apresenta as propriedades de elasticidade e estabilidade, sendo que a elasticidade, ou auto-organização, é a capacidade de reforçar ou enfraquecer o peso das relações causais e a estabilidade é a capacidade de o mapa evoluir, estabilizando-se em um ponto fixo ou após

um número máximo de iterações. Uma vantagem do FCM é a modularidade, a qual permite que um problema complexo seja representado por vários mapas modulares e outra vantagem é que os pesos das relações causais e dos conceitos podem ser obtidos via treinamento a partir dos dados históricos do sistema ou através de um algoritmo adaptativo, que atualiza os pesos constantemente.

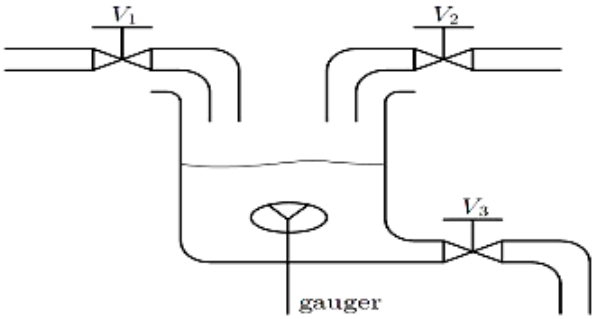

Figura 16: Aplicação do FCM em processo industrial.
Fonte: (STYLIOS; GROUMPOS, 2000)

Em (STYLIOS; GROUMPOS, 2000) os mapas cognitivos *fuzzy* são aplicados no controle de processos industriais. Um exemplo de aplicação é mostrado na figura 16, na qual é ilustrado um tanque com duas válvulas de entrada (V1 e V2) para diferentes tipos de líquidos, um misturador, uma válvula de saída (V3) para o líquido misturado e um medidor de massa específica (G) que mede a quantidade de líquido produzida. As válvulas V1 e V2 introduzem dois líquidos diferentes. Durante a mistura, o medidor de massa específica verifica quando o produto atingiu o ponto adequado e, desse modo, a válvula V3 é ativada e o produto da mistura é esvaziado. Analisando-se o problema, os seguintes conceitos podem ser definidos:

- Conceito 1: Volume de líquido no tanque, o qual depende do estado das válvulas V1, V2 e V3;
- Conceito 2: Estado da válvula 1 (fechada, aberta ou parcialmente aberta);
- Conceito 3: Estado da válvula 2 (fechada, aberta ou parcialmente aberta);
- Conceito 4: Estado da válvula 3 (fechada, aberta ou parcialmente aberta);
- Conceito 5: Valor de massa específica do líquido medido pelo sensor G.

O controlador do processo deve manter as variáveis V e G, sendo V o volume e G a massa específica do produto no tanque, dentro das faixas de operação $[V_{min}, V_{max}]$ e $[G_{min}, G_{max}]$, respectivamente.

Interligando-se os conceitos através de relações de causa e efeito, o FCM da figura 17 foi construído.

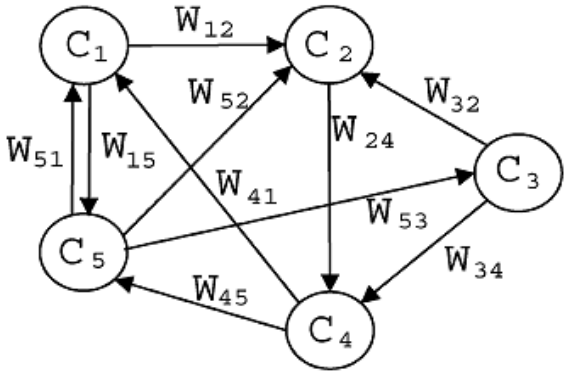

Figura 17: FCM do controlador.
Fonte: (STYLIOS; GROUMPOS, 2000)

Analisando-se o conhecimento dos especialistas, os pesos das relações são dados pelas inequações 10 a 17.

$$-0,50 < w_{12} < 0,30 \tag{10}$$

$$-0,40 < w_{13} < 0,20 \tag{11}$$

$$0,20 < w_{15} < 0,40 \tag{12}$$

$$0,30 < w_{21} < 0,40 \tag{13}$$

$$0,40 > w_{31} < 0,50 \tag{14}$$

$$-1,0 < w_{41} < 0,80 \tag{15}$$

$$0,50 < w_{52} < 0,70 \tag{16}$$

$$0,30 < w_{54} < 0,40 \tag{17}$$

O controlador do processo foi executado e, após a estabilização, obtiveram-se os pesos da matriz 18 e os valores dos conceitos da matriz 19. Os limites de V e G são reajustados para os valores correspondentes às equações 20 e 21, respectivamente, correspondendo ao ponto de

operação desejado.

$$W^{inicial} = \begin{pmatrix} 0,00 & -0,40 & -0,25 & 0,00 & 0,30 \\ 0,36 & 0,00 & 0,00 & 0,00 & w0,00 \\ 0,45 & 0,00 & 0,00 & 0,00 & 0,00 \\ -0,90 & 0,00 & 0,00 & 0,00 & 0,00 \\ 0,00 & 0,60 & 0,00 & 0,30 & 0,00 \end{pmatrix} \tag{18}$$

$$A^{inicial} = \begin{pmatrix} 0,10 & 0,45 & 0,39 & 0,04 & 0,01 \end{pmatrix} \tag{19}$$

$$0,68 < V < 0,70 \tag{20}$$

$$0,78 < G < 0,85 \tag{21}$$

Nesse exemplo, a estabilização (ou sintonia) foi realizada através de três métodos: RNA (Rede Neural Artificial), AG (Algoritmo Genético) e PSO (Particle Swarm Optimization ou Otimização por Enxame de Partículas).

Outra aplicação é descrita no artigo (MENDONÇA; ARRUDA; NEVES, 2011), na qual a abordagem FCM é empregada em navegação robótica. Nesse artigo, um modelo de FCM novo é implementado para suportar as condições dinâmicas dos sistemas de navegação, nas quais os valores das relações causais são modificados dinamicamente através da ocorrência de eventos especiais. Os autores do artigo chamaram esse modelo de ED-FCM (*Event-Driven Fuzzy Cognitive Map*).

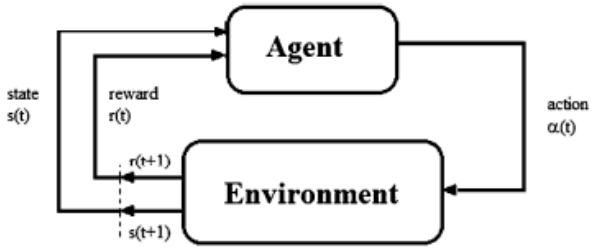

Figura 18: Algoritmo de aprendizado por reforço.
Fonte: (MENDONÇA; ARRUDA; NEVES, 2011)

O ajuste dos pesos das relações causais é efetuado por um algoritmo de aprendizado por

reforço, conforme ilustra a figura 18, e permite que o robô (agente) aprenda diretamente através de sua interação com o ambiente. A cada instante de tempo t, o agente estabelece, por meio de seus sensores, um estado $s(t)$ e, de acordo com suas regras, determina uma ação $\alpha(t)$ a ser efetuada pelos atuadores. Essa ação causa uma transição para o estado $s(t+1)$ e o ambiente retorna uma medida de reforço $r(t+1)$, que pode ser uma recompensa (caso a ação seja boa) ou uma punição (caso a ação seja ruim).

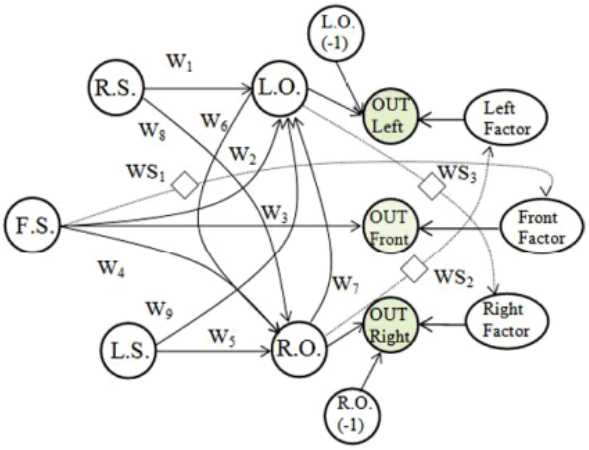

Figura 19: ED-FCM do comportamento reativo do robô.
Fonte: (MENDONÇA; ARRUDA; NEVES, 2011)

O ED-FCM descreve o comportamento reativo do robô (figura 19), no qual a leitura dos sensores de distância (esquerdo, frontal e direito) levam a uma ação imediata que interfere no movimento. Os conceitos RS, FS e LS representam as leituras dos sensores, os conceitos LO e RO representam as decisões de virar à esquerda ou virar à direita, respectivamente, decisões anteriores, representadas pelos conceitos LO(-1) e RO(-1), exercem influência sobre as decisões atuais e a saída do algoritmo é representada pelos conceitos *Out Left*, *Out Front* e *Out Right*. As relações causais do mapa são descritas na tabela 1 e as regras a seguir determinam o comportamento do mapa:

1. SE a intensidade do sensor frontal (FS) for maior que um limiar médio ENTÃO W_{lim} aplicado para computar o relacionamento w_3 é o valor máximo de WF_{max};

2. SE a intensidade do sensor frontal (FS) for menor que um limiar mínimo ENTÃO W_{lim} aplicado para computar o relacionamento w_3 é o valor mínimo de WF_{min};

3. SE a intensidade do sensor direito (RS) for maior que um limiar médio ENTÃO W_{lim} aplicado para computar o relacionamento w_1 é o valor máximo de WR_{max};

4. SE a intensidade do sensor direito (RS) for menor que um limiar mínimo ENTÃO W_{lim} aplicado para computar o relacionamento w_1 é o valor mínimo de WR_{min};

5. SE a intensidade do sensor esquerdo (LS) for maior que um limiar médio ENTÃO W_{lim} aplicado para computar o relacionamento w_5 é o valor máximo de WL_{max};

6. SE a intensidade do sensor direito (LS) for menor que um limiar mínimo ENTÃO W_{lim} aplicado para computar o relacionamento w_5 é o valor mínimo de WL_{min}.

Tabela 1: Relações causais do controlador do robô.

Relação causal	Descrição	Efeito	Intensidade
w_1	Sensor direito (RS) influencia a saída esquerda (LO)	Positivo	Forte
w_2	Sensor frontal (FS) influencia a saída esquerda (LO)	Positivo	Médio
w_3	Sensor frontal (FS) influencia a saída frontal (FO)	Positivo	Forte
w_4	Sensor frontal (FS) influencia a saída direita (RO)	Positivo	Médio
w_5	Sensor esquerdo (LS) influencia a saída direita (RO)	Positivo	Forte
w_6	Saída esquerda (LO) influencia a saída direita (RO)	Negativo	Fraco
w_7	Saída direita (RO) influencia a saída esquerda (LO)	Negativo	Fraco
w_8	Sensor direito (RS) influencia a saída direita (RO)	Negativo	Fraco
w_9	Sensor esquerdo (LS) influencia a saída esquerda (LO)	Negativo	Fraco

Fonte: (MENDONÇA; ARRUDA; NEVES, 2011)

Essas regras determinam a política de mudança de estados do mapa e os pesos dos relacionamentos são responsáveis pelas decisões de o robô virar à esquerda, acelerar ou virar à direita. Nesse contexto, o valor atual desses pesos depende da diferença entre os valores anteriores e o valor máximo admissível ponderado por um fator γ. O incremento dos pesos também leva em conta o valor da recompensa ou punição (r) e de um fator de aprendizagem α, os quais estão associados ao algoritmo de aprendizado por reforço escolhido (equação 22).

$$w_i(k) = w_i(k-1) + \alpha \times [r + \gamma \times W_{lim} - w_i(k-1)] \tag{22}$$

Por fim, o artigo descreve os resultados nos quais o robô, em simulação, foi capaz de desviar obstáculos à direita e à esquerda do mesmo ao longo da trajetória.

2.5.1 Considerações

O ED-FCM representa a solução de um problema em termos de conceitos e relações causais, podendo ser empregado em controladores de processos industriais ou no controle de robôs autônomos. O problema do desvio de obstáculos em navegação robótica pôde ser modelado,

conforme o exemplo apresentado (figura 19), através de três conceitos de entrada (RS, FS e LS), dois conceitos de decisão (RO e LO), três conceitos de saída (Right Out, Front Out e Left Out), relações causais, regras *SE-ENTÃO* e um algoritmo de aprendizado. O controlador proposto permitiu que o robô desviasse obstáculos reagindo à leitura de sensores que medem a distância de objetos posicionados à esquerda e à direita do mesmo. A ocorrência de eventos especiais permitiram modificar o estado do ED-FCM e alterar os pesos das relações causais de forma dinâmica. Em (MENDONÇA; ARRUDA; NEVES, 2011), esse modelo foi utilizado no problema de navegação e, por isso, concluiu-se que essa abordagem é adequada para esse trabalho.

3 DESENVOLVIMENTO

Este capítulo contém, em detalhes, o desenvolvimento do trabalho realizado pela equipe, dividido nas seguintes seções:

- Reconstrução da Plataforma Robótica Bellator: Todos os passos realizados para reconstruir e adaptar a plataforma robótica para utilização no projeto.
- Algoritmos de Navegação: Detalha a implementação dos algoritmos de navegação *fuzzy* e ED-FCM, de acordo com a revisão bibliográfica apresentada no capítulo 2.
- Testes e Análise de Resultados: Apresenta a metodologia, elaboração dos testes dos algoritmos e a análise dos resultados obtidos para ambos os algoritmos de navegação.
- Considerações: Conclusão do capítulo e considerações sobre o desenvolvimento do projeto.

3.1 RECONSTRUÇÃO DA PLATAFORMA ROBÓTICA BELLATOR

A reconstrução e adaptação da plataforma robótica Bellator, bem como a documentação da mesma para facilitar utilização em trabalhos futuros, são objetivos deste trabalho de conclusão de curso. Assim sendo, esta seção irá descrever de forma detalhada os passos realizados pela equipe no processo de reconstrução, incluindo os testes dos componentes recebidos no início do projeto, elaboração e instalação de novos componentes de *hardware* para o robô, documentação de componentes de *software* necessários para o funcionamento da plataforma robótica e para possibilitar a execução autônoma de algoritmos de navegação na mesma.

3.1.1 Teste dos Componentes

Após o recebimento do robô, foram realizados testes para garantir a funcionalidade dos componentes recebidos, já que o robô estava com suas peças empilhadas numa caixa e não era possível confiar no funcionamento adequado de nenhum dos componentes, além de que

a falha de alguns componentes implicaria na impossibilidade de continuar o projeto ou em atrasos significativos. Estes testes também foram necessários para determinar de forma mais precisa o que poderia ser reaproveitado do projeto Bellator. De acordo com a documentação do projeto Bellator (MARIN et al., 2010), o robô deveria ser capaz de funcionar como um sistema controlado remotamente. Como o objetivo deste projeto não envolve controlar o robô remotamente, foi testada apenas a camada de baixo nível.

O primeiro passo da etapa de testes foi verificar o funcionamento da placa C8051F340, peça fundamental para o desenvolvimento do projeto, que apresentou o funcionamento adequado, gerando os PWMs, cujo conceito é descrito na seção 2.3, dos motores conforme necessário (visualizados no osciloscópio), e realizando a leitura dos sensores e conversão A/D conforme esperado. Em seguida, foram iniciados os testes utilizando a placa de roteamento já existente, que apresentou defeito. Após alguns testes, foi constatado que a placa havia sido desconfigurada, sendo que várias soldas foram removidas e o circuito em si estava alterado. Então, a equipe reorganizou a placa, realizou novos testes, mas não obteve sucesso. Foi então verificado que o regulador de tensão não estava funcionando. Este foi substituído e a placa finalmente funcionou conforme esperado.

Com a placa de roteamento antiga funcionando, foi possível realizar o teste dos motores, utilizando os PWMs gerados pelo microcontrolador C8051F340 (entrada do *buffer* da placa de roteamento). Nesse teste, não ocorreram problemas, os motores funcionaram conforme esperado.

Das duas baterias inicialmente disponíveis, uma não estava funcionando conforme a especificação, o que levou a equipe a adquirir uma nova bateria de 12V para reposição da bateria danificada.

Com todos os componentes acima citados testados, o que faltava para completar os testes da camada de baixo nível era apenas o teste dos *encoders*. Esta etapa foi uma das mais difíceis, pois a equipe não tinha informação nem do modelo do *encoder*. Depois de muito procurar, foi encontrado um *datasheet* de um *encoder* equivalente ao presente no robô, *datasheet* este que foi fundamental para determinar como alimentar e testar o *encoder*. Em posse da informação de como usar o *encoder*, o teste foi realizado tanto para o *encoder* esquerdo como o direito. O *encoder* direito funcionou normalmente, porém o esquerdo não. Então, a equipe percebeu que a solda dos fios do *encoder* não estava boa. Após refazer as soldas, o *encoder* esquerdo foi testado novamente e funcionou.

Tendo realizado os testes dos componentes mais críticos, o passo seguinte foi tentar utilizar o PC Embarcado VIA EPIA ME6000. Após muitas tentativas falhas e busca por informações

sem resultados, a equipe optou por não utilizar este componente, já que sua documentação era escassa e o tempo perdido na tentativa de utilizá-la já estava acima do planejado (praticamente um mês foi gasto em tentativas frustradas). O PC Embarcado foi substituído pela placa TS-7260, que possui uma documentação muito melhor, poder de processamento superior, e funcionou nos primeiros testes. Esta substituição não gerou custos para o projeto, pois a placa TS-7260 já havia sido adquirida para um projeto anterior e não estava sendo utilizada, e foi então disponibilizada à equipe pelo professor orientador.

3.1.2 Levantamento da curva dos sensores

Esta seção apresenta como foi levantada a curva dos sensores, ponto fundamental para utilizar os dados de conversão, transformando-os para a distância em centímetros, que então pode ser passada aos algoritmos de navegação.

Primeiramente, foi montada uma tabela, contendo os valores de conversão para cada distância, de 5 em 5cm, começando em 15cm até 115cm, totalizando 21 medidas. Os valores obtidos estão na tabela 2.

Tabela 2: Valores de conversão dos sensores x distância

Valor da conversão	Distância (cm)
207	15
191	20
173	25
150	30
134	35
120	40
110	45
100	50
91	55
86	60
81	65
75	70
71	75
68	80
66	85
65	90
62	95
59	100
56	105
54	110
50	115

A partir destes valores, foi realizada uma interpolação polinomial para obtenção da equação aproximada da curva. O grau do polinômio interpolador escolhido foi o de grau 4 (graus inferiores traziam uma aproximação com erro muito elevado, enquanto graus superiores praticamente não alteravam o erro), que aproximou-se bem da curva real (medida), conforme é possível visualizar na figura 20, onde o eixo das abscissas contém os valores de conversão, e o eixo das ordenadas a distância correspondente. Segue a equação da interpolação polinomial:

$$y = 3.6404 \cdot 10^{-7}x^4 - 2.4435 \cdot 10^{-4}x^3 + 6.0732 \cdot 10^{-2}x^2 - 6.8962x + 339.361 \qquad (23)$$

onde y representa a distância em centímetros e x o valor da conversão analógica/digital.

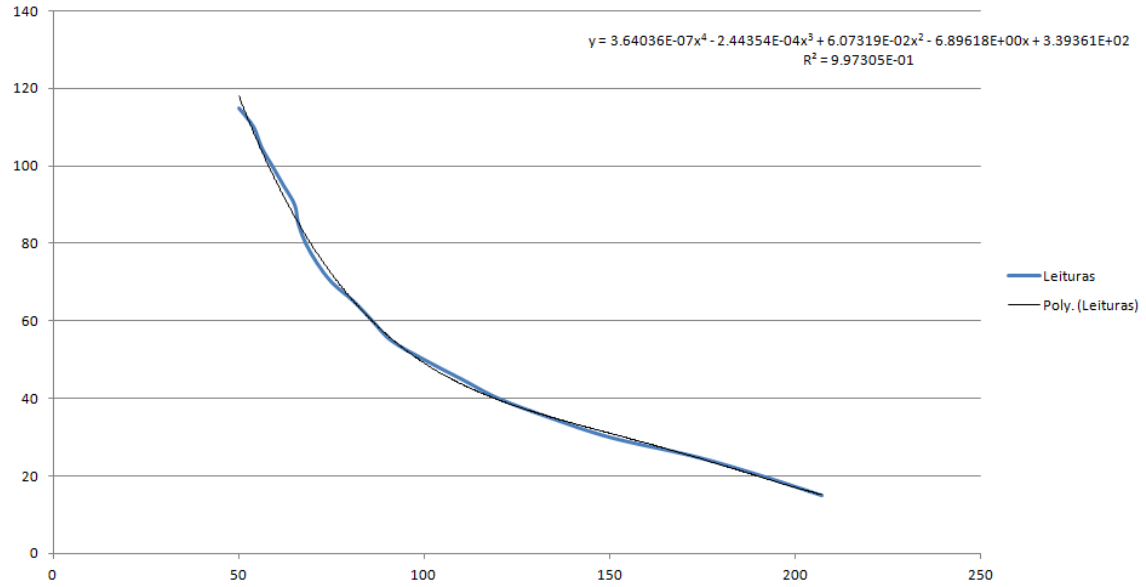

Figura 20: Curva real dos sensores e interpolação polinomial

Fonte: Autoria própria

Com esta equação, só foi necessário iterar pelos valores *x* possíveis de conversão (0 a 255, pois a conversão analógica/digital usa 8 bits) e montar a tabela para acessar com os valores de conversão e obter a distância em centímetros. Para exemplificar, suponha que a conversão analógica digital de algum sensor resultou num valor de $x = 120$. Para obter a distância correspondente em centímetros, basta acessar a tabela na posição 120, e o valor armazenado nesta posição é a distância em centímetros, que foi obtido diretamente da substituição de *x* na equação, ou seja, sendo $y = f(x)$, a distância em centímetros para a conversão analógica digital com valor 120 é f(120). A tabela foi montada simplesmente para evitar recálculos através da equação, e foi inserida no código da placa TS-7260, que será explicado em detalhes na seção 3.1.9.

3.1.3 Repositório de trabalho

Nesta seção será apresentado um dos pontos mais importantes do projeto: a ferramenta de controle de versão e armazenamento de arquivos *git*.

Para realizar o trabalho de forma mais organizada, garantir versionamento e sincronia dos arquivos relacionados ao projeto, foi utilizado um repositório *git* (GIT, 2012). Este repositório foi armazenado no GitHub (GITHUB, 2012a), site cujo propósito é disponibilizar uma *interface* para criação, administração e armazenamento remoto de repositórios *git*. Dentre os arquivos armazenados neste repositório, os principais são: códigos que foram utilizados neste projeto

(microcontrolador C8051F340, TS-7260, algoritmos Fuzzy e ED-FCM); *datasheets* dos componentes utilizados; diversas referências da monografia; o texto da monografia em si; figuras utilizadas na monografia. O repositório está disponível para cópia, podendo ser obtido através do seguinte comando:

```
git clone git@github.com:jcnborges/TCC.git
```

Para tal, é necessário antes configurar o *git* na máquina em que for utilizar, tarefa esta que pode ser realizada através do tutorial disponível no site do GitHub (GITHUB, 2012b). Após configuração e execução do comando acima, todos os arquivos armazenados no repositório serão copiados (em sua versão mais recente). Após configuração do *git* e execução do comando acima, todos os arquivos do repositório (na sua versão mais recente) serão copiados para a máquina em que o comando foi executado, e imediatamente disponíveis para alterações.

Esta seção apresentou a ferramenta *git*, utilizada para versionamento dos arquivos relacionados ao projeto, armazenados em um ambiente remoto, reduzindo significativamente os riscos de perda de trabalho e a consequente necessidade de retrabalho. Por este motivo, sua utilização foi peça fundamental para realização de um trabalho organizado e seguro. A equipe também considera que esta praticidade é fundamental para trabalhos futuros, sendo uma maneira muito simples e direta de obter os resultados deste trabalho para uma possível continuação do projeto.

3.1.4 Código do microcontrolador C8051F340

Esta seção apresentará as funções do código do microcontrolador, quais funções foram aproveitadas do projeto Bellator, quais foram modificadas e quais foram implementadas por completo.

O código que executa no microcontrolador C8051F340 tem diversas funcionalidades fundamentais do robô, sendo elas: leitura dos sensores de distância e conversão analógica/digital dos sinais dos mesmos, geração dos PWMs para ambos os motores, contagem dos pulsos dos *encoders*, recepção e envio de mensagens através da conexão serial.

A única funcionalidade que foi implementada no microcontrolador neste projeto foi o tratamento dos sinais dos *encoders* (para obter informações de odometria), informações estas essenciais para a execução dos algoritmos e consequente navegação autônoma. O código do projeto Bellator (MARIN et al., 2010) não continha o tratamento dos sinais dos *encoders* (os *encoders* não haviam sido utilizados no projeto Bellator). Portanto, foi necessário alterar o código do microcontrolador para tratar os sinais dos *encoders* e enviar informações de odome-

tria. Para tal tarefa, foram utilizadas duas interrupções externas da placa C8051F340 (uma para cada*encoder*). Nestas interrupções, cada pulso do *encoder* é contado (obviamente cada *encoder* possui seu contador separadamente). A informação de contagem é então enviada através da comunicação serial com a placa TS-7260 quando requerida pela mesma. Os *encoders* estão conectados à placa de roteamento, devido à necessidade de amplificação de seus sinais para utilização no microcontrolador. Na placa de roteamento, os sinais dos *encoders* são amplificados em um amplificador operacional LM324 (TEXASINSTRUMENTS,), e o sinal amplificado é então conectado ao Port 0 do microcontrolador.

A geração dos PWMs foi mantida como estava no projeto Bellator (MARIN et al., 2010), com 76 níveis possíveis de PWM (este número de níveis é resultado da forma como foi definida a frequencia da forma de onda resultante). Os PWMs são gerados através do Timer0 do microcontrolador C8051F340 e disponibilizados no Port 1 do mesmo, que é então conectado à placa de roteamento, onde os PWMs são utilizados como entrada para um *buffer* 74LS244 (PHILIPS,), cujas saídas são utilizadas como PWM para as pontes H dos motores. Este *buffer* é necessário pois as saídas do microcontrolador não possuem corrente suficiente para os motores.

A leitura dos sensores de distância, que é realizada através de uma varredura (ao fim de cada conversão, o próximo sensor é selecionado), teve seu período (este período inclui a conversão de todas as leituras) alterado (de 1s para 50ms). Esta mudança foi necessária para permitir que dados mais recentes sejam enviados ao robô, para este poder reagir de forma mais adequada ao ambiente (com o intervalo de 1s entre as conversões o robô demorava para desviar de obstáculos). Os sensores estão conectados à placa de roteamento, onde seus sinais de leitura são roteados para o Port 2 do microcontrolador, que, por sua vez, realiza as conversões A/D. Os valores obtidos são simplesmente a conversão do sinal de analógico para digital. Estes valores são enviados para a placa TS-7260 quando requeridos pela mesma, e somente na TS-7260 cada valor é utilizado para consultar a tabela (para obter o valor da distância em centímetros), obtida de acordo com o que foi descrito na seção 3.1.2, mapeando o valor da conversão para a distância em centímetros.

A parte de recepção e envio de mensagens do código do microcontrolador foi vastamente alterada, para adequação ao protocolo descrito na seção seguinte.

3.1.5 Protocolo de comunicação

Esta seção tem como objetivo explicar como funciona o protocolo de comunicação, que foi desenvolvido para padronizar a troca de mensagens entre o microcontrolador C8051F340 e a TS-7260.

O protocolo de comunicação define quais são os bytes para cada tipo de mensagem. Todas as mensagens, sejam elas recebidas ou enviadas pelo C8051F340, possuem o byte *END_CMD* para sinalizar o fim de um comando/mensagem. As mensagens reconhecidas pelo microcontrolador C8051F340 são as seguintes:

- **SYNC END_CMD**: quando o microcontrolador C8051F340 recebe esta mensagem, responde com as leituras mais recentes de cada sensor de distância, em seguida as leituras dos *encoders*;
- **LEFT_WHEEL valor END_CMD**: ao receber este comando, o microcontrolador utiliza o valor para definir o nível de PWM para a roda esquerda do robô. valor é representado por apenas um byte, onde o bit mais significativo indica o sentido de rotação da roda e os restantes a intensidade do PWM;
- **RIGHT_WHEEL valor END_CMD**: funcionamento idêntico ao comando LEFT_WHEEL, mas para a roda direita.

As mensagens enviadas pelo microcontrolador C8051F340 são apenas as respostas do comando SYNC:

- **OPTICAL_SENSOR_[0-5] valor END_CMD**: representa a leitura de cada sensor, onde valor é um byte, cuja faixa de variação é [0, 255].
- **ENCODER_[0-1] valor_high valor_low END_CMD**: representa a leitura de cada *encoder*, valor_high e valor_low juntos formam um inteiro de 16 bits que contém o valor da contagem do *encoder*.

3.1.6 Placa de Roteamento

A placa de roteamento é um componente fundamental do projeto, responsável por realizar a interligação entre a placa C8051F340 e os componentes de *hardware* do robô. A equipe constatou a necessidade deste componente devido aos seguintes fatores:

- Necessidade de alimentação dedicada para alguns componentes;
- Necessidade de tratamento dos sinais dos *encoders*;
- Roteamento das leituras de cada sensor para o pino de E/S correto da C8051F340;

- Necessidade de um *buffer* para o PWM;

- Melhor organização do robô.

Como descrito nas especificações do robô (seção 2.2), o robô Bellator possui, dentre outros componentes, cinco sensores, dois *encoders* e duas pontes H. Os cinco sensores e dois *encoders* necessitam de alimentação de aproximadamente 5 Volts para operação. Os *encoders* produzem um sinal que precisa ser amplificado antes da utilização, e as duas pontes H necessitam de um sinal de PWM de baixa impedância. Além disso, o consumo de corrente total destes componentes ultrapassa a capacidade de fornecimento de corrente da C8051F340. Medições durante os testes com os componentes foram realizadas, e foi constatado que cada sensor de distância utiliza cerca de 30mA de corrente, enquanto os *encoders* utilizam 20mA cada, totalizando quase 200mA para estes componentes.

Por fim, a dificuldade de conectar, de forma prática, todos os componentes do robô com a C8051F340 bem como a quantidade excessiva de fios resultante fez com que a equipe concluísse que a placa de roteamento é indispensável.

De acordo com os fatores descritos, a equipe construiu uma lista de requisitos para a placa de roteamento:

- Fornecer alimentação de aproximadamente 5 Volts DC;

- Capacidade de corrente suficiente;

- Conectores práticos para *interface* com a C8051F340 e o resto do robô;

- Fornecer um *buffer* para o sinal de PWM;

- Fornecer amplificação para o sinal do *encoder*.

Após a análise dos requisitos, a equipe iniciou o desenvolvimento de uma placa de circuito impresso para atender a todos os requisitos. A necessidade de uma alimentação de 5 Volts tornou essencial a utilização de um regulador de tensão, o LM317, que era o mesmo utilizado na versão antiga da placa, e suporta corrente de até 1.5A, valor suficiente para alimentar os componentes da placa. Também foi utilizado um *driver* de corrente (74LS244N) para o sinal de PWM (o sinal de saída do microcontrolador não possuía corrente suficiente para acionar as pontes H). Para a amplificação dos sinais dos *encoders* foi utilizado um amplificador operacional LM324N. O restante da placa consiste de pinos para conectar cabos *flat*, para a interação com a C8051F340, bem como pinos para conectores do *hardware* do robô Bellator. O diagrama

esquemático de tal placa foi produzido com o auxílio da versão gratuita, de uso exclusivamente não comercial do *software* Eagle (CADSOFT, 2012), versão 5.11.0, que mostrou-se suficiente para a elaboração da placa. O resultado pode ser visualizado na figura 21.

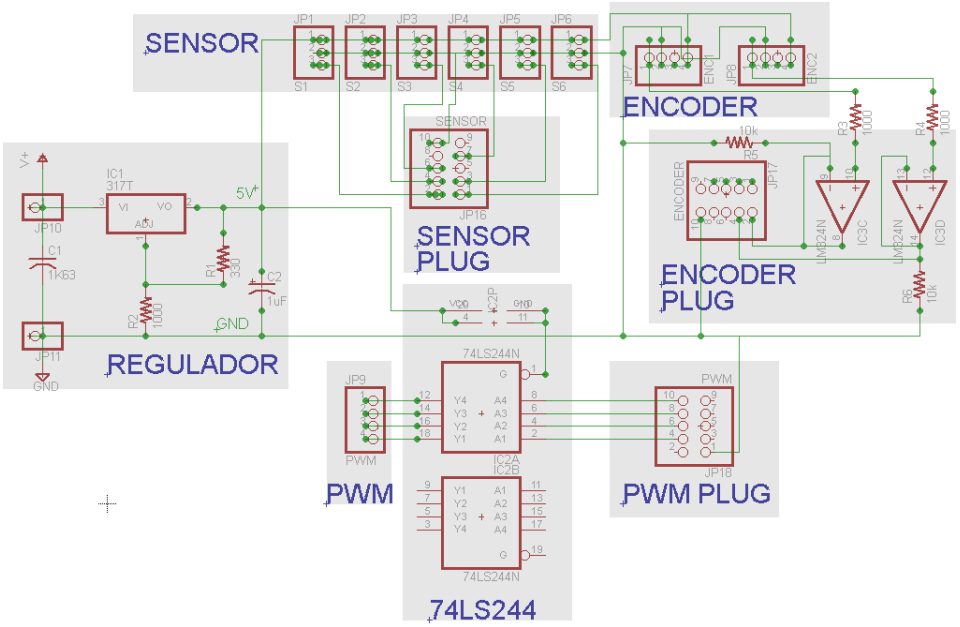

Figura 21: Diagrama esquemático da placa de roteamento.

Fonte: Autoria própria

De posse do diagrama esquemático apresentado na figura 21 a equipe realizou o roteamento de uma placa de circuito impresso que atenda ao mesmo diagrama esquemático, também utilizando o *software* Eagle.

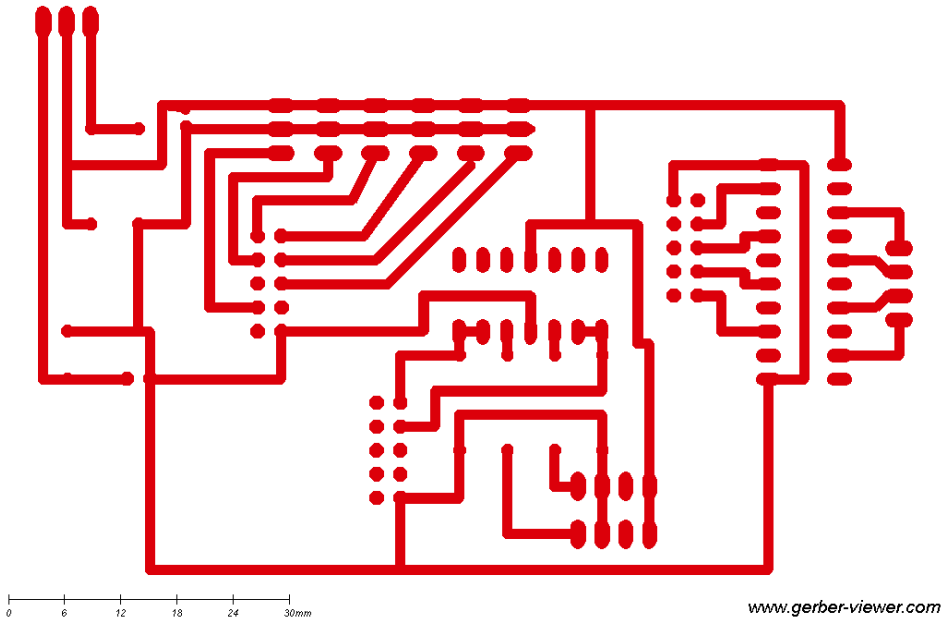

Figura 22: Roteamento final produzido pela equipe.

Fonte: Autoria própria

A placa de roteamento foi confeccionada manualmente em uma placa de circuito impresso. O desenho da placa foi impresso em uma folha de transparência A4 com uma impressora à laser. Esse desenho foi passado da transparência para a placa com o auxílio de um ferro de passar roupa e a placa foi corroída usando-se percloreto de ferro. A placa foi perfurada com broca e perfurador de placa e os componentes eletrônicos foram soldados com ferro de solda, estanho e pasta de solda. A figura 23 ilustra o resultado do processo de confecção da placa.

Figura 23: Resultado do processo de confecção da placa de roteamento.
Fonte: Autoria própria

A placa em utilização pode ser visualizada no Apêndice A, que mostra por meio de fotos como a placa de roteamento estava conectada ao restante do robô, ao final do projeto.

3.1.7 Placa TS-7260

Esta seção explicará como foi estruturado o *software* desenvolvido para a placa TS-7260, cuja função principal é realizar a *interface* entre o *software* do microcontrolador C8051F340 e o algoritmo de navegação (Fuzzy ou ED-FCM).

3.1.8 Configuração da TS-7260

O *software* da TS-7260 foi desenvolvido em C++, assim como os algoritmos de navegação, a serem explicados em detalhes na seção 3.2. A programação foi realizada em um computador comum. Para realização de testes na placa, o código foi compilado para a TS-7260 através da utilização de um *cross-compiler*, que pode ser baixado em (TECHNOLOGICSYSTEMS, 2012). O arquivo binário gerado pela compilação foi então movido para uma pasta compartilhada na rede. Esta pasta foi disponibilizada através de um servidor NFS (*Network File System*) instalado no computador onde o código foi desenvolvido. A placa TS-7260 contém em um

pendrive um linux compilado para a sua arquitetura, que disponibiliza o cliente NFS, através do qual foi possível montar a pasta compartilhada e executar o algoritmo. Posteriormente, para testes totalmente autônomos, o arquivo binário foi copiado da pasta compartilhada para o pendrive conectado à TS-7260, o que permite que o robô execute a navegação sem necessidade de conexão com o computador. O comando utilizado no terminal da TS-7260 para montar a pasta compartilhada foi:

```
mount 192.168.0.3:/files /mnt
```

onde 192.168.0.3 era o IP local da máquina onde o *software* foi desenvolvido, */files* o diretório compartilhado por rede e */mnt* a pasta onde deveria ser montado. Após montar a pasta, os seguintes comandos são executados:

```
cd /mnt
./tslogic fcm
```

ou

```
./tslogic fuzzy
```

Todas as configurações da TS-7260 foram realizadas através de um terminal disponibilizado pela porta serial. Através de um *software* como o HyperTerminal (Windows) ou o minicom (Linux) é possível receber/enviar dados. A placa TS-7260 possui três portas de comunicação serial, das quais duas foram utilizadas: COM1 e COM2. A porta COM2 foi utilizada para conectar a TS-7260 ao microcontrolador C8051F340, para possibilitar a troca de mensagens de comando com o microcontrolador (para alterar os níveis de PWM dos motores), assim como o recebimento das leituras dos sensores. A porta COM1, por sua vez, foi configurada para ser o terminal mencionado anteriormente. Basta conectar um cabo serial da COM1 da TS-7260 ao computador para utilizá-lo. As configurações da serial utilizadas foram (tanto COM1 quanto COM2): 115200 8N1, ou seja, *baud-rate* de 115200, caracteres de 8 (oito) bits, sem bit de paridade e com 1 (um) bit de parada. Através do terminal disponibilizado na COM1, é possível montar e acessar a pasta de rede compartilhada, conforme descrito no parágrafo anterior. As conexões seriais e *interface* com o restante do robô podem ser visualizadas no Apêndice A, em fotos do robô após o final do projeto.

3.1.9 Software da TS-7260

Os algoritmos de controle gravados no pen-drive conectado à placa TS-7260 são o que efetivamente controla o robô. A parte principal do código consiste de um loop infinito, que executa os seguintes passos:

- Envio de um comando SYNC para o microcontrolador C8051F340: isto faz com que o C8051F340 responda enviando um pacote com as leituras mais recentes dos sensores de distância, juntamente com as leituras dos *encoders* (esta comunicação é realizada via porta serial);
- Leitura do pacote enviado pelo C8051F340: o programa recebe os dados do C8051F340 e utiliza os dados de conversão dos sensores para acessar a tabela que mapeia o valor da conversão para a distância em centímetros, tabela esta que foi obtida conforme o procedimento especificado na seção 3.1.2. Os valores de distância para cada sensor são então armazenados, assim como as leituras dos *encoders*;
- Execução do algoritmo de navegação: utiliza os dados armazenados para executar o algoritmo de navegação, passando os valores de leitura dos sensores de distância. O algoritmo utiliza os dados para realizar a inferência, retornando valores que indicam qual ação o robô deve tomar. O algoritmo de navegação é executado também na placa TS-7260, mas não faz parte do mesmo *software*. Cada algoritmo representa uma biblioteca externa ao software de controle e implementa um método que recebe como parâmetros as distâncias lidas e retorna valores que definem como o robô deve se locomover (velocidade e direção);
- Ajuste dos *setpoints*: nesta etapa, os dados retornados pelo algoritmo de navegação são utilizados para o cálculo do *setpoint* de cada roda (velocidade desejada), o *setpoint* é então alterado de acordo com o valor calculado;
- Ajuste de velocidade: esta etapa consiste no ajuste dos níveis de PWM de cada roda, e utiliza para tal os valores de leitura dos *encoders* e o *setpoint* do passo anterior. O objetivo do ajuste de velocidade é fazer com que cada motor fique o mais próximo possível do *setpoint* estabelecido previamente. Este ajuste é apenas proporcional, simplesmente aumentando a velocidade caso a contagem do *encoder* esteja abaixo do *setpoint* ou diminuindo caso esteja acima;
- Envio das ações: os níveis de PWM definidos pelo ajuste de velocidade são enviados para o microcontrolador C8051F340, que efetiva a mudança.

O diagrama em blocos a seguir representa o funcionamento do *software*, demonstrando também em que etapas ocorre a comunicação com o microcontrolador (setas tracejadas) e quando o algoritmo de navegação é executado. Vale lembrar que o algoritmo de navegação (Fuzzy ou ED-FCM) executa na placa TS-7260 também, mas seu código não faz parte do *software* de controle do robô (conforme explicado acima), por este motivo está em um bloco separado no diagrama. As ações descritas no bloco C8051F340 são executadas no microcontrolador C8051F340.

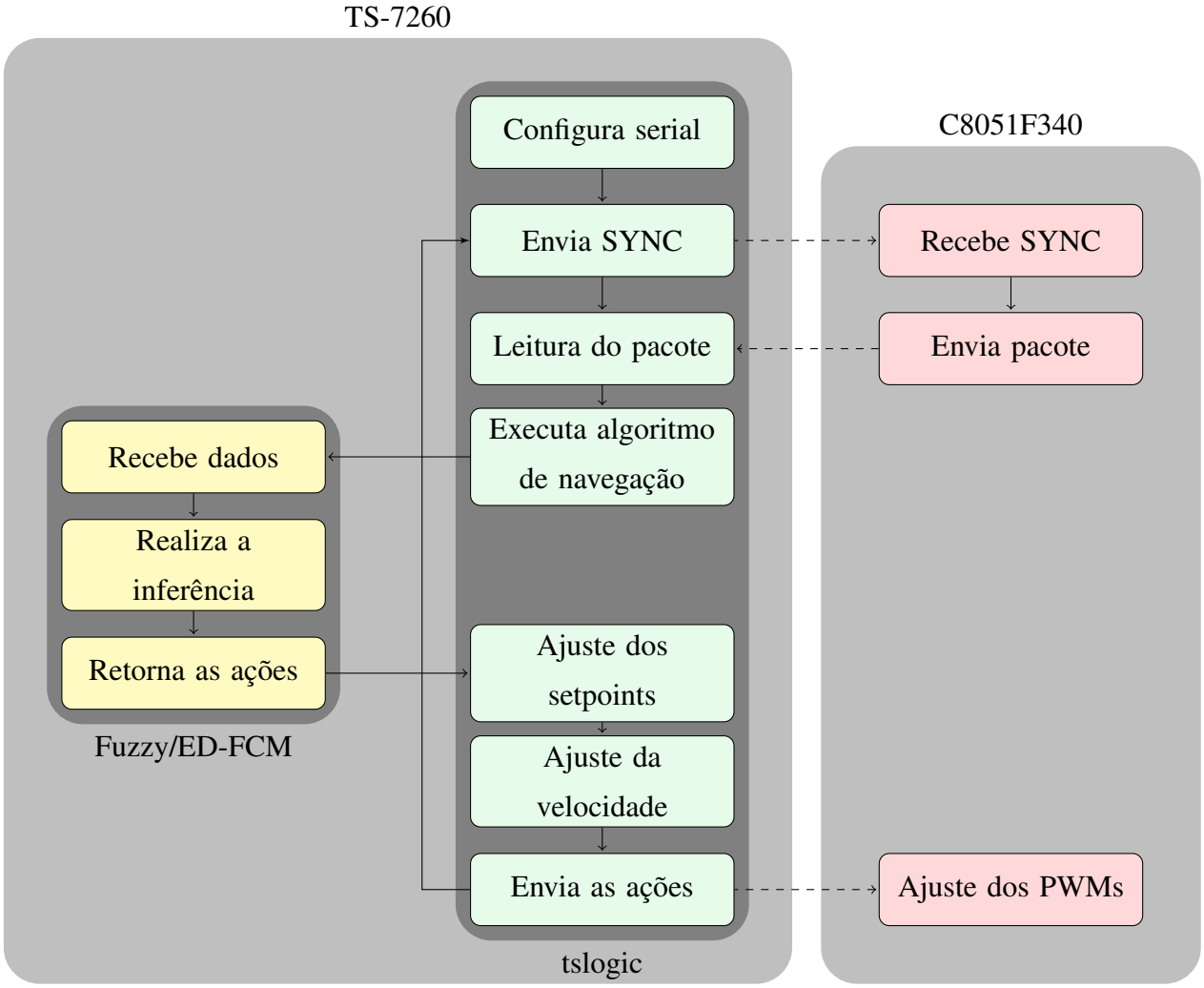

Figura 24: Diagrama em blocos demonstrando o funcionamento do software da placa TS-7260. Fonte: Autoria própria

3.1.10 Considerações

Nesta seção foram descritos os passos da equipe para alcançar o primeiro objetivo do projeto, a reconstrução da plataforma robótica Bellator, tornando-a apta para utilização no teste

e comparação de algoritmos de navegação. Foram descritos os testes iniciais com o robô, o projeto da nova placa de roteamento e a implementação do *software* da placa TS-7260.

3.2 ALGORITMOS DE NAVEGAÇÃO

Esta seção dedica-se à descrição da implementação dos algoritmos de navegação conforme descrição teórica dos mesmos apresentada na fundamentação teórica. Mais especificamente, serão descritos os códigos responsáveis pela tomada de decisões de ambos os algoritmos de navegação que executam na placa TS-7260, o algoritmo de navegação de lógica Fuzzy clássica, utilizando os mecanismos de inferência propostos por (MAMDANI; ASSILIAN, 1999) e o ED-FCM, proposto por (MENDONÇA; ARRUDA; NEVES, 2011).

3.2.1 Algoritmo de Navegação Fuzzy

O algoritmo de Navegação *Fuzzy* foi desenvolvido utilizando-se a biblioteca FLIE(*Fuzzy Logic Inference Engine*), (FABRO, 1996), que já implementa um ambiente para a definição de conjuntos *fuzzy* e regras de inferência sobre os mesmos, proporcionando à equipe mais tempo para o aprimoramento das regras e conjuntos e o teste dos mesmos. Este algoritmo será utilizado como base de comparação para determinação da eficiência do ED-FCM, descrito na seção 3.2.2. Nesta seção, será descrito como a biblioteca FLIE foi utilizada neste projeto e como foram definidos os conjuntos *fuzzy* e as regras de inferência, seguindo a fundamentação teórica sobre o assunto apresentada na seção 2.4.

Variáveis Linguísticas

A biblioteca FLIE possui estruturas de dados, ou classes, prontas para serem utilizadas para definição de variáveis linguísticas, bem como estruturas para a definição de regras de inferência baseadas neste conjunto de variáveis linguísticas, sendo apropriada para a elaboração de sistemas de controle *fuzzy*. A biblioteca funciona de acordo com os princípios teóricos apresentados na seção 2.4.

Para elaborar o controle *fuzzy* responsável pelo algoritmo de navegação, é necessário, primeiramente, definir as variáveis linguísticas e os conjuntos *fuzzy* associados às mesmas. Posteriormente, deve-se definir as variáveis linguísticas de entrada e saída do sistema e as associar apropriadamente. Em seguida, é necessário definir o método de defuzzificação e, finalmente, as regras de inferência. Cada um destes passos será descrito em detalhes a seguir.

A biblioteca FLIE permite a definição de variáveis linguísticas na classe *linguisticvariable*, que pode ser composta por diversos conjuntos *fuzzy*, definidos a partir da classe *category*. Cada *category* é definida por uma faixa de valores válidos (*range*), por quatro pontos que definem um trapézio com os intervalos de subida, máximo e descida do nível de pertinência do conjunto de entrada, para fuzzificação dos dados, e um nome apropriado, tal como Perto, Longe, Rápido, dependendo do contexto. Ou seja, cada valor de entrada é fuzzificado de acordo com seu nível de pertinência a cada categoria (conjunto *fuzzy*) pertencente à variável linguística. A figura 25 demonstra de forma gráfica esta estrutura.

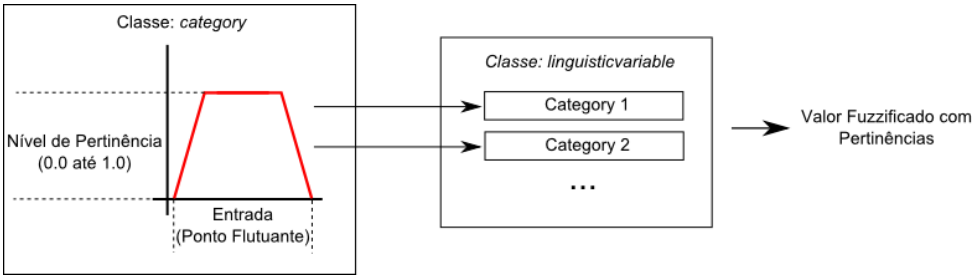

Figura 25: Estrutura de uma Variável Linguística no FLIE.

Fonte: Autoria própria

Utilizando esta estrutura, foram desenvolvidas três variáveis linguísticas para a entrada do sistema e duas variáveis linguísticas para a saída do sistema. Deve-se observar que foram definidas apenas três variáveis linguísticas de entrada por dois motivos: A biblioteca FLIE não aceita mais que três entradas em uma regra de inferência e um número maior de entradas levaria à uma quantidade excessiva de regras de inferência. As variáveis linguísticas de entrada correspondem aos cinco sensores do robô, sendo que o sensor frontal utiliza uma variável linguística, "SensorFrente" e os dois pares de sensores laterais são agrupados em duas variáveis linguísticas, "SensorEsquerda" e "SensorDireita" sendo que apenas o menor valor de leitura válida é selecionado nos pares de sensores laterais para ser fuzzificado. Cada uma destas variáveis linguísticas está definida com três conjuntos *fuzzy*, totalizando nove conjuntos *fuzzy* organizados de acordo com a tabela 3.

Tabela 3: Funções de Pertinência e Conjuntos Fuzzy dos Sensores.

Variável Linguística	Conjunto Fuzzy	Função de Pertinência
SensorEsquerda	PertoEsquerda	sp(d)
SensorEsquerda	MedioEsquerda	sm(d)
SensorEsquerda	LongeEsquerda	sl(d)
SensorFrente	Perto	sp(d)
SensorFrente	Medio	sm(d)
SensorFrente	Longe	sl(d)
SensorDireita	PertoDireita	sp(d)
SensorDireita	MedioDireita	sm(d)
SensorDireita	LongeDireita	sl(d)

As funções de pertinência sp(d), sm(d), sl(d), correspondentes a perto, médio e longe, são definidas pelos trapézios da figura 26:

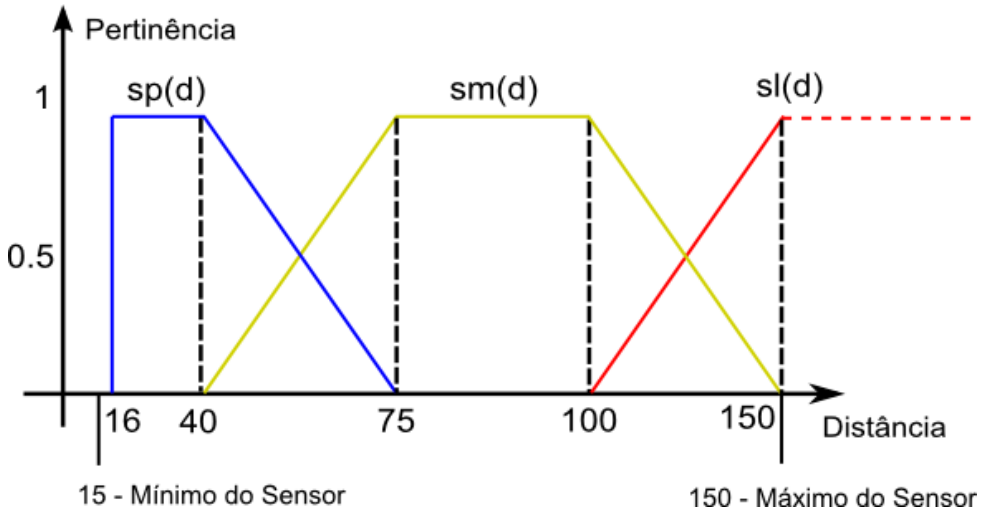

Figura 26: Definição das Funções de Pertinência perto, médio e longe.

Fonte: Autoria própria

Embora as funções de pertinência sejam as mesmas para os conjuntos *fuzzy* representando perto, médio e longe para as três variáveis linguísticas, cada variável linguística necessita de objetos da classe *category* exclusivos e únicos, por uma restrição da biblioteca FLIE, e não é possível utilizar o mesmo conjunto *fuzzy* para duas ou mais variáveis linguísticas diferentes.

De forma análoga, as variáveis linguísticas de saída do sistema são definidas pelas variáveis linguísticas “Velocidade Motor” e “Direção”, representando a velocidade e a direção do deslo-

camento do robô. Os conjuntos *fuzzy* associados à variável linguística Velocidade Motor foram definidos de forma a distribuir igualmente as faixas de PWM de 0 a 100% da capacidade do motor em três categorias: Lento, Médio e Rápido. As funções de pertinência que definem estes conjuntos *fuzzy* são pl(p), pm(p), pr(p), onde p é a percentagem do PWM. Pode-se visualizar estas funções na figura 27.

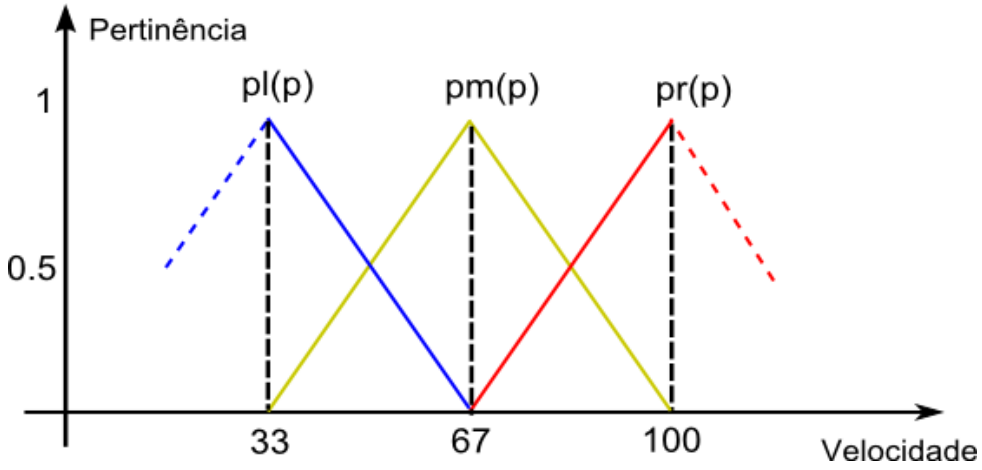

Figura 27: Definição das Funções de Pertinência lento, médio e rápido.
Fonte: Autoria própria

Similarmente, foram definidos 5 conjuntos *fuzzy* para a variável linguística "Direção": ViraEsquerda, ViraPoucoEsquerda, Reto, ViraPoucoDireita, ViraDireita, distribuindo igualmente entre estas classes todas as possibilidades de direção, desde virar totalmente para a esquerda (0^o) até totalmente para direita(180^o), sendo que 90^o representa movimento em linha reta. As funções de pertinência que definem estes conjuntos *fuzzy* são ve(a), vpe(a), rt(a), vpd(a), vd(a). O gráfico da figura 28 ilustra estas funções.

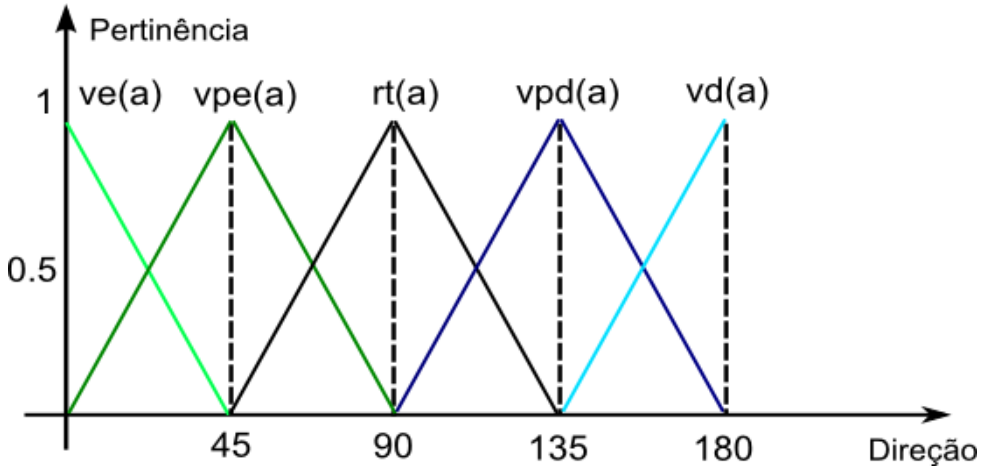

Figura 28: Definição das Funções de Pertinência de Direção.

Fonte: Autoria própria

Regras de Inferência

Após definidas as variáveis linguísticas, regras de inferência podem ser elaboradas através da classe *infrule*. Esta classe é composta de, no máximo, três variáveis linguísticas de entrada e uma variável linguística de saída, todas definidas pela classe *linguisticvariable*, seguindo a estrutura na figura 29.

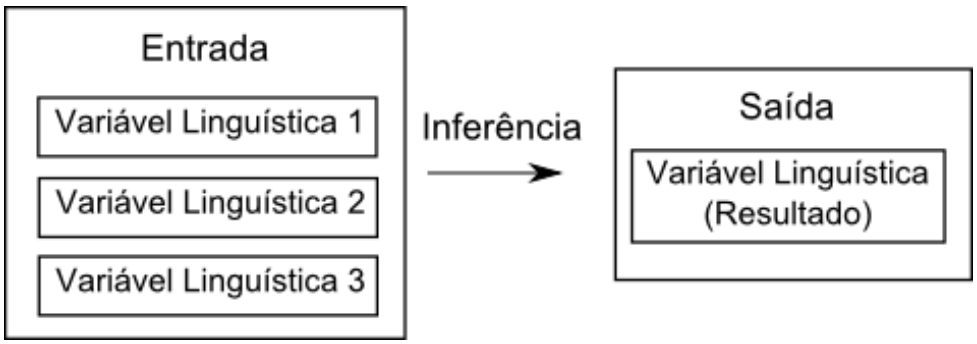

Figura 29: Estrutura de uma regra de inferência no FLIE.

Fonte: Autoria própria

Com as variáveis linguísticas de entrada e saída pode-se criar regras de inferência. Primeiramente, é necessário associar as variáveis linguísticas que compõem as regras de inferência de entrada e saída à um objeto da classe *fuzzy_control*. Em seguida, cada regra é definida por uma sequência de valores *fuzzy* pertencentes às variáveis linguísticas de entrada e saída associadas com o objeto da classe *fuzzy_control*, sendo o último valor da sequência a saída *fuzzy* desejada

para uma entrada correspondente a um, dois ou três valores antecedentes, que é o limite de entradas da biblioteca FLIE, como mencionado em 3.2.1. Como deseja-se obter duas saídas, uma para a velocidade e outra para a direção, foram necessários dois objetos desta classe, ambos com as mesmas entradas. Além disso, é necessário definir o método de defuzzificação a ser utilizado pelo controle *fuzzy*. A equipe utilizou o método da Média do Máximo, descrito na seção 2.4.3. Assim, as regras de inferência seguem o padrão abaixo, de acordo com as variáveis linguísticas definidas na seção 3.2.1:

SensorEsquerda, SensorFrente, SensorDireita, Velocidade Motor
SensorEsquerda, SensorFrente, SensorDireita, Direção

Em seguida, pode-se definir as regras através dos valores *fuzzy* válidos para cada variável linguística associada aos objetos *fuzzy_control*. A definição das regras é bastante intuitiva, empírica e altamente dependente da aplicação. A equipe optou por definir um conjunto de regras arbitrário, realizar testes e, analisando os resultados preliminares, aprimorar o conjunto de regras. A tabela 4 representa o conjunto definitivo de 27 regras elaboradas pela equipe, que correspondem a todas as combinações possíveis de entradas para o controle *fuzzy* (3 opções de classificação para cada entrada, sendo 3 entradas). Algumas regras podem ser redundantes, tendo em vista que, a partir dos testes, o melhor curso de ação observado para algumas situações diferentes é o mesmo.

Tabela 4: Regras de Inferência - Algoritmo Fuzzy.

SensorEsquerda	SensorFrente	SensorDireita	VelocidadeMotor	Direção
PertoEsquerda	Perto	PertoDireita	Lento	ViraDireita
PertoEsquerda	Perto	MedioDireita	Lento	ViraDireita
PertoEsquerda	Perto	LongeDireita	Lento	ViraDireita
PertoEsquerda	Medio	PertoDireita	Lento	Reto
PertoEsquerda	Medio	MedioDireita	Lento	ViraDireita
PertoEsquerda	Medio	LongeDireita	Lento	ViraPoucoDireita
PertoEsquerda	Longe	PertoDireita	Medio	Reto
PertoEsquerda	Longe	MedioDireita	Lento	ViraPoucoDireita
PertoEsquerda	Longe	LongeDireita	Lento	ViraDireita
MedioEsquerda	Perto	PertoDireita	Lento	ViraEsquerda
MedioEsquerda	Perto	MedioDireita	Lento	ViraDireita
MedioEsquerda	Perto	LongeDireita	Lento	ViraPoucoDireita
MedioEsquerda	Medio	PertoDireita	Lento	ViraEsquerda
MedioEsquerda	Medio	MedioDireita	Medio	Reto
MedioEsquerda	Medio	LongeDireita	Medio	Reto
MedioEsquerda	Longe	PertoDireita	Lento	ViraPoucoEsquerda
MedioEsquerda	Longe	MedioDireita	Rapido	Reto
MedioEsquerda	Longe	LongeDireita	Medio	Reto
LongeEsquerda	Perto	PertoDireita	Lento	ViraEsquerda
LongeEsquerda	Perto	MedioDireita	Lento	ViraPoucoEsquerda
LongeEsquerda	Perto	LongeDireita	Lento	ViraDireita
LongeEsquerda	Medio	PertoDireita	Lento	ViraPoucoEsquerda
LongeEsquerda	Medio	MedioDireita	Medio	Reto
LongeEsquerda	Medio	LongeDireita	Rapido	ViraDireita
LongeEsquerda	Longe	PertoDireita	Lento	ViraEsquerda
LongeEsquerda	Longe	MedioDireita	Medio	Reto
LongeEsquerda	Longe	LongeDireita	Rapido	Reto

Finalmente, as regras definidas podem ser utilizadas para realizar as inferências. A bilioteca FLIE recebe os dados de entrada não fuzzificados, os fuzzifica, atribuindo valores de pertinência para cada categoria definida pelo desenvolvedor e determina o nível de ativação de cada regra de inferência definida. Após esta etapa, a FLIE defuzzifica o resultado seguindo o modelo de defuzzificação escolhido (neste caso a média do máximo) e retorna o resultado final, que

pode ser utilizado para controlar a navegação do robô. Este resultado possui dois valores: a velocidade do motor, que é um valor percentual de 0 a 100%, e a direção, um valor de 0^o (que representa virar à esquerda) a 180^o (que representa virar à direita), sendo que 90^o seria reto.

Considerações

O algoritmo *fuzzy* produziu o comportamento adequado em navegação robótica permitindo que o robô fosse capaz de desviar obstáculos posicionados à direita, à esquerda e à frente do mesmo. Conforme será descrito na seção de testes, seção 3.3, esse algoritmo possibilitou o robô a navegar de forma autônoma recebendo como entrada as leituras dos sensores. A versão descrita nessa seção foi produzida após a execução e conclusão dos testes iniciais e avançados, seções 3.3.2 e 3.3.3, respectivamente. Durante esses testes, o algoritmo foi submetido a uma série de experimentos visando detectar erros de navegação, isolá-los e corrigí-los. Esse processo foi realizado repetidas vezes e as funções de pertinência e regras *fuzzy* foram aprimoradas gradualmente. A tabela de funções (tabela 3) e a tabela de regras (tabela 4) representam a versão utilizada nos testes comparativos, descritos na seção 3.3.4, que foi utilizada para obtenção do resultado final.

3.2.2 Algoritmo de Navegação Baseado em ED-FCM

Esta seção tem por objetivo descrever o projeto e a implementação do algoritmo de navegação baseado na abordagem ED-FCM, apresentada na seção 2.5 da fundamentação teórica. O projeto consistiu na definição dos conceitos de entrada, nível e saída, relações causais, respectivos pesos e um evento. A implementação foi desenvolvida em linguagem C++ e compilada para ser executada na placa TS-7260.

A idéia do algoritmo foi controlar a direção do robô ajustando a potência aplicada em cada roda, de forma independente, sendo que esta pode ser aplicada para a roda girar em ambos os sentidos. As seguintes hipóteses foram elaboradas para projetar o algoritmo:

1. Quando o robô detectar um objeto à esquerda, deve desviar para a direita;
2. Quando o robô detectar um objeto à direita, deve desviar para a esquerda;
3. Quando o robô não detectar objetos nas laterais, este deve seguir em frente.

Os movimentos “desviar para a direita”, “desviar para a esquerda” e “seguir em frente” podem ser obtidos controlando-se a potência e o sentido de giro aplicado em cada roda. Desse

modo, duas hipóteses foram imaginadas: as duas rodas girando para frente e as duas rotas girando em sentidos opostos. Os movimentos produzidos na primeira situação são:

1. Quando a roda direita girar mais rápido que a roda esquerda (no mesmo sentido), a mudança de direção é “desviar para a esquerda”;
2. Quando a roda esquerda girar mais rápido que a roda direita (no mesmo sentido), a mudança de direção é “desviar para a direita”;
3. Quando a roda esquerda girar na mesma intensidade e sentido que a roda direita, o movimento é “seguir em frente”.

Os movimentos produzidos na segunda situação são:

1. Quando a roda direita girar para frente e a roda esquerda girar para trás, o movimento é “desviar para a esquerda”;
2. Quando a roda esquerda girar para frente e a roda direita girar para trás, o movimento é “desviar para a direita”.

Deste modo, o algoritmo controla a direção do robô calculando a “intensidade de girar” a roda direita para trás ou para frente e a “intensidade de girar” a roda esquerda para trás ou para frente, sendo que esse cálculo recebe como entradas as distâncias registradas pelos sensores de distância na lateral direita, na frente e na lateral esquerda do robô. O mesmo agrupamento realizado para o algoritmo Fuzzy foi utilizado para o ED-FCM, tomando o mínimo de cada par de sensores laterais como uma entrada, totalizando 2 entradas, e o sensor frontal representando a terceira entrada.

Conceitos

Primeiramente, definiu-se a entrada do sistema de navegação, a qual correspondeu às leituras dos sensores de distância. Desse modo, foram criados três conceitos de entrada:

1. SE - Representa a leitura do sensor lateral esquerdo;
2. SF - Representa a leitura do sensor frontal;
3. SD - Representa a leitura do sensor lateral direito.

Esses conceitos guardam o valor da leitura da distância normalizada na faixa de 0 a 1, através da equação 24, sendo x o valor absoluto da distância (cm), MIN o valor mínimo suportado pelo sensor e MAX o valor máximo.

$$y = \frac{x - MIN}{MAX - MIN} \tag{24}$$

Em seguida, foram determinados os conceitos de nível do ED-FCM, os quais estabelecem as inferências resultantes dos valores dos conceitos de entrada. Foram definidos quatro conceitos de nível:

1. GDF - Representa a intensidade da decisão de girar a roda direita para frente;
2. GDT - Representa a intensidade da decisão de girar a roda direita para trás;
3. GEF - Representa a intensidade da decisão de girar a roda esquerda para frente;
4. GET - Representa a intensidade da decisão de girar a roda esquerda para trás.

Os níveis desses conceitos são ativados através de uma função sigmoidal dependente dos valores dos conceitos de entrada. As equações 25 e 26 foram adaptadas da equação 9, apresentada na fundamentação teórica, e operam em domínio $[0,1]$ e imagem $[0,1]$.

$$f_1(x) = \frac{1}{1 + e^{(-10x+4,5)}} \tag{25}$$

$$f_2(x) = 1 - \frac{1}{1 + e^{(-10x+4,5)}} \tag{26}$$

Por fim, definiu-se a saída do sistema de navegação, a qual são os níveis percentuais de potência de cada roda. Esse nível varia na faixa de -100% a +100% e o sinal indica o sentido de giro, sendo o sinal positivo “girar para frente” e o sinal negativo “girar para trás” . Desse modo, foram estabelecidos dois conceitos de saída:

1. RD Out - Representa o nível percentual de potência da roda direita;
2. RE Out - Representa o nível percentual de potência da roda esquerda.

Relações Causais

Os conceitos foram interligados através das relações causais ilustradas na figura 30. Esse ED-FCM conecta os conceitos de entrada SD, SF e SE, através dos conceitos de nível GDF, GDT, GEF e GET, aos conceitos de saída RD Out e RE Out.

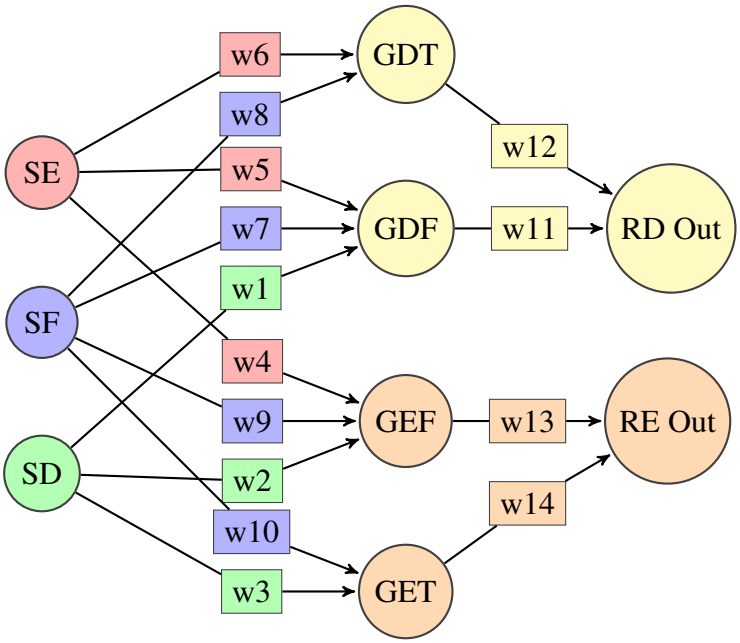

Figura 30: ED-FCM proposto.

Fonte: Autoria própria.

Para modificar o valor dos pesos em tempo de execução, introduziu-se ao sistema um conceito de estado que influencia os pesos de todas as relações causais. Os estados que o ED-FCM proposto pode assumir são “FRENTE” e “TRÁS”, sendo que os pesos das relações causais quando o ED-FCM estiver no primeiro estado estão de acordo com a tabela 5 e os pesos no segundo estado, com a tabela 6. O valor numérico das intensidades das relações causais estão de acordo com a tabela 7.

A transição de estado é feita através de uma regra “SE-ENTÃO” , sendo que os valores de L0 e L1 estão de acordo com a tabela 8.

- SE o conceito SF for menor que o limiar L0 e o estado for igual a FRENTE ENTÃO o estado muda para TRÁS.
- SENÃO SE o conceito SF for maior que o limiar L1 e o estado for igual a TRÁS ENTÃO o estado muda para FRENTE.

Tabela 5: Relações causais do controlador ED-FCM proposto (Estado = FRENTE).

Relação causal	Descrição	Efeito	Intensidade
w_1	SD influencia GDF	Positivo	Forte
w_2	SD influencia GEF	Positivo	Fraca
w_3	SD influencia GET	Positivo	Média
w_4	SE influencia GEF	Positivo	Forte
w_5	SE influencia GDF	Positivo	Fraca
w_6	SE influencia GDT	Positivo	Média
w_7	SF influencia GDF	Negativo	Média
w_8	SF influencia GDT	Positivo	Forte
w_9	SF influencia GEF	Negativo	Média
w_{10}	SF influencia GET	Positivo	Fraca
w_{11}	GDF influencia RD Out	Positivo	Forte
w_{12}	GDT influencia RD Out	Negativo	Média
w_{13}	GEF influencia RE Out	Positivo	Forte
w_{14}	GET influencia RE Out	Negativo	Média

Tabela 6: Relações causais do controlador ED-FCM proposto (Estado = TRÁS).

Relação causal	Descrição	Efeito	Intensidade
w_1	SD influencia GDF	Positivo	Fraca
w_2	SD influencia GEF	Positivo	Fraca
w_3	SD influencia GET	Positivo	Forte
w_4	SE influencia GEF	Positivo	Fraca
w_5	SE influencia GDF	Positivo	Fraca
w_6	SE influencia GDT	Positivo	Forte
w_7	SF influencia GDF	Negativo	Fraca
w_8	SF influencia GDT	Positivo	Média
w_9	SF influencia GEF	Negativo	Média
w_{10}	SF influencia GET	Positivo	Fraca
w_{11}	GDF influencia RD Out	Positivo	Fraca
w_{12}	GDT influencia RD Out	Negativo	Forte
w_{13}	GEF influencia RE Out	Positivo	Forte
w_{14}	GET influencia RE Out	Negativo	Fraca

Tabela 7: Valor numérico dos pesos.

Intensidade	Valor numérico
FRACA	0,125
MÉDIA	0,5
FORTE	1,0

Tabela 8: Valor numérico dos limiares L0 e L1.

Limiar	Valor numérico
L0	0,15
L1	0,25

Ativação dos Conceitos

As equações 27 a 30 determinam a ativação dos conceitos de nível GDF, GDT, GEF e GET, respectivamente. Essa ativação é gradual e herda a característica de suavidade das funções sigmoidais 25 e 26. Os conceitos de saída RD_{Out} e RE_{Out} são uma média ponderada entre os conceitos "girar para frente" e "girar para trás", aplicada às rodas direita e esquerda, conforme as equações 31 e 32, respectivamente.

$$GDF = \frac{w_1 f_2(SD) + w_5 f_1(SE) - w_7 f_2(SF)}{w_1 + w_5} \tag{27}$$

$$GDT = \frac{w_6 f_2(SE) + w_8 f_2(SF)}{w_6 + w_8} \tag{28}$$

$$GEF = \frac{w_4 f_2(SE) + w_2 f_1(SD) - w_9 f_2(SF)}{w_4 + w_2} \tag{29}$$

$$GET = \frac{w_3 f_2(SD) + w_{10} f_2(SF)}{w_3 + w_{10}} \tag{30}$$

$$RD_{Out} = 100 \times \frac{w_{11} GDF - w_{12} GDT}{FORTE} \tag{31}$$

$$RD_{Out} = 100 \times \frac{w_{13} GEF - w_{14} GET}{FORTE} \tag{32}$$

Considerações

O ED-FCM projetado ofereceu a capacidade de o robô desviar obstáculos posicionados à direita, à esquerda e à frente do mesmo. Conforme será descrito na seção de testes, seção 3.3, esse algoritmo possibilitou ao robô navegar de forma autônoma recebendo como entrada as leituras dos sensores. Contudo, a versão final do ED-FCM foi obtida após diversos testes, nos quais erros de navegação foram encontrados, isolados e corrigidos repetidas vezes. Nesse processo, os pesos das relações causais foram modificados até chegarem aos valores das tabelas 5 e 6. A necessidade de implementação de um ED-FCM com evento surgiu durante a execução de um experimento dos testes avançados, como é descrito na seção 3.3.3. O evento foi implementado para resolver especificamente aquele problema de indecisão. Com isso, a versão final do projeto de ED-FCM apresentada nessa seção foi capaz de resolver os problemas propostos

nos experimentos dos testes comparativos, seção 3.3.4, gerando resultados adequados para a comparação com o algoritmo *fuzzy*, apresentado na seção 3.2.1.

3.3 TESTES E ANÁLISE DE RESULTADOS

Esta seção tem por objetivo apresentar a metodologia e elaboração dos testes dos algoritmos de navegação implementados pela equipe, utilizando-se de desenhos esquemáticos e descrição detalhada dos objetivos de cada teste, além das alterações e ajustes aos parâmetros e conceitos dos algoritmos realizados a partir dos resultados dos testes. Nesta seção também é apresentada a análise e comparação do comportamento de ambos os algoritmos de navegação a partir dos resultados dos experimentos para cada algoritmo, após submetidos às mesmas condições de teste.

3.3.1 Elaboração dos Testes

A metodologia de elaboração dos testes da equipe visou possibilitar testes de sistema, verificando a funcionalidade básica dos algoritmos, testes de comportamento visando eliminar erros de desenvolvimento e comportamentos indesejados para ambos os algoritmos e, finalmente, comparar o comportamento dos algoritmos submetidos às mesmas condições iniciais de teste. Ao longo dos testes, várias alterações foram realizadas tanto nos conjuntos *fuzzy* e regras de inferência do algoritmo de navegação *fuzzy* quanto nos conceitos e funções sigmóides do algoritmo baseado em ED-FCM, antes de atingir a configuração final descrita nas seções 3.2.1 e 3.2.2.

Assim sendo, foram desenvolvidas três principais etapas de testes:

1. Testes Iniciais: Teste básico dos algoritmos visando observar a movimentação básica, sincronia das rodas e ajuste de parâmetros.

2. Testes Avançados: Testes realizados com mais obstáculos para avaliar a capacidade dos algoritmos em evitar colisões com obstáculos.

3. Testes Comparativos: Testes finais visando a comparação do comportamento entre os algoritmos em diversas situações

Tanto os testes iniciais quanto os avançados são essenciais para proporcionar uma análise de resultados confiável entre os algoritmos. Nas seções seguintes, para cada teste desenvolvido, são apresentados diagramas esquemáticos espaciais, representando a disposição dos obstáculos

e a posição inicial do robô, o comportamento apresentado pelo robô, possíveis alterações e ajustes realizados nos algoritmos por conta dos resultados do teste.

3.3.2 Testes Iniciais

Foram elaborados dois testes iniciais, o primeiro visando observar o deslocamento simples e detecção de um obstáculo simples próximo à parede em um corredor (Figura 31), e o segundo teste foi construído em um corredor mais largo que o primeiro, com alguns obstáculos obstruindo a passagem, definindo apenas um caminho possível entre os obstáculos (figura 32).

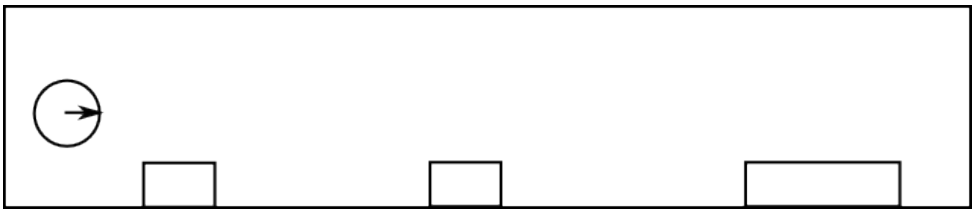

Figura 31: Configuração do primeiro teste inicial.
Fonte: Autoria própria

Ambos os algoritmos obtiveram resultados satisfatórios, sendo capazes de realizar o percurso sem colidir com os obstáculos. Contudo, o algoritmo de navegação *fuzzy* mostrou-se excessivamente lento próximo aos obstáculos. Concluiu-se que isto ocorreu devido ao valor máximo das funções de pertinência "Lento" e "Médio", serem muito baixos. A função de pertinência "Lento" apresentava um valor máximo de apenas 20% da velocidade máxima e "Médio" correspondia a 50%. A partir desta observação, as funções de pertinência da velocidade foram alteradas para os novos valores, descritos na seção 3.2.1 de 33% para "Lento" e 66% para "Médio". Após os ajustes, o algoritmo mostrou-se mais rápido no corredor, no mesmo teste.

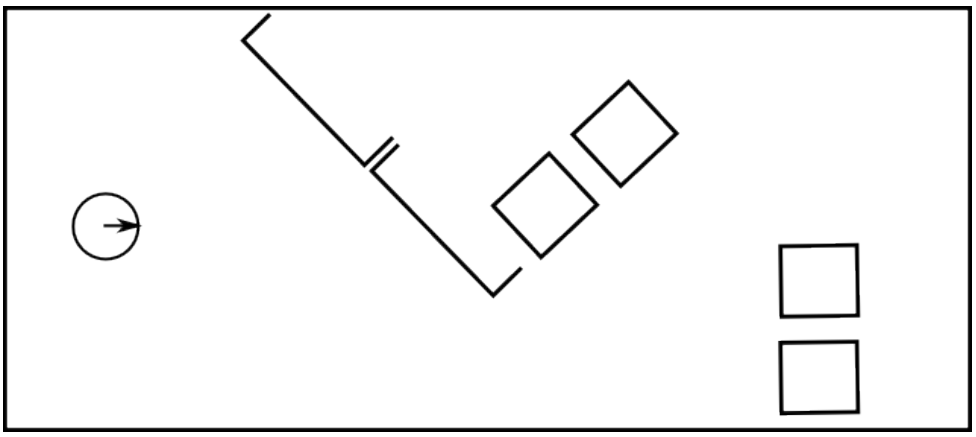

Figura 32: Configuração do segundo teste inicial.
Fonte: Autoria própria

O segundo teste, ao contrário do primeiro, força os algoritmos a desviar de uma barreira imediatamente no caminho do robô e encontrar a saída do corredor, logo após a barreira, como ilustrado na figura 32. Ao executar este teste pela primeira vez, o algoritmo *Fuzzy* não foi capaz de fazer o desvio e encontrar a saída, devido a uma reação muito tardia e, ao mesmo tempo, muito forte. Já o algoritmo ED-FCM foi capaz de realizar o trajeto sem alterações. A figura 33 apresenta o trajeto esperado, em verde, que é aproximadamente o mesmo que o realizado pelo algoritmo ED-FCM, e o trajeto percorrido pelo algoritmo *Fuzzy* clássico, em vermelho.

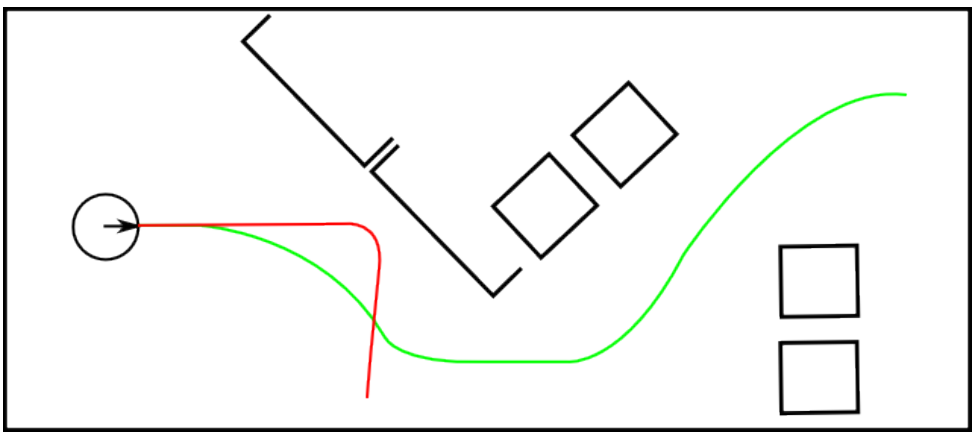

Figura 33: Caminhos percorridos no segundo teste inicial.
Fonte: Autoria própria

Para resolver este problema de reação tardia e tornar a navegação do algoritmo *Fuzzy* mais suave (em relação as curvas), a equipe aumentou a faixa de valores da função de pertinência da variável linguística perto, que era inicialmente 16 até 25 cm como distância de pertinência

máxima e descida até pertinência nula na distância de 35 centímetros para os valores finais apresentados na figura 26, função p(d). Além disso, os conjuntos *fuzzy* “ViraPoucoEsquerda”e “ViraPoucoDireita”, inexistentes na primeira versão, foram adicionados e utilizados nas regras de inferência da forma como foi apresentado na tabela 4, visando iniciar a reação do robô aos obstáculos mais cedo. Estas alterações fizeram com que o algoritmo *Fuzzy* clássico realizasse o trajeto esperado, após novo teste no mesmo ambiente.

3.3.3 Testes Avançados

Os testes avançados desenvolvidos pela equipe visam observar e otimizar o comportamento dos algoritmos, ou seja, além da simples capacidade de desvio de obstáculos, observar as escolhas realizadas pelos algoritmos em ambientes com mais de uma possibilidade de navegação além da velocidade e agilidade do robô nestes ambientes, com atenção especial a situações específicas que podem levar o robô a “travar” e parar de navegar.

A equipe montou duas situações distintas para cumprir estes objetivos: O primeiro em um corredor similar ao segundo teste inicial, porém mais longo e com mais obstáculos dispersos e o segundo em um corredor largo com obstáculos dispersos, provendo um ambiente aberto ao robô.

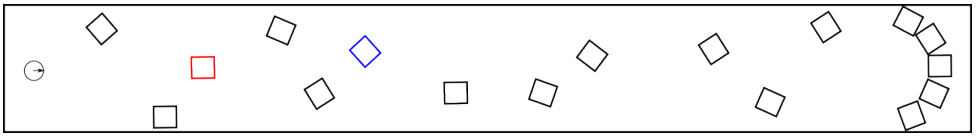

Figura 34: Ilustração do primeiro teste avançado.
Fonte: Autoria própria

O algoritmo *fuzzy* foi capaz de percorrer o primeiro teste avançado, ilustrado na figura 34, retornando ao ponto de partida após 5 minutos. Durante o percurso, houve apenas uma pequena colisão com o obstáculo destacado em azul na figura acima. Neste experimento foi possível observar lentidão excessiva do robô próximo a obstáculos, levando a equipe a alterar a base de regras de inferência levemente, de forma a reduzir o número de regras que contêm o conjunto *fuzzy* “Lento” como saída de velocidade, acelerando a navegação do robô no mesmo teste. O conjunto de regras final, obtido após as alterações feitas a partir deste experimento, pode ser observado na tabela 4. O algoritmo ED-FCM, em contrapartida, não foi capaz de percorrer o teste, apresentando indecisão ao aproximar-se do obstáculo destacado em vermelho na figura 34. Isto ocorreu devido à disponibilidade de espaços iguais para o desvio deste obstáculo em

ambas as opções de desvio, o que anulou a intensidade de PWM em ambas as rodas. O efeito de anulação é devido ao fato de as sigmóides utilizadas na ativação dos conceitos serem simétricas, conforme as equações 25 e 26. Desse modo, existe um ponto no qual a distância registrada pelos sensores laterais esquerdo e direito geram valores nos conceitos de entrada, SE e SD, que fazem com que os conceitos de nível GDF, GDT, GEF, GET (ver a explicação desses conceitos na seção 3.2.2) se igualem, ou seja, a tendência de girar a roda para frente é igual a tendência de girar a roda para trás, anulando, portanto, o PWM das rodas. Este problema foi corrigido através da implementação de um "evento", específico para auxílio à navegação nesta situação, que permitiu que o ED-FCM modificasse o peso das relações causais dinamicamente, conforme foi descrito na seção 3.2.2. Essa modificação retirou do ED-FCM o comportamento simétrico que resultava na anulação dos PWMs e permitiu que o algoritmo resolvesse o problema de indecisão, completando o percurso, sem novas indecisões e travamentos, após nova execução do experimento.

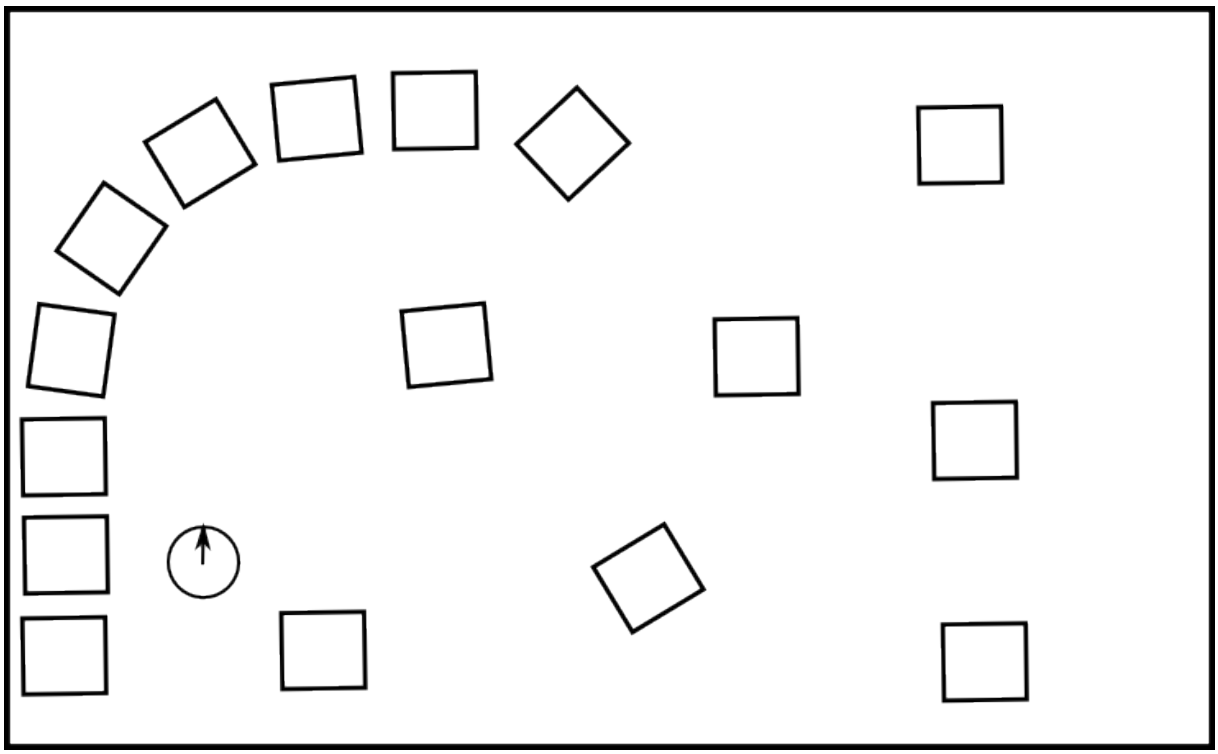

Figura 35: Ilustração do segundo teste avançado.
Fonte: Autoria própria

Após todas as correções realizadas até este ponto, a equipe elaborou um teste com diversos obstáculos esparsos e pontos de partida aleatórios, visando detectar quaisquer problemas e comportamentos indesejáveis remanescentes nos algoritmos de navegação. Em todas as execuções

dos algoritmos neste ambiente, representado na figura 35, ambos os algoritmos foram capazes de navegar sem indecisões, raramente encostando em um obstáculo. Visto que não foi mais detectada a necessidade de correções e ajustes específicos, este experimento permitiu concluir que os algoritmos estavam prontos para serem comparados em novos experimentos, descritos na seção a seguir.

3.3.4 Testes Comparativos

Os testes comparativos são, após o desenvolvimento, correção e aprimoramento de ambos os algoritmos, a oportunidade de avaliar, de forma empírica, a nova abordagem de navegação, proposta por Mendonça, utilizando mapas cognitivos *Fuzzy* em relação a uma abordagem utilizando um controle *Fuzzy* clássico. Foram elaborados dois experimentos visando expor ambos os algoritmos a diversos obstáculos e situações de indecisão, possibilitando assim observar qual dos algoritmos responde melhor a estas situações. Os resultados de ambos os experimentos serão analisados de forma detalhada, visando determinar as principais características que diferenciam os algoritmos e justificar o comportamento observado.

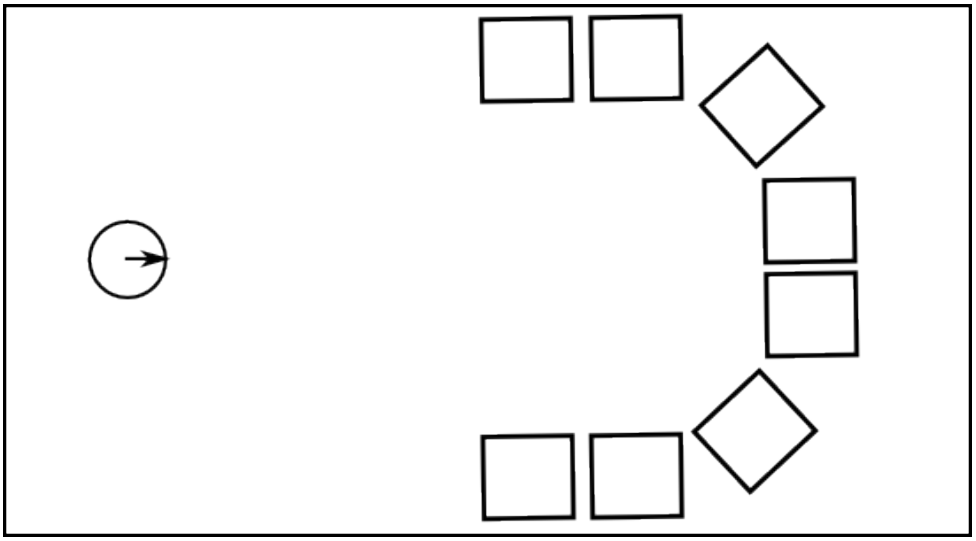

Figura 36: Ilustração do corredor sem saída.
Fonte: Autoria própria

O primeiro teste comparativo consiste em um corredor sem saída, observável na figura 36, visando comparar o comportamento de ambos os algoritmos em uma situação comum de indecisão, na qual só é possível sair através do mesmo caminho de entrada no corredor. Caso o algoritmo não seja capaz de detectar esta situação, poderá ficar preso. Neste experimento, ambos os algoritmos de navegação foram capazes de realizar o retorno e sair do corredor. O

algoritmo *Fuzzy* clássico entrou no corredor no centro do mesmo, realizou um giro de 180º aproximadamente e saiu do corredor em um total de 26 segundos, enquanto que o algoritmo baseado em ED-FCM realizou pequenas conversões ao longo da borda do corredor antes de encontrar a saída, o que levou cerca de 48 segundos. O caminho aproximado percorrido por ambos algoritmos pode ser visualizado no esboço da figura 37, onde o caminho executado pelo algoritmo *Fuzzy* está em vermelho e o ED-FCM em azul.

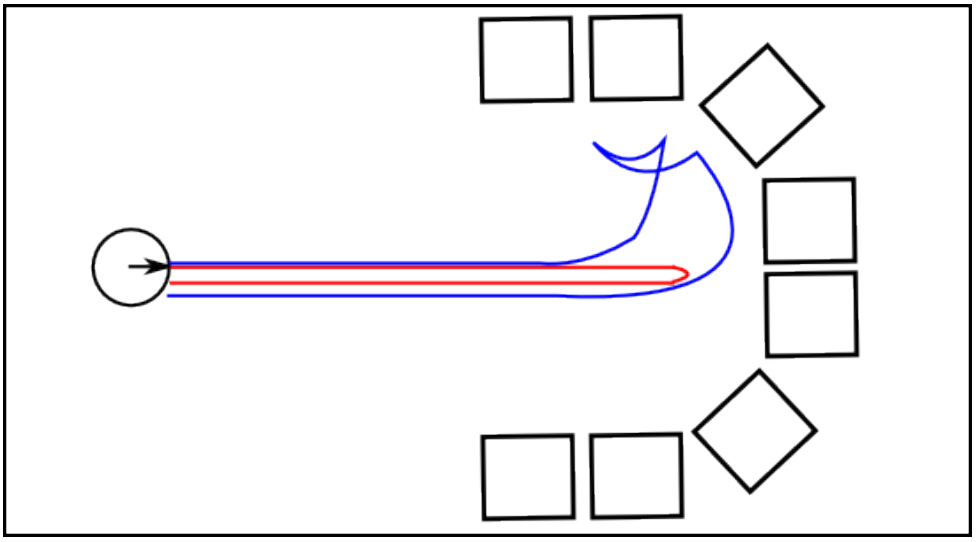

Figura 37: Esboço dos caminhos percorridos no primeiro teste comparativo.
Fonte: Autoria própria

Analisando este resultado, pode-se observar que o algoritmo *Fuzzy* clássico apresentou facilidade em retornar e sair do corredor. Isto deve-se à tendência de virar para a direita quando a entrada do mecanismo de inferência é "Perto"para o sensor frontal e igual nos sensores laterais. Esta tendência foi introduzida visando evitar a parada do robô em situações similares a esta, e foi capaz de resolver esse experimento rapidamente, como esperado. As três regras de inferência responsáveis por esta tendência, retiradas da tabela 4, são:

$$\text{PertoEsquerda, Perto, PertoDireita} \implies \text{Lento, ViraDireita}$$
$$\text{MedioEsquerda, Perto, MedioDireita} \implies \text{Lento, ViraDireita}$$
$$\text{LongeEsquerda, Perto, LongeDireita} \implies \text{Lento, ViraDireita}$$

Em compensação, o algoritmo baseado em ED-FCM apresentou dificuldades nesta situação. Na figura 37 é visível que o algoritmo ED-FCM tratou inicialmente o corredor como um obstáculo, tentando desviar-se do mesmo pela esquerda. Ao detectar obstáculos em todos os sensores, o evento, implementado como solução do problema detectado no segundo teste

avançado, descrito em 3.3.3, fazendo com que o robô retorne e tente, novamente, fazer o desvio do obstáculo pela direita. Porém, na segunda tentativa, é detectada maior disponibilidade de espaço à direita do robô, fazendo com que o mesmo contorne o corredor e encontre a saída.

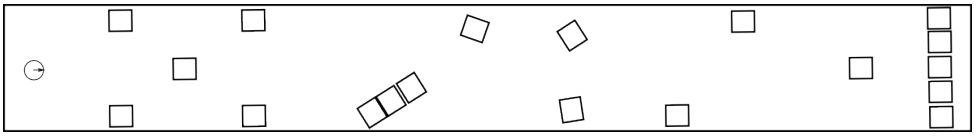

Figura 38: Esboço do segundo teste comparativo.
Fonte: Autoria própria

O segundo teste comparativo foi montado em um corredor longo, similarmente ao primeiro teste avançado, porém com mais situações que possam causar indecisões. Este teste possibilitou a observação da capacidade de decisão, evasão de obstáculos e tempo de ida e volta a partir do ponto de partida, visível na figura 38, para ambos os algoritmos. O algoritmo *fuzzy* foi capaz de percorrer o experimento em 4 minutos e 36 segundos, colidindo suavemente com um dos obstáculos, enquanto que o ED-FCM fez seu percurso em 4 minutos e 2 segundos, colidindo com dois obstáculos que compunham a parede do final do corredor enquanto tentava encontrar a saída do local. A figura abaixo mostra os caminhos percorridos por cada algoritmo, realçando os obstáculos com os quais houve colisão, sendo vermelho para o *Fuzzy* clássico e azul para o ED-FCM.

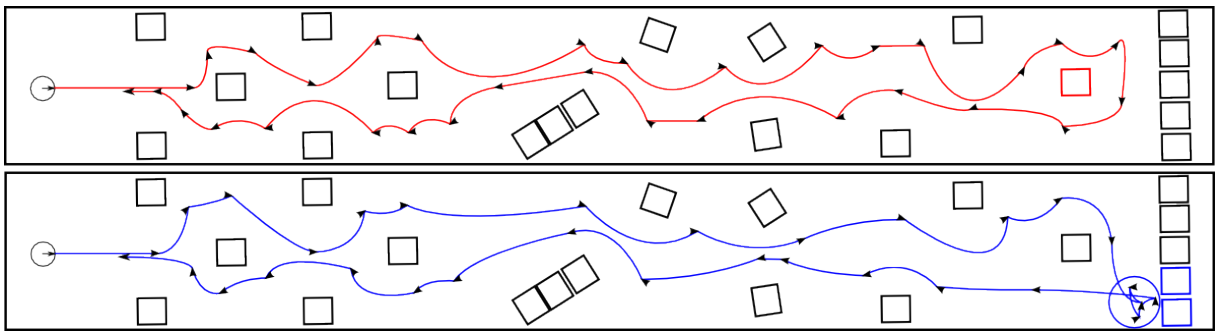

Figura 39: Esboço dos caminhos percorridos no primeiro teste comparativo.
Fonte: Autoria própria

Neste experimento foi notável a diferença de cerca de 14% no tempo total levado pelo algoritmo *fuzzy* para percorrer o experimento em relação ao algoritmo ED-FCM, apesar da colisão com os obstáculos que demarcavam o final da área de testes e da dificuldade do algoritmo ED-FCM em encontrar a saída deste local, circulado em azul na figura 39. Esta dificuldade

foi causada pela mesma situação encontrada pelo algoritmo no experimento anterior, no corredor sem saída. Ao tentar desviar do obstáculo pela esquerda, o algoritmo ED-FCM demorou cerca de 30 segundos para detectar a impossibilidade do desvio pela esquerda e voltar para o caminho correto. Se desconsiderarmos esta situação e os 30 segundos de atraso causados pela mesma, pode-se observar que o algoritmo ED-FCM percorreu o restante do experimento cerca de 23% mais rapidamente que o algoritmo *Fuzzy* clássico, sem pausas prolongadas. O algoritmo *Fuzzy*, apesar de ter colidido suavemente com um obstáculo, demarcado em vermelho na figura 39, mostrou-se mais cauteloso, não apresentando problemas durante o experimento como um todo. A colisão pode ser explicada devido à tentativa de desviar da parede, que fez com que o obstáculo aparecesse subitamente nos sensores do robô. Apenas o parafuso da roda encostou no obstáculo, deslocando a caixa em menos de 5 centímetros.

De uma forma geral, o segundo teste comparativo possibilitou à equipe concluir que ambos os algoritmos de navegação, apesar de algumas diferenças no tratamento de situações específicas e do tempo necessário para realizar o percurso, apresentaram resultados muito similares quanto à trajetória e decisões tomadas ao longo do experimento.

3.3.5 Considerações

Nesta seção, foram apresentadas todas as etapas de elaboração e execução dos testes, bem como todos os ajustes e correções dos algoritmos de navegação desenvolvidos neste projeto. Também foram apresentados os testes comparativos, seus resultados e análise. Foi possível observar que, com exceção à situações específicas, ambos os algoritmos apresentam comportamentos similares, embora o algoritmo baseado em ED-FCM apresente maior agressividade e o algoritmo *Fuzzy* clássico uma cautela maior próximo a obstáculos. A partir destas informações, a equipe concluiu que nenhum dos algoritmos pode ser considerado como "melhor" que o outro, porém, existem diversas vantagens e desvantagens para cada um dos algoritmos, o que pode tornar um mais apropriado que o outro em determinadas situações. A lista a seguir mostra as vantagens e desvantagens observadas pela equipe:

1. Implementação: A implementação inicial do algoritmo *Fuzzy* clássico é mais complexa, exigindo a definição de diversos conjuntos *Fuzzy*, variáveis linguísticas e regras de inferência, enquanto que o ED-FCM exige apenas a definição de conceitos, as relações causais entre eles e suas funções de ativação.

2. Alterações: Após a implementação inicial, o algoritmo *Fuzzy* clássico mostrou-se mais fácil de alterar, quando necessário mudar o comportamento do algoritmo, enquanto que o

ED-FCM é mais complexo de alterar, devido à dificuldade de determinar quais conceitos e relações estão produzindo o comportamento indesejado.

3. Comportamento: O algoritmo *Fuzzy* apresenta maior cautela próximo a obstáculos e maior facilidade de sair de situações com diversos obstáculos próximos, enquanto que o ED-FCM apresenta dificuldade nestas situações, porém maior agilidade em ambientes abertos e com menor número de obstáculos dispersos.

4 CONCLUSÃO

Este capítulo apresenta a conclusão da monografia, na qual serão discutidos os objetivos iniciais e os resultados alcançados, as conclusões sobre o uso de algoritmos de navegação em plataformas robóticas reais, as dificuldades encontradas durante o projeto e as sugestões para trabalhos futuros.

4.1 OBJETIVOS INICIAIS E RESULTADOS ALCANÇADOS

Este projeto surgiu a partir da inspiração em testar, empiricamente, a eficiência de uma metodologia de navegação chamada *Event-Driven Fuzzy Cognitive Maps* (ED-FCM) comparada à metodologia *Fuzzy*, que levou a equipe a desenvolver um trabalho de conclusão de curso com objetivos abrangentes, envolvendo desenvolvimento de *hardware* e *software*. No contexto da navegação robótica, surgiu a necessidade de se utilizar um robô real com a finalidade de se obterem resultados mais significativos, visto que o ED-FCM não havia ainda sido testado em ambientes reais, apenas em um ambiente simulado. O ambiente simulado da tese de (MENDONÇA; ARRUDA; NEVES, 2011) não levou em consideração ruídos dos sensores e diferenças mecânicas dos atuadores do robô. No Bellator, os motores DC tinham uma resposta diferente para um derterminado nível de tensão, sendo necessário implementar um ajuste via sistema microcontrolado para manter as velocidades das rodas em um mesmo nível. Outros problemas inerentes ao uso de uma plataforma real puderam ser observados, como a mudança de comportamento dos algoritmos quando a bateria dos motores foi trocada. Com a nova bateria, as ações do robô tornaram-se mais rápidas, exigindo que os algoritmos fossem modificados para manter o movimento suave do robô. Com a finalidade de observar esses efeitos nos algoritmos *Fuzzy* e ED-FCM, foi elaborado um projeto cujo escopo também foi reconstruir e adequar uma plataforma robótica previamente disponível, que é descrita na seção 2.1, contudo, que não estava em condições de uso imediato. A equipe procedeu com testes em laboratório de eletrônica afim de avaliar as condições iniciais do robô, conforme foi descrito na seção 3.1.1. Uma análise de *software* foi efetuada e o código original do microcontrolador C8051F340DK, disponibilizado como parte integrante do robô Bellator (MARIN et al., 2010), foi avaliado e

reconfigurado de acordo com as necessidades do projeto, o que é descrito em detalhes na seção 3.1.4. Havendo necessidade de implementação de *hardware*, a equipe projetou e construiu uma placa de roteamento para alimentar os sensores e *encoders*, assim como tratar os sinais destes e os de PWM, como é descrito na seção 3.1.6. Finalmente, o *hardware* acoplado foi configurado e o *software* responsável pela execução dos algortimos de navegação foi desenvolvido, conforme a seção 3.1.7. Com isso, a equipe foi capaz de obter uma plataforma robótica apropriada para o restante do projeto e possíveis utilizações futuras, concluindo a primeira parte do projeto.

Estando a plataforma robótica funcional, seguiram-se o projeto e implementação dos algoritmos de navegação propostos. A Lógica *Fuzzy* foi estudada e apresentada na seção 2.4 e a metodologia *Event-Driven Fuzzy Congnitive Maps* (ED-FCM) foi estudada e apresentada na seção 2.5, proporcionando à equipe a fundamentação teórica necessária para o desenvolvimento e aprimoramento dos algoritmos. Com a teoria fundamentada, a equipe projetou os algoritmos estudados e implementou-os na linguagem C++, compilando-os para execução na plataforma Linux embarcada da TS-7260, como descrito nas seções 3.2.1 e 3.2.2. Após a implementação, a equipe submeteu os algoritmos a uma série de testes básicos, denominada Testes Iniciais, que serviram para fornecer a primeira realimentação do projeto dos algoritmos, que permitiu o ajuste inicial dos mesmos e pode ser lido na seção 3.3.2. Finalizados esses testes, foram elaborados testes complexos, denominados Testes Avançados, para estressar os sistemas de navegação propostos e fornecer a segunda realimentação do projeto dos algoritmos, conforme foi descrito na seção 3.3.3, e que permitiu o aprimoramento dos algoritmos até a versão definitiva utilizada para obtenção dos resultados finais. Finalmente, após os testes avançados, os algoritmos resolviam problemas complexos de navegação, como o corredor sem saída e o problema de decisão quando dois obstáculos laterais e um frontal eram colocados diante do robô, e poderiam ser usados nos testes finais, que forneceram os dados para a análise de resultados e foram denominados Testes Comparativos, conforme foi descrito na seção 3.3.4. Com isso a equipe concluiu a segunda parte, que foi composta pelo projeto e implementação dos algoritmos de navegação e a elaboração e execução de uma metodologia de testes comparativos entre os algoritmos.

A equipe conclui esta monografia justificando que os objetivos descritos na introdução, seção 1.2, foram alcançados e estão de acordo com os requisitos mínimos de um curso de Engenharia de Computação.

4.2 USO DE ALGORITMOS DE NAVEGAÇÃO EM PLATAFORMAS ROBÓTICAS REAIS

Através deste trabalho, concluiu-se que os algoritmos de navegação apresentam uma solução para os problemas de navegação em ambientes reais, isto é, as metodologias *Fuzzy* e *ED-FCM* permitiram implementar um controlador capaz de guiar um robô autônomo de maneira que este pudesse desviar obstáculos e evitar a colisão, que é um dos problemas de navegação, conforme explica (FRACASSO; COSTA, 2005). O uso de um robô real ao invés de uma simulação computacional gerou resultados que contribuem para o estudo proposto. Um robô real apresenta diversos problemas práticos que em uma simulação computacional são dificilmente visualizados. O primeiro problema é a interferência que os sensores de distância podem sofrer apresentando ruídos na saída. Os sensores de distância não são perfeitos e apresentam respostas diferentes dependendo da forma geométrica das superfícies e, como é o caso do sensor infravermelho, até mesmo da cor das superfícies (SHARPCORPORATION, 2006). Outro problema é que os sensores apresentam uma faixa de operação, ou seja, uma distância mínima e máxima possível de serem medidas. Isso significa que os algoritmos devem ser projetados para operar de acordo com essa faixa. No caso do sensor infravermelho utilizado, o 2Y0A02F98 (explicado na seção 2.2.1), a faixa de operação suportada é de 20 a 150 centímetros. Dentro dessa faixa, o algoritmo deve gerar necessariamente um comando, caso contrário, o robô perderá o controle pois o sensor fornecerá leituras inconsistentes, principalmente quando a distância for menor que o limite mínimo. Além dos problemas associados aos sensores, a implementação em uma plataforma real demonstrou que as partes mecânicas do robô devem ser incluídas no projeto do sistema de navegação. O robô Bellator possui dois motores DC cuja resposta de velocidade de rotação não é igual quando aplicado um mesmo nível de tensão, isto é, quando aplicado um mesmo nível de PWM aos motores, estes não giram na mesma velocidade. Isso implicou a implementação de um mecanismo de ajuste que permitisse que as duas rodas girassem na mesma velocidade quando necessário. Esse mecanismo é descrito na seção 3.1.9. Outro problema inerente ao uso da plataforma real foi o que aconteceu quando a bateria dos motores foi trocada. Com a bateria nova, a corrente que alimentava os motores era maior e por isso as ações produzidas pelo robô tornaram-se mais intensas. Para manter o movimento do robô suave e evitar colisões por causa de curvas acentuadas, foi necessário reajustar os algoritmos. Concluiu-se que o nível de tensão e carga do sistema de alimentação dos motores do robô deve ser levado em conta no projeto dos algoritmos de navegação.

As conclusões sobre os algoritmos explorados nesse trabalho são que ambos, o algoritmo *Fuzzy* e o ED-FCM, apresentaram comportamentos semelhantes, conforme pode ser observado

nas trajetórias das figuras 37 e 39, e são indicados para utilização em sistemas de navegação robótica. Existem peculiaridades de cada um que podem ser levantadas. O algoritmo ED-FCM apresentou ações rápidas e curvas mais acentuadas que o algoritmo *fuzzy*. Isso porque as sigmóides utilizadas na ativação dos conceitos variam rapidamente do nível 0 para 1, significando que pequenas variações nas distâncias dos sensores produzem grandes variações nos conceitos de nível e, portanto, na potência dos motores. Esse efeito poderia ser reduzido suavizando as sigmóides ou criando mais níveis para os pesos das relações causais. Outra característica do ED-FCM é que essa implementação não necessita da criação de um banco de regras, como é o caso do algoritmo *fuzzy*, e as decisões são calculadas por meio de fórmulas matemáticas e mapeiam a leitura dos sensores diretamente em ações. Também a implementação do ED-FCM não necessitou do uso de uma biblioteca computacional específica, como foi o caso do *fuzzy*, determinando simplicidade de implementação e menor tamanho de código escrito. Por isso, concluiu-se que um ED-FCM poderia ser implementado em uma camada de mais baixo nível, um processador digital de sinais, por exemplo, que desse suporte a operações matemáticas de exponenciação, divisão, multiplicação e soma de números de ponto flutuante.

Alguns problemas de navegação foram melhor resolvidos pelo algoritmo *fuzzy*, explicando esse fato porque a base de regras permite maior número de ações influenciando a saída do algoritmo, que produz um comportamento mais abrangente e diversificado. O ED-FCM, por outro lado, tem a limitação de apresentar uma quantidade reduzida de conceitos, sendo que o projeto de um mapa com mais conceitos poderia implicar maior complexidade computacional. Um dos problemas observados na depuração do ED-FCM é que o processo de calibração dos pesos das relações causais pode gerar instabilidades. Nenhum método de aprendizado de máquina foi utilizado e os pesos foram determinados manualmente com o procedimentos de tentativa e erro. Por exemplo, a alteração de um peso determinava a mudança do comportamento do robô e, como a antecipação dessa mudança era difícil de ser visualizada, geravam-se resultados indesejados. Por isso o processo de ajuste do ED-FCM foi mais demorado que o do algorimo *fuzzy*. Concluiu-se que o ED-FCM gera resultados mais rapidamente que o *fuzzy* em termos de implementação, entretanto, a manutenção do ED-FCM é mais demorada e complexa que do *fuzzy*. Vale ressaltar que ambos os algoritmos tem vantagens e desvantagens um sobre o outro. O algoritmo *fuzzy* necessita de uma biblioteca específica que implemente a base de regras, necessita da formulação das funções de pertinência e das regras *fuzzy*, tendo como desvantagens maior tamanho de código e tempo de implementação, entretanto, estando esses elementos concluídos, o processo de manutenção do sistema é simples porque os resultados das alteraçãos podem ser antecipados com maior facilidade. O ED-FCM apresenta maior simplicidade de implementação e menor tamanho de código porque depende da definição dos conceitos e das

relações causais, sendo que os conceitos são ativados através de fórmulas matemáticas. Desse modo, os resultados da implementação de um ED-FCM são mais rápidos, mas a manutenção do sistema é mais complexa porque o efeito da alteração das propriedades do ED-FCM é mais dificilmente visualizado e antecipado. Os dois algoritmos são indicados para uso em uma plataforma robótica real, mesmo apesar dos problemas inerentes a um robô real, o *fuzzy* e o ED-FCM geraram comportamentos adequados de navegação e, cabe salientar, que os comportamentos observados foram previstos no projeto. Isto é, os algoritmos apresentam um comportamento controlável, permitindo projetar sistemas que podem ter usos inclusive mais específicos que o apresentado neste trabalho.

4.3 DIFICULDADES ENCONTRADAS

Os problemas encontrados na execução do projeto estão associados ao escopo abrangente do mesmo, o qual envolveu o desenvolvimento de *hardware* e *software* em um projeto integrador. A subdivisão do projeto em diversos objetivos, sendo um pré-requisito para o outro, foi inevitável para alcançar os resultados finais. A equipe encontrou dificuldades durante a reconstrução do robô, configuração da placa TS-7260, testes integrados de funcionamento do robô, implementação dos algoritmos, execução dos testes dos algoritmos e análise de resultados.

Durante a reconstrução, os componentes eletrônicos foram testados isoladamente, com risco de existência de componentes danificados, o que representaria atrasos no projeto. A placa TS-7260 apresentou complexidade para ser configurada pois não houve um técnico disponível para auxiliar a equipe, a qual teve que aprender a trabalhar com esse *hardware*. Nos testes de integração da C8051F340DK e da TS-7260, que determinaram o funcionamento da plataforma robótica, foram exigidos processos de depuração integrados, nos quais os problemas foram isolados e corrigidos repetidas vezes. A implementação dos algoritmos até a versão final, que foi utilizada nos testes comparativos, foi realizada paralelamente aos testes básicos e avançados, nas quais os problemas de navegação foram detectados, isolados e corrigidos repetidas vezes. A metodologia de testes escolhida foi elaborada pela equipe e foram efetuados vários experimentos com registro em vídeo até que se atingessem os resultados finais. Tendo com base os vídeos gravados e a experiência em campo observada, a equipe precisou analisar os resultados, discutí-los e extrair conclusões para finalizar o projeto.

4.4 TRABALHOS FUTUROS

Para trabalhos futuros utilizando a plataforma Bellator reconstruída, a equipe recomenda combinar sensores de ultrassom com sensores infravermelhos, pois os sensores de ultrassom apresentam uma faixa de operação cuja distância mínima é menor que a do infravermelho, podendo capturar distâncias de 2 cm, como o sensor SRF06 (DEVANTECHELETRONIC, 2012). Atualmente a distância mínima suportada pelo sistema de navegação é de 15 cm, com os sensores de ultrassom, os algoritmos poderiam operar em uma faixa mais abrangente. Outra sugestão é introduzir ao sistema uma realimentação por bússola pois nesse projeto a realimentação odométrica fornecida pelos *encoders* é utilizada para ajustar a velocidade das rodas e não faz uma interpretação da direção do robô.

Para tornar o robô seguro para o manuseio, sugere-se a fixação dos sensores parafusando-os no chassi do Bellator e acoplando uma carcaça que proteja os circuitos microcontrolados. A equipe também recomenda a reconstrução da placa de roteamento utilizando um método industrial para confeccção de placas de circuito impresso.

Para os sistemas de navegação, um trabalho futuro de grande riqueza seria introduzir ao sistema a capacidade de interpretar a posição do robô em relação a um referencial. Com isso, o robô seria capaz de resolver problemas nos quais este deve partir de um ponto inicial no espaço a um ponto final, guiando-se pelos sensores para evitar colisões e realimentar-se por um sistema de posicionamento para corrigir a trajetória.

Outro trabalho, produto deste, seria introduzir ao sistema uma memória a qual pudesse mapear os obstáculos capturados pelos sensores do robô, assim sendo, produzir-se-ia um artefato autônomo capaz de mapear terrenos. Outro aprimoramento da plataforma seria implementar um sistema de controle remoto, na qual uma base remota pudesse pilotar o robô via *joystick* - esta funcionalidade poderia ser usada para treinar o sistema de controle ED-FCM. Finalmente, a equipe sugere um projeto futuro no qual seja implementado um sistema de visão computacional por câmera de vídeo. Sobre os algoritmos, a equipe sugere que os estudos empíricos sejam continuados e que nos próximos trabalhos sejam apresentadas medidas quantitativas de desempenho para ambos algoritmos, *fuzzy* e ED-FCM, justificando a importância que essas medidas teriam na escolha de uma das abordagens no projeto de um sistema de navegação.

REFERÊNCIAS

AGILENTTECHNOLOGIES. **Small Optical Encoder Modules HEDS-9700 Series**. 2002. Disponível em: <http://www.digchip.com/datasheets/parts/datasheet/021/QEDS-9871-pdf.php>. Acesso em: 28 de Fevereiro de 2012.

ARM, E. **TS-7260 Datasheet**. 2009. Disponível em: <http://www.embeddedarm.com/documentation/ts-7260-datasheet.pdf>. Acesso em: 16 de Janeiro de 2012.

ARM, E. **TS-7260 Hardware Manual**. 2012. Disponível em: <http://www.embeddedarm.com/documentation/ts-7260-manual.pdf>. Acesso em: 16 de Janeiro de 2012.

CADSOFT. **CadSoft EAGLE Freeware Version**. 2012. Disponível em: <http://www.cadsoftusa.com/downloads/freeware/?language=en>. Acesso em: 05 de Março de 2012.

DEVANTECHELETRONIC. **SRF06 Ultra-Sonic Ranger with 4-20mA Output**. 2012. Disponível em: <http://www.robot-electronics.co.uk/htm/srf06tech.htm>. Acesso em: 29 de Fevereiro de 2012.

DOCTORROBOT. **X80 Robot**. 2012. Disponível em: <http://www.drrobot.com/products_item.asp?itemNumber=x80>. Acesso em: 06 de Fevereiro de 2012.

ELETRONICAORG. **Apostila Sobre Modulação PWM**. 2012. Disponível em: <http://www.eletronica.org/arq_apostilas/apostila_pwm.pdf>. Acesso em: 04 de Março de 2012.

FABRO, J. A. **Grupos neurais e sistemas nebulosos: aplicação a navegação autonoma**. Dissertação (Mestrado) — Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica, Fevereiro 1996.

FRACASSO, P. T.; COSTA, A. H. Navegação reativa de robôs móveis autônomos utilizando lógica nebulosa com regras ponderadas. **VII SBAI/ II IEEE LARS**, Setembro 2005.

GIT. **Git - Fast Version Control System**. 2012. Disponível em: <http://git-scm.com/>. Acesso em: 06 de Março de 2012.

GITHUB. **GitHub**. 2012. Disponível em: <https://github.com/>. Acesso em: 06 de Março de 2012.

GITHUB. **Help.GitHub**. 2012. Disponível em: <http://help.github.com/>. Acesso em: 06 de Março de 2012.

KOSKO, B. Fuzzy cognitive maps. **International Journal Man-Machine Studie**, v. 24, n. 1, p. 65–75, 1986.

MAMDANI, E. H.; ASSILIAN, S. An experiment in linguistic synthesis with a fuzzy logic controller. **Int. J. Hum.-Comput. Stud.**, Elsevier, v. 51, n. 2, p. 135–147, 1999.

MARIN, A. J. et al. **Desenvolvimento do Projeto Bellator**. Curitiba, 2010.

MENDONÇA, M. **Uma Contribuição ao Desenvolvimento de Sistemas Inteligentes Utilizando Redes Cognitivas Dinâmicas**. Doutorado — Universidade Tecnológica Federal do Paraná, 2011.

MENDONÇA, M.; ARRUDA, L.; NEVES, F. Autonomous navigation system using event driven-fuzzy cognitive maps. © **Springer Science+Business Media**, Outubro 2011.

PEDRYCZ, W.; GOMIDE, F. A. C. **Fuzzy Systems Engineering - Toward Human-Centric Computing**. [S.l.]: Wiley, 2007. I-XXII, 1-526 p. ISBN 978-0-471-78857-7.

PHILIPS. **74LS244 Octal buffer,line driver; 3-state**. Disponível em: <http://www.learn-c.com/74ls244.pdf>. Acesso em: 26 de Fevereiro de 2012.

SEMICONDUCTOR, N. **LM317 - 3-Terminal Adjustable Regulator**. 2012. Disponível em: <http://www.national.com/mpf/LM/LM317.html>. Acesso em: 21 de Março de 2012.

SHARPCORPORATION. **GP2Y0A02F98YK Datasheet**. 2006. Disponível em: <http://www.mindsensors.com/index.php?module=documents&JAS_DocumentManager_op=downloadFile&JAS_File_id=335>. Acesso em: 1 de Agosto de 2011.

SILICONLABS. **C8051F340DK Datasheet**. 2006. Disponível em: <http://datasheet.octopart.com/C8051F340DK-Silicon-Laboratories-datasheet-9512.pdf>. Acesso em: 1 de Agosto de 2011.

STYLIOS, C. D.; GROUMPOS, P. P. Fuzzy cognitive maps in modeling supervisory control systems. **Journal of Intelligent and Fuzzy Systems**, v. 8, n. 1, p. 83–98, 2000.

TECHNOLOGICSYSTEMS. **Linux for ARM on TS-7000 Embedded Computers**. 2012. Disponível em: <http://www.embeddedarm.com/software/software-arm-linux.php>. Acesso em: 27 de Fevereiro de 2012.

TEXASINSTRUMENTS. **LM124,LM224,LM324,LM2902 Low Power Quad Operational Amplifiers**. Disponível em: <http://www.national.com/ds/LM/LM124.pdf>. Acesso em: 25 de fevereiro de 2012.

ZADEH, L. A. Fuzzy sets. **Information and Control**, v. 8, n. 3, p. 338–353, 1965.

APÊNDICE A – REFERÊNCIA DE MONTAGEM DO ROBÔ

A figura 40 é uma composição de partes de fotos do robô Bellator em seu estágio final de montagem, ao final do projeto. A alimentação do motor em ”2”, bem como a ligação dos mesmos através da chave indicada abaixo de ”9”na visão geral do robô, e a ligação dos outros componentes em ”3”só deverão ser realizados após verificar todas as outras conexões do robô conforme indicadas na figura. Ligar incorretamente alguma das conexões pode levar à queima de componentes. Os cabos *flat* devem ser conectados de modo a alinhar o *GND* da placa de roteamento com o *GND* da C8051F340 no *port* correspondente. Também deve-se ficar atento ao nível de carga das baterias, que deverá ser de 10Volts ou mais. Após realizadas todas as conexões, a TS irá inicializar (cerca de 5 minutos) e poderá ser, enfim, utilizada conforme documentação em 3.1.7. A lista a seguir relaciona cada item numerado na figura com sua respectiva documentação ao longo do documento.

- 1 e 3) Especificação da C8051F340DK em 2.2.3, TS 7260 em 2.2.4 e placa de roteamento em 2.2.5.
- 2) Especificação dos motores em 2.2.6.
- 4 a 9) Diagramas esquemáticos e *interface* da placa de roteamento com a C8051F340 em 3.1.6.
- Especificação dos Sensores e Encoders em 2.2.1 e 2.2.2.
- 10) O pendrive contém o código executado na TS7260, descrito em 3.1.7.
- 11) As pontes H são alimentadas pelos conectores de alimentação do motor.

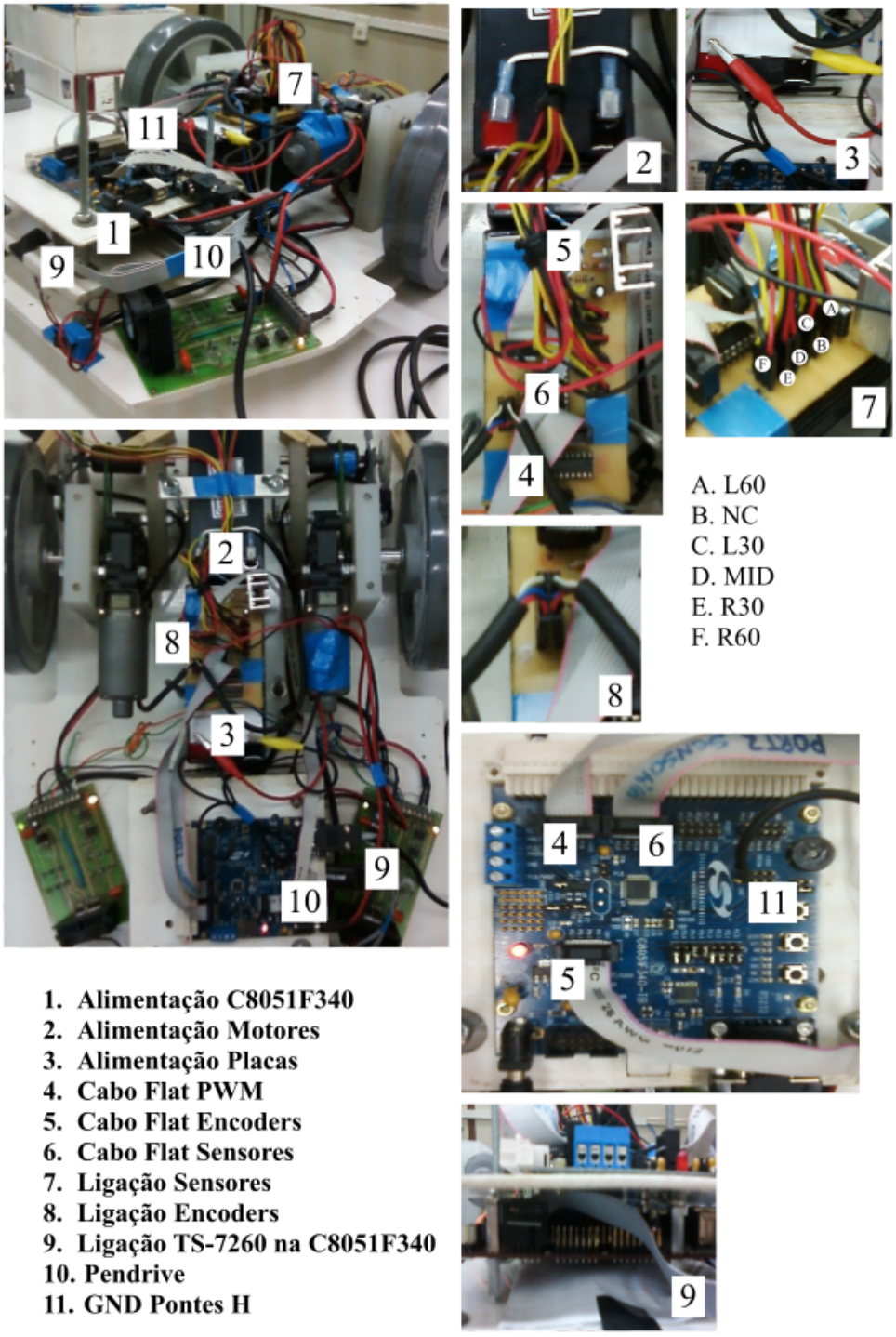

Figura 40: Referência de Montagem do Robô.

Fonte: Autoria própria