

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. 96 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, Reconhecimento de Padrões, 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. 96 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, Pattern Recognition, Machine Learning, Neural Network, Principal Components Analysis, Fourier Descriptors

LISTA DE FIGURAS

FIGURA 1 – IMAGENS COM E SEM RUÍDO E SEUS RESPECTIVOS HISTOGRAMAS	15
FIGURA 2 – IMAGENS LIMARIZADAS PELO MÉTODO DE <i>OTSU</i> E SEUS RESPECTIVOS HISTOGRAMAS	16
FIGURA 3 – EXEMPLOS DE ELEMENTOS ESTRUTURANTES.	17
FIGURA 4 – IMAGEM BINÁRIA, ELEMENTO ESTRUTURANTE E OPERAÇÃO DE B EM A	18
FIGURA 5 – EXEMPLO DE EROÇÃO DO CONJUNTO A POR B	19
FIGURA 6 – EXEMPLO DE DILATAÇÃO DO CONJUNTO A POR B	19
FIGURA 7 – CONJUNTO A	20
FIGURA 8 – CONJUNTO $A \circ B$	21
FIGURA 9 – CONJUNTO $A \bullet B$	21
FIGURA 10 – CONJUNTO A , DISCO MÁXIMO CENTRADO EM DIVERSOS PONTOS Z E ESQUELETO COMPLETO.	22
FIGURA 11 – EXEMPLO DE ESQUELETONIZAÇÃO E PODA.	23
FIGURA 12 – IMAGENS DE EXEMPLO E SUAS RESPECTIVAS MATRIZES DE CO-VARIÂNCIA.	25
FIGURA 13 – IMAGENS DE EXEMPLO E SEUS RESPECTIVOS AUTOVETORES.	26
FIGURA 14 – IMAGENS NORMALIZADAS.	27
FIGURA 15 – REPRESENTAÇÃO DA ÁREA PARA CÁLCULO DO MOMENTO DE INÉRCIA.	29
FIGURA 16 – IMAGEM ORIGINAL E IMAGENS RECONSTITUÍDAS UTILIZANDO N DESCRITORES DE <i>FOURIER</i>	31
FIGURA 17 – MODELO DE <i>PERCEPTRON</i>	33
FIGURA 18 – GRÁFICO DA FUNÇÃO DE ATIVAÇÃO SIGMOIDE.	34
FIGURA 19 – MODELO DE <i>MULTI-LAYER FEEDFORWARD NETWORK</i>	35
FIGURA 20 – ESTRUTURAS DE REDES NEURAIS E SUA RESPECTIVA CAPACIDADE DE DECISÃO.	36
FIGURA 21 – VISÃO GERAL DO SISTEMA DESENVOLVIDO.	41
FIGURA 22 – PRIMEIRA ETAPA DA SEGMENTAÇÃO.	43
FIGURA 23 – SEGUNDA ETAPA DA SEGMENTAÇÃO.	44
FIGURA 24 – PRIMEIRA ETAPA DA DESCRIÇÃO.	45
FIGURA 25 – PRIMEIRA ETAPA DA CLASSIFICAÇÃO.	47
FIGURA 26 – SEGUNDA ETAPA DA DESCRIÇÃO.	48
FIGURA 27 – SEGUNDA ETAPA DA CLASSIFICAÇÃO.	49
FIGURA 28 – CURVA REAL DOS SENSORES E INTERPOLAÇÃO POLINOMIAL .	54
FIGURA 29 – DIAGRAMA ESQUEMÁTICO DA PLACA DE ROTEAMENTO.	59
FIGURA 30 – ROTEAMENTO FINAL PRODUZIDO PELA EQUIPE.	60
FIGURA 31 – RESULTADO DO PROCESSO DE CONFECÇÃO DA PLACA DE ROTEAMENTO.	61
FIGURA 32 – DIAGRAMA EM BLOCOS <i>SOFTWARE</i> TS-7260	64
FIGURA 33 – ESTRUTURA DE UMA VARIÁVEL LINGUÍSTICA NO FLIE.	66

FIGURA 34 – DEFINIÇÃO DAS FUNÇÕES DE PERTINÊNCIA PERTO, MÉDIO E LONGE.	67
FIGURA 35 – DEFINIÇÃO DAS FUNÇÕES DE PERTINÊNCIA LENTO, MÉDIO E RÁPIDO.	68
FIGURA 36 – DEFINIÇÃO DAS FUNÇÕES DE PERTINÊNCIA DE DIREÇÃO	68
FIGURA 37 – ESTRUTURA DE UMA REGRA DE INFERÊNCIA NO FLIE.	69
FIGURA 38 – ED-FCM PROPOSTO	74
FIGURA 39 – CONFIGURAÇÃO DO PRIMEIRO TESTE INICIAL.	78
FIGURA 40 – CONFIGURAÇÃO DO SEGUNDO TESTE INICIAL.	79
FIGURA 41 – CAMINHO PERCORRIDO PELO ALGORITMO <i>FUZZY</i> (VERMELHO) E O CAMINHO ESPERADO (VERDE).	79
FIGURA 42 – ILUSTRAÇÃO DO PRIMEIRO TESTE AVANÇADO.	80
FIGURA 43 – ILUSTRAÇÃO DO SEGUNDO TESTE AVANÇADO.	81
FIGURA 44 – ILUSTRAÇÃO DO CORREDOR SEM SAÍDA.	82
FIGURA 45 – ESBOÇO DO CAMINHO PERCORRIDO PELO <i>FUZZY</i> (VERMELHO) E ED-FCM(AZUL).	83
FIGURA 46 – ESBOÇO DO SEGUNDO TESTE COMPARATIVO.	84
FIGURA 47 – ESBOÇO DO CAMINHO PERCORRIDO PELO <i>FUZZY</i> (VERMELHO) E ED-FCM(AZUL).	84
FIGURA 48 – REFERÊNCIA DE MONTAGEM DO ROBÔ	96

LISTA DE TABELAS

TABELA 1	– DESCRITOR PARA PRIMEIRA ETAPA	46
TABELA 2	– VALORES DE CONVERSÃO DOS SENSORES X DISTÂNCIA	53
TABELA 3	– FUNÇÕES DE PERTINÊNCIA E CONJUNTOS FUZZY DOS SENSORES	67
TABELA 4	– REGRAS DE INFERÊNCIA - ALGORITMO FUZZY	70
TABELA 5	– RELAÇÕES CAUSAIS DO CONTROLADOR ED-FCM PROPOSTO (ESTADO = FRENTE)	75
TABELA 6	– RELAÇÕES CAUSAIS DO CONTROLADOR ED-FCM PROPOSTO (ESTADO = TRÁS)	75
TABELA 7	– VALOR NUMÉRICO DOS PESOS	75
TABELA 8	– VALOR NUMÉRICO DOS LIMIARES L0 E L1	75

LISTA DE SIGLAS

LISTA DE SÍMBOLOS

SUMÁRIO

1 INTRODUÇÃO	11
1.1 MOTIVAÇÃO E JUSTIFICATIVA	11
1.2 OBJETIVOS	11
1.3 METODOLOGIA	12
1.4 ESTRUTURA DO DOCUMENTO	12
2 FUNDAMENTAÇÃO TEÓRICA	13
2.1 SEGMENTAÇÃO	13
2.1.1 Subtração de fundo	13
2.1.2 Limiarização	14
Limiarização global pelo método de <i>Otsu</i>	15
2.1.3 Operações morfológicas básicas	16
Erosão	18
Dilatação	19
Abertura e Fechamento	20
2.1.4 Esqueletonização	21
Poda (<i>Pruning</i>)	23
2.1.5 Normalização	24
Matriz de covariâncias	24
Autovetores e autovalores	25
Análise de componentes principais	25
Normalização	26
2.2 DESCRIÇÃO	27
2.2.1 Propriedades de regiões	27
Área	28
Área convexa	28
Área Preenchida	28
Solidez	28
Extensão	28
Excentricidade	28
Tamanho do eixo principal e eixo secundário	29
Número de Euler	29
2.2.2 Descritores de <i>fourier</i>	29
2.3 CLASSIFICAÇÃO	32
2.3.1 Redes neurais	32
Treinamento de <i>perceptrons</i>	36
Treinamento de <i>multi-layer perceptrons</i>	38
2.4 CONSIDERAÇÕES	40
3 DESENVOLVIMENTO	41
3.1 SEGMENTAÇÃO: ETAPA 1	42
3.2 SEGMENTAÇÃO: ETAPA 2	44
3.3 DESCRIÇÃO: ETAPA 1	45

3.4 CLASSIFICAÇÃO: ETAPA 1	46
3.5 DESCRIÇÃO: ETAPA 2	48
3.6 CLASSIFICAÇÃO: ETAPA 2	49
3.7 CLASSIFICAÇÃO: TREINAMENTO	50
3.8 CONSIDERAÇÕES	50
3.9 RECONSTRUÇÃO DA PLATAFORMA ROBÓTICA BELLATOR	50
3.9.1 Teste dos Componentes	51
3.9.2 Levantamento da curva dos sensores	52
3.9.3 Repositório de trabalho	54
3.9.4 Código do microcontrolador C8051F340	55
3.9.5 Protocolo de comunicação	56
3.9.6 Placa de Roteamento	57
3.9.7 Placa TS-7260	61
3.9.8 Configuração da TS-7260	61
3.9.9 Software da TS-7260	63
3.9.10 Considerações	64
3.10 ALGORITMOS DE NAVEGAÇÃO	65
3.10.1 Algoritmo de Navegação Fuzzy	65
Variáveis Linguísticas	65
Regras de Inferência	69
Considerações	71
3.10.2 Algoritmo de Navegação Baseado em ED-FCM	71
Conceitos	72
Relações Causais	74
Ativação dos Conceitos	76
Considerações	76
3.11 TESTES E ANÁLISE DE RESULTADOS	77
3.11.1 Elaboração dos Testes	77
3.11.2 Testes Iniciais	78
3.11.3 Testes Avançados	80
3.11.4 Testes Comparativos	82
3.11.5 Considerações	85
4 CONCLUSÃO	87
4.1 OBJETIVOS INICIAIS E RESULTADOS ALCANÇADOS	87
4.2 USO DE ALGORITMOS DE NAVEGAÇÃO EM PLATAFORMAS ROBÓTICAS	
REAIS	89
4.3 DIFICULDADES ENCONTRADAS	91
4.4 TRABALHOS FUTUROS	92
REFERÊNCIAS	93
APÊNDICE A – REFERÊNCIA DE MONTAGEM DO ROBÔ	95

1 INTRODUÇÃO

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 (LIN, 2008), (LIN, 2007) a inspeção do acabamento em objetos metálicos (ZHENG; KONG; NAHAVANDI, 2002), a produção de circuitos impressos (MAR; YARLAGADDA; FOOKES, 2011), (ZENG; MA; ZHENG, 2011), 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.

A grande quantidade de objetos distintos que podem estar em uma mesma linha de produção ou linha de montagem podem confundir os trabalhadores, estando estes portanto sujeitos a erros e falhas. A identificação automática de objetos, assim como a identificação automática de sua posição, tendem a diminuir o erro cometido na classificação ou reposicionamento manual para cada objeto. Diminuindo o erro de classificação diminui se consequentemente o custo gerado por uma linha de montagem parada e dispensa-se a necessidade de parte dos operadores.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

A principal motivação para o desenvolvimento deste projeto é a posterior aplicação em uma linha de produção real. Espera-se que a utilização do projeto reduza os custos operacionais da linha de produção, diminua a margem de erro e aumente a velocidade da produção. A aplicação específica do projeto não será detalhada devido ao posterior interesse comercial do autor do projeto.

1.2 OBJETIVOS

O objetivo geral 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 ob-

jetos previamente conhecidos pelo sistema utilizando-se técnicas de aprendizagem de máquina.

TODO: ACRESCENTAR CASO SEJA FEITA A IMPLEMENTAÇÃO EM C++

Para tanto, temos como objetivos específicos três objetivos que possibilitarão o cumprimento do objetivo geral. O primeiro objetivo específico é analisar e aplicar técnicas de segmentação de imagens, como algoritmos de detecção de bordas, algoritmos de limiarização, segmentação baseada em crescimento de regiões. O segundo objetivo é analisar e aplicar técnicas de descrição de imagens, como algoritmos de descrição de contornos, descrição de regiões, entre outros. O terceiro objetivo é o emprego de uma técnica de aprendizagem de máquina, como redes neurais ou máquina de vetores de suporte, que possibilite a classificação de objetos previamente conhecidos pelo sistema.

1.3 METODOLOGIA

TODO: REMOVER CASO NÃO SEJA FEITA A IMPLEMENTAÇÃO EM C++

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.

1.4 ESTRUTURA DO DOCUMENTO

TODO: REVISAR DEPOIS DE PRONTO

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 apresenta a fundamentação teórica, ou seja, o estudo das técnicas e algoritmos utilizados durante o trabalho. O capítulo divide-se em três seções.

A seção 2.1 apresenta técnicas e algoritmos que serão úteis para realizar a segmentação de imagens. A segmentação de imagens consiste em subdividir a imagem original em regiões ou objetos até que se tenha identificado a região ou objeto de interesse.

A seção 2.2 apresenta técnicas e algoritmos para realizar a descrição de imagens. Após a segmentação de uma imagem é necessário representar a imagem para que se possa processá-la. Existem duas abordagens para esse problema (GONZALEZ; WOODS, 2006): representação por meio de características externas da região, como bordas, ou representação por meio de características internas, como textura. A representação por características externas normalmente é utilizada quando se está interessado na forma da região. Já a representação por características internas é utilizada quando o foco está na cor ou textura da região. Nessa seção aborda-se principalmente a representação por meio de características externas de imagens.

Finalmente a seção 2.3 apresenta uma visão geral de reconhecimento de padrões utilizando redes neurais. O reconhecimento de padrões é feito tomando como base descritores, que podem ser obtidos a partir de técnicas como as vistas na seção 2.2.

2.1 SEGMENTAÇÃO

Esta seção apresenta técnicas e algoritmos para realizar a segmentação de imagens, tais como técnicas de subtração de fundo, limiarização, operações morfológicas, esqueletonização e análise de componentes principais.

2.1.1 Subtração de fundo

Técnicas de subtração de fundo são amplamente empregadas para a detecção de movimento em sequências de imagens de vídeo. A detecção do movimento se dá pela subtração da imagem

de referencia chamada "fundo", ou em inglês *background image*, da imagem a ser segmentada. A imagem de fundo não deve possuir objetos em movimento e deve estar sempre atualizada com relação à variações de iluminação (PICCARDI, 2004).

O método mais simples de subtração de fundo consiste em subtrair a imagem de fundo da imagem a ser segmentada e em seguida aplicar um limiar no resultado. Pode-se resumir esse método na equação 1. Onde, $D(t+1)$ é a imagem segmentada, $V(x,y,t+1)$ é a imagem de entrada, $V(x,y,t)$ é a imagem de referencia, ou fundo e Th é um limiar, que pode ser escolhido conforme os métodos apresentados na seção 2.1.2.

$$D(t+1) = |V(x,y,t+1) - V(x,y,t)| > Th \quad (1)$$

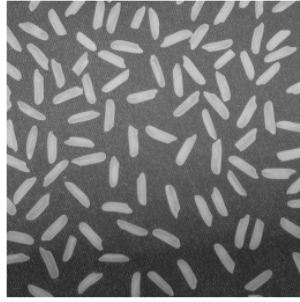
Existem outros métodos mais complexos, como *Running Gaussian average*, *Temporal median filter*, entre outros, que geram melhores resultados com fundos que não são completamente estáticos (PICCARDI, 2004), porém não são de interesse para este projeto.

2.1.2 Limiarização

Pela simplicidade na implementação e pela baixa complexidade computacional os algoritmos de limiarização são bastante utilizados na segmentação de imagens. Os métodos de limiarização consistem na partição da imagem baseado na intensidade dos pixels. Caso a partição seja feita com base em um único limiar de intensidade a limiarização se diz global. Caso a partição seja feita levando em conta os valores de intensidade dos pixels em uma região de vizinhança de um certo pixel, então o método é chamado de limiarização local (GONZALEZ; WOODS, 2006). A equação 2 descreve esse processo, $g(x,y)$ é a imagem limiarizada, $f(x,y)$ é a imagem de entrada e T é um limiar, que pode variar para diferentes valores de x e y caso a limiarização seja local.

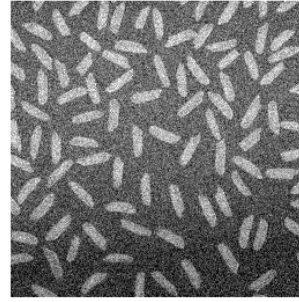
$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) > T \\ 0 & \text{if } f(x,y) \leq T \end{cases} \quad (2)$$

Na figura 1 mostra-se duas imagens e seus respectivos histogramas. A figura 1c mostra o histograma para a imagem sem ruído e a figura 1d mostra o histograma para a imagem com ruído gaussiano. Como pode-se ver na figura 1c, a escolha de um valor para o limiar T para a imagem sem ruído é uma tarefa fácil, porém a escolha do limiar para a imagem com ruído da figura 1d não é mais trivial. A seguir descreve-se um métodos para obtenção de limiar para limiarização global que pode ser aplicado nos dois casos.



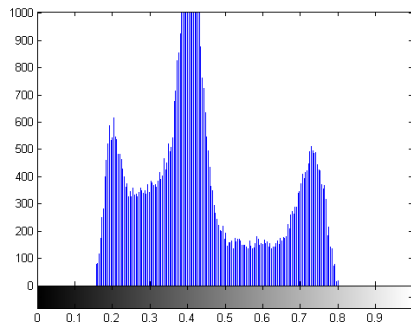
(a) Imagem sem ruído

Fonte: (MATHWORKS, 2013a)



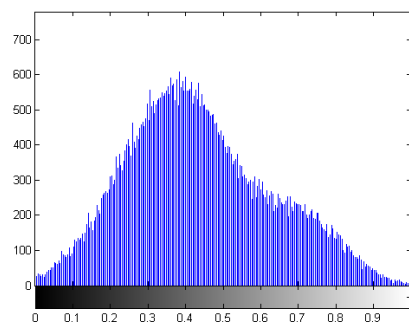
(b) Imagem com ruído

Fonte: Autoria própria



(c) Histograma da imagem sem ruído

Fonte: Autoria própria



(d) Histograma da imagem com ruído

Fonte: Autoria própria

Figura 1: Imagens com e sem ruído e seus respectivos histogramas

Limiarização global pelo método de *Otsu*

O método para determinar o valor do limiar T proposto por *Otsu* é um método ótimo que busca maximizar a variância entre as duas classes. Isso baseia-se no fato de que classes bem divididas possuem valores de intensidades distintos, e o valor ótimo para T é o valor tal que haja maior separação entre as intensidades (GONZALEZ; WOODS, 2006).

O cálculo do valor limiar pelo método de *Otsu* se dá pela equação 3.

$$\sigma_B^2(t) = \frac{[m_G P_1(t) - m(t)]^2}{P_1(t) [1 - P_1(t)]} \quad (3)$$

Onde:

- $\sigma_B^2(t)$ é a variância entre as classes para um valor limiar t ;
- m_G é a média global de intensidades da imagem;

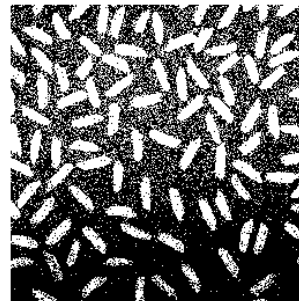
- $P_1(t)$ é a probabilidade de um pixel qualquer da imagem estar na classe 1 dado o valor limiar t .
- $m(t)$ é a média acumulada das intensidades de $t = 0$ até a intensidade t ;

Portanto, para o cálculo do valor limiar pelo método de *Otsu* basta encontrar o valor limiar t tal que a variância entre as classes, $\sigma_B^2(t)$, é máximo. A dedução da equação 3 está fora do escopo deste projeto, mas pode ser encontrada em (GONZALEZ; WOODS, 2006).

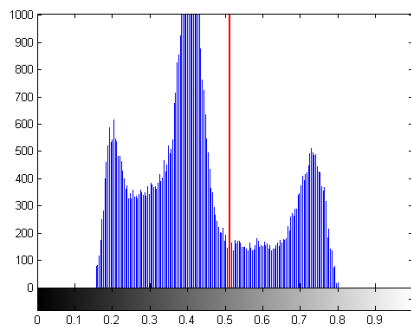
Na figura 2 mostram-se as imagens da figura 1 limiarizadas pelo método de *Otsu* e seus respectivos valores de limiarização.



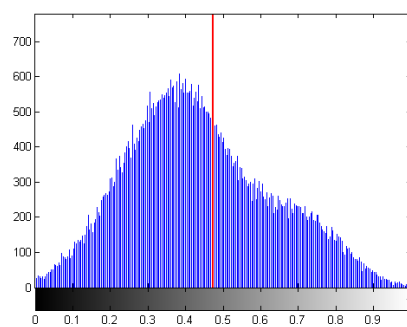
(a) Imagem sem ruído limiarizada



(b) Imagem com ruído limiarizada



(c) Histograma da imagem sem ruído com o valor do limiar T



(d) Histograma da imagem com ruído com o valor do limiar T

Figura 2: Imagens limiarizadas pelo método de *Otsu* e seus respectivos histogramas

Fonte: Autoria própria

2.1.3 Operações morfológicas básicas

Operações morfológicas como abertura e fechamento podem ser úteis para filtrar imagens. Principalmente para filtrar imagens binárias onde a operação de fechamento pode ser aplicada para diminuir ruído.

Para o estudo das operações morfológicas de abertura e fechamento precisa-se primeiramente esclarecer alguns pontos:

- Podemos representar imagens binárias como sendo um conjunto de pixels no espaço Z^2 , onde cada pixel é representado por uma tupla (x,y) e x e y são as coordenadas do pixel.
- A translação de um conjunto B para um ponto $z = (z_1, z_2)$, denotado por $(B)_z$ é o conjunto de pontos em B com as respectivas coordenadas (x,y) tendo sido substituídas pela coordenadas $(x + z_1, y + z_2)$.
- Elemento estruturante é um conjunto de pixels que é utilizado para fazer operações sobre uma imagem. Cada elemento estruturante possui um pixel de origem. A figura 3 mostra exemplos de elementos estruturantes e suas respectivas origens.

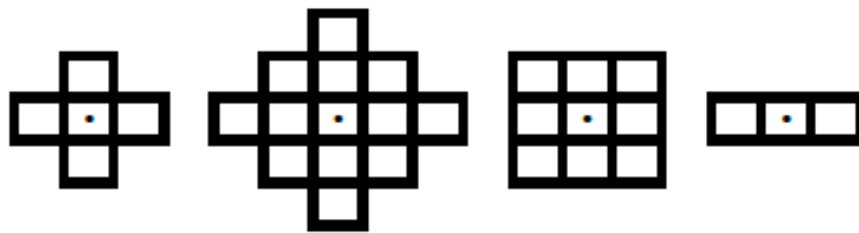
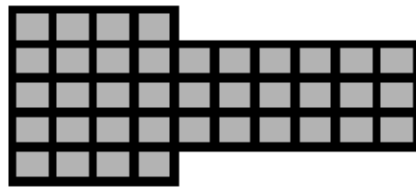
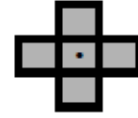
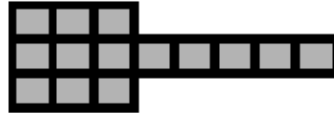


Figura 3: Exemplos de elementos estruturantes.

Fonte: Autoria própria

- O elemento estruturante pode ser utilizado para fazer operações sobre uma imagem binária. Considere por exemplo a operação no conjunto A , feita pelo elemento estruturante B (figuras 4a e 4b): Cria-se um novo conjunto C sobrepondo B em A da maneira que a origem de B passe por todos os pixels de A . Em cada ponto que a origem de B passa verifica-se se B está completamente contido em A . Se estiver marca-se esse ponto no conjunto C . A figura 4c mostra o resultado da operação, ou seja, o conjunto C .

(a) Imagem binária A (b) Elemento estruturante B (c) C , resultado da operação de B em A **Figura 4:** Imagem binária, elemento estruturante e operação de B em A **Fonte:** Autoria própria

A seguir apresentam-se as operações de erosão e de dilatação, que são necessárias para as operações de abertura e fechamento que são apresentadas na sequência.

Erosão

Sendo A e B conjuntos no espaço Z^2 , a erosão do conjunto A pelo elemento estruturante B , denotada por $A \ominus B$, é definida na equação 4 (GONZALEZ; WOODS, 2006).

$$A \ominus B = \{z \mid (B)_z \subseteq A\} \quad (4)$$

Ou seja, a erosão de A por B é o conjunto de todos os pontos z tal que B , transladado por z , está completamente contido em A . Pode-se ver melhor esse processo na figura 5, onde o conjunto A é erodido pelo elemento estruturante B formando o conjunto $A \ominus B$.

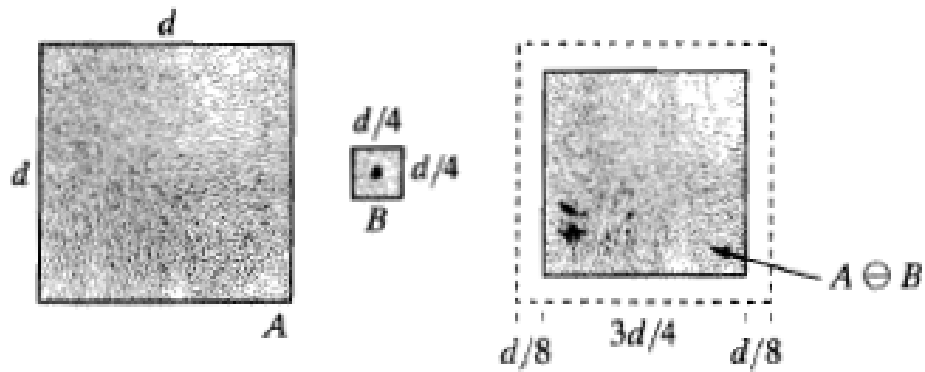


Figura 5: Exemplo de erosão do conjunto A por B.

Fonte: (GONZALEZ; WOODS, 2006)

Dilatação

Sendo A e B conjuntos no espaço Z^2 , a dilatação do conjunto A pelo elemento estruturante B , denotada por $A \oplus B$, é definida na equação 5 (GONZALEZ; WOODS, 2006).

$$A \oplus B = \{z \mid [(\hat{B})_z \cap A] \subseteq A\} \quad (5)$$

Onde \hat{B} é o conjunto B espelhado. Quando o elemento estruturante é simétrico então $\hat{B} = B$.

A equação 5 diz que a dilatação do conjunto A pelo elemento estruturante B é o conjunto de todos os pontos z tal que se \hat{B} e A se sobrepõem em pelo menos um ponto, então z faz parte de $A \oplus B$. Pode-se ver melhor esse processo na figura 6, onde o conjunto A é dilatado pelo elemento estruturante B formando o conjunto $A \oplus B$.

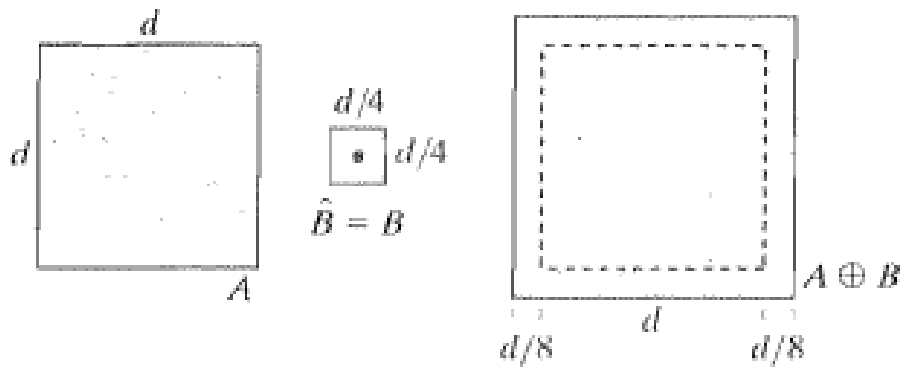


Figura 6: Exemplo de dilatação do conjunto A por B.

Fonte: (GONZALEZ; WOODS, 2006)

Abertura e Fechamento

A abertura do conjunto A pelo elemento estruturante B é definida pela equação 6 (GONZALEZ; WOODS, 2006).

$$A \circ B = (A \ominus B) \oplus B \quad (6)$$

O fechamento do conjunto A pelo elemento estruturante B é definido pela equação 7 (GONZALEZ; WOODS, 2006).

$$A \bullet B = (A \oplus B) \ominus B \quad (7)$$

Na figura 8 pode-se ver o resultado da execução da operação de abertura em cima do conjunto A da figura 7. Em seguida na figura 9 pode-se ver o resultado da execução da operação de fechamento em cima do mesmo conjunto da figura 7. Como pode-se ver, a operação de abertura "abre" a imagem em pontos estreitos, já a operação de fechamento "fecha" a imagem onde existem cavidades pequenas.

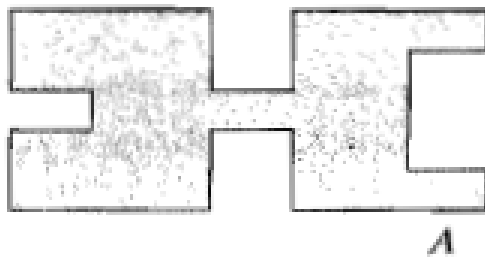


Figura 7: Conjunto A .

Fonte: (GONZALEZ; WOODS, 2006)

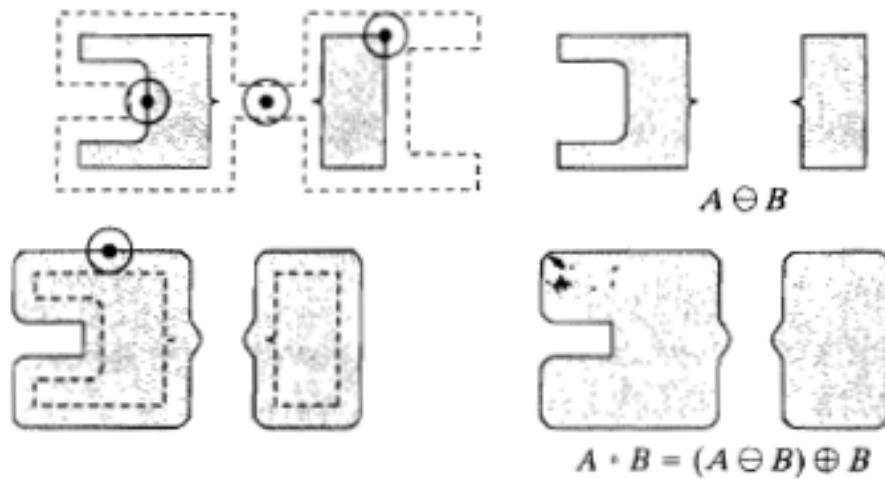


Figura 8: Conjunto $A \circ B$.

Fonte: (GONZALEZ; WOODS, 2006)

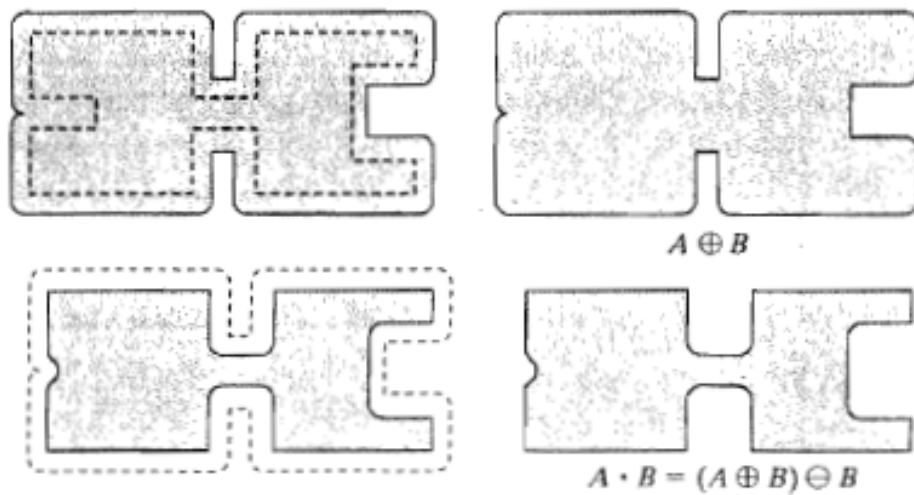


Figura 9: Conjunto $A \bullet B$.

Fonte: (GONZALEZ; WOODS, 2006)

2.1.4 Esqueletonização

Define-se esqueleto conforme segue (GONZALEZ; WOODS, 2006).

Se $S(A)$ é o esqueleto de A e $(D)_z$ o maior disco centrado em z e contido em A , pode se dizer que :

- Se z faz parte de $S(A)$ então não é possível encontrar um disco maior que $(D)_z$, não necessariamente centrado em z , que contenha $(D)_z$ e que esteja completamente incluso em A ;

- O disco $(D)_z$ toca a borda de A em pelo menos dois pontos distintos.

Isso pode ser visto melhor na figura 10.

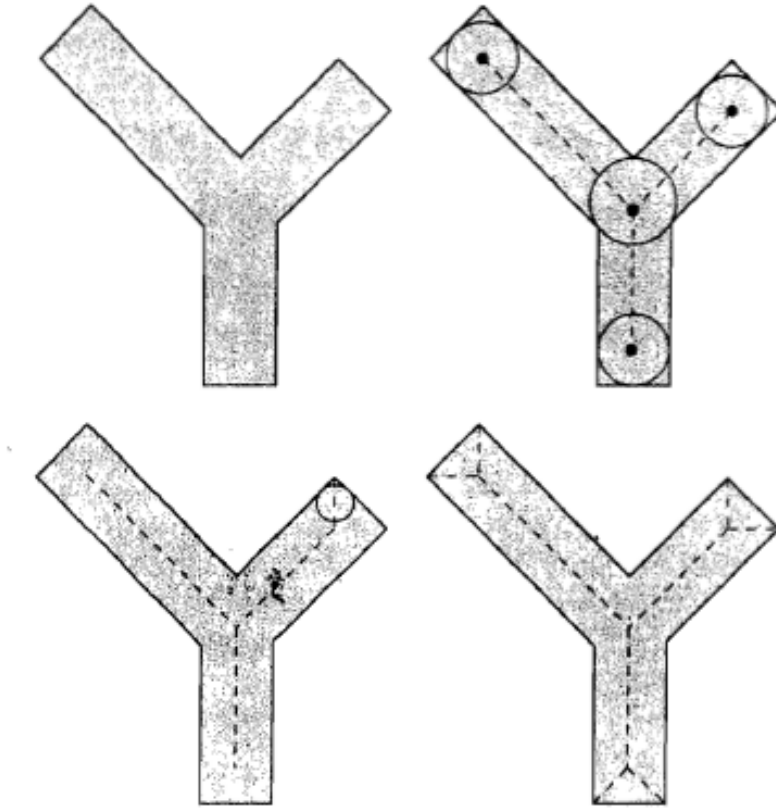


Figura 10: Conjunto A , disco máximo centrado em diversos pontos z e esqueleto completo.

Fonte: (GONZALEZ; WOODS, 2006)

Pode-se definir também o esqueleto $S(A)$ em função de erosões e aberturas sucessivas como mostrado nas equações 8, 9, 10 e 11 (GONZALEZ; WOODS, 2006).

$$S(A) = \bigcup_{k=0}^K S_k(A) \quad (8)$$

Com:

$$S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B \quad (9)$$

Onde B é um elemento estruturante e $(A \ominus kB)$ indica k sucessivas erosões de A :

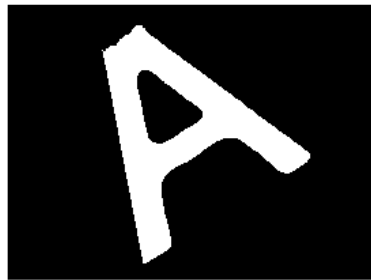
$$(A \ominus kB) = (((...(A \ominus B) \ominus B) \ominus ...) \ominus B) \quad (10)$$

Sendo K o ultimo passo antes de A erodir para um conjunto vazio. Ou seja:

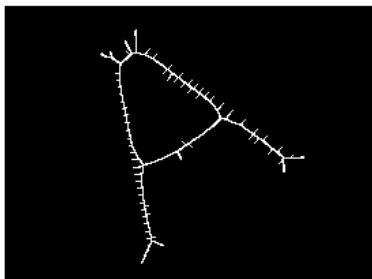
$$K = \max \{k \mid (A \ominus kB) \neq \emptyset\} \quad (11)$$

Poda (*Pruning*)

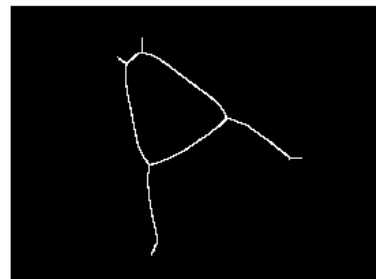
Aplicando-se o método de eskeletonização mencionado anteriormente em figuras reais, como a figura 11a, pode-se obter resultados como a figura 11b. Devido a esse tipo de resultados aplica-se após a eskeletonização um método de poda. A poda do esqueleto consiste em avaliar o tamanho dos ramos gerados, caso sejam menores que um valor limiar são removidos. A figura 11c mostra o método de poda aplicado com um limiar de 20 pixels.



(a) Imagem de entrada



(b) Imagem eskeletonizada



(c) Imagem eskeletonizada e podada

Figura 11: Exemplo de eskeletonização e poda.

Fonte: Autoria própria

2.1.5 Normalização

A análise de componentes principais pode ser empregada para normalizar imagens em função de rotação, translação e tamanho (GONZALEZ; WOODS, 2006). Explica-se a seguir alguns conceitos necessários para efetuar a normalização utilizando a análise de componentes principais e em seguida o processo de normalização utilizando componentes principais.

Matriz de covariâncias

Considere as imagens da figura 12. Os pontos da imagem, podem ser considerados como sendo vetores $x_k = (x_1, x_2)^T$, onde x_1 e x_2 são as coordenadas do ponto x_k , e T indica a matriz transposta. Os pontos x portanto formam uma população de vetores no espaço Z^2 . Sendo assim pode-se calcular a matriz de covariâncias C_x e o vetor da média m_x para a população de vetores.

Para K amostras de uma população aleatória pode-se aproximar a média m_x pela equação 12 (GONZALEZ; WOODS, 2006).

$$m_x = \frac{1}{K} \sum_{k=1}^K x_k \quad (12)$$

Agora utilizando a média m_x , pode-se aproximar a matriz de covariâncias C_x pela equação 13 (GONZALEZ; WOODS, 2006).

$$C_x = \frac{1}{K} \sum_{k=1}^K (x_k x_k^T) - m_x m_x^T \quad (13)$$

Na matriz de covariâncias os elementos c_{ii} de C_x representam a variância entre os valores na coordenada x_i . Os elementos c_{ij} representam a covariância entre os valores na coordenada x_i com os valores da coordenada x_j . Portanto, a matriz de covariância possui informações que dizem respeito à distribuição dos pixels da imagem.

Pode-se ver isso melhor nas matrizes de covariância apresentadas nas figuras 12d, 12e e 12f que foram calculadas para as imagens das figuras 12a, 12b e 12c respectivamente. Percebe-se que na figura 12d a maior variância ocorre entre os elementos da coordenada x_1 , visto que c_{11} possui o maior valor na matriz de covariâncias. Semelhantemente percebe-se que na figura 12e a maior variância ocorre entre os elementos da coordenada x_2 . A terceira imagem, figura 12f, mostra uma distribuição mais continua em x_1 e x_2 , mas valores maiores para as covariâncias entre os elementos das coordenadas x_1 e x_2 . A terceira imagem mostra portanto o relacionamento entre as duas coordenadas da imagem: quando x_1 aumenta, x_2 tende a aumentar também,

e vice-versa.

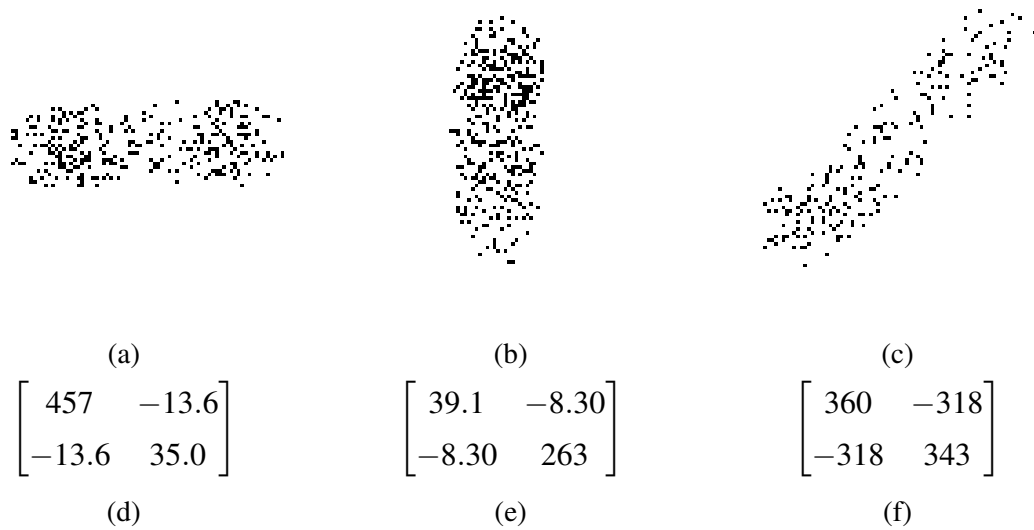


Figura 12: Imagens de exemplo e suas respectivas matrizes de covariância.

Fonte: Autoria própria

Autovetores e autovalores

Um autovetor de uma matriz A é um vetor não nulo v , que quando a matriz é multiplicada por v tem-se um vetor que é múltiplo de v por uma constante λ denominada autovalor. Ou seja, a equação 14 é verdadeira (LEON, 1998).

$$Av = \lambda v \quad (14)$$

Algumas propriedades de autovetores:

- Os autovetores podem ser calculados apenas para matrizes quadradas (LEON, 1998);
- Para uma matriz $n \times n$ existem n autovetores (LEON, 1998);
- Todos os autovetores são ortogonais entre si (GONZALEZ; WOODS, 2006).

Análise de componentes principais

Pode-se demonstrar que o autovetor associado ao maior autovalor de uma matriz de covariâncias aponta na direção de maior variação dos dados (GHAOUI, 2013). Pode-se demonstrar também que o segundo autovetor, associado ao segundo maior autovalor, aponta na segunda

direção de maior variação dos dados. A esses autovetores se dá o nome de componentes principais da imagem.

Considere por exemplo as imagens da figura 13. Calculamos os autovetores para cada uma das imagens tomando como base as matrizes de covariância apresentadas nas figuras 12d, 12e e 12f. Os autovetores podem ser vistos nas figuras 13d, 13e e 13f. O autovetor associado ao maior autovalor está na primeira linha de cada matriz e o segundo autovetor está na segunda linha. Os autovetores foram representados sobre as respectivas imagens nas figuras 14a, 14b e 14c.

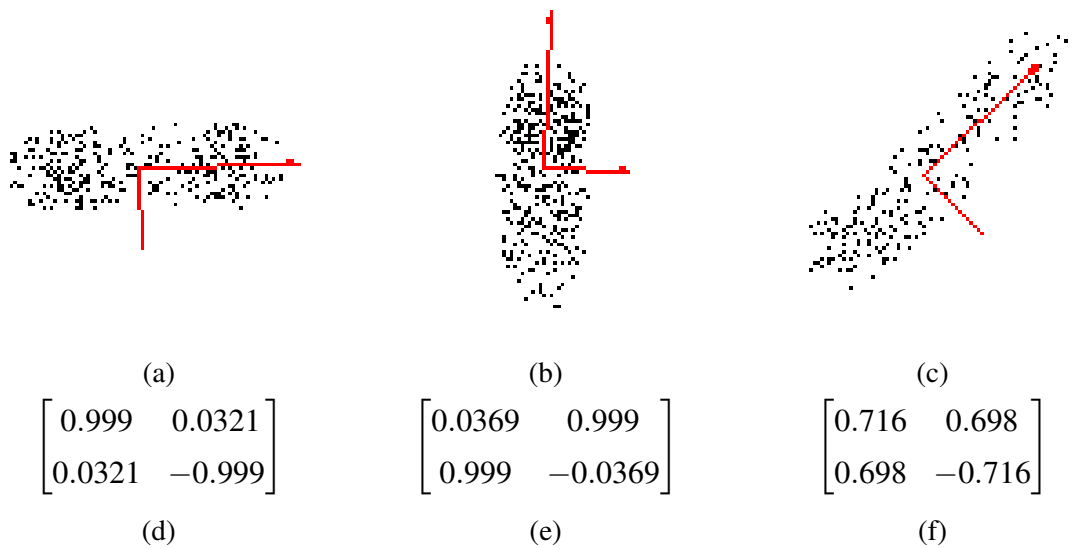


Figura 13: Imagens de exemplo e seus respectivos autovetores.

Fonte: Autoria própria

Vê-se portanto que é possível definir um eixo absoluto para cada imagem com base na distribuição dos pixels sobre a imagem.

Normalização

A normalização das imagens pode ser feita por meio de uma matriz de transformação que mapeia valores do sistema de coordenadas original da imagem para o sistema de coordenadas formado pelos autovetores da imagem (GONZALEZ; WOODS, 2006). A transformação em questão pode ser expressa pela equação 15.

$$y = A(x - m_x) \quad (15)$$

Onde:

- y é o valor no novo sistema de coordenadas formado pelos autovetores.
- x o valor da coordenada no sistema original.
- A a matriz de autovetores.
- m_x vetor da média.

As imagens da figura 12 podem ser vistas normalizadas na figura 14.

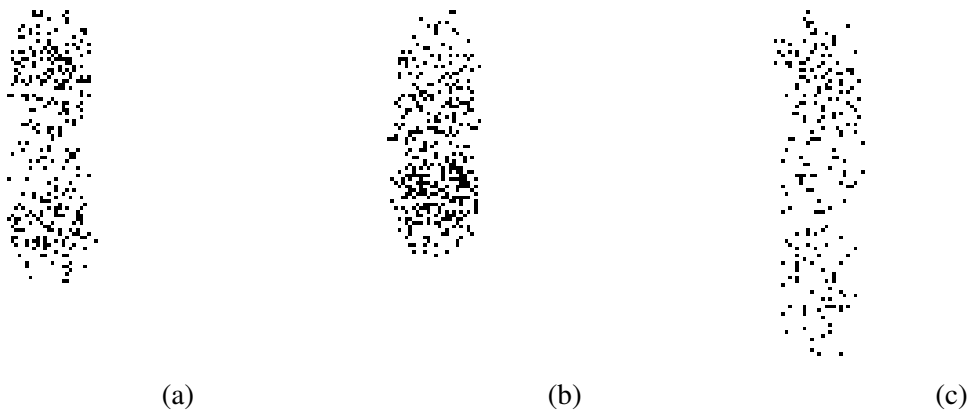


Figura 14: Imagens normalizadas.

Fonte: Autoria própria

Caso se deseje normalizar a imagem com relação ao tamanho pode-se dividir o valor das coordenadas y pelos autovalores correspondentes (GONZALEZ; WOODS, 2006).

2.2 DESCRIÇÃO

Esta seção apresenta técnicas e algoritmos para realizar a descrição de imagens, tais como extração de propriedades básicas de regiões e descritores de *fourier*.

2.2.1 Propriedades de regiões

Propriedades básicas de regiões, como área e excentricidade, podem ser usadas para descrever imagens. A seguir descreve-se algumas propriedades básicas de regiões, principalmente aquelas que possuem como saída um valor escalar, visto que são facilmente adicionados à um vetor descritor.

Área

A área de uma região é o número de pixels da região (MATHWORKS, 2013b).

Área convexa

A área convexa é dada pela área do menor polígono capaz de conter a região (MATHWORKS, 2013b).

Área Preenchida

Área preenchida corresponde a área da região cujas cavidades foram todas preenchidas (MATHWORKS, 2013b).

Solidez

A solidez de uma região é dada pela razão Área / Área convexa (MATHWORKS, 2013b).

Extensão

Semelhantemente à área convexa, a extensão é dada pela razão entre a área da região (definida anteriormente) e a área do menor retângulo capaz de conter a região (MATHWORKS, 2013b).

Excentricidade

Para a obtenção da excentricidade de uma região deve-se primeiramente obter o momento de inércia de área da região (MATHWORKS, 2013b). O momento de inércia de área para uma região qualquer no sistema de coordenadas polares é definido pela equação 16:

$$J_{BB} = \int_A \rho^2 dA \quad (16)$$

Onde A é a área da região sobre a qual se deseja calcular o momento de inércia de área e ρ é a distância do centro do sistema de coordenadas até dA . Uma representação da área pode ser vista na figura 15.

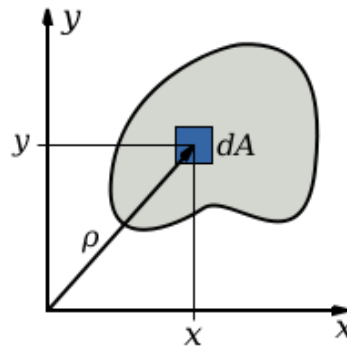


Figura 15: Representação da área para cálculo do momento de inércia.

Fonte: (WIKIPEDIA, 2013)

A excentricidade da região é então a excentricidade da elipse que possui o mesmo momento de inércia que a área da região. A excentricidade para uma elipse com semi-eixo maior a e semi-eixo menor b é dada pela equação 17:

$$e = \sqrt{1 - \frac{b^2}{a^2}} \quad (17)$$

Tamanho do eixo principal e eixo secundário

O tamanho do eixo principal de uma região diz respeito ao tamanho do maior eixo da elipse que possui o mesmo momento de inércia de área que a área da região, conforme visto anteriormente na seção Excentricidade (MATHWORKS, 2013b). Da mesma forma o tamanho do eixo secundário diz respeito ao tamanho do menor eixo da elipse que possui o mesmo momento de inércia de área que a área da região.

Número de Euler

O número de Euler é dado pelo total de objetos desconexos da região subtraindo o total de cavidades da região (MATHWORKS, 2013b).

2.2.2 Descritores de *fourier*

Considere a figura 16a. Os pixels com valor 1 da imagem binária podem ser representados como sendo pares de coordenadas $(x_0, y_0), (x_1, y_1), (x_2, y_2), \dots, (x_{K-1}, y_{K-1})$, onde K é o número de pixels na imagem. Além disso os pixels podem ser representados como uma sequência

$x(k) = x_k$ e $y(k) = y_k$ ou como $s(k) = [x(k), y(k)]$ para $k = 0, 1, 2, \dots, K - 1$. Além disso pode-se tratar as coordenadas dos pixels como números complexos: $s(k) = x(k) + jy(k)$.

Tem-se que a transformada discreta de *fourier* de um sinal $s(k)$ é dado pela equação 18 (GONZALEZ; WOODS, 2006):

$$a(u) = \sum_{k=0}^{K-1} s(k) e^{-j2\pi uk/K}, u = 0, 1, 2, \dots, K - 1 \quad (18)$$

Os coeficientes complexos $a(u)$ da transformada discreta de *fourier* são chamados de descritores de *fourier* (GONZALEZ; WOODS, 2006).

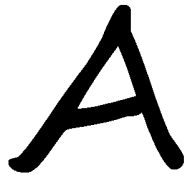
Da mesma forma que se aplicou a transformada discreta de *fourier* para obter os descritores de *fourier* pode-se aplicar a transformada discreta inversa de *fourier* nos coeficientes complexos para obter novamente os números complexos representando as coordenadas dos pixels (equação 19).

$$s(k) = \frac{1}{K} \sum_{u=0}^{K-1} a(u) e^{j2\pi uk/K}, k = 0, 1, 2, \dots, K - 1 \quad (19)$$

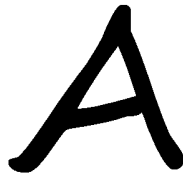
Suponha agora que ao invés de utilizar todos os descritores de *fourier* utiliza-se apenas os P primeiros descritores para reconstituir $s(k)$. Tem-se portanto uma aproximação para $s(k)$ dada por $\hat{s}(k)$ na equação 20.

$$\hat{s}(k) = \frac{1}{P} \sum_{u=0}^{P-1} a(u) e^{j2\pi uk/P}, k = 0, 1, 2, \dots, K - 1 \quad (20)$$

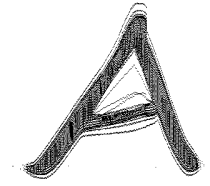
Quando os P coeficientes de menor frequência são escolhidos para reconstituir a imagem tem-se uma imagem aproximada da imagem original porém que mantém as principais características. Pode-se ver esse comportamento nas imagens da figura 16. A imagem da figura 16b foi formada utilizando todos os 19410 coeficientes da imagem original, a imagem 16c foi formada com 9705 coeficientes, e assim por diante, até apenas 2 coeficientes na figura 16o.



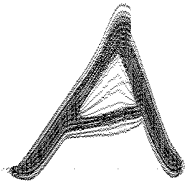
(a) Imagem original



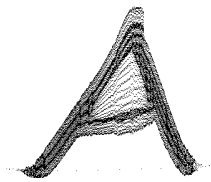
(b) 19410 coeficientes



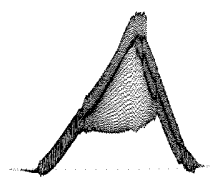
(c) 9705 coeficientes



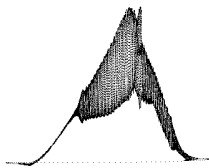
(d) 4852 coeficientes



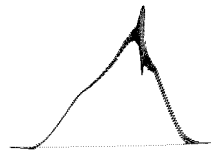
(e) 2426 coeficientes



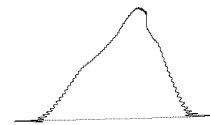
(f) 1213 coeficientes



(g) 606 coeficientes



(h) 303 coeficientes



(i) 151 coeficientes



(j) 75 coeficientes



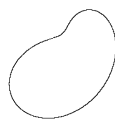
(k) 37 coeficientes



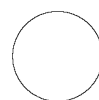
(l) 18 coeficientes



(m) 9 coeficientes



(n) 4 coeficientes



(o) 2 coeficientes

Figura 16: Imagem original e imagens reconstituídas utilizando n descritores de *fourier*.

Fonte: Autoria própria

2.3 CLASSIFICAÇÃO

Esta seção apresenta uma visão geral de reconhecimento de padrões utilizando redes neurais.

2.3.1 Redes neurais

Redes neurais são elementos não lineares (chamados neurônios) interconectados da mesma maneira que se acredita que os neurônios estão conectados no cérebro. Usam-se essas redes para tomadas de decisão após seus coeficientes terem sido ajustados por meio de apresentações sucessivas de conjuntos de dados de treinamento (GONZALEZ; WOODS, 2006).

Na forma mais simples no entanto a rede é um elemento linear e é composta por um único neurônio, também chamado *perceptron*. Esse neurônio então aprende uma função linear para separar dois conjuntos de dados de treinamento linearmente separáveis. A figura 17 apresenta o modelo básico para um *perceptron*. A resposta desse *perceptron* é baseada na soma ponderada das entradas, dada pela equação 21.

$$d(x) = \sum_{i=1}^n (\omega_i x_i) + \omega_{n+1} \quad (21)$$

Os coeficientes $w_i, i = 1, 2, \dots, n, n + 1$, são chamados de pesos e modificam a soma antes que os valores de entrada passem pelo elemento de ativação, também chamado de função de ativação. Os pesos podem ser comparados às sinapses que ocorrem no cérebro humano. No caso particular da figura 17 a saída do *perceptron* será +1 caso a entrada pertença a classe ω_1 e -1 caso a entrada pertença a classe ω_2 .

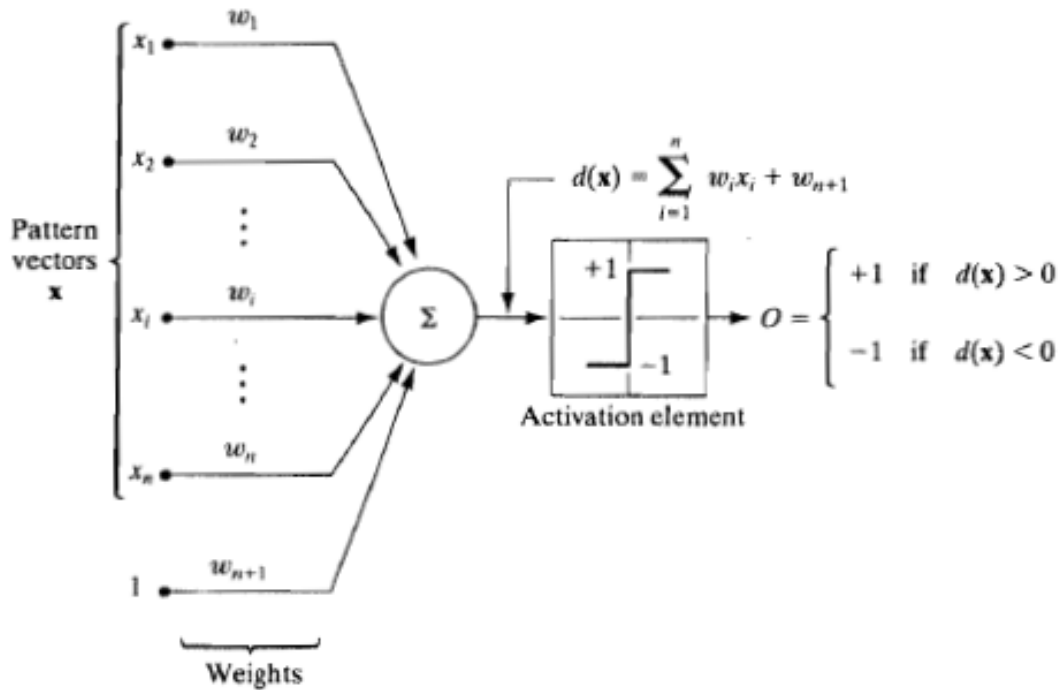


Figura 17: Modelo de *perceptron*.

Fonte: (GONZALEZ; WOODS, 2006)

Esse tipo de abordagem formada por um único *perceptron* não é capaz de separar mais do que duas classes e também não é capaz de separar classes que não são linearmente separáveis. Isso se dá devido ao fato de que a equação 21 quando expandida define um hiperplano em um espaço n -dimensional.

Utilizam-se portanto para problemas mais complexos redes de neurônios, também chamada de *multi-layer perceptron* ou *multi-layer feedforward network*. Cada neurônio é formado por uma estrutura do mesmo tipo da estrutura do *perceptron*, com a diferença de a função de ativação ser geralmente uma função sigmoide ao invés de um simples limiar. A função sigmoide pode ser vista na equação 22 e na figura 18. As saídas dos neurônios estão interconectadas com as entradas de outros neurônios. Essa estrutura pode ser vista na figura 19.

$$h_j(I_j) = \frac{1}{1 + e^{-(I_j + \theta_j)/\theta_0}} \quad (22)$$

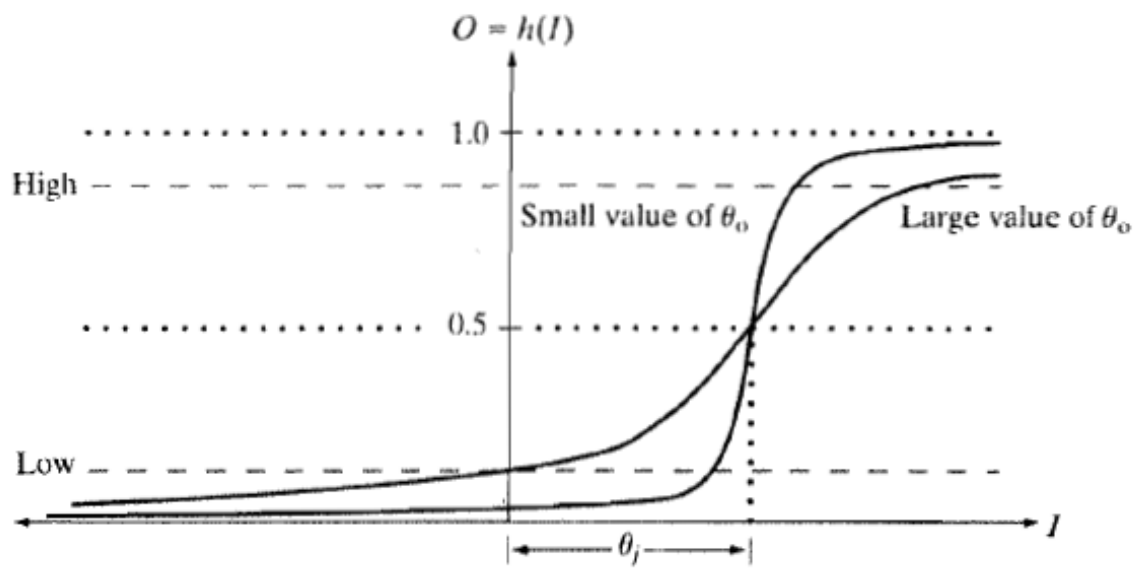


Figura 18: Gráfico da função de ativação sigmoide.

Fonte: (GONZALEZ; WOODS, 2006)

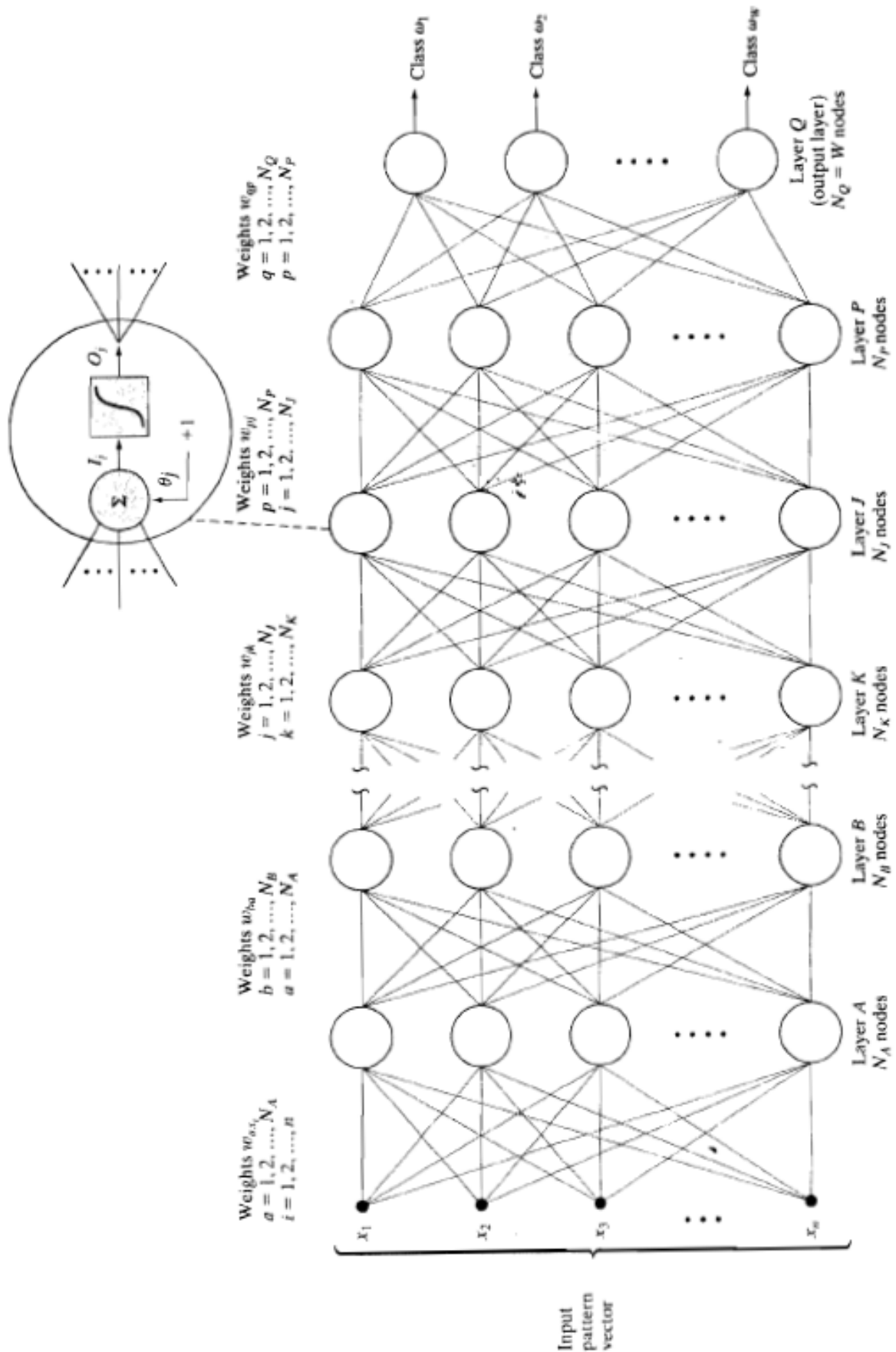


Figura 19: Modelo de *multi-layer feedforward network*.

Fonte: (GONZALEZ; WOODS, 2006)

Nesse tipo de rede a primeira camada, A , possui o mesmo número de neurônios que a dimensão do vetor descritor da entrada, $N_A = n$. Semelhantemente o número de neurônios da camada de saída, Q , é igual ao número de classes que a rede deve reconhecer, $N_Q = W$. A rede reconhece uma entrada como sendo parte de uma classe quando a saída da classe é um e as demais saídas são zero.

Pode-se demonstrar que redes desse tipo são capazes de separar regiões convexas no espaço n -dimensional de entradas utilizando duas camadas de neurônios. São também capazes de separar classes em regiões arbitrárias utilizando três camadas de neurônios (GONZALEZ; WOODS, 2006). A complexidade das regiões separadas por uma rede de três camadas se dá pelo número de neurônios em cada camada. Isso está resumido na figura 20.


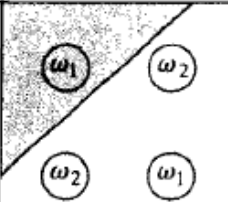
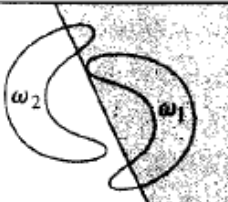
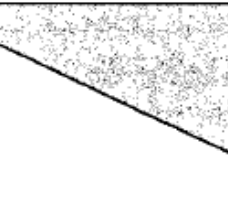
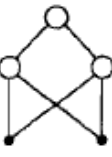
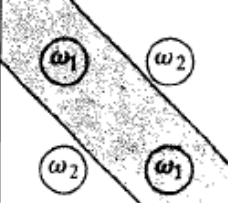
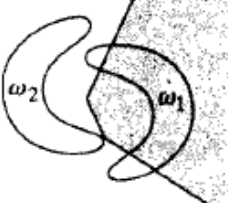

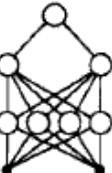
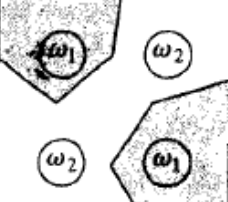
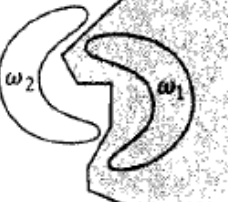
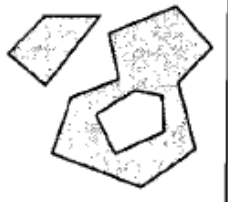
Network structure	Type of decision region	Solution to exclusive-OR problem	Classes with meshed regions	Most general decision surface shapes
Single layer 	Single hyperplane			
Two layers 	Open or closed convex regions			
Three layers 	Arbitrary (complexity limited by the number of nodes)			

Figura 20: Estruturas de redes neurais e sua respectiva capacidade de decisão.

Fonte: (GONZALEZ; WOODS, 2006)

Treinamento de *perceptrons*

O treinamento de *perceptrons* consiste em determinar os valores adequados de ω_i para que a soma ponderada das entradas forme a saída adequada para a função de ativação. Existem diversos métodos de treinamento para determinar os valores de ω_i para um único perceptron.

Apresenta-se a seguir o método denominado *delta rule* ou *least-mean-square*.

A principal característica do método *delta rule* é que ele minimiza o erro entre a saída desejada e a saída real do *perceptron* em cada passo de treinamento. Considere a função da equação 23.

$$J(w) = \frac{1}{2}(r - w^T y)^2 \quad (23)$$

$J(w)$ representa o erro quadrático para um dado conjunto de pesos w . O erro é calculado como sendo a diferença entre a saída desejada do *perceptron* r e a saída real dada pelo produto do vetor de pesos transposto w^T com o valor de entrada do conjunto de treinamento y .

O objetivo do treinamento pelo método *delta rule* é minimizar $J(w)$ ajustando os valores dos pesos w .

Se $w(k)$ representa o vetor de pesos no passo de treinamento k , então pode-se calcular $w(k+1)$ de forma a reduzir o valor de $J(w)$ utilizando a equação 24 (GONZALEZ; WOODS, 2006).

$$w(k+1) = w(k) - \alpha \left[\frac{\partial J(w)}{\partial w} \right]_{w=w(k)} \quad (24)$$

Onde $\alpha > 0$ é uma constante escolhida de acordo com o grau de correção que se deseja a cada passo. Da equação 23 tem-se que:

$$\frac{\partial J(w)}{\partial w} = -(r - w^T y)y \quad (25)$$

Substituindo a equação 25 na equação 24 resulta na equação 26 que pode ser utilizada para o cálculo dos coeficientes $w(k+1)$ dados os coeficientes $w(k)$, a saída desejada r e o valor de entrada do conjunto de treinamento y .

$$w(k+1) = w(k) + \alpha [r(k) - w^T(k)y(k)]y(k) \quad (26)$$

Pode-se demonstrar que utilizando esse método para calcular os coeficientes, tem-se a redução do erro do *perceptron* em um fator de $\alpha y(k)^T y(k)$ a cada ciclo de treinamento (GONZALEZ; WOODS, 2006).

Treinamento de *multi-layer perceptrons*

O treinamento de redes multicamadas pode ser efetuado de maneira semelhante ao treinamento de um único *perceptron*. Considere a equação 27. O erro quadrático E_Q obtido na camada de saída é dado pela soma dos erros quadráticos em todos os *perceptrons* da camada de saída (*perceptrons* de 1 a N_Q). O erro em cada *perceptron* é dado pela diferença entre a saída desejada r_q , no *perceptron* q , e a saída obtida O_q .

$$E_Q = \frac{1}{2} \sum_{q=1}^{N_Q} (r_q - O_q)^2 \quad (27)$$

Novamente deseja-se minimizar o erro E_Q em cada passo de treinamento. Minimizando a equação 27 chega-se a dois casos. O primeiro é quando o *perceptron* está na última camada da rede. Nesse casos sabe-se o valor esperando na saída do *perceptron*, que é dado por O_q . Por outro lado quando o *perceptron* está em uma camada intermediária é necessário que primeiro se defina o erro em função de parâmetros conhecidos na rede. Esse processo foi desenvolvido por (GONZALEZ; WOODS, 2006) e a seguir apresenta-se o resultado.

Para *perceptrons* na camada de saída Q os pesos $w(k+1)$ podem ser calculado pelas equações 28 e 29. K é a camada que precede a camada de saída Q .

$$w_q(k+1) = w_q(k) + \alpha \delta_q O_K \quad (28)$$

$$\delta_q = (r_q - O_q) h'_q(I_q) \quad (29)$$

Onde:

- $w_q(k+1)$ são os pesos atualizados do *perceptron* q ;
- $w_q(k)$ são os pesos antigos do *perceptron* q ;
- $\alpha > 0$ é uma constante escolhida de acordo com o grau de correção que se deseja a cada passo;
- O_K são as saídas dos *perceptrons* da camada K ;
- r_q é a saída esperada do *perceptron*, dada pelo conjunto de treinamento;
- O_q é a saída atual no *perceptron* q ;

- $h'_q(I_q)$ é a derivada da função de ativação do *perceptron* q para a entrada I_q .

Para *perceptrons* em uma camada intermediária J , onde a camada seguinte é a camada P e a camada antecedente é a camada K , os pesos $w(k+1)$ podem ser calculados pelas equações 30 e 31.

$$w_j(k+1) = w_j(k) + \alpha \delta_j O_K \quad (30)$$

$$\delta_j = h'_j(I_j) \sum_{p=1}^{N_P} \delta_p \omega_{jp} \quad (31)$$

Onde:

- $w_j(k+1)$ são os pesos atualizados do *perceptron* j ;
- $w_j(k)$ são os pesos antigos do *perceptron* j ;
- $\alpha > 0$ é uma constante escolhida de acordo com o grau de correção que se deseja a cada passo;
- O_K são as saídas dos *perceptrons* da camada K ;
- $h'_j(I_j)$ é a derivada da função de ativação para a entrada I_j ;
- p até N_P são todos os *perceptrons* da camada P ;
- δ_p é o valor de δ que havia sido calculado para o *perceptron* p da camada P quando os pesos da camada P foram atualizados.
- ω_{jp} é o valor do peso ω que havia sido calculado para o *perceptron* j em cada *perceptron* da camada P .

Caso a função de ativação escolhida seja a função sigmoide da equação 22, com $\theta_O = 1$, pode-se demonstrar que $h'_j(I_j)$ assume o valor da equação 32 e semelhantemente $h'_q(I_q)$ assume o valor da equação 33 (GONZALEZ; WOODS, 2006).

$$h'_j(I_j) = O_j(1 - O_j); \quad (32)$$

Onde:

- O_j é a saída atual no *perceptron* j ;

$$h'_q(I_q) = O_q(1 - O_q); \quad (33)$$

Onde:

- O_q é a saída atual no *perceptron* q ;

Como se pode ver é necessário iniciar o processo de atualização dos pesos da rede pela camada de saída. Na camada de saída da rede é o único lugar onde se sabe qual é a saída que cada *perceptron* deveria ter. Para as camadas antecedentes a atualização deve ser feita em função dessas saídas. Portanto cada camada depende de parâmetros da camada seguinte na rede. Esse processo de atualização dos pesos iniciando pela ultima camada e voltando até a primeira recebe o nome de *back propagation*. O método mencionado anteriormente recebe o nome de *delta rule*, no entanto existem diversos métodos diferentes mas que utilizam a mesma estratégia de propagação das correções na rede. Todos esses métodos são classificados como métodos de *back propagation*.

2.4 CONSIDERAÇÕES

TODO.

3 DESENVOLVIMENTO

Este capítulo apresenta a o desenvolvimento do projeto, detalhando cada parte do sistema desenvolvido. A descrição apresentada a seguir referencia os conceitos apresentados no capítulo 2.

A figura 21 apresenta uma visão geral do sistema desenvolvido.

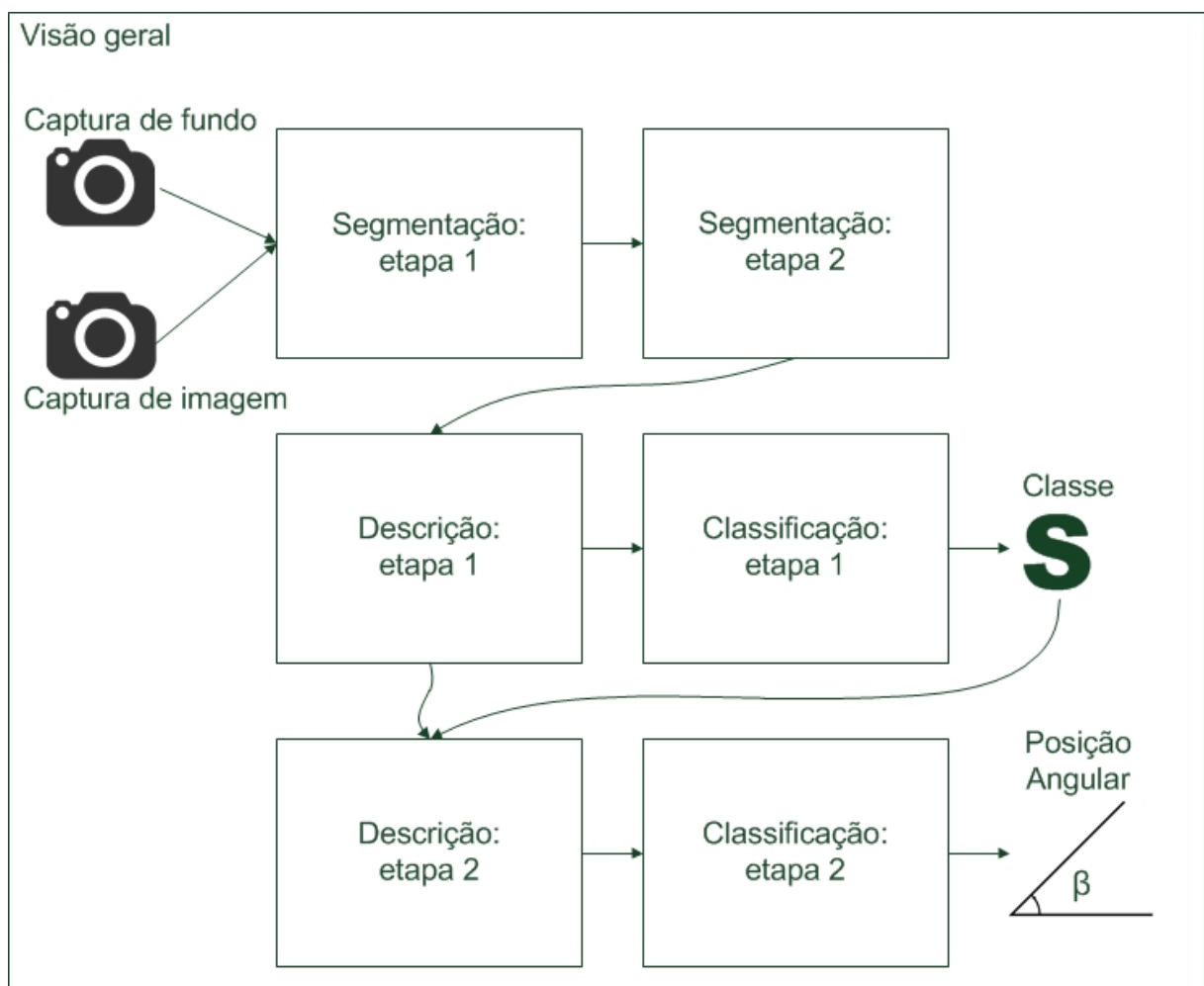


Figura 21: Visão geral do sistema desenvolvido.

Fonte: Autoria própria

O capítulo divide-se em oito seções. As primeiras seis seções apresentam os blocos da figura 21: A seção 3.1 e a seção 3.2 mostram como os conceitos da seção 2.1 foram aplicados para segmentar a imagem capturada. A seção 3.3 e a seção 3.5 apresentam como os descritores de *fourier* e as propriedades estudados na seção 2.2.1 foram aplicados para formar um descritor para as imagens. As seções 3.4 e 3.6 apresentam como os descritores foram utilizados para classificar e determinar o ângulo das imagens. Na seção 3.7 apresenta-se o processo de treinamento do sistema e ao final, na seção 3.8, algumas considerações sobre o desenvolvimento do projeto.

3.1 SEGMENTAÇÃO: ETAPA 1

A primeira etapa da segmentação consiste em uma série de limiarizações e filtros para localizar a região de interesse na imagem. A saída da primeira etapa é uma imagem binária contendo a região de interesse. O processo da primeira etapa pode ser visto na figura 22.

O processo inicia pela subtração da imagem RGB de fundo da imagem RGB com o objeto. Como resultado tem-se uma imagem RGB contendo as diferenças inseridas pelo objeto na imagem. Para não perder informações intrínsecas da imagem RGB no processo de conversão para escala de cinza dividem-se os canais R, G e B da imagem. Cada canal é processado separadamente. Primeiramente os canais são limiarizados utilizando o método global de *Otsu*, discutido na seção 2.1.2. Em seguida os canais são submetidos à operação morfológica de fechamento, vista na seção 2.1.3. A operação de fechamento é aplicado para reduzir o ruído do tipo pimenta (pixeis pretos espalhados aleatoriamente pela imagem). Na sequência os três canais processados separadamente são unidos através da operação lógica OU para formar uma única imagem. Finalmente a maior região conexa da imagem é selecionada para ser a região de interesse.

Embora não se possa ver visualmente nenhum progresso significativo em alguns passos da sequência de imagens da figura 22, a primeira etapa da segmentação necessita de todos eles. Todos os passos apresentados anteriormente garantem a robustez do sistema, fornecendo a região de interesse para a etapa seguinte.

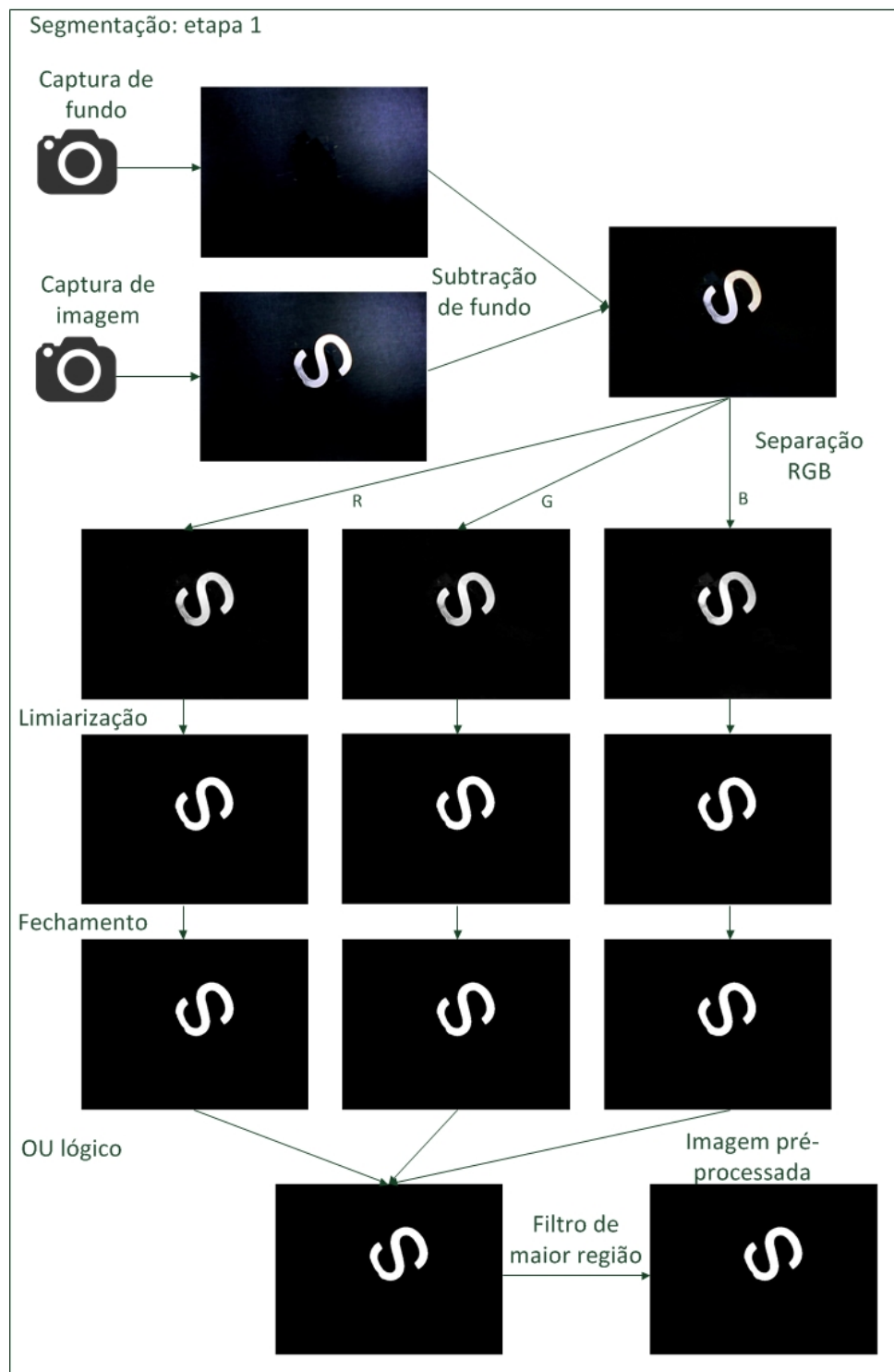


Figura 22: Primeira etapa da segmentação.

Fonte: Autoria própria

3.2 SEGMENTAÇÃO: ETAPA 2

A segunda etapa da segmentação é responsável por normalizar a imagem. Esta etapa recebe como entrada a região de interesse e apresenta como saída duas opções de normalização da imagem, além de um autovetor e seu oposto. O processo pode ser visto na figura 23.

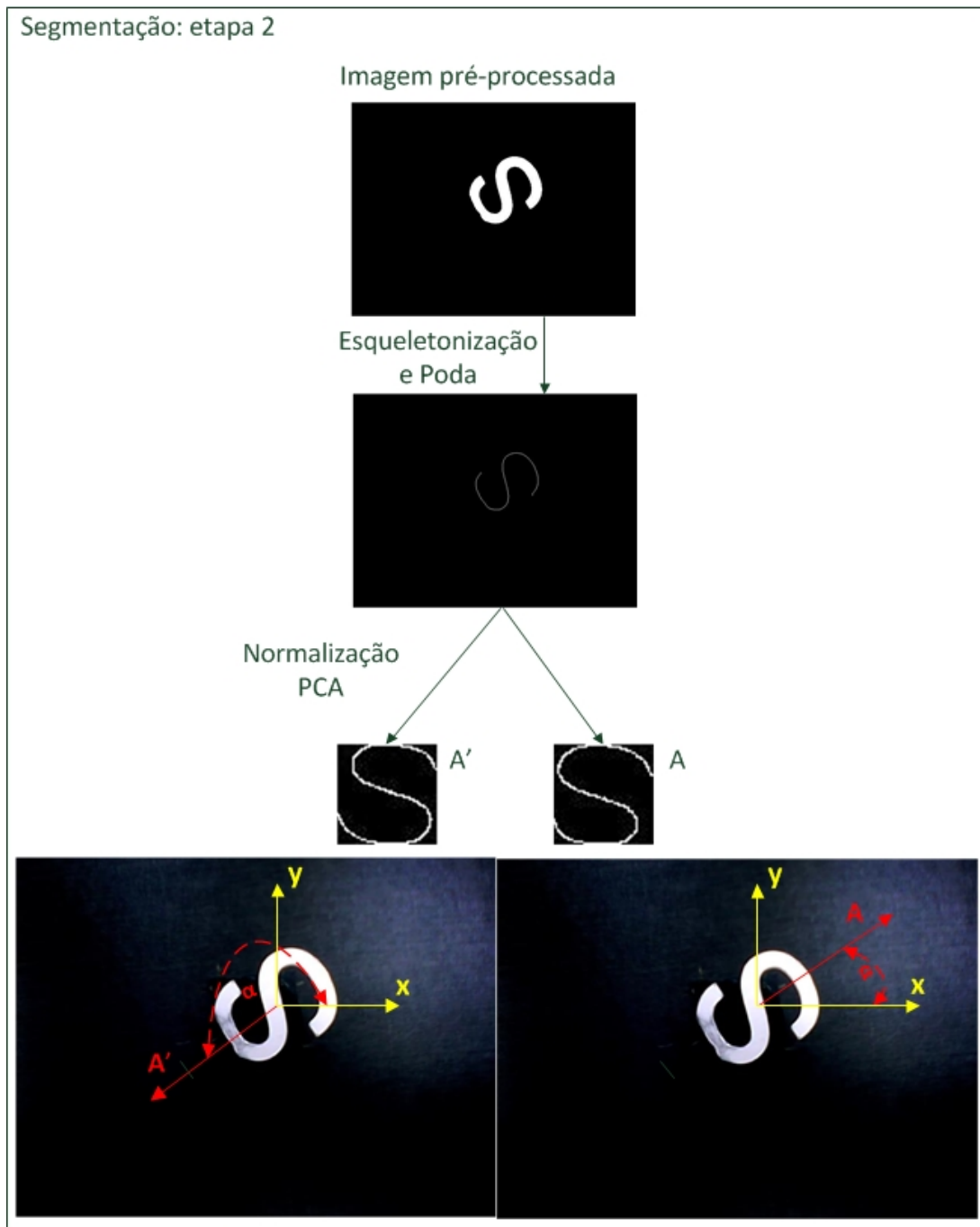


Figura 23: Segunda etapa da segmentação.

Fonte: Autoria própria

A região de interesse obtida da primeira etapa de segmentação é esqueletonizada e podada pelos processos descritos na seção 2.1.4. Em seguida o processo de normalização da seção 2.1.5 é aplicado.

Como dito na seção 2.1.5, o autovetor associado ao maior autovalor da matriz de covariâncias aponta na direção de maior variação dos dados. Porém o autovetor não aponta no sentido de maior variação dos dados. Empiricamente percebe-se que o autovetor aponta sempre no sentido positivo dos eixos da imagem. Esse comportamento leva a duas possíveis normalizações para cada imagem. A normalização obtida utilizando o autovetor A e a normalização pelo oposto do autovetor, $A' = -A$.

Essas duas opções de normalização são então apresentadas a etapa seguinte juntamente com os respectivos autovetores.

3.3 DESCRIÇÃO: ETAPA 1

Essa etapa de descrição é responsável por gerar um descritor para a imagem normalizada utilizando o autovetor A (chamada mais adiante de imagem pré-normalizada). A imagem normalizada utilizando o autovetor A' não é utilizada. A figura 24 resume o processo desta etapa.

O descritor é composto por 20 descritores de *fourier*, $d1, d2, \dots, d20$, concatenados com 9 propriedades básicas da imagem, $p1, p2, \dots, p9$. A tabela 1 detalha o descritor gerado.

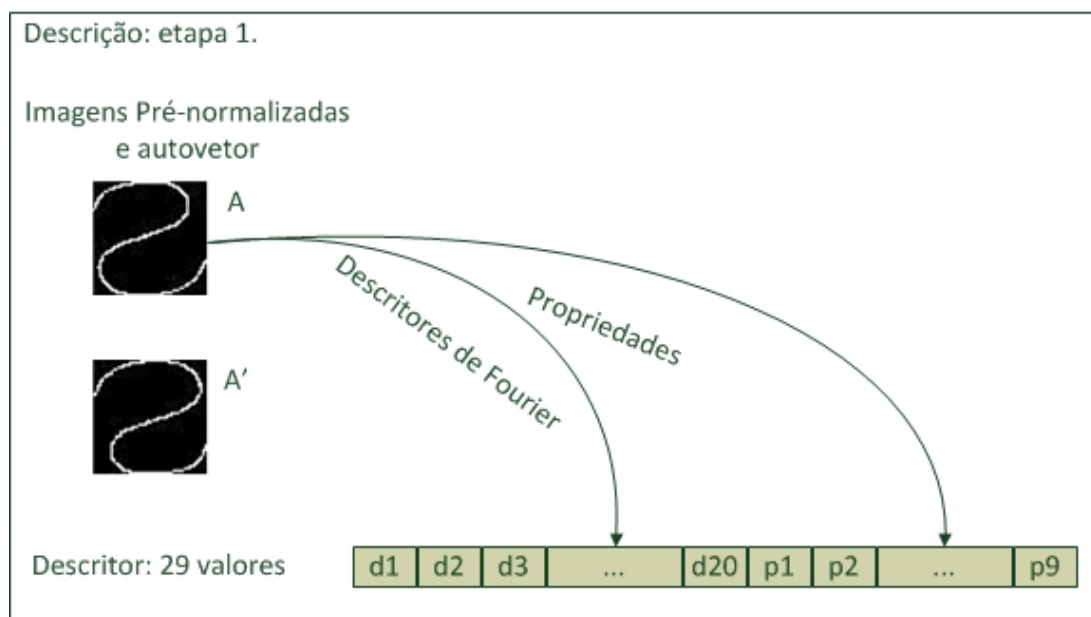


Figura 24: Primeira etapa da descrição.

Fonte: Autoria própria

Tabela 1: Descritor para primeira etapa.

Descritor	Descrição
$d1\dots d20$	20 descritores de <i>fourier</i>
$p1$	Área
$p2$	Área convexa
$p3$	Excentricidade
$p4$	Número de Euler
$p5$	Extensão
$p6$	Área preenchida
$p7$	Tamanho do eixo principal
$p8$	Tamanho do eixo secundário
$p9$	Solidez

Fonte: Autoria própria

3.4 CLASSIFICAÇÃO: ETAPA 1

A função da primeira etapa de classificação é tomar como entrada o descritor da imagem pré-normalizada e fornecer como saída a classe a qual a imagem pertence. Além disso é responsável por dizer qual a normalização correta para a imagem. Ou seja, se a imagem normalizada corretamente utiliza o autovetor A ou o oposto A' . Esse processo é mostrado na figura ??.

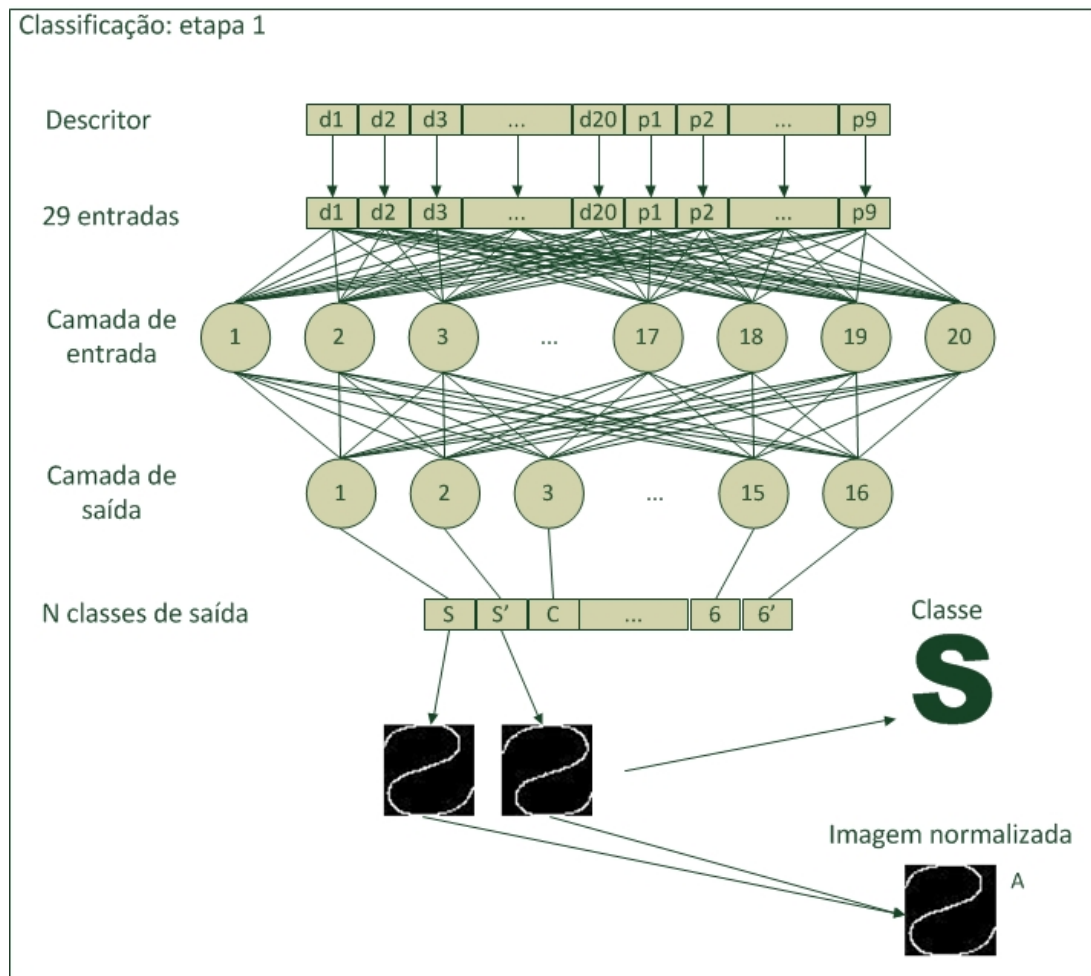


Figura 25: Primeira etapa da classificação.

Fonte: Autoria própria

3.5 DESCRIÇÃO: ETAPA 2

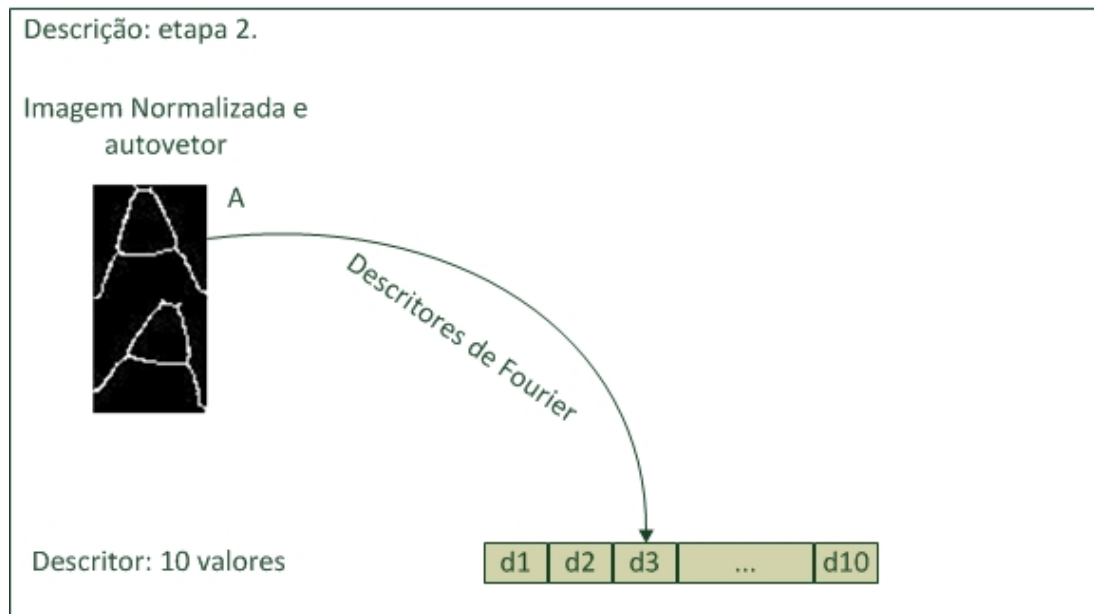
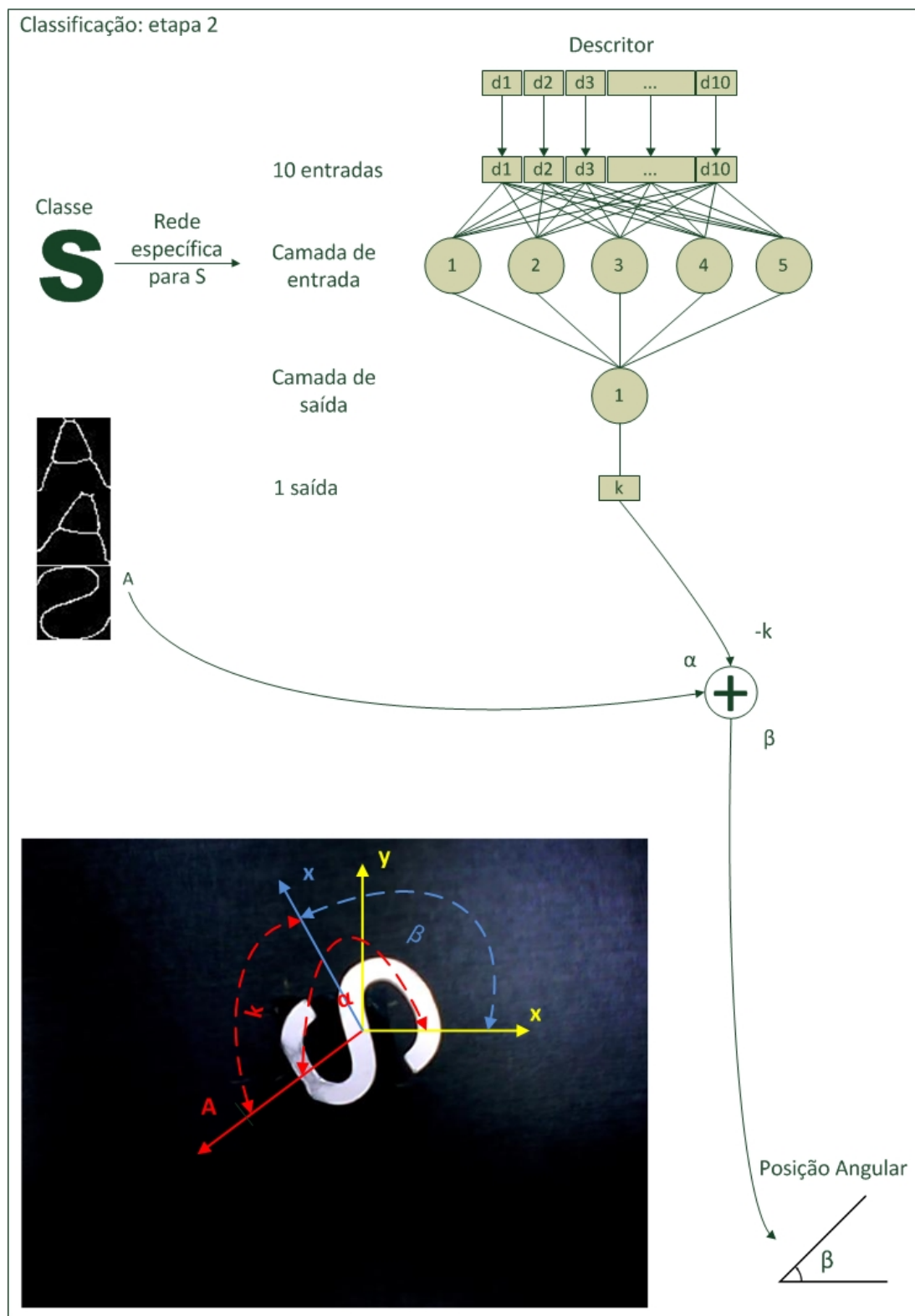


Figura 26: Segunda etapa da descrição.

Fonte: Autoria própria

3.6 CLASSIFICAÇÃO: ETAPA 2

**Figura 27:** Segunda etapa da classificação.**Fonte:** Autoria própria

3.7 CLASSIFICAÇÃO: TREINAMENTO

3.8 CONSIDERAÇÕES

A seção 2.1 apresenta técnicas e algoritmos que serão úteis para realizar a segmentação de imagens. A segmentação de imagens consiste em subdividir a imagem original em regiões ou objetos até que se tenha identificado a região ou objeto de interesse.

A seção 2.2 apresenta técnicas e algoritmos para realizar a descrição de imagens. Após a segmentação de uma imagem é necessário representar a imagem para que se possa processá-la. Existem duas abordagens para esse problema (GONZALEZ; WOODS, 2006): representação por meio de características externas da região, como bordas, ou representação por meio de características internas, como textura. A representação por características externas normalmente é utilizada quando se está interessado na forma da região. Já a representação por características internas é utilizada quando o foco está na cor ou textura da região. Nessa seção aborda-se principalmente a representação por meio de características externas de imagens.

Finalmente a seção 2.3 apresenta uma visão geral de reconhecimento de padrões utilizando redes neurais. O reconhecimento de padrões é feito tomando como base descritores, que podem ser obtidos a partir de técnicas como as vistas na seção 2.2.

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.

3.9 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.

3.9.1 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 ??, 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.

3.9.2 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 28, onde o eixo das abscissas contém os valores de conversão, e o eixo das ordenadas a distância correspondente. Segue a equação da interpolação polinomial:

$$y = 3.6404 \cdot 10^{-7}x^4 - 2.4435 \cdot 10^{-4}x^3 + 6.0732 \cdot 10^{-2}x^2 - 6.8962x + 339.361 \quad (34)$$

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

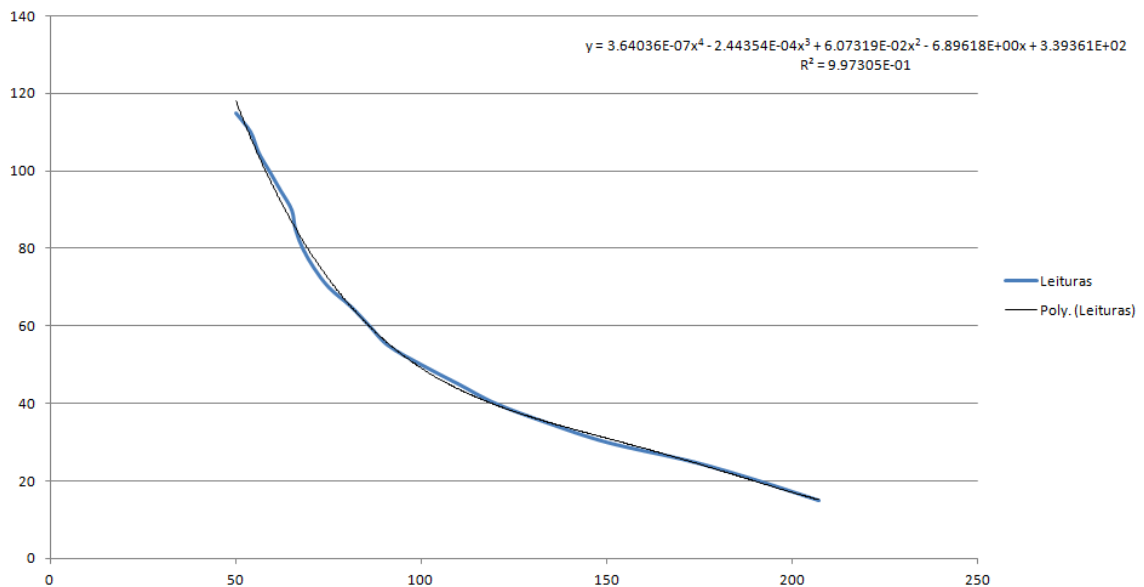


Figura 28: Curva real dos sensores e interpolação polinomial

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 3.9.9.

3.9.3 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 com-

ponentes 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.

3.9.4 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 odometria. 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 frequência 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 3.9.2, 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.

3.9.5 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*.

3.9.6 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 ??), 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 29.

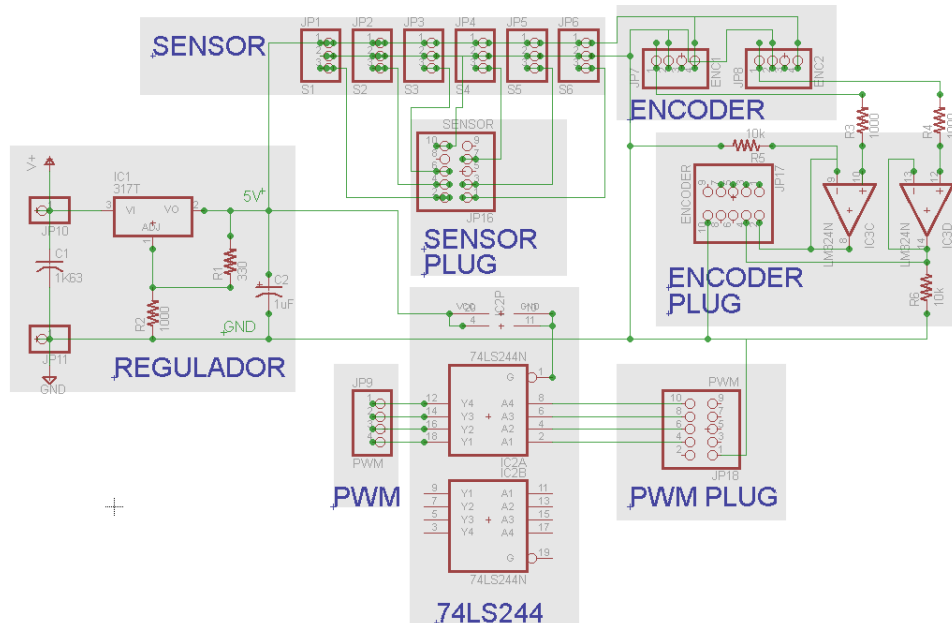


Figura 29: Diagrama esquemático da placa de roteamento.

Fonte: Autoria própria

De posse do diagrama esquemático apresentado na figura 29 a equipe realizou o roteamento de uma placa de circuito impresso que atenda ao mesmo diagrama esquemático, também utilizando o *software* Eagle.

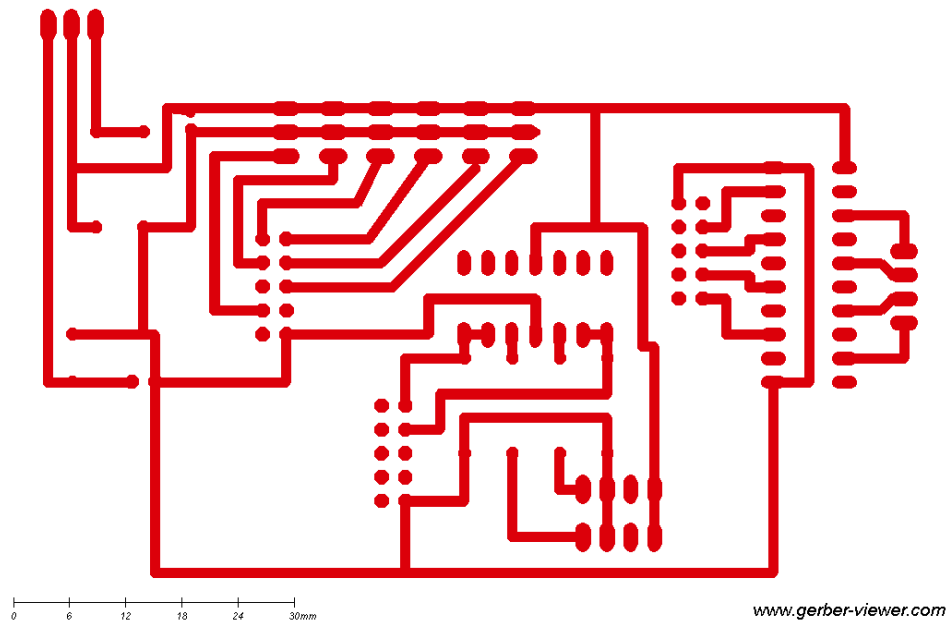


Figura 30: Roteamento final produzido pela equipe.

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 percloroeto 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 31 ilustra o resultado do processo de confecção da placa.

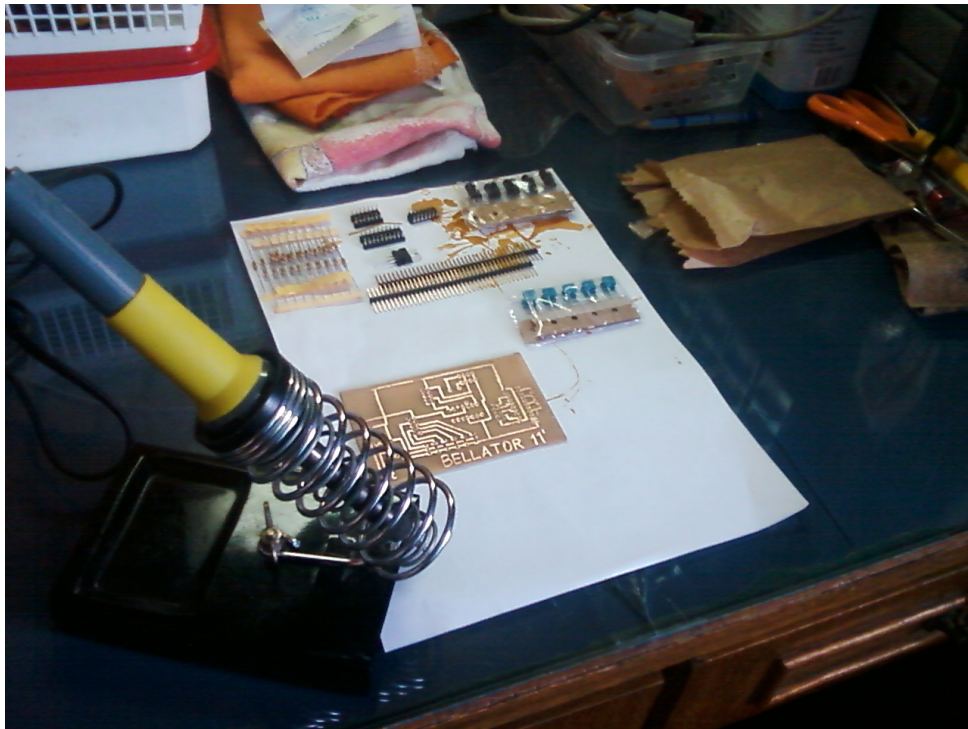


Figura 31: Resultado do processo de confecção da placa de roteamento.

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.

3.9.7 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).

3.9.8 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 3.10. 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.

3.9.9 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 3.9.2. 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.

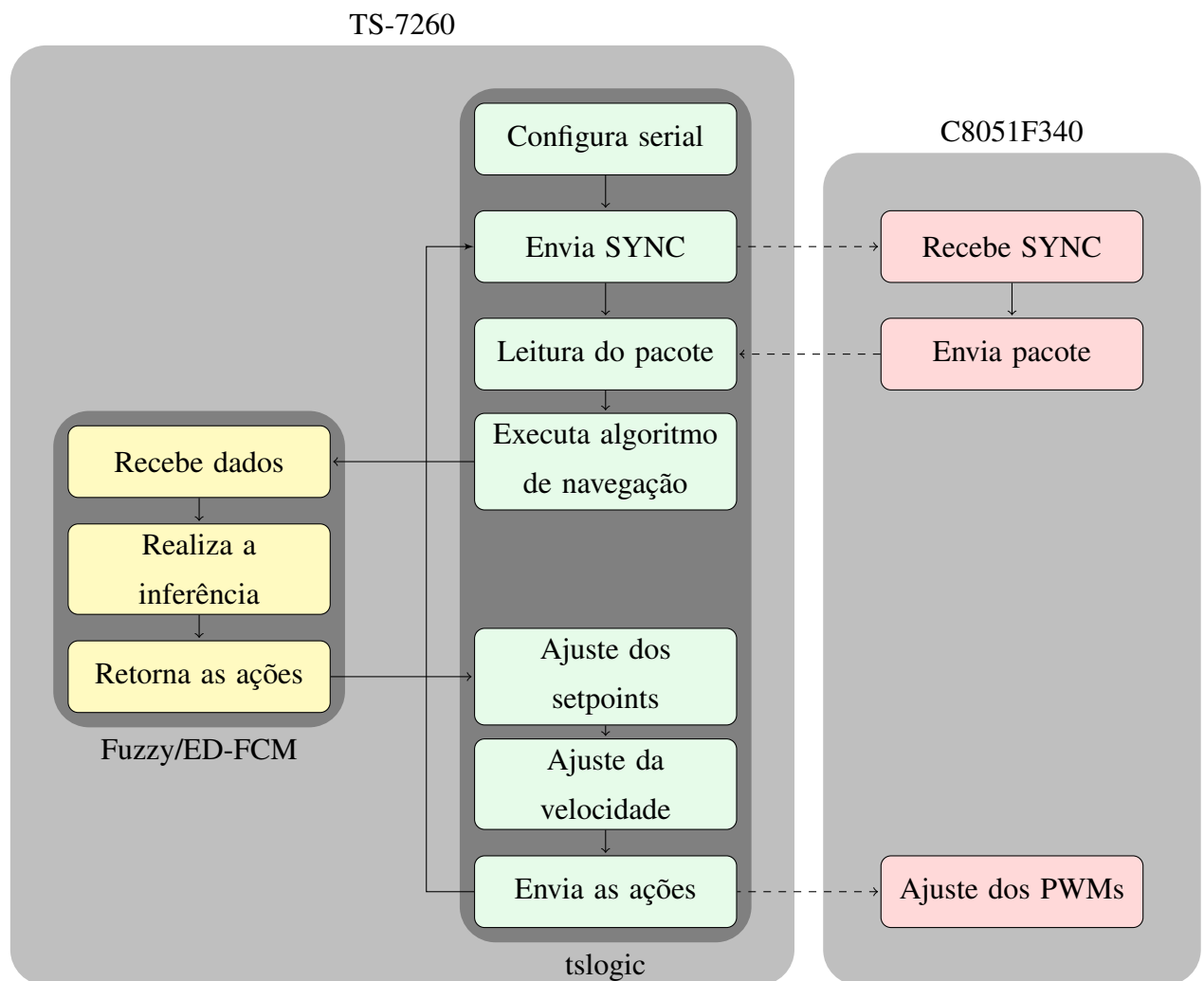


Figura 32: Diagrama em blocos demonstrando o funcionamento do software da placa TS-7260.

Fonte: Autoria própria

3.9.10 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.

3.10 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).

3.10.1 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 3.10.2. 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 ??.

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 ??.

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 33 demonstra de forma gráfica esta estrutura.

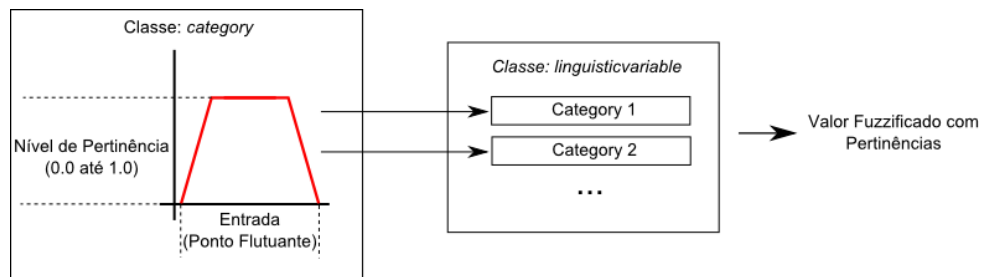


Figura 33: Estrutura de uma Variável Linguística no FLIE.

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.

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 35.

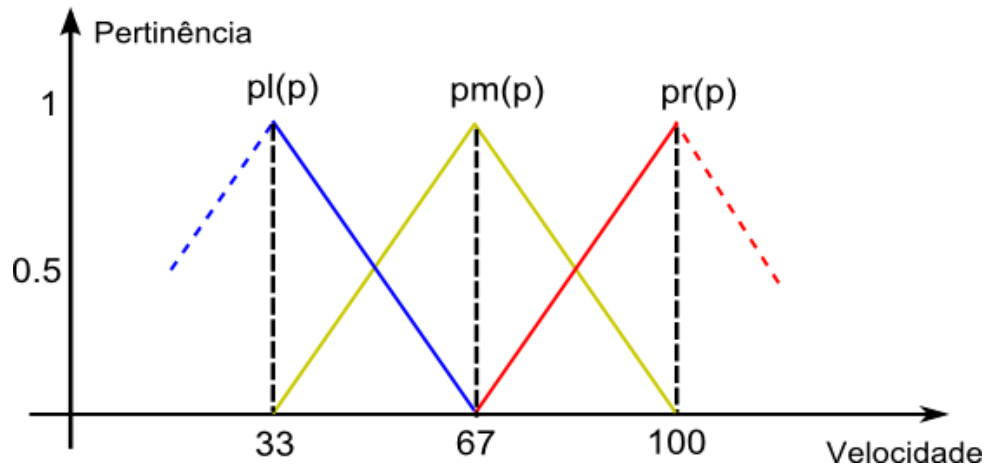


Figura 35: Definição das Funções de Pertinência lento, médio e rápido.

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°) até totalmente para direita (180°), sendo que 90° 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 36 ilustra estas funções.

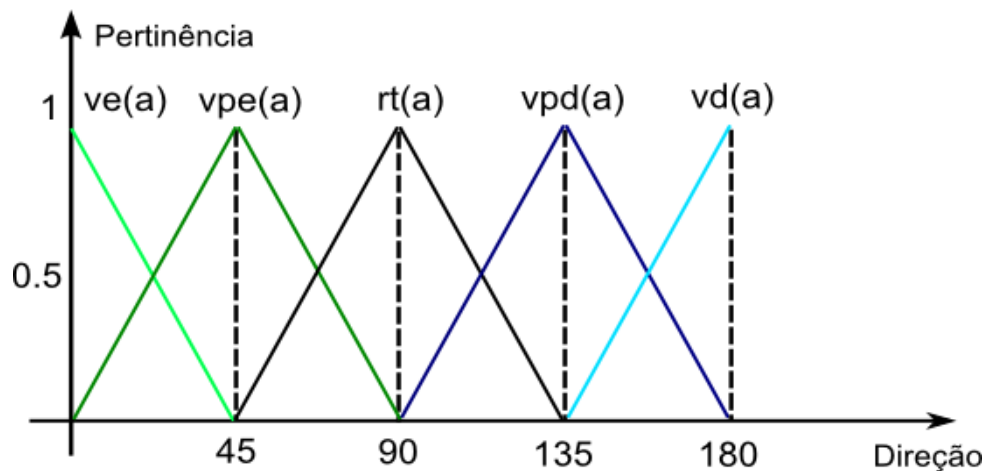


Figura 36: Definição das Funções de Pertinência de Direção.

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 37.

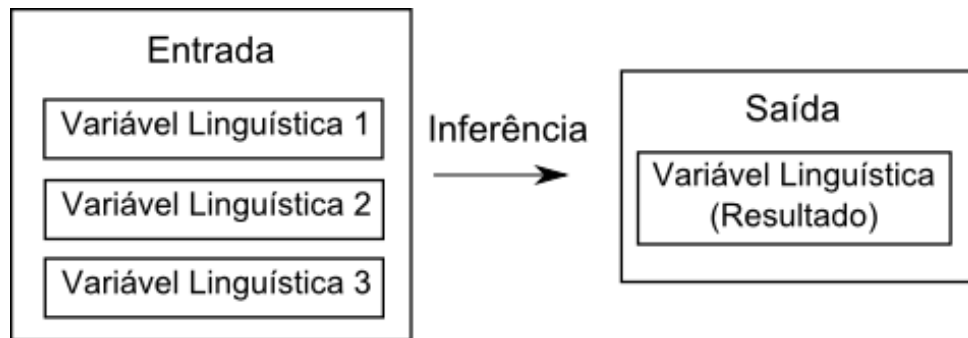


Figura 37: Estrutura de uma regra de inferência no FLIE.

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 a 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 3.10.1. 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 ???. Assim, as regras de inferência seguem o padrão abaixo, de acordo com as variáveis linguísticas definidas na seção 3.10.1:

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 biblioteca 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° (que representa virar à esquerda) a 180° (que representa virar à direita), sendo que 90° 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 3.11, 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 3.11.2 e 3.11.3, respectivamente. Durante esses testes, o algoritmo foi submetido a uma série de experimentos visando detectar erros de navegação, isolá-los e corrigi-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 3.11.4, que foi utilizada para obtenção do resultado final.

3.10.2 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 ?? 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 rodas 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 35, sendo x o valor absoluto da distância (cm), MIN o valor mínimo suportado pelo sensor e MAX o valor máximo.

$$y = \frac{x - MIN}{MAX - MIN} \quad (35)$$

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 sigmoideal dependente dos valores dos conceitos de entrada. As equações 36 e 37 foram adaptadas da equação ??, apresentada na fundamentação teórica, e operam em domínio $[0, 1]$ e imagem $[0, 1]$.

$$f_1(x) = \frac{1}{1 + e^{(-10x+4,5)}} \quad (36)$$

$$f_2(x) = 1 - \frac{1}{1 + e^{(-10x+4,5)}} \quad (37)$$

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 38. 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.

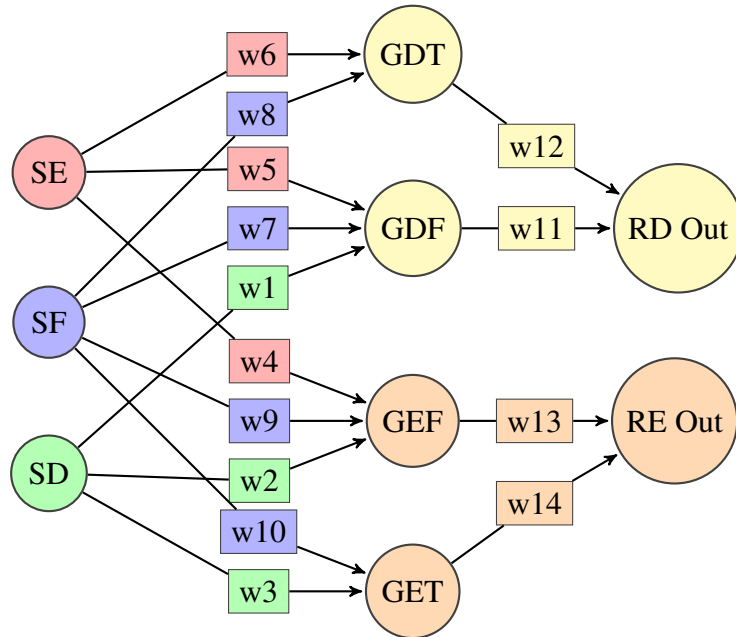


Figura 38: ED-FCM proposto.

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 38 a 41 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 36 e 37. 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 42 e 43, respectivamente.

$$GDF = \frac{w_1 f_2(SD) + w_5 f_1(SE) - w_7 f_2(SF)}{w_1 + w_5} \quad (38)$$

$$GDT = \frac{w_6 f_2(SE) + w_8 f_2(SF)}{w_6 + w_8} \quad (39)$$

$$GEF = \frac{w_4 f_2(SE) + w_2 f_1(SD) - w_9 f_2(SF)}{w_4 + w_2} \quad (40)$$

$$GET = \frac{w_3 f_2(SD) + w_{10} f_2(SF)}{w_3 + w_{10}} \quad (41)$$

$$RD_{Out} = 100 \times \frac{w_{11} GDF - w_{12} GDT}{FORTE} \quad (42)$$

$$RE_{Out} = 100 \times \frac{w_{13} GEF - w_{14} GET}{FORTE} \quad (43)$$

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 3.11, 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 3.11.3. 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 3.11.4, gerando resultados adequados para a comparação com o algoritmo *fuzzy*, apresentado na seção 3.10.1.

3.11 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.

3.11.1 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 3.10.1 e 3.10.2.

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.

3.11.2 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 39), 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 40).

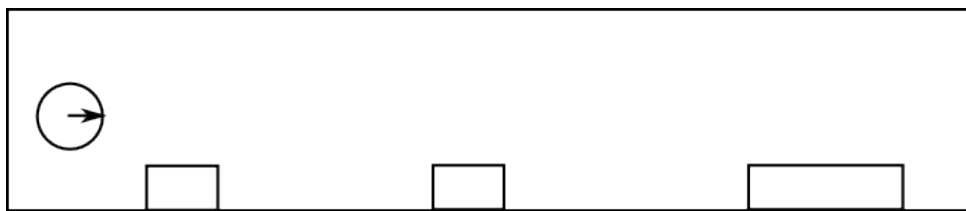


Figura 39: Configuração do primeiro teste inicial.

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 3.10.1 de 33% para “Lento” e 66% para “Médio”. Após os ajustes, o algoritmo mostrou-se mais rápido no corredor, no mesmo teste.

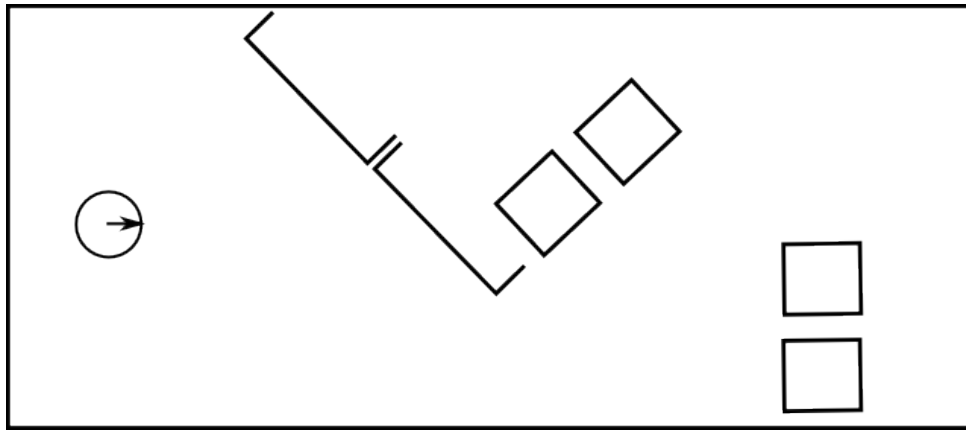


Figura 40: Configuração do segundo teste inicial.

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 40. 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 41 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.

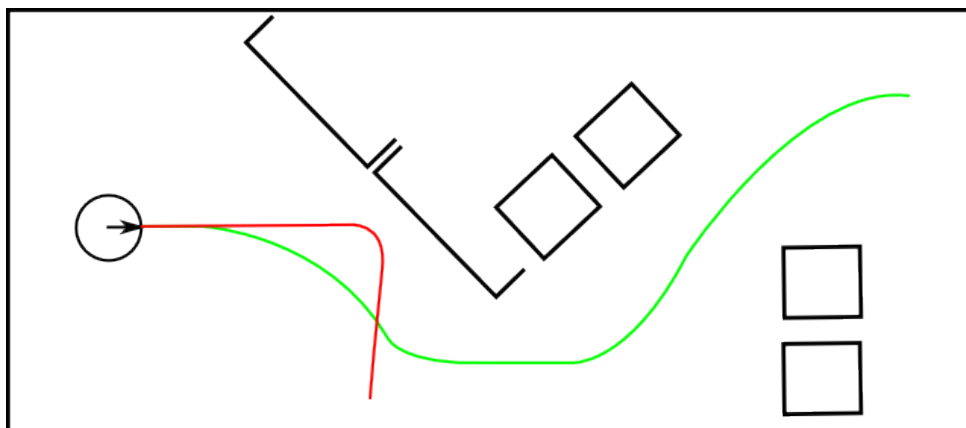


Figura 41: Caminhos percorridos no segundo teste inicial.

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 34, 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.

3.11.3 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.

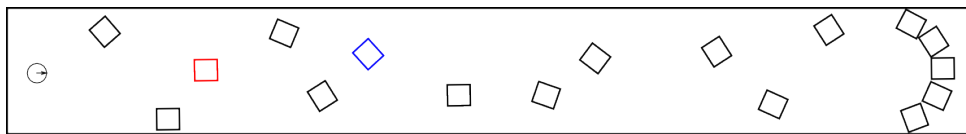


Figura 42: Ilustração do primeiro teste avançado.

Fonte: Autoria própria

O algoritmo *fuzzy* foi capaz de percorrer o primeiro teste avançado, ilustrado na figura 42, 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 42. 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 36 e 37. 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 3.10.2) 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 3.10.2. 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.

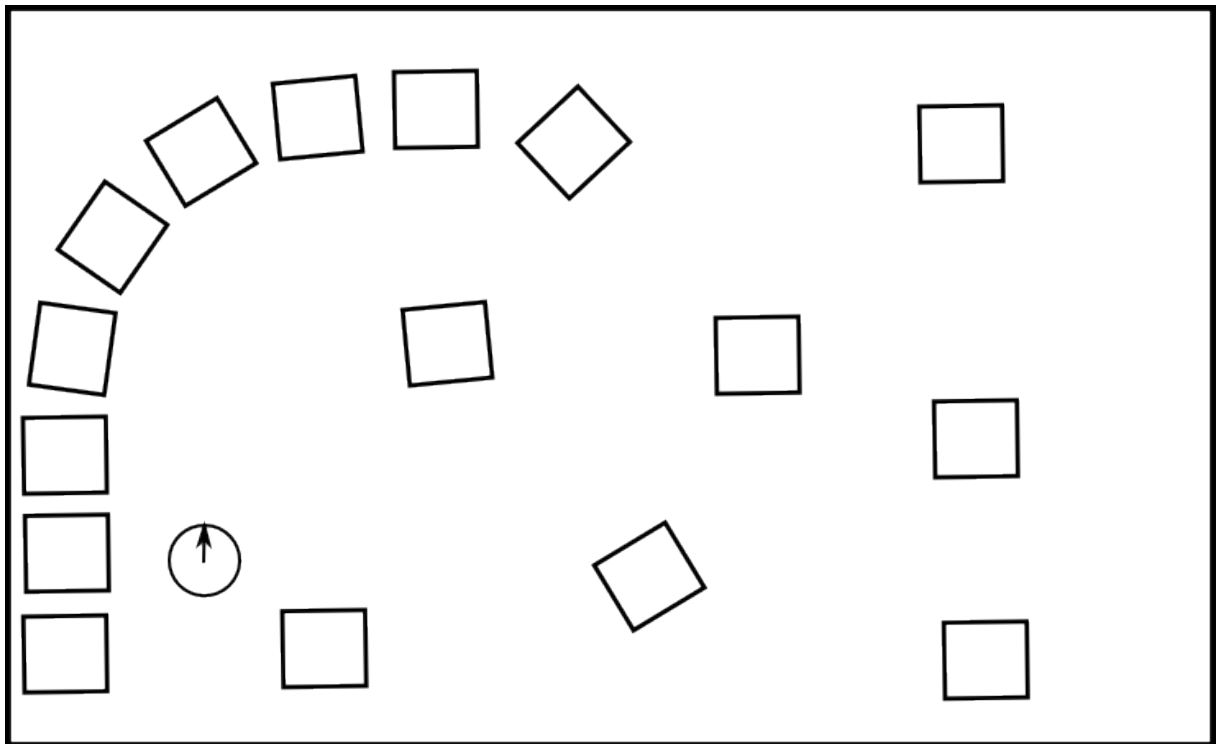


Figura 43: Ilustração do segundo teste avançado.

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 43, 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.

3.11.4 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.

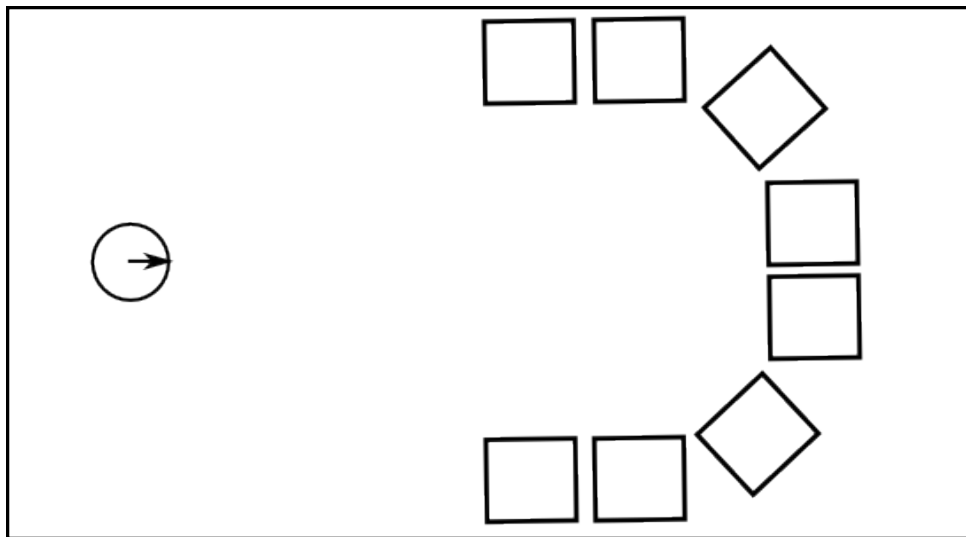


Figura 44: Ilustração do corredor sem saída.

Fonte: Autoria própria

O primeiro teste comparativo consiste em um corredor sem saída, observável na figura 44, 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 180° 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 45, onde o caminho executado pelo algoritmo *Fuzzy* está em vermelho e o ED-FCM em azul.

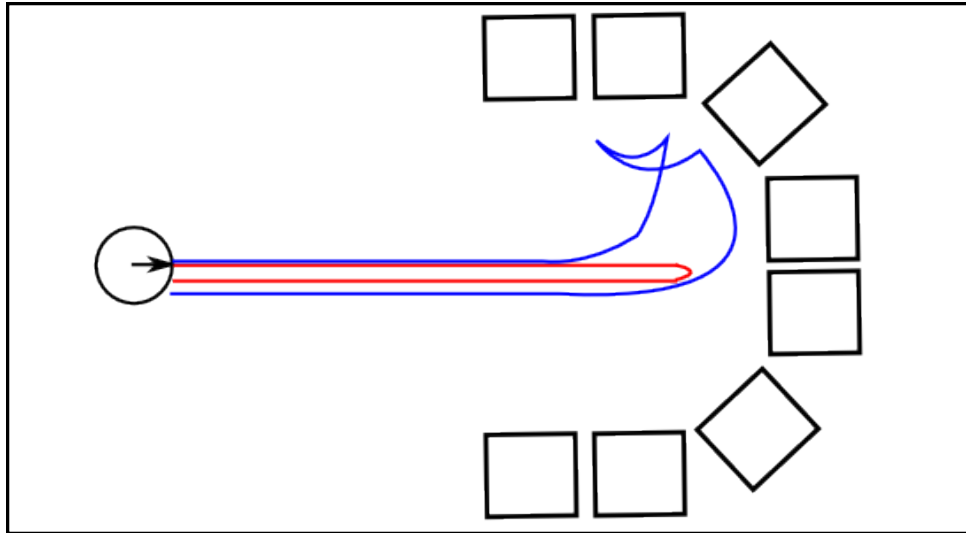


Figura 45: Esboço dos caminhos percorridos no primeiro teste comparativo.

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:

PertoEsquerda, Perto, PertoDireita \implies Lento, ViraDireita
 MedioEsquerda, Perto, MedioDireita \implies Lento, ViraDireita
 LongeEsquerda, Perto, LongeDireita \implies Lento, ViraDireita

Em compensação, o algoritmo baseado em ED-FCM apresentou dificuldades nesta situação. Na figura 45 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 3.11.3, 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.

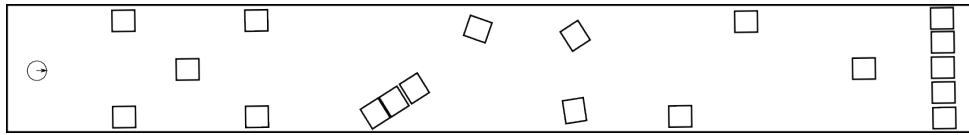


Figura 46: Esboço do segundo teste comparativo.

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 46, 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.

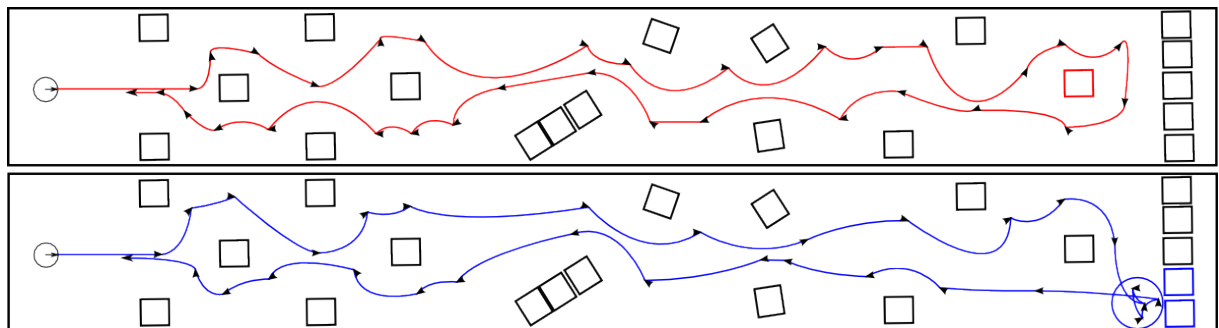


Figura 47: Esboço dos caminhos percorridos no primeiro teste comparativo.

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 47. 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 47, 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.

3.11.5 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 às 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 determinado 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 ??, 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 3.9.1. 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 3.9.4. 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 3.9.6. Finalmente, o *hardware* acoplado foi configurado e o *software* responsável pela execução dos algoritmos de navegação foi desenvolvido, conforme a seção 3.9.7. 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 ?? e a metodologia *Event-Driven Fuzzy Cognitive Maps* (ED-FCM) foi estudada e apresentada na seção ??, 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 3.10.1 e 3.10.2. 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 3.11.2. 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 3.11.3, 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 3.11.4. 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 ??, 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 ??), 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 3.9.9. 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 45 e 47, 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 procedimento 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 algoritmo *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 alterações 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 atingissem 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 confecção 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

- 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.
- 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.
- GHAOUI, L. E. **Hyper-Textbook: Optimization Models and Applications**. 2013. Disponível em: <<https://inst.eecs.berkeley.edu/ee127a/book/login/index.html>>. Acesso em: 30 de julho de 2011.
- 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.
- GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing (3rd Edition)**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 013168728X.
- LEON, S. **Linear algebra with applications**. Prentice Hall, 1998. ISBN 9780138493080. Disponível em: <<http://books.google.ie/books?id=BeAZAQAAIAAJ>>.
- LIN, H.-D. Computer-aided visual inspection of surface defects in ceramic capacitor chips. **Journal of Materials Processing Technology**, v. 189, p. 19 – 25, 2007. ISSN 0924-0136. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0924013607000052>>.
- LIN, H.-D. Tiny surface defect inspection of electronic passive components using discrete cosine transform decomposition and cumulative sum techniques. **Image and Vision Computing**, v. 26, n. 5, p. 603 – 621, 2008. ISSN 0262-8856. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0262885607001151>>.
- 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.

MAR, N.; YARLAGADDA, P.; FOOKES, C. Design and development of automatic visual inspection system for {PCB} manufacturing. **Robotics and Computer-Integrated Manufacturing**, v. 27, n. 5, p. 949 – 962, 2011. ISSN 0736-5845. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0736584511000457>>.

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

MATHWORKS. **Image Enhancement and Analysis**. 2013. Disponível em: <<http://www.mathworks.com/help/images/image-enhancement-and-analysis.html>>. Acesso em: 06 de Março de 2012.

MATHWORKS. **regionprops**. 2013. Disponível em: <<http://www.mathworks.com/help/images/ref/regionprops.html>>. Acesso em: 06 de Março de 2012.

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

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

PICCARDI, M. Background subtraction techniques: a review. In: **Systems, Man and Cybernetics, 2004 IEEE International Conference on**. [S.l.: s.n.], 2004. v. 4, p. 3099–3104 vol.4. ISSN 1062-922X.

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.

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.

WIKIPEDIA. **Second moment of area**. 2013. Disponível em: <http://en.wikipedia.org/wiki/Second_moment_of_area>. Acesso em: 29 de julho de 2011.

ZENG, Z.; MA, L.; ZHENG, Z. Automated extraction of pcb components based on specularly using layered illumination. **Journal of Intelligent Manufacturing**, Springer US, v. 22, n. 6, p. 919–932, 2011. ISSN 0956-5515. Disponível em: <<http://dx.doi.org/10.1007/s10845-009-0367-6>>.

ZHENG, H.; KONG, L.; NAHAVANDI, S. Automatic inspection of metallic surface defects using genetic algorithms. **Journal of Materials Processing Technology**, v. 125-126, n. 0, p. 427 – 433, 2002. ISSN 0924-0136. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0924013602002947>>.

APÊNDICE A – REFERÊNCIA DE MONTAGEM DO ROBÔ

A figura 48 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 3.9.7. 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 ??, TS 7260 em ?? e placa de roteamento em ??.
- 2) Especificação dos motores em ??.
- 4 a 9) Diagramas esquemáticos e *interface* da placa de roteamento com a C8051F340 em 3.9.6.
- Especificação dos Sensores e Encoders em ?? e ??.
- 10) O pendrive contém o código executado na TS7260, descrito em 3.9.7.
- 11) As pontes H são alimentadas pelos conectores de alimentação do motor.

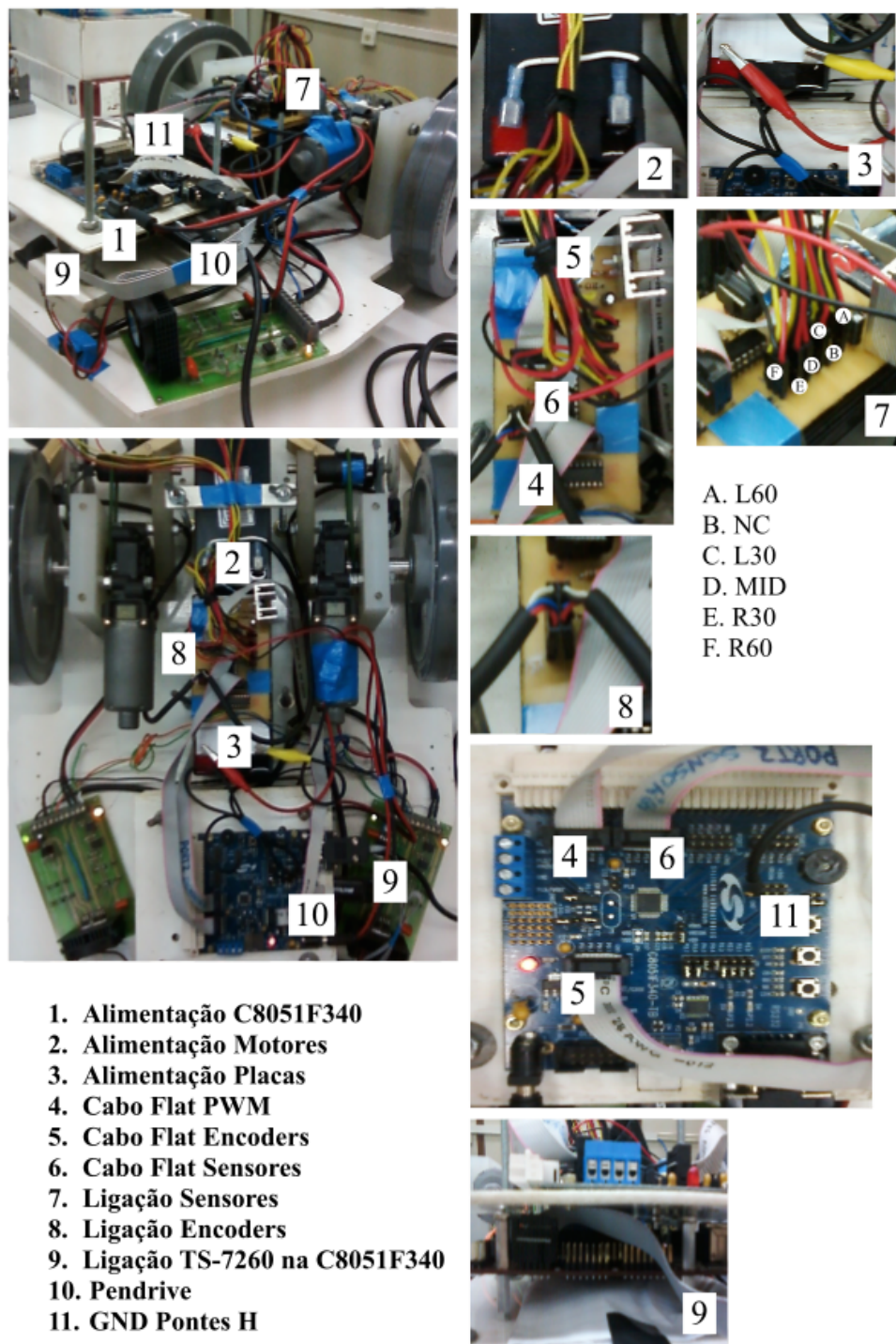


Figura 48: Referência de Montagem do Robô.

Fonte: Autoria própria