Universidade Federal de Juiz de Fora

Faculdade de Engenharia

Programa de Graduação em Engenharia Elétrica - Habilitação em Robótica e Automação Industrial

Frederick Tavares Marliere

Sistema Fuzzy de Apoio à Decisão e Estratégia Aplicados ao Futebol de Robôs

Juiz de Fora

2017

Frederick Tavares Marliere

Sistema Fuzzy de Apoio à Decisão e Estratégia Aplicados ao Futebol de Robôs

Trabalho de Conclusão de Curso apresentado ao Programa de Graduação em Engenharia Elétrica - Habilitação em Robótica e Automação Industrial da Universidade Federal de Juiz de Fora, na área de concentração em Lógica Fuzzy Aplicada ao Futebol de Robôs , como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Leonardo Rocha Olivi

Coorientador: Lucas Corrêa Netto Machado

Juiz de Fora

2017

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Marliere, Frederick Tavares.
Sistema Fuzzy de Apoio à Decisão e Estratégia Aplicados ao Futebol de Robôs / Frederick Tavares Marliere. -- 2017.
111 f.

Orientador: Leonardo Rocha Olivi
Coorientador: Lucas Corrêa Netto Machado
Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Juiz de Fora, Faculdade de Engenharia, 2017.

1. Lógica Fuzzy. 2. Futebol de Robôs. 3. Multi-robôs. I. Olivi, Leonardo Rocha, orient. II. Machado, Lucas Corrêa Netto, coorient. III. Título.

Frederick Tavares Marliere

Sistema Fuzzy de Apoio à Decisão e Estratégia Aplicados ao Futebol de Robôs

Trabalho de Conclusão de Curso apresentado ao Programa de Graduação em Engenharia Elétrica - Habilitação em Robótica e Automação Industrial da Universidade Federal de Juiz de Fora, na área de concentração em Lógica Fuzzy Aplicada ao Futebol de Robôs , como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Aprovado em: 12/07/2017

BANCA EXAMINADORA

Professor Dr. Eng. Leonardo Rocha Olivi - Orientador
Universidade Federal de Juiz de Fora

Ms. Eng. Lucas Correa Netto Machado - Coorientador
Universidade Federal de Juiz de Fora

Professor Ms. Eng. Exuperry Barros Costa
Universidade Federal de Juiz de Fora

Professora Dra. Eng. Ana Sophia Cavalcanti Alves Vilas Boas
Universidade Federal de Juiz de Fora

Universidade Federal de Juiz de Fora
Faculdade de Engenharia
Engenharia Elétrica – Robótica e Automação Industrial

ATA DE APRESENTAÇÃO DE TRABALHO FINAL DE CURSO

DATA DA DEFESA: **12/07/2017**

CANDIDATO: **FREDERICK TAVARES MARLIÉRE**

ORIENTADOR: **PROF. LEONARDO ROCHA OLIVI**

TÍTULO DO TRABALHO: **SISTEMA *FUZZY* DE APOIO À DECISÃO E ESTRATÉGIA APLICADOS AO FUTEBOL DE ROBÔS**

BANCA EXAMINADORA/INSTITUIÇÃO:

PREDIDENTE: **LEONARDO ROCHA OLIVI - UFJF**

AVALIADOR 1: **ANA SOPHIA CAVALCANTI COSTA VILAS BOAS - UFJF**

AVALIADOR 2: **EXUPERRY BARROS COSTA - UFJF**

AVALIADOR 3: **LUCAS CORRÊA NETTO MACHADO - UFJF**

HORÁRIO E LOCAL: **13H00, ANFIEATRO I, ED. ITAMAR FRANCO, FACULDADE DE ENGENHARIA, UFJF.**

Nesta data, em sessão pública, após exposição oral de 34 minutos, o candidato foi arguido pelos membros da banca. Em decorrência desta arguição, a banca considerou o candidato:

(X) **APROVADO**

( ) **REPROVADO**

Na forma regulamentar foi lavrada a presente Ata que é abaixo assinada pelos membros da banca na ordem determinada e pelo candidato:

PREDIDENTE: Leonardo Rocha Olivi

AVALIADOR 1: Ana Sophia Cavalcanti Costa Vilas Boas

AVALIADOR 2: Exuperry Barros Costa

AVALIADOR 3: Lucas Corrêa Netto Machado

CANDIDATO: Frederick Tavares Marliere

Juiz de Fora, 12 de julho de 2017.

Vistos: *(coordenador do curso ou presidente da comissão de TFCs do curso).*

Engenharia Elétrica
Robótica e Automação
COORDENAÇÃO DE CURSO
Faculdade de Engenharia - UFJF

AGRADECIMENTOS

Diante da execução deste Trabalho de Conclusão de Curso, mais uma etapa é vencida. Agora mudam-se as metas e as expectativas para novas conquistas. Agradeço primeiramente e acima de tudo à Deus, Sublime Arquiteto do Universo, que nos proporciona a coragem para vencer os desafios e a determinação para escrever nosso próprio destino, agradeço por me guiar durante toda minha jornada.

Agradeço aos meus pais Dartagnan e Rosane, exemplos de equilíbrio e sabedoria, que me deram a vida, me ensinaram a vivê-la com dignidade e honestidade acima de tudo, a quem devo tudo que sou e que conquisto, agradeço por todo auxílio, amor e carinho, eternamente. Aos meus irmãos, Monique e Daniel, pelo companheirismo e apoio. Aos meus avós, exemplos de superação e força fundamentais em minha formação acadêmica e pessoal.

Agradeço ao meu orientador Leonardo Rocha Olivi, não só pelo apoio neste trabalho, mas por ser o expoente que é na Faculdade de Engenharia da UFJF, exemplo de profissionalismo, caráter e dedicação. Mostrando que ser professor é muito além de ter titulações, e sim ter o dom de ensinar, e fazer isso com amor e empenho. Sem dúvidas é um dos grandes responsáveis pelo avanço e melhoria da Faculdade de Engenharia Elétrica - Robótica e Automação.

Um agradecimento especial aos amigos que fiz nessa caminhada, Gustavo Hofstätter e Pedro Alcântara, que participaram de todas as conquistas e trabalhos realizados. Juntos fomos capazes de compartilhar os prazeres e dificuldades impostas durante esse curso, quem sempre serão lembrados em minha vida pessoal e profissional.

Agradeço ao núcleo de Estratégia da *Rinobot Team*, em especial aos amigos Gabriel Barros, João Paulo Pimentel, Matheus Pimentel e Rodrigo Gonçalves. Pessoas com um futuro brilhante, exemplos de determinação e foco. Juntos, trabalhamos em tornar possível este trabalho sair do ramo das ideias para chegar à implementação prática. À esses amigos, que não poderia deixar de citar, fica expressa minha gratidão.

Finalmente, deixo expresso meu agradecimento à minha namorada Fabiana, por toda a compreensão, carinho e dedicação que lhe é peculiar. Por me ajudar a ultrapassar todas as dificuldades e barreiras impostas nesse curso de graduação. Por me dar forças e empenho no cumprimento dessa jornada.

“ Minha religião consiste em uma humilde
admiração ao espírito superior e ilimitado
que se revela através dos leves detalhes
que somos capazes de perceber
com nossa mente frágil e débil.”
(Albert Einstein)

RESUMO

Este trabalho aborda o estudo sobre a lógica *fuzzy* e implementação da mesma em um sistema de apoio à decisão voltado para a estratégia da *Rinobot Team* da Universidade Federal de Juiz de Fora (UFJF) no contexto do futebol de robôs autônomos. Esse estudo possibilita a criação de um sistema autônomo que identifica as condições instantâneas de jogo e gera uma decisão a ser tomada no instante desejado. Este trabalho ainda aborda o estudo e implementação acerca dos métodos de navegação Campos Potenciais Harmônicos (CPH) e Campos Potenciais Orientados (CPO). Com a decisão gerada pelo sistema baseado em lógica *fuzzy*, é possível implantar um método de navegação à cada possível decisão. Assim, a cada possível saída do sistema *fuzzy*, uma navegação com características distintas é acionada, garantindo a dinâmica desejável ao jogo. Os resultados mostram uma solução efetiva e computacionalmente leve para um problema altamente não-linear multi-robôs.

Palavras-chave: Lógica Fuzzy. Futebol de Robôs. Multi-robôs.

ABSTRACT

This work studies and applies fuzzy logic and its implementation in a decision support system in the context of autonomous robot soccer, focused on the strategy of the Rinobot Robot Soccer Team of the Federal University of Juiz de Fora (UFJF). This study enables the creation of an autonomous system that identifies instantaneous conditions of play and generates a decision of strategy at all moments of the match. This work also addresses the study and implementation of the Harmonic Potential Fields (HPF) and Potential Guided Fields (PGF) navigation methods. With the decision generated by the system based on fuzzy logic, it is possible to deploy a navigation method to each possible decision. Thus, each possible output of the fuzzy system triggers a navigation with different characteristics, guaranteeing the desirable dynamics to the game. The results show an effective and computationally light solution for a highly nonlinear multi-robot problem.

Key-words: Fuzzy logic. Soccer Robotics. Multi-robot task.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de robô móvel: Pioneer P3-DX. 16
Figura 2 – Sistema geral de futebol de robôs autônomos. 17
Figura 3 – Áreas da Inteligência Computacional. 20
Figura 4 – Exemplo de conjunto clássico. 21
Figura 5 – Exemplo de conjunto *fuzzy*. 21
Figura 6 – Função de pertinência clássica e *fuzzy*. 22
Figura 7 – Função de pertinência triangular. 23
Figura 8 – Função de pertinência trapezoidal. 23
Figura 9 – Função de pertinência em formato de sino ou gaussiana. 24
Figura 10 – Exemplos de funções de pertinência *fuzzy*. 25
Figura 11 – Resultado da operação União. 25
Figura 12 – Resultado da operação Interseção 26
Figura 13 – Resultado da operação Complementar em relação ao conjunto *fuzzy* A. 26
Figura 14 – Visão geral de um sistema *fuzzy*. 27
Figura 15 – Exemplo de fuzzyficação. 28
Figura 16 – Exemplo inferência de Mamdani. 31
Figura 17 – Visão gráfica do método centroide. 32
Figura 18 – Visão gráfica método média dos máximos. 33
Figura 19 – Visão geral do futebol de robôs VSSS. 36
Figura 20 – Visão superior do campo indicando as demarcações das áreas de ataque e defesa. 37
Figura 21 – Ilustração exemplo para cálculo de FA. Considere o ângulo A como sendo o ângulo entre o robô e a bola e o ângulo B como sendo o ângulo do robô. 40
Figura 22 – Ilustração das funções de pertinência adotadas no parâmetro FD. . . . 43
Figura 23 – Ilustração das funções de pertinência adotadas no parâmetro FC. . . . 43
Figura 24 – Ilustração das funções de pertinência adotadas no parâmetro FA. . . . 44
Figura 25 – Ilustração das funções de pertinência adotadas na saída do sistema *fuzzy*. 45
Figura 26 – Orientação capturada pelo sistema à partir da visão superior do robô. . 46
Figura 27 – Proposição de possíveis comportamentos do sistema modelado a partir de situações comuns ou típicas de jogo. 47
Figura 28 – Base de Regras proposta para o sistema *fuzzy* 48
Figura 29 – Exemplo de Campos Potenciais Harmônicos (CPH). 51
Figura 30 – Influência do vetor $v\angle 45°$ no CPO. 54
Figura 31 – Influência do vetor $v\angle 90°$ no CPO. 54
Figura 32 – Influência do vetor $v\angle 135°$ no CPO. 55
Figura 33 – Comparação entres as trajetórias segundo várias taxas de influência ϵ. . 55
Figura 34 – Campo instável, $\epsilon = 4$. 56

Figura 35 – Resultados obtidos em relação às entradas propostas na criação da base de regras *fuzzy*. 64
Figura 36 – Sistema *fuzzy* simulado em [24] 64
Figura 37 – Resultado obtido para uma entrada [0.53, 0.03, 0.66] 65
Figura 38 – Resultado obtido para uma entrada [0.34, 0.59, 0.06] 65
Figura 39 – Resultado obtido para uma entrada [0.95, 0.95, 0.55] 66
Figura 40 – Exemplo de movimentação relativa à função de jogo *Defesa Pesada*. . . 67
Figura 41 – Exemplo de movimentação relativa à função de jogo *Defesa Leve*. . . . 67
Figura 42 – Exemplo de movimentação relativa à função de jogo *Ataque Leve*. . . . 68
Figura 43 – Exemplo de movimentação relativa à função de jogo *Ataque Pesado*. . . 68
Figura 44 – Simulação de comportamento sem utilizar o drible. 74
Figura 45 – Simulação de comportamento utilizando o drible. 74
Figura 46 – Fotos 1, 2 e 3 77
Figura 47 – Fotos 4, 5 e 6 77
Figura 48 – Fotos 7, 8 e 9 77
Figura 49 – Fotos 10, 11 e 12 77
Figura 50 – Fotos 13, 14 e 15 77
Figura 51 – Fotos 16, 17 e 18 78
Figura 52 – Fotos 19, 20 e 21 78
Figura 53 – Fotos 22, 23 e 24 78
Figura 54 – Fotos 25, 26 e 27 78
Figura 55 – Fotos 28, 29 e 30 78
Figura 56 – Fotos 31, 32 e 33 78
Figura 57 – Fotos 34, 35 e 36 79
Figura 58 – Fotos 37, 38 e 39 79
Figura 59 – Fotos 40, 41 e 42 79
Figura 60 – Fotos 43, 44 e 45 79
Figura 61 – Fotos 46, 47 e 48 79
Figura 62 – Fotos 49, 50 e 51 79
Figura 63 – Fotos 52, 53 e 54 80
Figura 64 – Fotos 55, 56 e 57 80
Figura 65 – Fotos 58, 59 e 60 80
Figura 66 – Fotos 61, 62 e 63 80
Figura 67 – Fotos 64, 65 e 66 80
Figura 68 – Fotos 67, 68 e 69 80
Figura 69 – Fotos 70, 71 e 72 81
Figura 70 – Foto 73 81

LISTA DE ABREVIATURAS E SIGLAS

UFJF	Universidade Federal de Juiz de Fora
VSSS	Very Small Size Soccer
PID	Proporcional, Integral e Derivativo
CP	Campos Potenciais
CPH	Campos Potenciais Harmônicos
CPO	Campos Potenciais Orientados
IA	Inteligência Artificial
IC	Inteligência Computacional
SBR	Sistemas Baseados em Regras
FD	Fator Defesa
FC	Fator Competição
FA	Fator Ângulo
PVC	Problema de Valor de Contorno
DP	Defesa Pesada
DL	Defesa Leve
AL	Ataque Leve
AP	Ataque Pesado
ATK	Ataque
RNA	Redes Neurais Artificiais

LISTA DE SÍMBOLOS

$\in$	Pertence
$\mathbb{R}$	Conjunto dos Números Reais
$\mathbb{R}^2$	Plano Cartesiano de Duas Dimensões
$\|v\|$	Norma do Vetor v
$v\angle\alpha$	Vetor v com ângulo de orientação α graus.
∇	Gradiente
$\forall$	Para todo
$\subset$	Está contido
$\cup$	União
$\cap$	Interseção
$\neg$	Complemento
∂	Derivada de primeira ordem
∂^2	Derivada de segunda ordem
$\|v\|$	Módulo de v

SUMÁRIO

1	**INTRODUÇÃO**	**15**
1.1	FUTEBOL DE ROBÔS	16
1.2	OBJETIVOS	18
1.3	ORGANIZAÇÃO DO TRABALHO	19
2	**LÓGICA FUZZY**	**20**
2.1	CONJUNTOS FUZZY	20
2.1.1	**Graus e Funções de Pertinência**	21
2.1.2	**Operações Básicas em Conjuntos *Fuzzy***	24
2.2	SISTEMAS BASEADOS EM REGRAS FUZZY	26
2.2.1	**Fuzzyficação**	27
2.2.2	**Inferência e Base de Regras**	28
2.2.3	**Defuzzyficação**	31
2.3	APLICAÇÕES DA LÓGICA FUZZY	33
3	**IMPLEMENTAÇÃO SISTEMA FUZZY DE APOIO À DECISÃO**	**35**
3.1	CONTEXTO DE FUTEBOL DE ROBÔS	35
3.2	MODELAGEM DAS CONDIÇÕES DE JOGO	36
3.2.1	**Fator Defesa (FD)**	36
3.2.2	**Fator Competição (FC)**	38
3.2.3	**Fator Ângulo (FA)**	39
3.3	FUNÇÕES DE PERTINÊNCIA	42
3.4	BASE DE REGRAS	45
4	**SISTEMAS DE NAVEGAÇÃO**	**49**
4.1	VISÃO GERAL	49
4.2	CAMPOS POTENCIAIS HARMONICOS (CPH)	50
4.3	CAMPOS POTENCIAIS ORIENTADOS (CPO)	51
5	**IMPLEMENTAÇÃO DE SISTEMAS DE NAVEGAÇÃO EM FUNÇÕES DE JOGO**	**57**
5.1	FUNÇÃO DEFESA PESADA	57
5.2	FUNÇÃO DEFESA LEVE	58
5.3	FUNÇÃO ATAQUE LEVE	60
5.4	FUNÇÃO ATAQUE PESADO	61

6 RESULTADOS OBTIDOS 63
6.1 SISTEMA *FUZZY* DE APOIO À DECISÃO 63
6.2 SISTEMAS DE NAVEGAÇÃO APLICADOS ÀS FUNÇÕES DE JOGO 66

7 CONCLUSÕES 70
7.1 PONTOS POSITIVOS 70
7.2 PONTOS NEGATIVOS E LIMITAÇÕES DO SISTEMA 71
7.3 TRABALHOS FUTUROS E POSSÍVEIS MELHORIAS 72

REFERÊNCIAS 75

APÊNDICE A – Fotos dos Testes para Criação da Base de Regras 77

APÊNDICE B – Código em C++ do Sistema Real VSSS - arquivo fuzzy.cpp 82

APÊNDICE C – Código em C++ do Método CPH 96

APÊNDICE D – Código em C++ do Método CPO 103

1 INTRODUÇÃO

Atualmente, segundo [1], a área da robótica é um ramo de estudo que cresce com o aumento das pesquisas, investimentos, e principalmente, pela busca do aperfeiçoamento em ramos como o industrial, no qual a robótica visa a realização de atividades de forma mais rápida, correta, segura e eficaz, propiciando maior produtividade e qualidade ao produto final, que é o objetivo de empresas e indústrias, minimizando operações e sendo capaz de trabalhar longas horas ininterruptas.

A robótica no cenário atual de um mundo tecnológico e globalizado se faz presente e necessária em mais variados tipos de atividades [2]. Segundo [3], pode-se citar, por exemplo, a robótica assistiva (propósito de auxiliar as limitações físicas ou mecânicas de um ser humano portador de deficiência); a robótica com propósito de produção industrial (operar mais rápido, e mais eficaz que o trabalho manual); a robótica móvel com fins de exploração de ambientes; dentre outras inúmeras aplicações que a robótica abrange, dispondo de uma tecnologia eficaz, e contribuindo com o avanço da ciência e da engenharia.

De acordo com as características e aplicações da robótica expostas em [1], [2] e [3], pode-se inferir que um sistema robótico é composto por vários subsistemas que necessitam trabalhar em conjunto para propiciar um perfeito funcionamento, sobretudo no contexto da robótica móvel. Um robô, para ter um grau de autonomia, ou seja, capacidade de executar ações sem a intervenção humana, necessita, no mínimo, de subsistemas relacionados à propulsão, sensoriamento, atuadores, processamento, comunicação e alimentação. A título ilustrativo, a Figura 1 apresenta um exemplo de robô móvel (Pioneer P3-DX).

O subsistema associado ao movimento (tração/propulsão) atua com rodas, esteiras, pernas. Um subsistema de sensoriamento é capaz de medir grandezas do ambiente e bem como do próprio robô. Um subsistema de atuadores, como braços robóticos, garras, ferramentas, é capaz de interagir com o ambiente. O processamento é capaz de executar as informações de *software* do robô. A comunicação conecta o robô com o mundo exterior. Finalmente, um subsistema de alimentação fornece energia ao robô para realizar suas tarefas.

Figura 1 – Exemplo de robô móvel: Pioneer P3-DX.

Fonte: Retirado de [4]

1.1 FUTEBOL DE ROBÔS

No contexto da robótica móvel e sistemas robóticos, a Universidade Federal de Juiz de Fora (UFJF) possui um grupo de projeto denomidado *Rinobot Team* cujo desenvolvimento de atividades, dentre outras, é a participação em projetos e competições de futebol de robôs autônomos.

A categoria desse desenvolvimento de atividades é a *IEEE "Very Small Size Soccer"* (VSSS). Segundo [5], nessa categoria os robôs competidores não devem ultrapassar o tamanho limite de 7,5cm x 7,5cm x 7,5cm; os robôs utilizados pela Equipe *Rinobot* são robôs diferenciais, ou seja, possuem duas rodas motorizadas; são dispostos três robôs para o jogo de futebol. Nesse contexto de futebol de robôs, o jogo ocorre sem a intervenção humana, os robôs são radiocontrolados por um computador que processa todas as informações de jogo e envia comandos aos robôs de maneira integralmente autônoma, conforme ilustrado na Figura 2.

Nesse cenário de futebol de robôs VSSS, existem vários desafios e várias áreas as quais devem funcionar em harmonia para que essa atividade ocorra com sucesso. Ainda em [5], o futebol VSSS para ocorrer necessita de algumas funcionalidades primordiais, são essas:

- Sistema de visão computacional: capaz de identificar as posições dos agentes robóticos em campo, bem como suas orientações, retornando essas informações com o mínimo de ruídos ou incertezas possíveis, contribuindo para a robustez do sistema, em geral.
- *Hardware*: montagem e construção do robô, de forma que suas múltiplas peças e eletrônica como circuito de acionamento, motores, carcaça física, circuito de comunicação e microprocessador estejam em harmonia com a integralidade física do agente robótico.

- Sistema de Controle e Comunicação: os robôs devem ser controlados em seu baixo nível, ou seja, devem conseguir receber informações provenientes de um *host* (computador), como velocidades e, dessa forma, devem possibilitar que exista o controle dos motores de acionamento para que o robô reaja a esse comando de maneira satisfatória e gerando as devidas velocidades, possibilitando sua mobilidade de maneira satisfatória, uma vez que o futebol VSSS necessita de que os agentes tenham um comportamento desejável no decorrer do jogo.

- Sistema de Estratégia e *Software*: essa área deve-se concentrar na criação da inteligência da partida. O sistema de navegação *(Path Planning)* possibilitará o robô atingir um ponto ou uma meta desejável, evitando colisões e ou possíveis obstáculos, bem como a inteligência embarcada aos robôs, que irá processar as informações de jogo e alterar a cada percepção da câmera as decisões e/ou comportamentos desejáveis no decorrer da partida são tarefas primordiais nessa área do projeto.

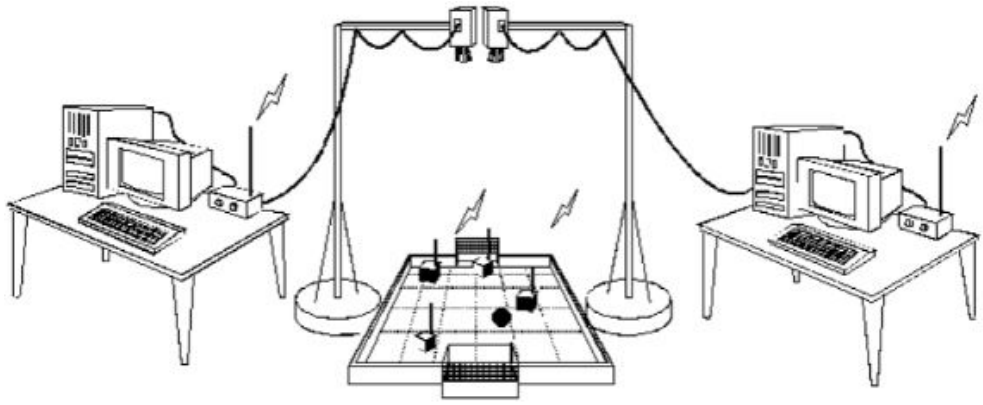

Figura 2 – Sistema geral de futebol de robôs autônomos.

Fonte: Retirado de [5]

O futebol de robôs VSSS integra todas as áreas descritas em um único sistema, que deve ser robusto e competitivo o suficiente. O sistema deve, através da visão computacional, identificar as posições dos agentes, ângulos de orientação dos agentes robóticos do time, as áreas de defesa e ataque delimitadas, posicionamento da bola em campo. A fragilidade nessa área quanto à identificação por cores dos robôs em jogo e/ou flutuações em ângulos de orientação pode comprometer o sistema de estratégia adotado.

O controle dos agentes robóticos passa por ajustes de controladores Proporcionais - Integrais - Derivativo (PID) para garantir que as velocidades calculadas pela estratégia e passadas como referência sejam atingidas pelas rodas, de forma que o robô atinja a velocidade ideal de forma mais precisa possível, necessitando de um controlador de baixo nível e bem como um sistema de comunicação adequados (radiofrequência, *wifi* e *bluetooth*

são os exemplos mais comuns) robusto o suficiente e que não perca a comunicação com os agentes robóticos. Essa área engloba a parte de *hardware*, relacionada aos circuitos e eletrônica interna do robô e bem como sua estrutura física.

A parte de *Software* do sistema deve ser inteligente o bastante e autônoma de forma a adotar a melhor estratégia mediante as circunstâncias de jogo. Essa área visa, por exemplo, a adoção de sistemas de navegação adequados e algoritmos ou técnicas para o controle de alto nível dos robôs em campo. Esse documento aborda especificamente práticas nessa área de atuação, em que o sistema possa ter autonomia para tomada de decisões e bem como utilizar de navegações próprias para cada decisão tomada.

1.2 OBJETIVOS

No contexto geral do futebol de robôs autônomos VSSS, percebe-se a importância de um sistema de estratégia robusto, o qual define o comportamento dos agentes robóticos de forma instantânea e autônoma. Essa peculiaridade torna o sistema de estratégia de forma geral como o centro ou a fonte de tomada de decisão do sistema como um todo, pois é o núcleo que define o comportamento desejado dos três robôs dispostos no campo para o futebol.

Este trabalho em questão é motivado pelo aprimoramento e avanço na implementação de técnicas possíveis no âmbito da estratégia em futebol de robôs autônomos. Visando assim aprimorar a utilização de estratégias de jogo da *Equipe Rinobot* da UFJF.

A estratégia abordada pela Equipe *Rinobot* passava por uma programação e seleção manual de comportamentos desejados mediante a ocorrência de eventos de interesse durante o jogo. De forma sucinta, a estratégia se baseava em comandos do tipo <SE ocorre A, então faça B>, sendo Campos Potenciais (CP) o único sistema de navegação adotado, o qual apresenta limitações a serem vistas neste trabalho. No entanto, é natural o aprimoramento do sistema de forma a ocorrer uma maior autonomia em tomadas de decisões, bem como a superação dos limitantes da navegação utilizada assim como a utilização de navegações próprias para cada decisão gerada.

Em síntese, o objetivo deste trabalho é utilizar no sistema de futebol de robôs um algoritmo *Fuzzy* de apoio à decisão. Assim, gerando um sistema autônomo que identifica as condições instantâneas de jogo, analisa as peculiaridades no mesmo instante e gera uma decisão no instante desejado. Gerando assim, um agente supervisor decisório que representa atitudes esperadas de cada agente robótico, mediante uma condição de jogo.

Com a decisão gerada, é possível implementar e implantar o sistema de navegação de forma particular à cada possível decisão, assim, a cada saída do sistema *fuzzy*, torna-se necessário implementar uma estratégia distinta, garantindo a dinâmica desejável ao jogo. Este trabalho visa implantar um sistema sobretudo autônomo para o VSSS.

1.3 ORGANIZAÇÃO DO TRABALHO

O presente trabalho encontra-se dividido em sete capítulos. Sendo o presente capítulo onde se encontram as noções introdutórias do trabalho, no âmbito da robótica móvel e do futebol de robôs autônomos, bem como a explicitação dos objetivos primordiais do mesmo, possibilitando uma contextualização prévia do estudo a ser feito.

O Capítulo 2 apresenta as fundamentações técnicas básicas da lógica e teoria *fuzzy*, base da implementação deste trabalho. Apresentando assim seus conceitos básicos, formulação matemática e métodos que possibilitam a criação de um sistema decisório implantado no ambiente do futebol de robôs, objetivo do trabalho atual.

O Capítulo 3 descreve de forma detalhada a implementação realizada de um sistema de apoio à decisão baseado na lógica *fuzzy* e aplicado ao futebol de robôs, apresentando a ideia principal da implementação, os testes realizados e o método de decisão utilizado.

O Capítulo 4 faz uma abordagem de revisão teórica e básica a respeito dos sistemas de navegação no contexto dos Campos Potenciais (CP), envolvendo os Campos Potenciais Harmônicos (CPH) e Campos Potenciais Orientados (CPO), o qual é importante para garantir uma navegação robusta e aceitável aos agentes robóticos no contexto do jogo de futebol.

O Capítulo 5 apresenta a implementação dos sistemas de navegação utilizados para cada saída do sistema de decisão implementado. Assim, apresenta a estratégia para cada decisão tomada, enfocando assim a navegação utilizada e a adaptação das mesmas para cada necessidade gerada pelo sistema decisório criado.

O Capítulo 6 se baseia na apresentação dos resultados obtidos por este trabalho realizado. Os resultados acerca da implementação do sistema *fuzzy* para apoio à decisão são expostos nesse capítulo. São apresentados também os resultados gerados em simulação da implementação proposta para os sistemas de navegação utilizados mediante cada decisão gerada pelo sistema baseado em algoritmo *fuzzy*.

O Capítulo 7 enfoca as conclusões gerais e considerações do trabalho feito. Relatando os pontos fortes da estratégia implementada, os pontos negativos da mesma, as limitações, as possíveis mudanças e melhorias, dando a perspectiva para futuros trabalhos no presente ramo da robótica aplicada ao futebol de robôs autônomos.

Por fim, os anexos ou apêndices do trabalho apresentam as fotografias dos testes realizados para criação do sistema *fuzzy*, bem como os códigos criados e implementados para utilização do sistema de apoio à decisão criado e bem como a navegação adotada para as estratégias das funções de jogo criadas.

2 LÓGICA FUZZY

Este Capítulo aborda de forma teórica os conceitos da lógica *fuzzy* e teorias básicas, contextualizando a implementação de um sistema de apoio à decisão utilizando a teoria *fuzzy* de análise quantitativa/qualitativa.

Atualmente, com o advento das novas tecnologias e sistemas autônomos ou inteligentes, existe uma sub-área da Inteligência Artificial (IA) que é identificada por Inteligência Computacional (IC). Nesse contexto, em [6] trata-se IC como uma área da IA que possui técnicas para modelar comportamentos de forma a se desenvolver máquinas inteligentes ou autônomas. Ainda em [6], a IC se divide em três grupos básicos e principais, conforme a Figura 3, sendo esses: a computação ou sistema neural; os sistemas evolutivos e a computação *fuzzy* ou nebulosa, que será o enfoque deste capítulo.

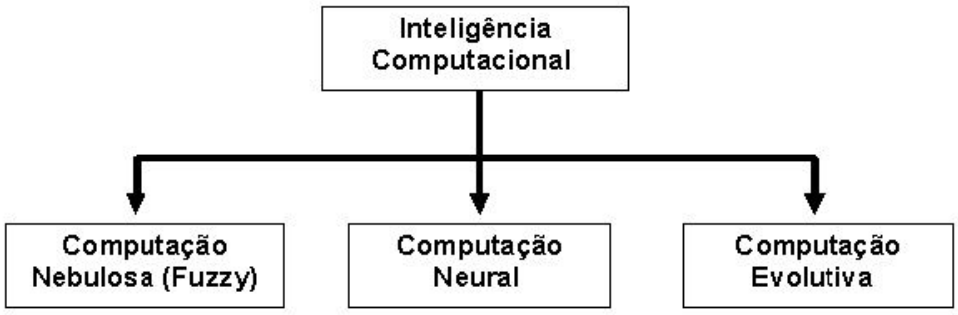

Figura 3 – Áreas da Inteligência Computacional.

Fonte: Retirado de [7]

2.1 CONJUNTOS FUZZY

Inicialmente, em 1965, a teoria *fuzzy* foi formulada por Lotfi Asker Zadeh, um matemático. Em sua publicação [8] propõe a ideia de uma modelagem matemática para termos linguísticos subjetivos. Seu objetivo era modelar matemáticamente e computacionalmente conceitos vagos, a exemplo de conceitos tipicamente humanos. Desse objetivo surge a palavra "*fuzzy*", que da origem inglesa pode significar incerto; vago; subjetivo; ou, em sua tradução mais utilizada em engenharia, nebuloso, já utilizado anteriormente como uma sub-área da IC.

Ainda em [8], o objetivo da lógica *fuzzy* é a formulação matemática de conceitos ambíguos ou imprecisos. Algo até então diferente da lógica *booleana* utilizada no cenário da computação. Esses conceitos vagos traduzem típicamente o pensamento humano. O objetivo então era fornecer meios para que fosse possível implementar em meios computacionais uma forma de agir conforme os conceitos da mente humana, sendo possível a tomada de decisões, por exemplo, em um ambiente de incertezas.

2.1.1 **Graus e Funções de Pertinência**

A teoria dos conjuntos *fuzzy*, diferentemente da teoria de conjuntos clássica, define um conjunto *fuzzy* por uma ou várias propriedade(s) incerta(s). Na lógica de conjuntos tradicional booleana, os elementos pertencem, ou não, a um dado conjunto de acordo com uma propriedade. Existindo apenas o verdadeiro ou falso em relação à pertinência do suposto elemento ao suposto conjunto. Na teoria *fuzzy*, um elemento pode pertencer mais a um determinado conjunto, e menos em outro conjunto, possuindo assim uma função de pertinência que identifica o quanto um elemento pertence a um conjunto e a outro, ou seja, o grau de pertinência.

O grau de pertinência de um elemento em relação a um conjunto, na teoria *fuzzy*, é um número contido em um intervalo fechado entre zero e um. Enquanto na teoria de conjuntos clássica ou tradicional, o grau de pertinência é apenas o valor zero ou um, significando que o elemento está não contido ou contido em um conjunto [8].

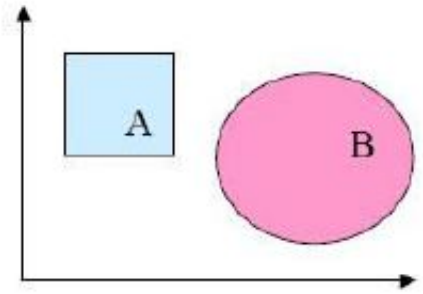

Figura 4 – Exemplo de conjunto clássico.

Fonte: Retirado de [7]

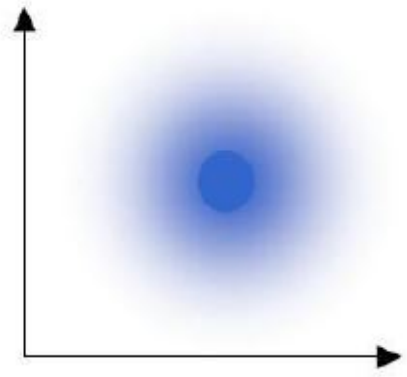

Figura 5 – Exemplo de conjunto *fuzzy*.

Fonte: Retirado de [7]

Na Figura 4, representa-se um exemplo típico de conjuntos clássicos, onde seus limites são bem definidos por propriedades bem definidas. Ainda que houvesse interseção dos conjuntos A e B, os elementos pertencentes à interseção teriam graus de pertinência unitário em relação aos dois conjuntos. Ou seja, os elementos da interseção pertencem integralmente aos conjuntos A e B.

Na Figura 5, por sua vez, é apresentado um exemplo de conjunto *fuzzy*, onde os limites não são bem definidos. Ou seja, um elemento pode pertencer parcialmente a determinado conjunto, ou até integralmente. No entanto existe a variação dos graus de pertinência, não limitando-se em apenas pertencer ou não a um determinado conjunto específico.

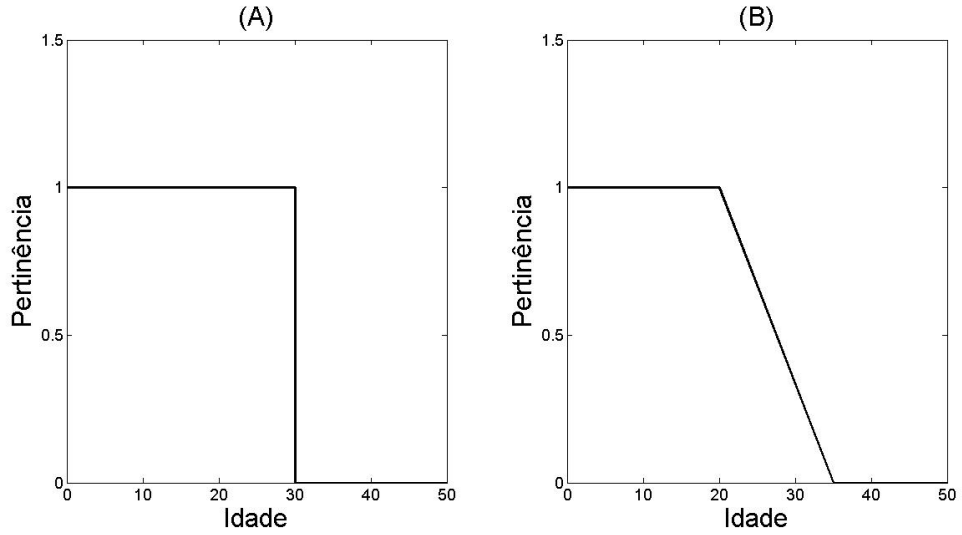

Figura 6 – Função de pertinência clássica e *fuzzy*.

Fonte: Baseado em [9]

Na Figura 6 pode-se perceber a diferença primordial entre as funções de pertinência dos conjuntos clássicos e dos conjuntos *fuzzy*. Na teoria clássica de conjuntos, denominada conjuntos *crisp*, a função de pertinência sugere os graus de pertinência unitário ou nulo. Indicando a inclusão ou exclusão integral do elemento ao conjunto idade, por exemplo. Enquanto [8] sugere elementos mais pertencentes ao conjunto do que outros. O grau de pertinência assume valores entre zero e um, indicando completa exclusão e completa pertinência, respectivamente.

De acordo com [10], na teoria *fuzzy* pode-se definir as funções de pertinência como funções que associam valores de um universo de discurso real a um número real contido no intervalo fechado entre zero e um.

$$\mu_F : U \rightarrow [0,1] \text{ ou } [0,100\%] \tag{2.1}$$

Em [11] afirma-se como sendo as funções de pertinência em sistemas *fuzzy* funções contínuas que podem possuir diversos tipos e formatos distintos. Os exemplos mais comuns de funções de pertinência são apresentados a seguir, de acordo com o seu tipo. As mais comuns são funções triangulares, trapezoidais e exponenciais de uma forma geral. Entretanto, existem inúmeras e diversas formas de funções de pertinência para modelar um sistema *fuzzy*. A escolha desse tipo de função depende da análise da aplicação e contexto proposto para modelagem *fuzzy*. As funções podem ser representadas por μ(x) ou φ(x).

- *Função Triangular:* A função de pertinência triangular pode ser apresentada sob o seguinte formato:

$$\varphi_A(x) = \begin{cases} 0, \text{ se } x \leq a, \\ \frac{x-a}{u-a}, \text{ se } a < x \leq u, \\ \frac{x-b}{u-b}, \text{ se } u \leq x < b, \\ 0, \text{ se } x \geq b. \end{cases} \tag{2.2}$$

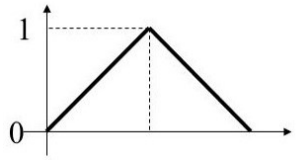

Figura 7 – Função de pertinência triangular.

Fonte: Baseado em [17]

- *Função Trapezoidal:* A função de pertinência trapezoidal pode ser apresentada sob o seguinte formato:

$$\varphi_A(x) = \begin{cases} \frac{x-a}{b-a}, \text{ se } a \leq x < b, \\ 1, \text{ se } b \leq x \leq c \\ \frac{d-x}{d-c}, \text{ se } c < x \leq d, \\ 0, \text{ caso contrário.} \end{cases} \tag{2.3}$$

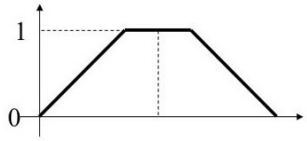

Figura 8 – Função de pertinência trapezoidal.

Fonte: Baseado em [17]

- *Função Sino ou Gaussiana:* A função de pertinência em formato de sino pode ser apresentada sob o seguinte aspecto:

$$\varphi_A(x) = \begin{cases} exp(\frac{-(x-u)^2}{a}), \text{ se } u - \delta \leq x \leq u + \delta, \\ 0, \text{ caso contrário.} \end{cases} \tag{2.4}$$

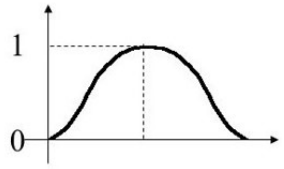

Figura 9 – Função de pertinência em formato de sino ou gaussiana.

Fonte: Baseado em [17]

2.1.2 Operações Básicas em Conjuntos *Fuzzy*

Segundo [8], as operações básicas em conjuntos *fuzzy* são as mesmas utilizadas em teoria de conjuntos clássica. Assim, as três operações básicas em conjuntos *fuzzy* são: união, interseção e o complementar ou negação. Sendo assim, apresenta-se as três operações básicas de forma detalhada:

- *União*:
  A união entre conjuntos *fuzzy* pode ser representada por $\mathbf{A} \cup \mathbf{B}$ ou $\mathbf{A} + \mathbf{B}$. A função de pertinência resultante da união de dois conjuntos *fuzzy* pode ser vista a seguir, sendo que o operador união corresponde ao conectivo "OU"/"OR".

$$\mu_{A \cup B} = max[\mu_A(x_i), \mu_B(x_i)] \tag{2.5}$$

- *Interseção*:
  A interseção entre conjuntos *fuzzy* pode ser representada por $\mathbf{A} \cap \mathbf{B}$ ou $\mathbf{A} \cdot \mathbf{B}$. A função de pertinência resultante da interseção de dois conjuntos *fuzzy* pode ser vista a seguir, sendo que o operador interseção corresponde ao conectivo "E".

$$\mu_{A \cap B} = min[\mu_A(x_i), \mu_B(x_i)] \tag{2.6}$$

- *Complemento*:
  O complementar de um conjunto *fuzzy* pode ser representado por $\neg\mathbf{A}$. A função de pertinência resultante da operação complementar de um conjunto *fuzzy* pode ser vista a seguir, sendo que o operador complemento corresponde ao conectivo "NOT".

$$\mu_{\neg A}(x_i) = 1 - \mu_A(x_i) \tag{2.7}$$

A seguir apresenta-se um exemplo das operações básicas sobre duas funções de pertinência *fuzzy* exemplificadas em [7].

Sejam as funções de pertinência mostradas na Figura 10:

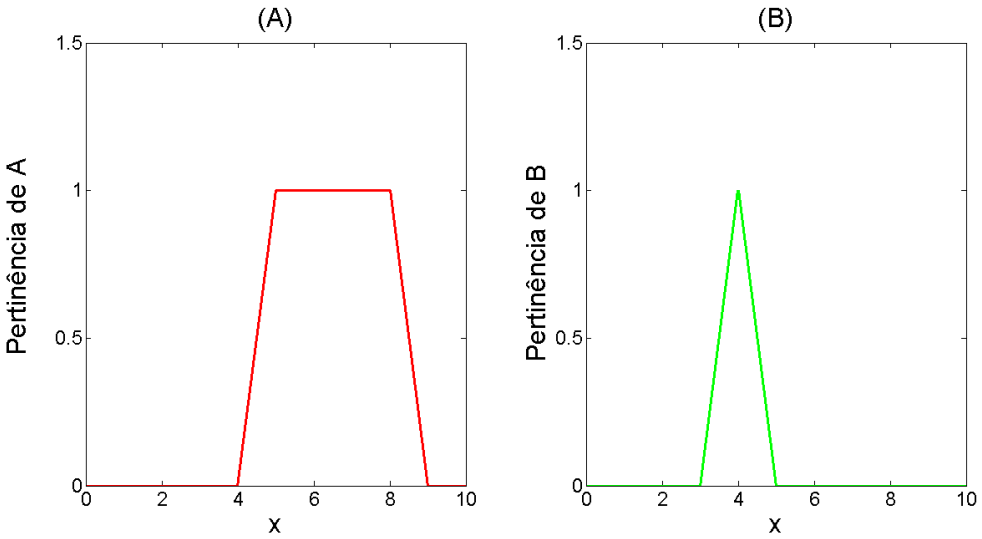

Figura 10 – Exemplos de funções de pertinência *fuzzy*.

Fonte: Baseado em [7]

Os resultados das três operações básicas estão nas Figuras 11, 12 e 13, a função de pertinência resultante é a representada em azul:

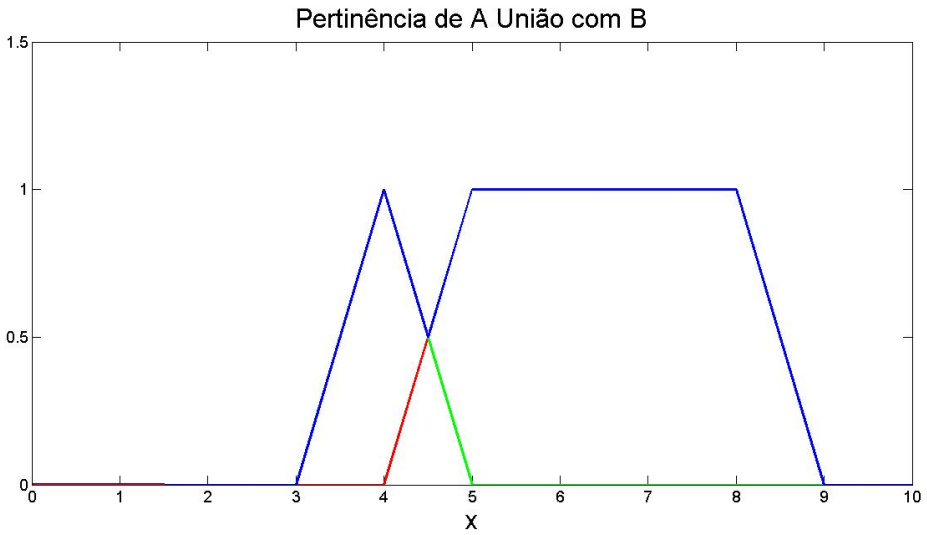

Figura 11 – Resultado da operação União.

Fonte: Baseado em [7]

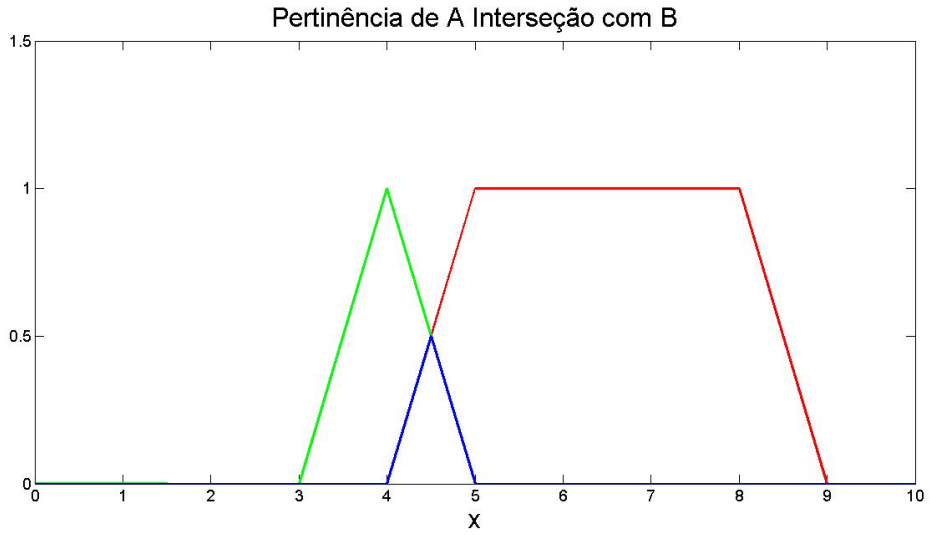

Figura 12 – Resultado da operação Interseção

Fonte: Baseado em [7]

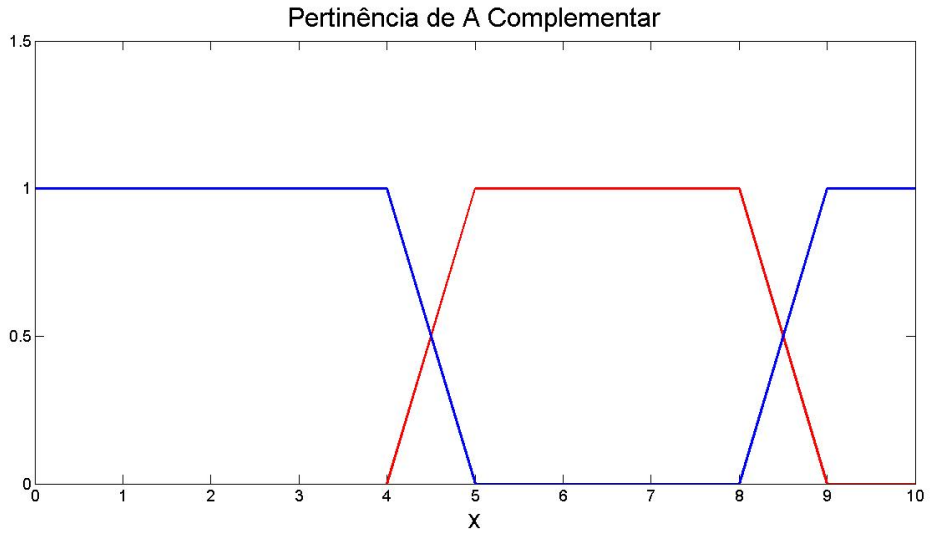

Figura 13 – Resultado da operação Complementar em relação ao conjunto *fuzzy* A.

Fonte: Baseado em [7]

2.2 SISTEMAS BASEADOS EM REGRAS FUZZY

Os Sistemas Baseados em Regras *Fuzzy* (SBR - *Fuzzy*) se subdividem em três etapas básicas: a etapa de fuzzyficação; a etapa do módulo de inferência, que é associado à base de regras; e a última etapa que é a defuzzyficação.

A visão geral de um sistema *fuzzy* pode ser apresentada conforme a Figura 14:

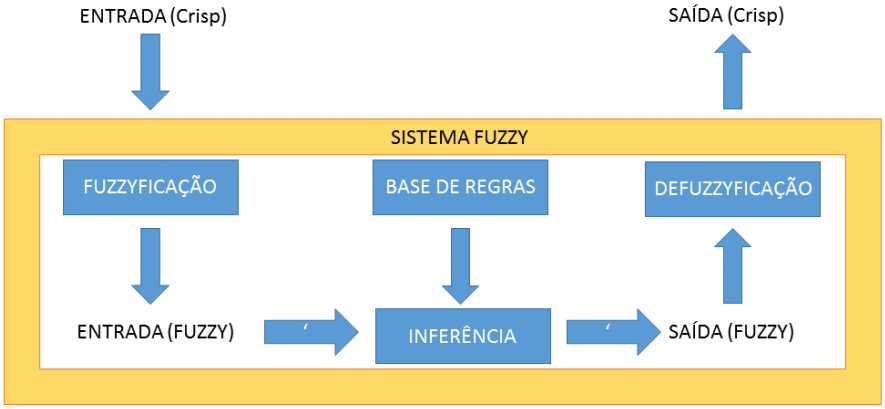

Figura 14 – Visão geral de um sistema *fuzzy*.

Fonte: Baseado em [15]

De acordo com [12] as informações *fuzzy* devem passar por um módulo de inferência juntamente com a utilização de uma base de regras, gerando uma saída *fuzzy*. A entrada e saída de um sistema *fuzzy* devem ser valores *crisp*. Segundo [13], na teoria clássica os conjuntos são denominados *crisp* e um dado elemento do universo em discurso (domínio) pertence ou não pertence ao referido conjunto. Enquanto na teoria dos conjuntos *fuzzy* existe um grau de pertinência de cada elemento a um determinado conjunto.

Ainda em [12], para entrar no sistema *fuzzy*, uma entrada *fuzzy* é necessária, assim, se a entrada for um escalar (*crisp*), deve ser passada por uma etapa de fuzzyficação, convertendo um escalar para uma grandeza *fuzzy*. Da mesma forma, a saída do módulo de inferência sendo uma saída *fuzzy*, deve ser passada por um processo chamado de defuzzyficação, que converte um número *fuzzy* em um número real (*crisp*). Dessa forma, o sistema *fuzzy* consegue para uma entrada *crisp* construir uma saída que também seja *crisp*.

2.2.1 **Fuzzyficação**

Em [12] define-se fuzzyficação como sendo a conversão dos dados de entrada. O valor *crisp* é convertido em um valor fuzzyficado chamado *singleton*, que ocupa todo o universo de discurso. Nessa etapa do sistema *fuzzy*, o objetivo é calcular o grau de pertinência de cada entrada do sistema, no universo de discurso pré estabelecido, em relação a cada conjunto *fuzzy*. Assim, o conhecimento do especialista no sistema a ser modelado é primordial nessa etapa, uma vez que para cada entrada *crisp*, de acordo com a base de conhecimento do sistema, associa-se um grau de pertinência adequado.

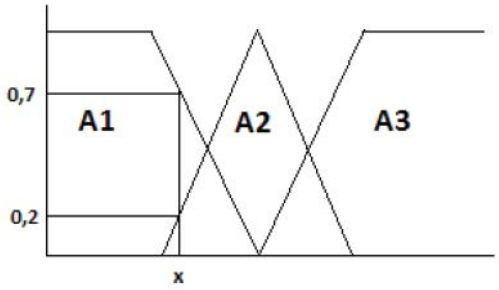

Figura 15 – Exemplo de fuzzyficação.

Fonte: Retirado de [16]

Na Figura 15 pode-se verificar a fuzzyficação para a entrada x. São calculados os graus de pertinência de x. De acordo com a base de conhecimento do especialista do sistema a ser modelado, pode-se associar a esta entrada "x"graus de pertinência de dois conjuntos *fuzzy* modelados, representados por A1 e A2, gerando os graus de pertinência de 0,7 e 0,2 respectivamente.

2.2.2 **Inferência e Base de Regras**

Em um sistema *fuzzy*, para o processamento das informações de entrada e geração das informações de saída *fuzzy*, antes do módulo defuzzyficador, é necessário a existência de uma base de regras sólida, capaz de tratar as informações obtidas pelo módulo fuzzyficador, e essas regras serão avaliadas e tratadas pelo chamado módulo de inferência. De acordo com [16], é importante observar que na fase da criação da base de regras, ou base de conhecimento, é primordial o conhecimento do operador ou especialista no sistema a ser modelado.

Segundo [16], quanto mais experiência naquele sistema o agente humano tiver, mais completa a base de regras, e mais eficaz será o sistema *fuzzy*, valorizando assim o conhecimento do operador. Por isso é primordial a criação de quantas regras forem necessárias para cobrir totalmente as combinações existentes entre as variáveis. Sendo assim, em [8], cita-se como objetivo da base de regras de um sistema *fuzzy* organizar o conhecimento acerca dos conjuntos definidos pelas funções de pertinência.

Em [12] define-se base de regras como um conjunto de regras estruturadas sob a forma: *SE <condicionais>, ENTÃO <consequente>*. Gerando, assim, uma descrição, mais completa possível, qualitativa sobre o sistema a ser modelado. [11].

A etapa do módulo de inferência de um sistema *fuzzy* é uma etapa de avaliação das regras definidas na base de conhecimento do sistema, e juntamente com as entradas do sistema possibilita a obtenção de conclusões significativas. Nesta etapa ocorre de fato a

tomada de decisões. Existem vários métodos de inferência, o mais utilizado é o *Método Mamdani*.

O método de inferência Mamdani foi criado pelo professor Ebrahim Mamdani da Universidade de Londres (Reino Unido) [18]. Esse método se tornou por muitos anos um padrão utilizado. Em [17] afirma-se que, como a maioria das aplicações de interesse possui sistemas convencionais de aquisição e atuação baseados em grandezas numéricas, o modelo de Mamdani inclui módulos de interface que transformam as variáveis de entrada em conjuntos *fuzzy* equivalentes e, posteriormente, as variáveis *fuzzy* geradas em variáveis numéricas proporcionais, adequadas para os sistemas de atuação existentes, conforme explicado e apresentado na Figura 13. Ainda em [17], afirma-se que o modelo de Mamdani é chamado de inferência Máx-Min pois utiliza as operações de união e de interseção entre conjuntos da mesma forma que [8], através dos operadores de máximo e de mínimo.

O antecedente em cada regra é um conjunto de condições sob a forma de variáveis *fuzzy* que quando são satisfeitas inteira ou parcialmente geram o consequente da regra em questão através do chamado método de inferência, gerando assim o chamado disparo da regra [16]. O consequente representa um conjunto de ações gerados em consequência do disparo da regra. Os consequentes das regras disparadas geram uma resposta para cada variável de saída do sistema.

De acordo com [17], para cada regra ativada, é gerado um coeficiente de disparo para a mesma. Assim, para a k-ésima regra da Base de Conhecimento um coeficiente de disparo D(k) é gerado conforme a Equação 2.8, onde os índices k nos conjuntos *fuzzy* denotam os termos primários que compõem a regra k na Base de Conhecimento. A expressão do coeficiente de disparo é apresentada na Equação 2.8, retirada de [17], onde μ representa o grau de pertinência.

$$D^{(k)} = T[\mu_{A_1^k}(x_1), \mu_{A_2^k}(x_2), ..., \mu_{A_p^k}(x_p)] = min[\mu_{A_1^k}(x_1), \mu_{A_2^k}(x_2), ..., \mu_{A_p^k}(x_p)] \tag{2.8}$$

Em [17], entende-se por uma regra disparada aquela cujo processamento do antecedente para as entradas atuais gerou graus de pertinência não nulos. A relação fuzzy entre as entradas e os termos primários do antecedente é maior que zero. Os coeficientes de disparo limitam os valores máximos dos conjuntos fuzzy de saída. E uma operação global de união vai compor um conjunto *fuzzy* para cada variável de saída, contendo as informações sobre todas as regras disparadas para as entradas atuais, segundo [17].

$$\mu_{B_i'}(y) = \max_{k=1..n}[min(D^{(k)}, \mu_{B_i}(y))], \forall y \in U_{y_2} \tag{2.9}$$

A união dos conjuntos *fuzzy* de saída limitados pelos coeficientes de disparo se denomina fase de agregação. Nessa fase todas as funções membro dos consequentes de

cada regra são agregadas em um único conjunto fuzzy.

Esquema do método de Mamdani. Regra Semântica: Máx - Min. [7]
1. Condicionais: Intersecção nebulosa entre os graus de pertinência das entradas atuais nos termos primários. Coeficiente de disparo $D^{(k)}$, para cada regra k.
$D^{(k)} = min[\mu_{A_1^k}(x_1), \mu_{A_2^k}(x_2), ..., \mu_{A_p^k}(x_p)]$

2. Todas as regras com $D[k] > 0$ disparam.

3. Consequentes: limitados pelo coeficiente de disparo nos seus valores máximos dos conjuntos de saída.

4. Operação global de *união* compõe um conjunto *fuzzy* para cada variável de saída (informações de todas as regras).
$\mu_{B_i'}(y) = \max_{k=1..n}[min(D^{(k)}, \mu_{B_i}(y))], \forall y \in U_{y_i}$

A seguir, a título de esclarecimento e ilustração utiliza-se o exemplo do método de Mamdani proposto em [7].

Assim, considerando as seguintes regras:

- Regra 1: Se x = A1 e y = B1, então z = C1;
- Regra 2: Se x = A2 e y = B2, então z = C2;

Assim, sendo os valores de entrada x_0 e y_0 obtém-se os valores μ_{A1} e μ_{B1} por meio da fuzzyficação. O mesmo princípio se aplica para determinar μ_{A2} e μ_{B2}.

Sabendo que tanto a regra 1 quanto a regra 2, o conector lógico usado é o "E" ou "AND", deve-se aplicar o mínimo entre as duas regras. Observa-se agora que o gráfico no termo C1 é ativado com o menor valor das pertinências μ_{A1} e μ_{B1}.

Assim, em termos de equações matemáticas, obtemos:

μC1 = $\mu_{A1} \cap \mu_{B1}$ = min(μ_{A1},μ_{B1}) = A1 $\cap$ B1

O mesmo acontece com o termo C2 e sua nova pertinência μ_{C2}:

$\mu_{C2} = \mu_{A2} \cap \mu_{B2}$ = min(μ_{A2},μ_{B2}) = A2 $\cap$ B2

O último passo do método de Mamdani é a união dos termos iguais, logo, se deve atribuir o maior valor de μ_{C1} e μ_{C2} para μ_C:

$\mu_C = \mu_{C1} \cup \mu_{C2}$ = max(A,B) = A $\cup$ B

A Figura 16 expressa o método gráfico bem como a saída C sendo a união dos gráficos C1 e C2.

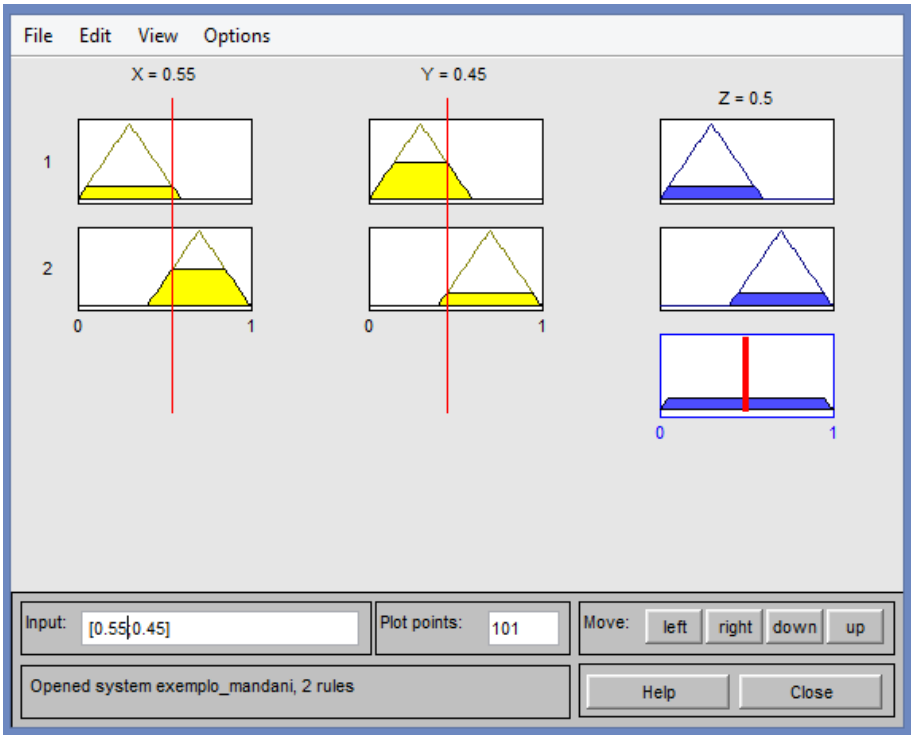

Figura 16 – Exemplo inferência de Mamdani.

Fonte: Baseado em [19], elaborado em [24]

No presente trabalho, irá se utilizar o método de inferência *Mandami*. Entretanto, outro método de inferência muito utilizado é o método de Takagi e Sugeno. [20].

De acordo com [16], os controladores *fuzzy* do tipo Mamdani, são de fácil modelagem por basear-se na intuição, são bons quando um controle grosseiro é aceitável. Os controladores que utilizam o método de Takagi e Sugeno apresentam desempenho superior, porém a modelagem tem o prejuízo de não ser mais intuitiva e sim matemática, pois o resultado da inferência de suas regras é dado por um valor numérico através de funções das variáveis linguísticas de entrada, isto é:

Se x é B, então y é C -> tipo Mamdani. [18]

Se x é B, então y é f(x) -> tipo Sugeno. [20]

2.2.3 **Defuzzyficação**

A última etapa do sistema *fuzzy* é a defuzzyficação, a qual converte uma saída *fuzzy* para um valor numérico (*crisp*), ou seja, transforma informações qualitativas em informações quantitativas. Existem diversos métodos de defuzzyficação, como: centro de massa, também conhecido como centroide (o método mais comum e utilizado), média dos máximos, e outros. [19]. Cada método fornece respostas diferentes, sendo necessário então, que se escolha um método mais adequado visando a aplicação necessária [16].

- *Método Centroide ou Centro de Massa*
  Para um determinado conjunto de saída *fuzzy* proveniente de uma base de conhecimento processada, o método do centro de massa calcula a abscissa (no universo de discurso definido para a variável em questão) do ponto de centro de massa correspondente e a utiliza como valor escalar de saída [20]. A expressão deste método é mostrada na Equação 2.10, retirado de [20] e ilustrada na Figura 17:

$$\hat{y_2} = \frac{\sum_{y \in U_{y2}} y . \mu_{B_i'}(y)}{\sum_{y \in U_{y2}} \mu_{B_i'}(y)} \tag{2.10}$$

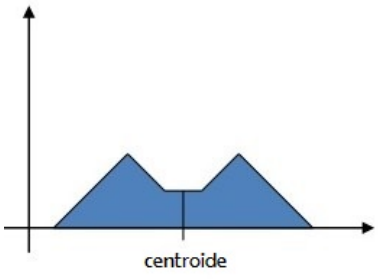

Figura 17 – Visão gráfica do método centroide.

Fonte: Adaptado de [14]

- *Método Média dos Máximos*
  No método da média dos máximos, o valor numérico da saída corresponde ao ponto do universo de discurso que corresponde à média dos pontos de máximo locais da função de pertinência do conjunto de saída *fuzzy*, produzida pelo processo de inferência [16]. A expressão deste método é mostrada na Equação 2.11, retirado de [20] e ilustrada na Figura 18:

$$y_2 = \frac{\sum_{y' \in U_{y2}} y'_k . \mu_{B_i'}(y'_k)}{n_{y'}}; \text{ onde } y' = \max_{y \in U', U' \subset U_{y2}} [\mu_{B_i'}(y)] \tag{2.11}$$

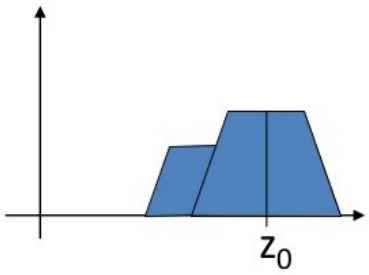

Figura 18 – Visão gráfica método média dos máximos.

Fonte: Retirado de [14]

2.3 APLICAÇÕES DA LÓGICA FUZZY

A lógica *fuzzy* tem um diverso campo de atuação e aplicações. Em [21], trata-se o campo da Inteligência Artificial o qual a lógica *fuzzy* coloca-se como o principal instrumento para uma representação mais adequada do conhecimento devido à sua capacidade de lidar com incertezas, raciocínio aproximado, termos vagos e ambíguos. Assim, a lógica *fuzzy* considera aspectos inerentes à incerteza e aos processos realísticos.

Ainda em [21], enumera-se alguns exemplos de aplicações da lógica *fuzzy* no contexto da Inteligência Artificial:

- Reconhecimento de padrões;
- Robótica;
- Sistemas de controle inteligentes;
- Sistemas de apoio à tomada de decisão;
- Algoritmos genéticos;

Ainda no contexto das aplicações da lógica *fuzzy*, [22] apresenta características de sistemas onde a aplicação da lógica *fuzzy* é benéfica:

- Sistemas complexos que são difíceis ou impossíveis de modelar;
- Sistemas controlados por especialistas;
- Sistemas que se utilizam da observação humana como entradas ou como base para regras;
- Sistemas que são naturalmente “vagos”, como os que envolvem ciências sociais e comportamentais, cuja descrição é extremamente complexa;

Em [16] cita-se algumas vantagens da lógica *fuzzy* como o mecanismo de raciocínio similar ao do ser humano, por meio do uso de termos linguísticos; modelagem de conhecimento de senso comum; conhecimento ambíguo e conhecimento impreciso, mas racional; técnica de aproximação universal; robustez e tolerância à falha; além do baixo custo de desenvolvimento e de manutenção.

Ainda em [16], cita-se algumas desvantagens ou limitações da lógica *fuzzy* como a geração das regras *fuzzy* e a definição das funções de pertinência; ambas, baseadas em uma avaliação subjetiva do conhecimento do especialista, além da inexistência de técnicas de aprendizado.

3 IMPLEMENTAÇÃO SISTEMA FUZZY DE APOIO À DECISÃO

Neste Capítulo será apresentada a implementação de um SBR *Fuzzy* no contexto do apoio à decisão em futebol de robôs autônomos. Assim, obtém-se a modelagem do contexto das condições durante o jogo em um VSSS, bem como a criação de uma base de regras e a modelagem de um sistema baseado em regras aplicado no contexto de apoio à decisão, utilizando a lógica *fuzzy*.

3.1 CONTEXTO DE FUTEBOL DE ROBÔS

No cenário do futebol de robôs autônomos VSSS, existe uma preocupação primordial na programação dos robôs que é a estratégia adotada pela equipe. Isso é, saber o que cada robô deve fazer no decorrer da partida, modelar seu comportamento desejado de acordo com o contexto geral da partida. Assim, é necessário um sistema que identifique em tempo real as condições de jogo. Para ilustrar o ambiente de futebol de robôs é apresentada a Figura 19.

A identificação pode ser feita de várias formas. O objetivo é, por exemplo, saber se o jogador está com chances claras de fazer gol, se está em uma posição crítica de risco na qual o time adversário poderá fazer um gol, se está em uma situação igualitária onde a posse de bola está sendo disputada entre as duas equipes envolvidas, entre outras. O objetivo é identificar ou modelar algum tipo de identificação do jogo em tempo real, para então saber qual é a atitude esperada para cada robô membro da equipe, atacando ou defendendo, por exemplo.

Esse sistema de regras irá gerar a decisão de determinadas funções a serem desempenhadas pelo robô específico. Cada uma dessas funções será uma tática de jogo diferente, modelando comportamentos esperados para um determinado robô, de acordo com as condições instantâneas de jogo as quais são modeladas e inseridas como entradas nesse SBR.

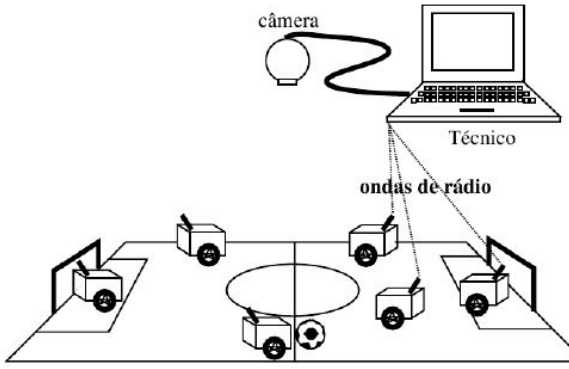

Figura 19 – Visão geral do futebol de robôs VSSS.

Fonte: Retirado de [23]

3.2 MODELAGEM DAS CONDIÇÕES DE JOGO

Para obter-se um sistema de apoio à decisão, nesse caso, baseado em regras *fuzzy*, é necessário modelar as condições reais de jogo sobre as quais o sistema irá decidir. Ou seja, é necessário modelar fatores que indicam quando um robô está em condições favoráveis ou não durante o jogo, para que possa definir um comportamento desejado fazendo o jogo fluir de acordo com o interesse.

Em [23] é proposto um modelo para mensurar as condições de jogo baseado em três fatores: o fator defesa (FD), fator competição (FC) e fator ângulo (FA). O presente trabalho apresenta uma estratégia baseada nestes fatores, porém com uma modelagem diferente da usada por [23]. A combinação dos valores de cada um desses fatores servirá de entrada para que o sistema gere a decisão de um comportamento esperado para um robô. Então, inicialmente, será apresentada a forma abordada neste trabalho para a modelagem dos fatores FD, FC e FA.

3.2.1 **Fator Defesa (FD)**

O Fator Defesa (FD) é um parâmetro que relaciona a posição do robô-jogador com relação às áreas de ataque e defesa. Assim como em [23] o fator defesa é expresso por uma exponencial, gerando valores confinados no intervalo dos números reais entre zero e um.

$$FD = e^{-c.\frac{d_{R,atk}}{d_{R,def}}} \tag{3.1}$$

Na equação de FD, pode-se definir, neste trabalho, como $d_{R,atk}$ sendo a distância euclidiana entre o robô jogador e o centroide de ataque do campo, ou seja, o ponto situado exatamente no centro da área do gol de ataque do time. De forma análoga, define-se $d_{R,def}$

como sendo a distância euclidiana entre o robô jogador e o centroide, nesse caso, de defesa do campo em questão.

A Figura 20 ilustra a delimitação das áreas de ataque e defesa no campo obtidos por meio da visão computacional.

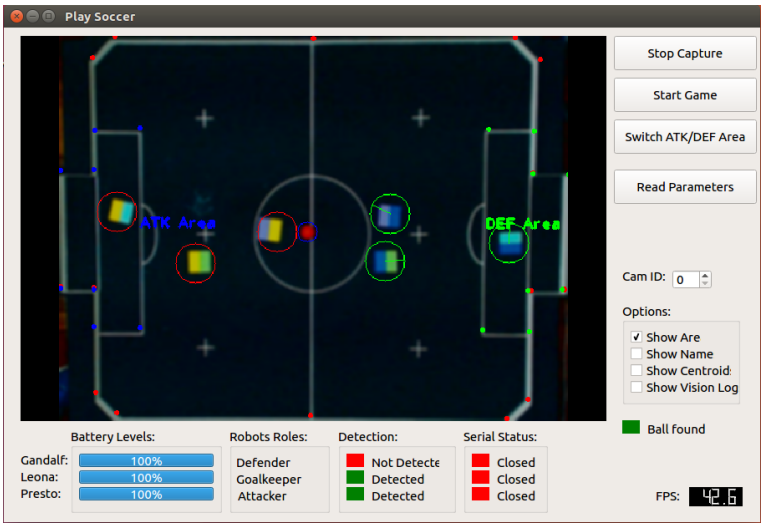

Figura 20 – Visão superior do campo indicando as demarcações das áreas de ataque e defesa.

Fonte: Feito no simulador da Equipe *Rinobot*

Sabendo que, por ser uma exponencial com expoente negativo, o valor de FD varia entre zero e um. Pode-se observar que quando o valor de FD se aproxima de um, menor é a razão $\frac{d_{R,atk}}{d_{R,def}}$, isso significa que a distância do robô até a área de defesa é maior que a distância até a área de ataque. Assim, quanto maior o FD, mais bem posicionado o robô está em relação à decisão de levar a bola para o gol adversário. Para análise de regras, quanto menor o valor de FD mais defensivo o robô deve ser, uma vez que esse fator considera apenas sua posição em relação aos gols de ataque e de defesa, o quanto avançado ou recuado o mesmo se encontra.

O valor c é uma constante de proporcionalidade calculada previamente. Pela análise anterior, sabe-se que o valor de FD varia conforme a posição do robô em seu campo de ataque ou defesa. Assim, para determinar essa constante empírica, foi utilizada a seguinte análise: se a distância entre o robô e o centroide de ataque for igual à distância entre o robô e o centroide de defesa, o robô deve estar em uma posição intermediária no campo. Assim, como FD varia entre zero e um, assume-se o valor $0,5$ como saída esperada.

Logo, temos:

$FD = e^{-c.\frac{d_{R,atk}}{d_{R,def}}}$ mas $\frac{d_{R,atk}}{d_{R,def}} = 1$

Então: $0,5 = e^{-c}$, logo $-c = \ln(0,5)$.

Assim, assumimos, neste trabalho, o valor de c:

$$c = -\ln(0,5) = 0,6931 \tag{3.2}$$

3.2.2 Fator Competição (FC)

O Fator Competição (FC) se baseia na noção do quanto o jogador ou robô alvo está em condições de competição, ou seja, o quanto é competitivo no instante de jogo sob as condições momentâneas da partida.

Para o FC, o objetivo é determinar o quão o jogador está bem posicionado em relação à bola, em detrimento do jogador adversário. Ou seja, se sua posição está favorecida ou desfavorecida em relação à proximidade com a bola e com o jogador adversário mais próximo. Pode-se notar que o FC também é um valor definido no intervalo entre zero e um, definido por uma exponencial conforme a Equação 3.3.

$$FC = e^{-k.\frac{d_{ball,R}}{d_{ball,enemy}}} \tag{3.3}$$

Na equação de FC, pode-se definir, neste trabalho, como $d_{ball,R}$ sendo a distância euclidiana entre o robô jogador e a posição da bola. De forma análoga, define-se $d_{ball,enemy}$ como sendo a distância euclidiana entre a bola e o robô do time adversário mais próximo. O jogador adversário considerado em questão é aquele em que a sua distância euclidiana em relação à bola seja menor. Assim, dos três robôs adversários, o robô que estiver mais próximo da posição da bola é o considerado no cálculo de FC na parcela $d_{ball,enemy}$.

Sabendo que, por ser uma exponencial com expoente negativo, o valor de FC varia entre zero e um. Pode-se observar que quando o valor de FC se aproxima de um, menor é a razão $\frac{d_{ball,R}}{d_{ball,enemy}}$, isso significa que a distância da bola até o robô adversário mais próximo é maior que a distância da bola até o robô jogador do time considerado. Assim, quanto maior o FC, mais bem posicionado o robô está em relação à bola, em detrimento do posicionamento do time adversário, também em relação à bola. Para análise de regras, quanto maior o valor de FC mais competitivo o robô considerado está, ou seja, mais favorável é a situação de jogo considerada se o FC for próximo da unidade, significando uma boa oportunidade uma vez que o jogador está mais próximo da bola em relação ao jogador adversário mais próximo da mesma. Assim, FC avalia a vantagem ou desvantagem em relação ao domínio de bola.

O valor k é uma constante de proporcionalidade calculada previamente. Pela análise anterior, sabe-se que o valor de FC varia conforme a posição do robô considerado, da bola e do robô adversário mais próximo. Assim, para determinar essa constante empírica, foi utilizado a seguinte análise: se a distância entre a bola e o robô do nosso time for igual à distância entre o robô adversário mais próximo da bola e a mesma, os dois robôs estão em

uma situação igualmente competitiva. Assim, como FC varia entre zero e um, assumimos o valor 0.5 como saída esperada.

Logo, temos:

$FC = e^{-k.\frac{d_{ball,R}}{d_{ball,enemy}}}$ mas $\frac{d_{ball,R}}{d_{ball,enemy}} = 1$

Então: $0,5 = e^{-k}$, logo $-k = \ln(0,5)$.

Assim, assumimos, neste trabalho, o valor de k:

$$k = -\ln(0,5) = 0,6931 \tag{3.4}$$

3.2.3 **Fator Ângulo (FA)**

O Fator Ângulo (FA) tem o objetivo de mensurar o quanto o robô jogador está bem posicionado angularmente em relação à bola. Ou seja, o FA avalia a facilidade do domínio de bola em termos angulares.

De uma forma geral o FA complementa o FC comentado na Seção 3.2.2, uma vez que se o robô jogador tiver o FC alto, significa que está com vantagem em relação ao jogador oponente e está próximo da bola. O FA por sua vez, avalia o quanto essa proximidade com a bola é favorável, uma vez que mensura se a faixa angular existente entre o robô e a bola é favorável a uma situação de ataque.

O cálculo de FA é feito da seguinte forma:

- A partir das posições (x,y) da bola e do robô jogador, calcula-se o vetor bola-robô como sendo a diferença das posições, ou seja $(x_{robô} - x_{bola}, y_{robô} - y_{bola})$.
- Calcula-se o ângulo entre o vetor bola-robô e o eixo x de acordo com o cálculo de ângulo entre dois vetores apresentado na Equação 3.5, uma vez que trabalha-se apenas em valores cartesianos dispostos no primeiro quadrante:

  Sejam dois vetores genéricos (x_1, y_1) e (x_2, y_2) o ângulo entre os mesmos é dado pela Equação 3.5:

$$\theta = cos^{-1}\left(\frac{x_1.x_2 + y_1.y_2}{\sqrt{x_1^2 + y_1^2}.\sqrt{x_2^2 + y_2^2}}\right) \tag{3.5}$$

- Finalmente, o ângulo entre o robô jogador e a bola é dado pela diferença entre o ângulo formado pelo vetor bola-robô e o eixo x e o ângulo de orientação do próprio robô (que é o ângulo formado pelo robô e o eixo x) gerando assim o ângulo que compõe o FA que é o ângulo entre o robô jogador e a bola, conforme a Figura 21.

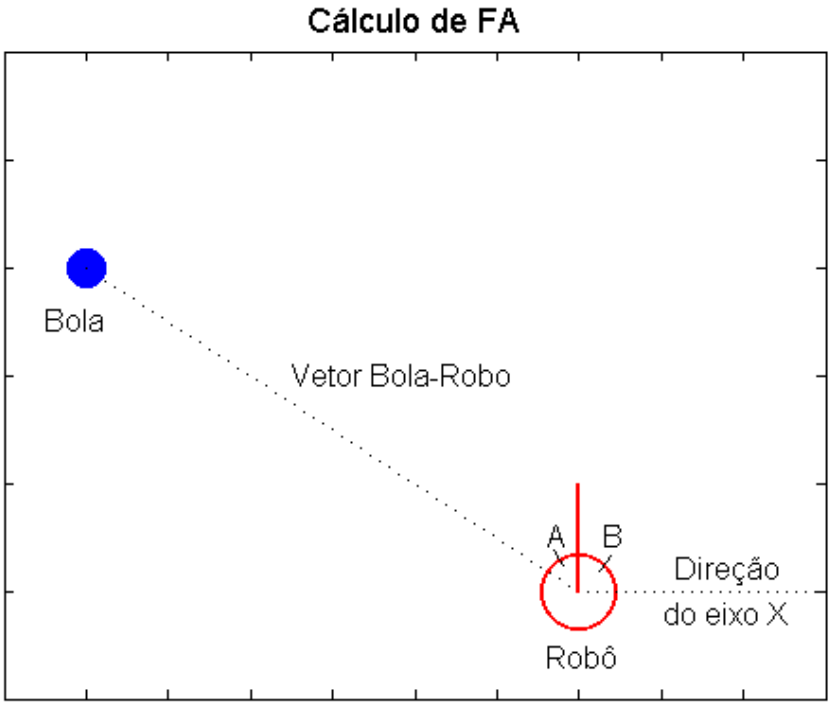

Figura 21 – Ilustração exemplo para cálculo de FA. Considere o ângulo A como sendo o ângulo entre o robô e a bola e o ângulo B como sendo o ângulo do robô.

Estando a bola entre o robô e o gol de defesa, formando um ângulo nulo entre o robô e a bola, o FA seria elevado por indicar uma situação favorável em termos angulares, o que não é desejável uma vez que o robô está virado ao contrário do sentido do gol de ataque, sendo essa uma situação perigosa que deve ser evitada. Assim, a análise considera também o ângulo entre o robô jogador e o centroide de ataque, sendo possível então perceber o quão melhor o robô está posicionado em relação angular com a bola e com o gol de ataque e também de defesa.

O FA desenvolvido é formado por uma parcela que é representada pelo ângulo entre o robô jogador e a bola, mas também uma parcela que é formada pelo ângulo entre o robô jogador e o centroide do gol de ataque. Dessa forma, pode-se medir também o quão bem orientado o robô está e relação à bola, mas também em relação ao gol de ataque. Essa análise é feita para evitar situações indesejáveis nas quais o robô está posicionado de frente para o gol de defesa e o ângulo formado entre o mesmo e a bola é zero, ou seja, a bola está exatamente à frente do robô.

O cálculo do ângulo entre o robô e o centroide de ataque é inteiramente análogo ao cálculo já descrito para o ângulo entre o robô e a bola, uma vez que no lugar do posicionamento da bola, insere-se o ponto fixo que é o centroide da área de ataque.

Tendo os dois ângulos principais calculados (ângulo entre o robô e a bola, e o ângulo

entre o robô e o centroide de ataque), pode-se assim definir o Fator Ângulo (FA). Dessa forma, esses ângulos são calculados para os intervalos em graus de $[-180, 180]$. Inspirado em [3], tais ângulos devem ser ajustados conforme mostrado nas Equações 3.6, 3.7 e 3.8 de forma generalizada para um ângulo α qualquer, garantindo a pertinência dos valores angulares no intervalo em graus mencionado.

$$\alpha > 180 \iff \alpha = \alpha - 360 \tag{3.6}$$

$$\alpha < -180 \iff \alpha = \alpha + 360 \tag{3.7}$$

$$-180 \leq \alpha \leq 180 \iff \alpha = \alpha \tag{3.8}$$

Dados os cálculos e ajustes anteriores, pode-se definir FA conforme mostrado na Equação 3.13.

Considerando β como sendo o ângulo robô-bola, tem-se:

Se $-90 \leq \beta \leq 90$ então,

$$k_1 = \frac{(90 - |\beta|)}{90} \tag{3.9}$$

Se $\beta > 90$ ou $\beta < (-90)$ então,

$$k_1 = \frac{(-90 + |\beta|)}{90} \tag{3.10}$$

Considerando γ como sendo o ângulo robô-centroide de ataque, tem-se:

Se $-90 \leq \gamma \leq 90$ então,

$$k_2 = \frac{(90 - |\gamma|)}{90} \tag{3.11}$$

Se $\gamma > 90$ ou $\gamma < (-90)$ então,

$$k_2 = \frac{(-90 + |\gamma|)}{90} \tag{3.12}$$

Assim, pode-se definir:

$$FA = 0,7.k_1 + 0,3.k_2 \tag{3.13}$$

Os ajustes apresentados são feitos de forma a garantir que o valor de FA esteja confinado em um intervalo de zero até um. De forma que quando k_1 é nulo, a posição da bola é perpendicular à direção de movimentação do robô, da mesma forma que ocorre quando k_2 é zero, sendo o centroide de ataque perpendicular em relação à direção de movimento do agente robótico. E quando k_1 vale um, a bola está logo a frente ou atrás do agente, uma vez que o robô consegue se movimentar em ambos sentidos. O mesmo acontece com k_2 em relação ao ponto do centroide de ataque.

O fator FA é a combinação das duas parcelas k_1 e k_2 em percentuais de 70% e 30% de forma que FA representa um fator que varia de zero a um. Quando é nulo ou próximo de zero significa que de uma forma geral, o robô não está alinhado com a bola e nem com o centroide de ataque. A análise dos percentuais prevê que não adianta estar alinhado apenas com a bola, uma vez que 30% de FA é o alinhamento com o centroide de ataque. Assim, FA próximo de um significa uma situação de posicionamento angular muito favorável, estando alinhado com a bola e com o centroide de ataque.

Assim a variação de FA define um parâmetro satisfatório para a determinação do posicionamento angular do robô em relação à bola e ao gol adversário durante toda a partida, uma vez que esse cálculo se repete a cada instante ou a cada momento de captura da câmera.

3.3 FUNÇÕES DE PERTINÊNCIA

A entrada do sistema *fuzzy* de apoio à decisão é a combinação dos três fatores apresentados anteriormente (FD, FC e FA). Os três valores estão confinados em um intervalo entre zero e um. Assim, para a análise *fuzzy* dos três parâmetros combinados é necessário criar as funções de pertinência dos grupos relativos à cada um desses fatores, e também uma base de regras sólida e compacta que será explicada na Seção 3.4.

Como os valores de FD, FC e FA estão entre zero e um, para a análise de regras, escolheu-se a função de pertinência no formato triangular. Assim, foi dividido o intervalo entre zero e um em quatro faixas de igual espaçamento para classificação de cada intervalo obtido. Assim sendo, utilizando [24], pode-se ilustrar a situação obtida para cada um dos fatores de análise.

Nas Figuras 22, 23 e 24, elaboradas no simulador [24], pode-se verificar a lógica feita. As funções de pertinência adotadas foram da forma triangular mostrada na Seção 2.1.1. Assim, considerou-se quatro intervalos de pertinência para os três fatores. Sendo considerado o grupo *fuzzy BAIXO* para os valores de intervalo $[-0,3; 0; 0,3]$, sendo esses, de acordo com a Seção 2.1.1, os respectivos valores $[a; u; b]$. O grupo intermediário $MÉDIO_1$ foi associado aos valores $[0,111; 0,333; 0,555]$. O próximo grupo *fuzzy* intermediário $MÉDIO_2$ foi associado aos valores $[0,444; 0,666; 0,888]$. E, por fim, o grupo $ALTO$ foi associado aos valores $[0,7; 1; 1,3]$.

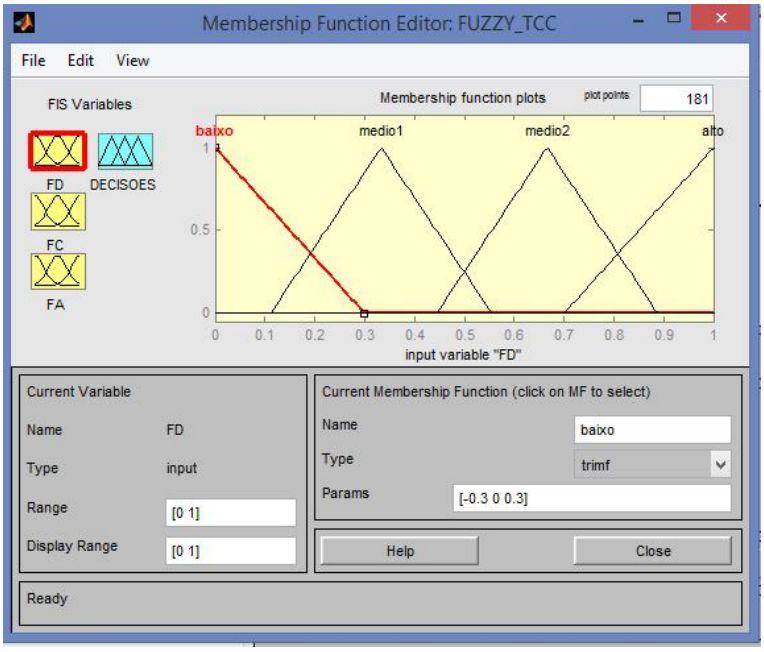

Figura 22 – Ilustração das funções de pertinência adotadas no parâmetro FD.

Fonte: Feito no Simulador em [24]

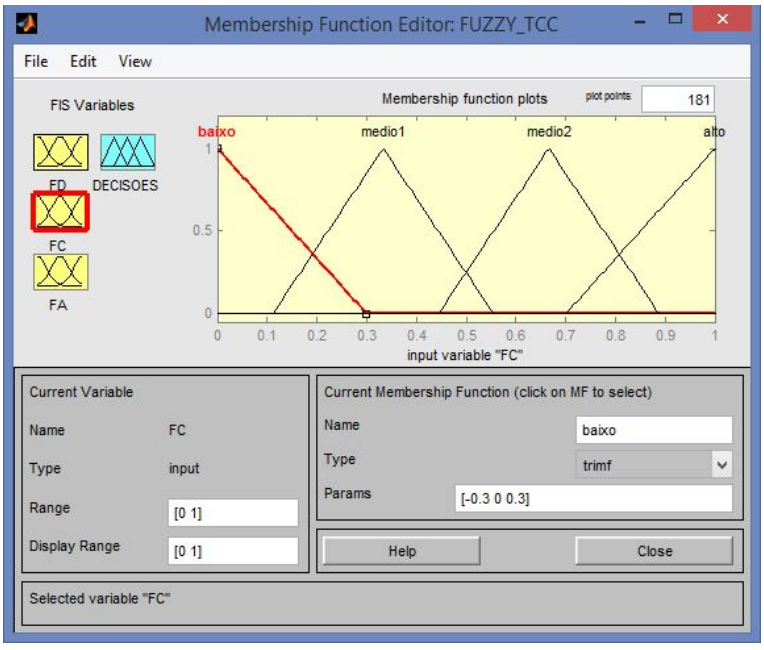

Figura 23 – Ilustração das funções de pertinência adotadas no parâmetro FC.

Fonte: Feito no Simulador em [24]

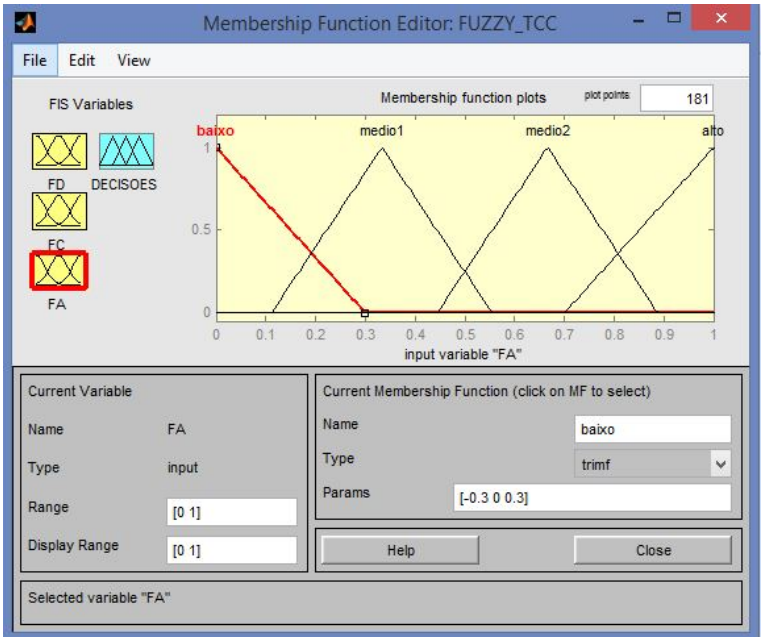

Figura 24 – Ilustração das funções de pertinência adotadas no parâmetro FA.

Fonte: Feito no Simulador em [24]

O simulador [24] foi utilizado a título de complementação e verificação dos resultados obtidos em testes, sendo interessante para obter uma visão gráfica da análise *fuzzy* no processo de decisão. Utilizado anteriormente para ilustração gráfica dos parâmetros adotados na curva de pertinência.

A divisão dos intervalos das variáveis de entrada em quatro subpartes de igual abrangência se fez de forma que pudesse facilitar a análise das combinações obtidas entre os valores dos três fatores. A construção da base de regras *fuzzy* se dá pelas combinações entre as características dispostas em cada fator de análise. Utilizando quatro grupos pode-se obter diversas situações encontradas em jogo, sendo adotadas como satisfatórias para cobrir o leque de possibilidades ocorridas durante uma partida.

Para as saídas do sistema *fuzzy* de decisão foram adotadas quatro funções de pertinência, da mesma forma em que foram construídas as de entrada, ou seja, na forma triangular mostrada na Seção 2.1.1. O grupo *fuzzy DEFESA PESADA* foi adotado para os valores de intervalo $[-0,3; 0; 0,3]$, sendo esses, de acordo com a Seção 2.1.1, os respectivos valores $[a; u; b]$. O grupo intermediário *DEFESA LEVE* foi associado aos valores $[0,111; 0,333; 0,555]$. O próximo grupo intermediário *ATAQUE LEVE* foi associado aos valores $[0,444; 0,666; 0,888]$. E, por fim, o *ATAQUE PESADO* foi associado aos valores $[0,7; 1; 1,3]$.

Os grupos apresentados representam os domínios sob os quais o sistema *fuzzy* irá gerar sua decisão, sendo cada um desses grupos representações de um comportamento

esperado para o agente robótico de acordo com a demanda e condições de jogo. Os detalhes sobre os comportamentos esperados e as funções de jogo para a saída do sistema serão apresentados no Capítulo 5.

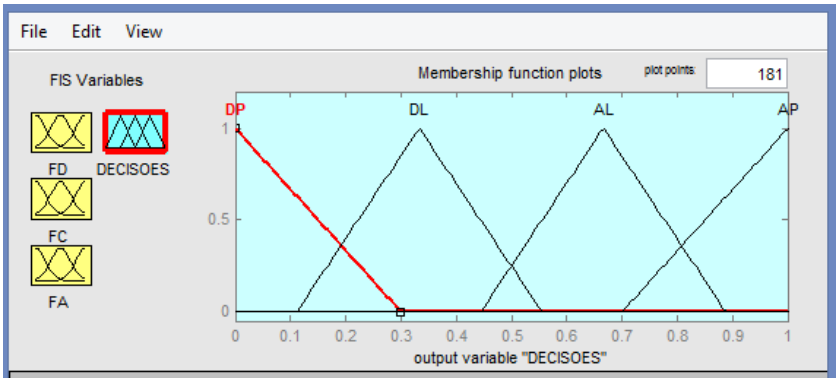

Figura 25 – Ilustração das funções de pertinência adotadas na saída do sistema *fuzzy*.

Fonte: Feito no Simulador em [24]

A saída resultante da defuzzyficação e a tomada de decisão é feita de acordo com o método do *centroide* ou *centro de massa*, explicado na Seção 2.2.3. Através desse método, obtém-se após o processo de defuzzyficação uma saída numérica do sistema confinada nos valores entre zero e um. A análise desse valor de saída é feita avaliando os graus de pertinência do valor em questão em relação aos grupos de saída (*defesa pesada, defesa leve, ataque leve e ataque pesado*). A função de saída adotada é aquela indicada pelo maior grau de pertinência em que o valor de saída possui. Assim, tendo um valor numérico de saída, a função de saída será aquela em que o grau de pertinência desse determinado valor for o maior, ou seja, o mais considerável.

3.4 BASE DE REGRAS

Um processo de criação de um sistema *fuzzy* de apoio à decisão, principalmente no contexto do futebol de robôs autônomos VSSS, se faz mediante a criação de uma base de regras sólida, completa e satisfatória.

A motivação desta etapa do sistema, portanto, é a criação das mais diversas situações e condições de jogo prováveis em uma partida do VSSS. A análise dessas situações juntamente com os fatores de entrada (FD, FC, FA) e identificação de possíveis regras auxiliam na tomada de decisões gerando os comportamentos esperados para as várias condições e situações diversas.

Para a criação de uma base de regras, foram feitos e criados aproximadamente setenta situações individuais de jogo. Situações nas quais é possível inferir o melhor comportamento esperado do agente robótico entre os quatro criados (ataque ou defesa,

leve ou pesados). Assim, criou-se situações típicas como saída de bola com posse em ambos os lados, falta na área de defesa em ambos os lados, falta na área de ataque em ambos os lados, tiro de meta, saída de bola do adversário, entre outras. Também foram criadas situações não clássicas mas que ocorrem frequentemente como as situações de bola presa nos cantos do campo, bem como nas áreas de defesa ou ataque.

Nas fotografias anexas no Apêndice A deste documento ilustra-se diversas situações computadas para a base de dados do sistema. É importante observar que as situações foram feitas utilizando apenas a parte superior do robô, a qual é dividida em cores para a captação da câmera, assim, o sistema pôde verificar a posição e calcular os fatores FD, FC e FA. A parte superior do robô utilizado para esses testes foi a de cores roxo e azul, o qual está circulado nas imagens-exemplo. A orientação que o sistema adota está ilustrado na Figura 26, onde a linha reta no interior da circunferência demarcada representa a frente do robô (ângulo de zero grau).

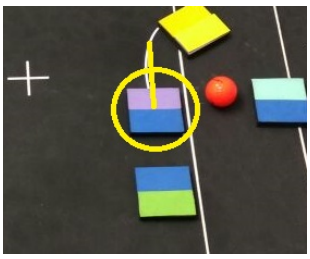

Figura 26 – Orientação capturada pelo sistema à partir da visão superior do robô.

As fotos das diversas posições e condições de jogo testadas estão disponíveis no *Apêndice A* deste documento. A Figura 27 representa os dados tabelados dos testes efetuados. De acordo com a situação de jogo apresentada pelas fotos, o sistema calcula os fatores FD, FC e FA automaticamente, na primeira parte da tabela a ser apresentada na Figura 27.

De acordo com a condição de jogo apresentada, e com os valores obtidos para os fatores em cada situação, analisou-se a reação esperada pelo sistema. Na coluna "*Regra Proposta*", infere-se o comportamento esperado do sistema a ser criado e também a maior proximidade dos valores de cada fator com os grupos de pertinência criados. Assim, se o valor de FD, por exemplo, for mais próximo do valor central ("u", definido na Seção 3.3), classifica-se esse valor como B (*Baixo*), M1 ($Médio_1$), M2 ($Médio_2$) ou A (*Alto*).

A proposta é utilizar os comportamentos armazenados na base de dados e criar as regras inerentes ao sistema *fuzzy*, já que se dispõe de diversas classificações e valores exemplo para FD, FC e FA e assim concluir regras sobre o sistema modelado, supondo que o conjunto de situações avaliadas (expostas nas imagens do *Apêndice A*) sejam satisfatórias para a criação de um sistema de apoio à decisão apropriado às partidas no VSSS.

Situações	FD	FC	FA
Foto 1	0,45	0,86	0,99
Foto 2	0,16	0,44	0,96
Foto 3	0,09	0,05	0,8
Foto 4	0,27	0,08	0,95
Foto 5	0,03	0,4	0,78
Foto 6	0,09	0,51	0,94
Foto 7	0,02	0,41	0,66
Foto 8	0,11	0,52	0,88
Foto 9	0,58	0,44	0,91
Foto 10	0,45	0,18	0,83
Foto 11	0,57	0,48	0,88
Foto 12	0,48	0,21	0,8
Foto 13	0,09	0,84	0,57
Foto 14	0,02	0,7	0,44
Foto 15	0,5	0	0,45
Foto 16	0,51	0,02	0,86
Foto 17	0,72	0,77	0,95
Foto 18	0,46	0,39	0,96
Foto 19	0,55	0,02	0,74
Foto 20	0,53	0,03	0,66
Foto 21	0,54	0,03	0,7
Foto 22	0,14	0,61	0,69
Foto 23	0,23	0,06	0,74
Foto 24	0,17	0,02	0,46
Foto 25	0,45	0,57	0,65
Foto 26	0,75	0,05	0,03
Foto 27	0,52	0,01	0,81
Foto 28	0,74	0,53	0,68
Foto 29	0,53	0,4	0,75
Foto 30	0,73	0,26	0,34
Foto 31	0,74	0,62	0,81
Foto 32	0,67	0,81	1
Foto 33	0,04	0,42	0,83
Foto 34	0,46	0,28	0,68
Foto 35	0,1	0,25	0,34
Foto 36	0,02	0,28	0,15
Foto 37	0,42	0	0,72
Foto 38	0,57	0,05	0,72
Foto 39	0,87	0,16	0,77
Foto 40	0,83	0,65	0,97
Foto 41	0,19	0,12	0,74
Foto 42	0,49	0,63	0,95
Foto 43	0,76	0,38	0,69
Foto 44	0,13	0,61	0,7
Foto 45	0,14	0,4	0,83
Foto 46	0,06	0,16	0,28
Foto 47	0,67	0,86	0,1
Foto 48	0,46	0,47	0,87
Foto 49	0,68	0,82	0,42
Foto 50	0,34	0,59	0,6
Foto 51	0,49	0,17	0,75
Foto 52	0,74	0,03	0,98
Foto 53	0,86	0,21	0,93
Foto 54	0,58	0,46	0,78
Foto 55	0,73	0,67	0,64
Foto 56	0,42	0,88	0,99
Foto 57	0,24	0,23	0,61
Foto 58	0,26	0,51	0,26
Foto 59	0,26	0,51	0,26
Foto 60	0,31	0,32	0,84
Foto 61	0,73	0,26	0,67
Foto 62	0,74	0,11	0,77
Foto 63	0,44	0	0,75
Foto 64	0,02	0,88	0,09
Foto 65	0,02	0,6	0,13
Foto 66	0,02	0,56	0,16
Foto 67	0,61	0,43	0,28
Foto 68	0,44	0,25	0,16
Foto 69	0,38	0,46	0,42
Foto 70	0,72	0,51	0,58
Foto 71	0,27	0,64	0,74
Foto 72	0,08	0,93	0,95
Foto 73	0,68	0,39	0,73

Regra Proposta			
Reação Esperada	FD	FC	FA
Ataque Leve	M1	A	A
Defesa Leve	B	M1	A
Defesa Leve	B	B	M2
Defesa Leve	M1	B	A
Defesa Leve	B	M1	M2
Ataque Leve	B	M2	A
Defesa Leve	B	M1	M2
Ataque Leve	B	M2	A
Ataque Pesado	M2	M1	A
Defesa Pesada	M1	M1	M2
Ataque Pesado	M2	M1	A
Defesa Pesada	M1	M1	M2
Ataque Leve	B	A	M2
Defesa Leve	B	M2	M1
Defesa Leve	M2	B	M1
Ataque Leve	M2	B	A
Ataque Pesado	M2	M2	A
Ataque Leve	M1	M1	A
Defesa Leve	M2	B	M2
Defesa Leve	M2	B	M2
Defesa Leve	M2	B	M2
Defesa Leve	B	M2	M2
Defesa Leve	M1	B	M2
Defesa Leve	M1	B	M1
Ataque Leve	M1	M2	M2
Defesa Leve	M2	B	B
Defesa Leve	M2	B	M2
Ataque Leve	M2	M2	M2
Ataque Leve	M2	M1	M2
Defesa Leve	M2	M1	M1
Ataque Leve	M2	M2	M2
Ataque Pesado	M2	M2	A
Defesa Leve	B	M1	M2
Defesa Leve	M1	M1	M2
Defesa Pesada	B	M1	M1
Defesa Pesada	B	M1	B
Defesa Leve	M1	B	M2
Defesa Leve	M2	B	M2
Ataque Leve	A	B	M2
Ataque Pesado	M2	M2	A
Defesa Leve	M1	B	M2
Ataque Leve	M1	M2	A
Ataque Leve	M2	M1	M2
Defesa Leve	B	M2	M2
Defesa Leve	B	M1	M2
Defesa Leve	B	B	M2
Ataque Leve	M2	A	B
Ataque Leve	M1	M1	A
Ataque Leve	M2	M2	M1
Ataque Leve	M1	M2	M2
Defesa Pesada	M1	M1	M2
Ataque Leve	M2	B	A
Ataque Pesado	A	M1	A
Ataque Leve	M2	M1	M2
Ataque Leve	M2	M2	M2
Ataque Leve	M1	A	A
Defesa Pesada	M1	M1	M2
Defesa Leve	M1	M2	M1
Defesa Leve	M1	M2	M1
Ataque Leve	M1	M1	A
Ataque Leve	M2	M1	M2
Ataque Leve	M2	B	M2
Defesa Leve	M1	B	M2
Defesa Leve	B	A	B
Defesa Leve	B	M2	B
Defesa Leve	B	M2	B
Defesa Leve	M2	M1	M1
Defesa Leve	M1	M1	B
Ataque Leve	M1	M1	M1
Ataque Leve	M2	M2	M2
Ataque Leve	M1	M2	M2
Ataque Leve	B	A	A
Ataque Leve	M2	M1	M2

Figura 27 – Proposição de possíveis comportamentos do sistema modelado a partir de situações comuns ou típicas de jogo.

Com a combinação das diversas situações de jogo, pode-se inferir um acervo de comportamentos esperados do agente robótico. Assim, gerou-se a combinação dos grupos de pertinência (Baixo, Médio$_1$, Médio$_2$ e Alto) para cada um dos três fatores de análise (FD, FC e FA). Ao todo, elaborou-se sessenta e quatro regras para o sistema *fuzzy* afim de cobrir todas ou satisfatoriamente a maioria das situações típicas e atípicas de jogo. A base de regras elaborada para o sistema modelado está apresentada na Figura 28. Assim, para cada uma das sessenta e quatro situações ou combinações dos fatores possíveis, uma decisão ideal é associada, tais como DP, DL, AL ou AP (Defesa Pesada, Defesa Leve, Ataque Leve ou Ataque Pesado).

As possíveis decisões para cada situação requerem um sistema de navegação apropriado ou adaptado para a execução de comportamentos esperados pelo robô em análise por parte do sistema *fuzzy*, essas funções de jogo serão explicadas e detalhadas posteriormente.

BASE DE REGRAS - FUZZY -

FD - BAIXO

FC \FA	BAIXO	MEDIO 1	MEDIO 2	ALTO
BAIXO	DP	DP	DL	DL
MEDIO1	DP	DP	DL	DL
MEDIO2	DL	DL	DL	AL
ALTO	DL	DL	AL	AL

FD - MEDIO 1

FC \FA	BAIXO	MEDIO 1	MEDIO 2	ALTO
BAIXO	DP	DL	DL	DL
MEDIO1	DL	AL	DP	AL
MEDIO2	DL	DL	AL	AL
ALTO	DL	DL	AL	AL

FD - MEDIO 2

FC \FA	BAIXO	MEDIO 1	MEDIO 2	ALTO
BAIXO	DL	DL	DL	AL
MEDIO1	DL	DL	AL	AP
MEDIO2	AL	AL	AL	AP
ALTO	AL	AL	AP	AP

FD - ALTO

FC \FA	BAIXO	MEDIO 1	MEDIO 2	ALTO
BAIXO	DL	DL	AL	AL
MEDIO1	DL	DL	AL	AP
MEDIO2	AL	AL	AP	AP
ALTO	AL	AL	AP	AP

ACOES:

AP	Ataque Pesado	39,43,46,47,55,58,59,62,63.
AL	Ataque Leve	11,14,15,21,23,26,27,30,31,35,38,40,41,42,44,45,50,51,54,56,57,60,61
DL	Defesa Leve	2,3,6,7,8,9,10,12,13,17,18,19,20,24,25,28,29,32,33,34,36,37,48,49,52,53.
DP	Defesa Pesada	0,1,4,5,16,22

Figura 28 – Base de Regras proposta para o sistema *fuzzy* .

4 SISTEMAS DE NAVEGAÇÃO

Neste Capítulo será apresentada uma revisão bibliográfica básica de forma a fazer uma abordagem teórica a respeito dos sistemas de navegação no contexto dos Campos Potenciais (CP), envolvendo os Campos Potenciais Harmônicos (CPH) e Campos Potenciais Orientados (CPO), os quais são importantes para garantir uma navegação robusta e aceitável dos agentes robóticos no contexto do jogo de futebol VSSS.

4.1 VISÃO GERAL

Baseado em [3], pode-se dizer que o tradicional método denominado Campos Potenciais (CP) é o mais utilizado e largamente aplicável em robótica tanto móvel quanto de manipuladores por ser um algoritmo que planeja trajetórias desviando de obstáculos e, portanto, evitando colisões. Esse método foi proposto por [25] utilizado para os agentes robóticos móveis. O método CP se baseia na ação de forças repulsivas e atrativas, portanto em cargas elétricas existentes sob a ação de um campo potencial. Assim, a meta e o agente considerado dotados de cargas de sinais contrários se atraem. Enquanto os obstáculos são dotados de cargas elétricas de mesmo sinal em relação ao agente robótico, repelindo-se.

No contexto da criação de um sistema de navegação que evita colisões dos agentes robóticos com obstáculos, e, ao mesmo tempo, orientando a trajetória e controlando o movimento, ou seja, gerando as velocidades para o robô, o conceito de Campos Potenciais é largamente aplicado. A aplicação dessa técnica, se feita de forma a gerar nova força resultante ao agente robótico, ou seja, novas velocidades, a cada passo dado do robô, facilita a aplicabilidade em ambientes desconhecidos onde deve-se desviar de obstáculos com maior robustez.

A técnica dos Campos Potenciais proposta por [25] apresenta algumas limitações desvantajosas para a aplicação, dentre as desvantagens apresentadas em [3], pode-se dizer que a ocorrência de *mínimos locais* durante a trajetória do agente móvel é prejudicial uma vez que o método não consegue convergir para meta, ficando o robô preso em uma condição em que as forças de atração e repulsão se anulam.

Com base na análise feita, este trabalho utiliza a abordagem de um sistema de navegação baseado nos Campos Potenciais de [25], porém com melhorias e superação das limitações inerentes ao CP tradicional. Assim, as funções de jogo utilizadas no sistema proposto por esse trabalho utilizam basicamente duas técnicas de navegação que são melhorias do CP tradicional. Será analisado o método dos Campos Potenciais Harmônicos (CPH) e o método dos Campos Potenciais Orientados (CPO). Neste capítulo será introduzida noções básicas teóricas desses métodos, para posteriormente no capítulo seguinte apresentar a aplicação dos mesmos no sistema proposto para o VSSS.

4.2 CAMPOS POTENCIAIS HARMONICOS (CPH)

Sabendo das limitações e desvantagens do método CP de navegação, surge a utilização de funções harmônicas para a solução do problema dos mínimos locais. Essa solução denomina-se Campos Potenciais Harmônicos. De acordo com [26], tal modelagem foi inicialmente proposta por [27] e utiliza-se de funções harmônicas para calcular o campo potencial de ambientes conhecidos.

Define-se como *Função Harmônica* qualquer solução não trivial e cuja as derivadas de primeira e segunda ordem sejam contínuas para a Equação de Laplace, apresentada na Equação 4.1 [28].

$$\sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2} = 0 \tag{4.1}$$

A Equação de Laplace, segundo [29] tem sua aplicabilidade por modelar o potencial gravitacional ou eletrostático em pontos do espaço vazio. As soluções para tal não são simples, sendo usual a utilização de métodos numéricos para encontrá-las, não sendo esse o escopo do presente trabalho.

As Funções Harmônicas, segundo [26], possuem propriedades muito úteis ao desenvolvimento de sistemas robóticos:

- **Integridade:** o método representa um sistema de planejamento completo, encontrando sempre um caminho livre entre dois pontos, caso exista.
- **Robustez:** consegue lidar com a presença de obstáculos não conhecidos a priori.

O planejador de caminhos CPH atua sobre um ambiente discretizado em forma de grade ou *grid*. O cálculo inerente à esse método, exposto e apresentado em [3], representa um Problema de Valor de Contorno (PVC) na região de atuação do agente robótico, utilizando a condição de Dirichlet, na qual as células que representam obstáculos são marcadas com potencial unitário e as metas com potencial zero, conforme [30].

As células livres, ou seja, que não são nem metas e nem células bloqueadas com potencial unitário representando paredes ou obstáculos, estão dotadas de um valor de potencial contido no intervalo $0 < p_{i,j} < 1$. Onde i representa a linha da matriz de potenciais e j representa a coluna na referida matriz. A Equação 4.2, conforme o trabalho de [26], interpola os valores dos potenciais para todas as células livres entre os obstáculos e a meta.

$$p_{i,j} = \frac{1}{4}(p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1}) \tag{4.2}$$

Conforme comentado anteriormente o método CPH apresenta integridade, permitindo o agente robótico sempre alcançar o objetivo seguindo as linhas de força, caso exista um caminho possível. Tal fato ocorre uma vez que as soluções da Equação de Laplace não apresentam mínimos ou máximos locais. A Figura 29 ilustra esse fato. Pode-se observar que as linhas de força geradas não orientam o agente robótico à uma situação de mínimo local, ou seja, a meta sempre é alcançada.

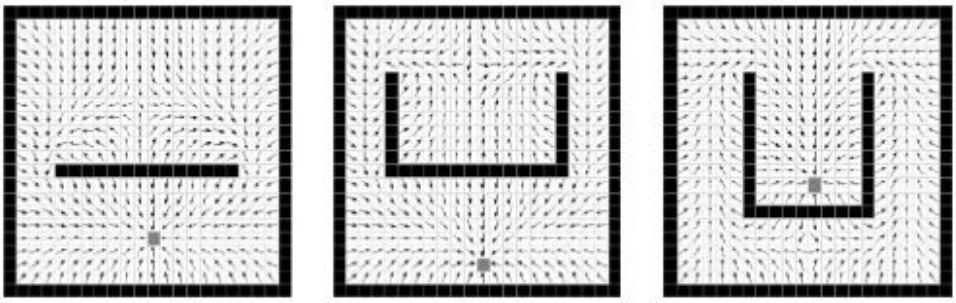

Figura 29 – Exemplo de Campos Potenciais Harmônicos (CPH).

Fonte: Retirado de [26]

No método CPH o robô é direcionado até sua meta seguindo o caminho em um gradiente decrescente dos potenciais de cada célula, sendo possível o desvio de obstáculos que detém um potencial unitário em suas células. [27].

A Equação 4.3 apresenta a direção Θ a ser seguida pelo agente robótico dada a posição do mesmo em uma determinada célula (i,j).

$$\Theta_{i,j} = -\arctan(p_{i-1,j} - p_{i+1,j}, p_{i,j-1} - p_{i,j+1}) \tag{4.3}$$

Sendo Θ pertencente ao intervalo $[-\pi, \pi]$.

4.3 CAMPOS POTENCIAIS ORIENTADOS (CPO)

O método CPO de navegação surge com um conjunto de funções que, assim como a solução da Equação de Laplace, também gera um campo potencial o qual não irá apresentar mínimos locais, sendo esse conjunto um caso generalista dentro do qual está incluído a própria Equação de Laplace.

No estudo de [31], apresenta-se uma família de funções escalares geradas por um PVC e que também apresenta a peculiaridade de não gerar mínimos locais. Esse conjunto

de funções é dado por:

$$\nabla^2 P + F(\nabla P) = 0 \tag{4.4}$$

Sendo o ambiente robótico inserido em um plano, no contexto do universo $\mathbb{R}^2$, podemos definir, para a Equação 4.4:

$$\nabla P = \frac{\partial P}{\partial x} + \frac{\partial P}{\partial y} \tag{4.5}$$

$$\nabla^2 P = \frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} \tag{4.6}$$

Sendo $F(0) = 0$ e $F(\nabla P)$ uma função contínua.

Ainda no estudo de [31], é sugerido uma função linear para o novo termo $F(\nabla P)$:

$$F(\nabla P) = \epsilon \cdot \nabla P \cdot v \tag{4.7}$$

No qual $\epsilon \in \mathbb{R}$ é um escalar e $v \in \mathbb{R}^2$ é um vetor constante ($v = \begin{bmatrix} v_x & v_y \end{bmatrix}$) com $\|v\| = 1$. Substituindo a Equação 4.7 em 4.4, obtém-se:

$$\nabla^2 P + \epsilon \cdot \nabla P \cdot v = 0 \tag{4.8}$$

Pode-se notar que a Equação 4.1 é um caso particular da Equação 4.8, quando $\epsilon = 0$. Ficando, portanto, a Equação 4.7 a seguir, inteiramente análoga à Equação 4.1 para um caso bidimensional, conforme está se tratando.

$$\nabla^2 P = 0 \tag{4.9}$$

Para o método CPO é necessário a discretização da Equação 4.8, não sendo essa discretização parte do escopo deste trabalho, a seguir serão apresentados apenas os resultados do cálculo de discretização, que pode ser visto em [3].

- **Solução Numérica Utilizando Diferenças Centradas**:

  A Equação 4.10 a seguir representa a solução discreta para a Equação 4.8.

$$p_{i,j} = \frac{1}{4}[(1 + \lambda v_y)p_{i+1,j} + (1 - \lambda v_y)p_{i-1,j} + (1 + \lambda v_x)p_{i,j+1} + (1 - \lambda v_x)p_{i,j-1}] \tag{4.10}$$

  Sendo $\Delta x = \Delta y = h$ e definindo $\lambda = \frac{\epsilon h}{2}$

Conforme apresentado em [3], [32] afirma que esse tipo de discretização impõe um limite em λ. Quanto maior for ϵ, menor precisa ser o espaçamento da malha.

Ainda em [32], é dito que quando $\epsilon < 2$, o novo potencial de uma determinada célula é uma média ponderada dos seus vizinhos. Tal ponderação leva em consideração o vetor v e o seu módulo ϵ.

- **Solução Numérica Utilizando Diferenças *Up Wind***

  Em [3], são definidos quatro casos, uma vez que a Equação 4.8 possui duas derivadas de primeira ordem:

  Para $v_x > 0$ e $v_y > 0$:

$$p_{i,j} = \frac{(1 + h\epsilon v_x)p_{i,j+1} + (1 + h\epsilon v_y)p_{i+1,j} + p_{i,j-1} + p_{i-1,j}}{4 + h\epsilon(v_x + v_y)} \tag{4.11}$$

  Para $v_x > 0$ e $v_y < 0$:

$$p_{i,j} = \frac{(1 + h\epsilon v_x)p_{i,j+1} + (1 - h\epsilon v_y)p_{i-1,j} + p_{i,j-1} + p_{i+1,j}}{4 + h\epsilon(v_x - v_y)} \tag{4.12}$$

  Para $v_x < 0$ e $v_y > 0$:

$$p_{i,j} = \frac{(1 - h\epsilon v_x)p_{i,j-1} + (1 + h\epsilon v_y)p_{i+1,j} + p_{i,j+1} + p_{i-1,j}}{4 + h\epsilon(-v_x + v_y)} \tag{4.13}$$

  Para $v_x < 0$ e $v_y < 0$:

$$p_{i,j} = \frac{(1 - h\epsilon v_x)p_{i,j-1} + (1 - h\epsilon v_y)p_{i-1,j} + p_{i,j+1} + p_{i+1,j}}{4 + h\epsilon(-v_x - v_y)} \tag{4.14}$$

Conforme apresentado em [3], [32] afirma que independente do valor de ϵ, o novo potencial de uma determinada célula, sempre será uma média ponderada entre os seus vizinhos.

O método CPO denomina-se Orientado pela existência do vetor v na orientação do campo gradiente (Equação 4.8). Utilizando [26], pode-se apresentar o resultado do método de navegação para diferentes vetores de orientação v, gerado a partir das equações discretizadas apresentadas anteriormente. Nas Figuras 30, 31 e 32 [3] apresenta *grids*

gerados pelo método CPO para três vetores de orientação diferentes, sendo $v\angle 45°$, $v\angle 90°$ e $v\angle 135°$, respectivamente.

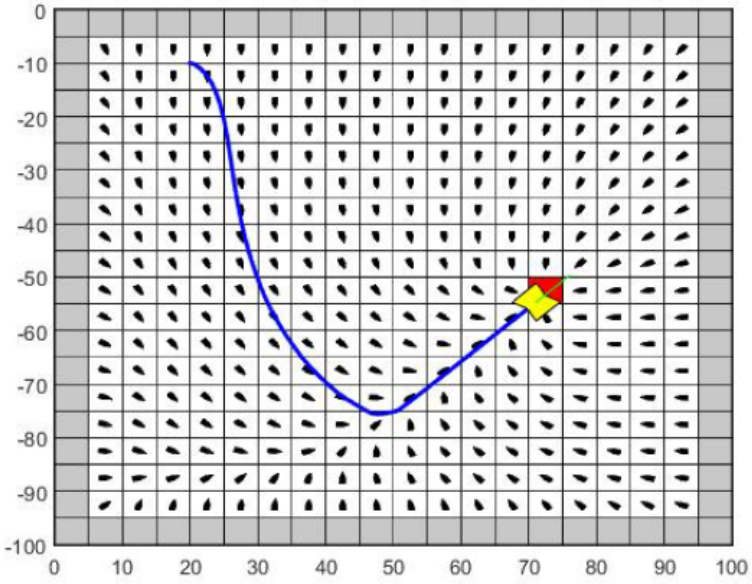

Figura 30 – Influência do vetor $v\angle 45°$ no CPO.

Fonte: Retirado de [3]

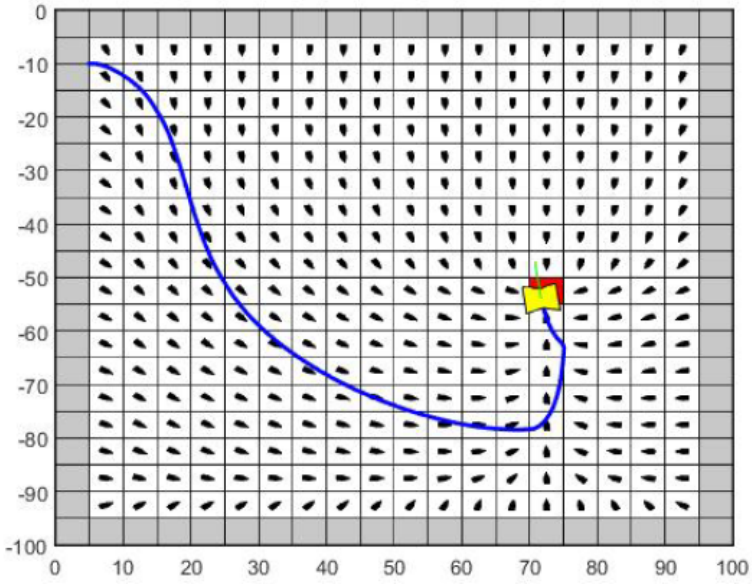

Figura 31 – Influência do vetor $v\angle 90°$ no CPO.

Fonte: Retirado de [3]

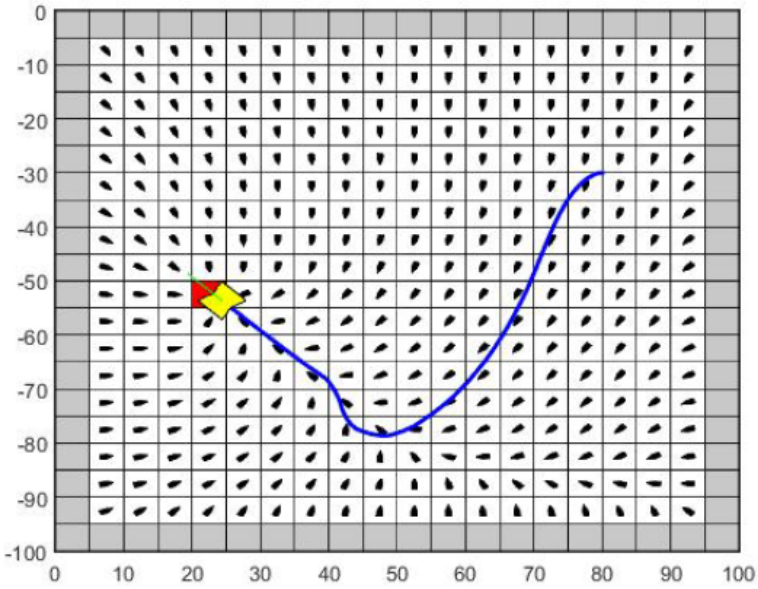

Figura 32 – Influência do vetor $v\angle 135°$ no CPO.

Fonte: Retirado de [3]

No método CPO, segundo [26], a variável ϵ representa a taxa de influência do vetor v sobre o campo potencial gerado. Assim, na Figura 33, [26] apresenta o comportamento do agente robótico inserido no *grid* gerado segundo valores diferentes de ϵ. Em [26] compara-se as trajetórias geradas com ϵ iguais a 0; 0,2; 0,5; 1,0 e 1,5. Pode-se verificar que, quanto maior o valor de ϵ, mais a trajetória se aproxima da direção do vetor $v\angle -90°$.

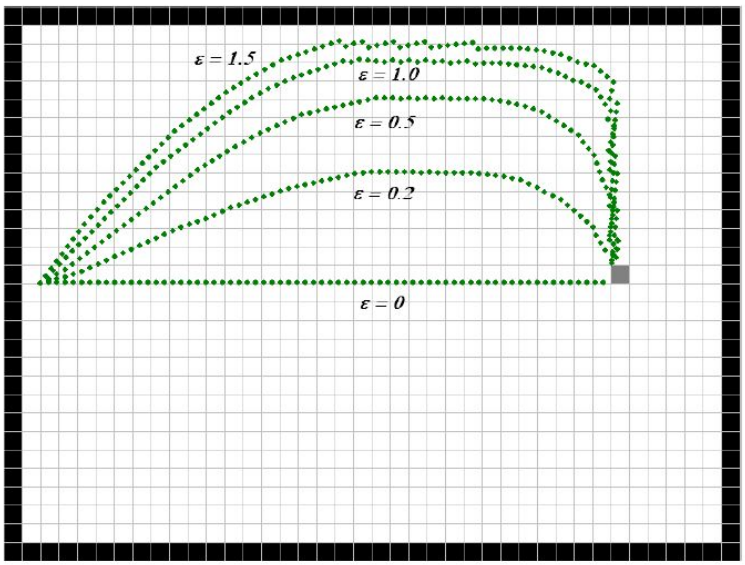

Figura 33 – Comparação entres as trajetórias segundo várias taxas de influência ϵ.

Fonte: Retirado de [26]

Na Figura 34, [26] mostra que o valor de ϵ utilizado no CPO não pode crescer indefinidamente, a depender da discretização utilizada. Gerando um campo instável, gerado por Diferenças Centradas, com o valor de $\epsilon = 4$, não garantindo uma trajetória até a meta.

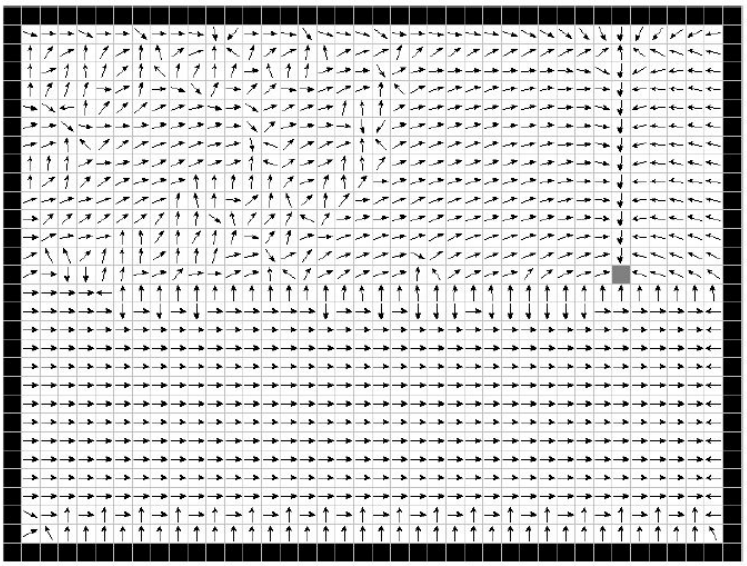

Figura 34 – Campo instável, $\epsilon = 4$.

Fonte: Retirado de [26]

Pode-se dizer que a utilização de um sistema de navegação baseado em CPO e CPH para a criação das funções de jogo é interessante por que evita colisões dos agentes robóticos com obstáculos, e, ao mesmo tempo, não gera mínimos locais. Este capítulo abordou as noções básicas teóricas de ambos os métodos, para tornar-se possível a implementação dessa navegação no sistema proposto para o VSSS. Essa implementação é apresentada no capítulo seguinte, o qual aborda as características das funções de jogo baseadas em CPH e CPO para cada saída do sistema *fuzzy* de apoio à decisão criado.

5 IMPLEMENTAÇÃO DE SISTEMAS DE NAVEGAÇÃO EM FUNÇÕES DE JOGO

Este capítulo é destinado a descrever a implementação das funções de jogo de interesse no âmbito do VSSS. As funções de jogo são criadas a partir dos métodos de navegação descritos no capítulo anterior.

De acordo com a decisão gerada do sistema de apoio *fuzzy*, um comportamento é definido para o agente robótico. Esse comportamento desejado é traduzido pelos métodos de navegação CPO e CPH, com algumas modificações e aplicações necessárias para obter uma movimentação desejada para o robô em questão. Este capítulo, portanto, descreve a aplicabilidade dos métodos de navegação utilizados para guiar a movimentação dos robôs em jogo.

É importante registrar duas regras básicas da categoria VSSS que influenciam diretamente a criação e análise estratégica para as funções de jogo. São essas:

- Um time atacando com mais de um robô, na área do gol adversário, deve ser penalizado por um tiro de meta batido pelo time do goleiro.
- Um time defendendo com mais de um robô na área do gol, quando a bola esta dentro da área, deve ser penalizado com um pênalti, exceto se o time atacante marcar um gol. Se a bola não estiver na área do gol, não há penalizações para os times com dois robôs de defesa dessa área.

5.1 FUNÇÃO DEFESA PESADA

A função de jogo denominada Defesa Pesada foi criada com o intuito de modelar o comportamento do agente robótico nas situações defensivas com risco iminente. Conforme pode-se notar na Base de Regras *fuzzy* propostas, a situação Defesa Pesada (DP) ocorre mediante a ocorrência de FD, FC ou FA em níveis majoritariamente baixos, ou seja, valores pequenos dos fatores indicam uma situação crítica de jogo a qual deve-se reagir de maneira defensiva. Para esse comportamento defensivo, propõe-se a função Defesa Pesada.

A Defesa Pesada, por sua vez, é modelada por uma navegação baseada em CPH. Esse método, conforme tratado no Capítulo 4, é um método de navegação eficaz por não gerar mínimos locais. Além disso, gera um *grid* de células dotadas de uma orientação a ser seguida bem como potenciais associados às mesmas, assim, o agente robótico segue a trajetória gerada por essa grade no sentido ao menor potencial obtido, sendo que os obstáculos são demarcados com potencial máximo, que é unitário.

Nesta função de jogo, a técnica utilizada é a do CPH, uma vez que se torna possível atingir um ponto objetivo de uma maneira eficiente. Como adaptação ao método de navegação, é favorável que os robôs identificados como oponentes sejam setados com

potencial unitário, tratando-os como "obstáculos". Ou seja, as células ocupadas pelos robôs adversários são tratadas com potencial unitário, para que o método de navegação, CPH, não gere trajetórias passando pelo ponto ocupado pelos adversários, gerando confrontos ou obstruções indesejadas.

Outra adaptação ao sistema de navegação utilizado foi estabelecer a meta do CPH para fora da área de defesa no momento em que a bola se encontra na mesma. A partir dos centroides, o sistema reconhece a sua área de ataque e de defesa. Sendo o goleiro programado para não sair da área, estando esse robô, portanto, livre da troca de funções proposta pelo sistema *fuzzy*. Sendo fixa sua função de goleiro, o robô que assumir a função Defesa Pesada não deve entrar na área quando a bola estiver nesse espaço do campo. Assim a meta do CPH é estabelecida sempre com trinta e cinco centímetros a frente do centroide de defesa, garantindo a segurança de não haver dois robôs habitando a área de defesa.

Nos momentos em que a bola estiver no campo de defesa, ou seja, não estiver na área do goleiro, o robô em sua função de Defesa Pesada deve proteger a bola e tirá-la de uma situação de risco. No entanto, se o CPH tratar a bola como meta simplesmente, o robô correria o risco de chegar até a bola e empurrar para seu próprio gol ou para sua área de defesa. Assim, outra proposta foi colocar a posição da bola como meta para o CPH, mas, no entanto, colocar uma barreira fictícia entre a bola e o campo de ataque. Isso obriga o robô a atingir a meta (bola) de qualquer modo exceto empurrando contra seu próprio campo. Utiliza-se uma barreira de três células a frente da bola, estabelecendo para essas células o potencial unitário, no CPH, representando células bloqueadas, ou obstáculos, fazendo com que o robô atinja a bola de forma a evitar uma situação de risco de gol.

5.2 FUNÇÃO DEFESA LEVE

A função de jogo denominada Defesa Leve foi criada com o intuito de modelar o comportamento do agente robótico em situações defensivas mas não tão críticas como na função Defesa Pesada, e sim em situações em que o robô deve agir de forma satisfatória em seu campo de defesa, ou seja, em um comportamento também defensivo porém mais ligado à proteção e acompanhamento da bola no campo de defesa.

Conforme pode-se notar na Base de Regras *fuzzy* proposta, a situação Defesa Leve (DL) ocorre mediante a ocorrência de FD, FC ou FA em níveis baixos, mas não tão baixos ou críticos como na situação anterior (DP). Ou seja, valores pequenos mas não tão críticos dos fatores indicam uma situação na qual deve-se reagir de maneira a acompanhar a bola no setor defensivo, agindo, portanto, defensivamente porém não tão agressivamente como no caso da função DP. Assim, para esse comportamento esperado, propõe-se a função Defesa Leve.

A Defesa Leve também é modelada por uma navegação baseada em CPH. Nesta função de jogo, como adaptação ao método CPH adotado, é favorável que os robôs identificados como oponentes sejam estabelecidos com potencial unitário, tratando-os como obstáculos. Ou seja, as células ocupadas pelos robôs adversários são estabelecidas com potencial unitário para que o método de navegação não gere trajetórias passando pelo ponto ocupado pelos adversários, gerando confrontos ou obstruções indesejadas.

A função Defesa Leve representa uma função de jogo relativa à uma posição adotada em situações comuns de jogo, ou medianas, visto que trata situações que não são de risco crítico ou iminente do time adversário pontuar ou do time a favor pontuar. Mediante a esse contexto, outra adaptação ao sistema de navegação desenvolvido é a proposta do acompanhamento da bola em termos defensivos pelo agente robótico.

O acompanhamento se faz de forma que, quando a bola se encontra nas laterais do campo, a meta estabelecida ao CPH é a posição da bola. Ao chegar na meta, o comportamento programado é o robô girar em seu próprio eixo, no sentido de conseguir tirar a bola. Esse comportamento é útil para os casos comuns em que a bola fica presa na lateral do campo, pelos próprios robôs. Assim, a função DL planeja tratar esses casos gerando a meta sendo a bola e programando-o para rotacionar no sentido de jogar a bola para o campo de ataque.

Na função DL, quando a bola não está nas laterais do campo mas sim no campo de ataque, a função planeja um acompanhamento da mesma em seu campo de defesa, uma proteção ao direcionamento da bola no campo defensivo. Isso se faz definindo um vetor com a direção definida entre a posição da bola e o centroide de defesa (ponto central do gol defensivo).

É definido um percentual da distância entre o centroide de defesa e a posição da bola. Dessa forma, a meta é definida sendo a posição da bola e acrescentando o percentual da distância entre a bola e o centroide de defesa na direção do vetor entre esses dois pontos. Assim sendo, o robô acompanha a direção da bola em seu campo defensivo. O valor do percentual imposto é obtido em testes empíricos. Esse comportamento objetiva acompanhar a posição da bola até que essa esteja no campo defensivo. Esse comportamento está apresentado nos comandos de código a seguir.

```
        Point2d vec_ball_def = centroid_def - ball_pos;
        double aux = (0.45/150)*euclidean_dist(centroid_def,
           ball_pos);

        meta = ball_pos + vec_ball_def*aux;
```

Quando a bola está no campo defensivo, o método de navegação CPH trata a meta

como sendo a posição da bola obtida.

5.3 FUNÇÃO ATAQUE LEVE

A função de jogo denominada Ataque Leve foi criada com o intuito de modelar o comportamento do agente robótico em situações ofensivas mas não tão claras ou eficazes como na função Ataque Pesado (explicada na próxima seção), e sim em situações em que o robô deve agir de forma satisfatória em seu campo de ataque, ou seja, em um comportamento também atacante porém mais ligado à movimentação e acompanhamento da bola.

Conforme pode-se notar na Base de Regras *fuzzy* proposta, a situação Ataque Leve (AL) ocorre mediante a ocorrência de FD, FC ou FA em níveis altos, mas não tão altos ou críticos como na situação a ser explicada relativa ao Ataque Pesado (AP). Ou seja, valores altos mas não tão críticos dos fatores indicam uma situação na qual deve-se reagir de maneira a acompanhar a bola no setor de ataque, agindo não tão agressivamente como no caso da função AP. Para esse comportamento esperado propõe-se a função Ataque Leve.

O Ataque Leve é modelado por uma navegação baseada em CPO. O agente robótico segue a trajetória gerada pelo *grid* no sentido ao menor potencial obtido, sendo que os obstáculos são demarcados com potencial máximo, que é unitário. Essas características são inteiramente análogas ao CPH, no entanto o CPO contém a peculiaridade de ser orientado pela existência de um vetor v que representa uma orientação a ser seguida no campo gradiente. Assim, o uso do CPO é destinado a situações nas quais seja favorável atingir a meta sob certa orientação.

Nesta função de jogo também é favorável que os robôs identificados como oponentes sejam estabelecidos com potencial unitário, tratando-os como obstáculos. Além disso, nessa função, caso a bola esteja no campo defensivo, distante da área e campo de ataque, a função Ataque Leve gera um CPH a partir do CPO formulado, definindo-se nulo o valor de ϵ, conforme explicitado no capítulo anterior. Esse CPH tem como meta o meio do campo, visando que, enquanto a bola estiver numa posição de defesa, o Ataque Leve gera como meta o centro do campo, até que a posição da bola mude e o Ataque Leve possa, efetiva e objetivamente, movimentar o robô.

Se a bola estiver se movimentando longe do campo defensivo, a função Ataque Leve deve garantir a movimentação do agente robótico objetivando conduzir a bola até o gol, para pontuar. Ou, ainda que não seja possível, chegar com a bola próximo ao gol de ataque, pois assim estará favorecendo a ocorrência da função de Ataque Pesado, a ser explicada na próxima seção. Para esse caso, o desenvolvido foi o CPO com a meta sendo fixada na posição ocupada pela bola.

A taxa de influência utilizada para o CPO será detalhada na seção de Resultados

Obtidos. A angulação ou orientação para o CPO foi desenvolvida como sendo o ângulo entre a posição da bola e o gol de ataque. Assim, a função se torna capaz de fazer o agente robótico se deslocar e atingir a meta (bola) sob orientação definida favorecendo o chute a gol.

O cálculo da orientação para o CPO é feito a partir do vetor de direção entre os dois pontos (posição da bola e centroide de ataque), depois mede-se o ângulo formado entre esse vetor e a direção do eixo x (Equação 3.5). Sendo essa, portanto, a orientação do CPO utilizado. Antes de fixar a orientação, porém, é necessário garantir que esse ângulo esteja no intervalo $[-180, 180]$. O cálculo, em código, utilizado para fixar a orientação da navegação é apresentado abaixo.

```
//Calcula angulo entre a bola e o gol de ataque
Point2d vec_ball_atk = centroid_atk-ball_pos;
double ang_ball_atk = angle_two_points(vec_ball_atk,
   eixo_x);
if (vec_ball_atk.y < 0)
ang_ball_atk = -ang_ball_atk;
//ajusta angulos para menores que 180 e maiores que -180
if (ang_ball_atk > 180) ang_ball_atk = ang_ball_atk -
   360;
else if (ang_ball_atk < -180) ang_ball_atk =
   ang_ball_atk + 360;
//cout << "Angulo bola atk: " << ang_ball_atk << endl;
orientation = ang_ball_atk;
```

5.4 FUNÇÃO ATAQUE PESADO

A função de jogo denominada Ataque Pesado foi criada com o intuito de modelar o comportamento do agente robótico em situações ofensivas claras ou críticas, onde existe a iminência de efetuar um gol a favor. Nessas situações o robô deve agir de forma objetiva e clara visando a pontuação, tendo um comportamento puramente atacante ou ofensivo.

Conforme pode-se notar na Base de Regras *fuzzy* proposta, a situação Ataque Pesado (AP) ocorre mediante a ocorrência de FD, FC ou FA em níveis majoritariamente altos, ou seja, valores elevados dos fatores indicam uma situação crítica de jogo a qual deve-se reagir de maneira puramente ofensiva ou atacante. Para esse comportamento, propõe-se a função Ataque Pesado.

O Ataque Pesado é modelado por uma navegação baseada em CPO uma vez que é interessante que o robô atinja a meta sob certa orientação. Assim como nas outras funções de jogo, as posições dos robôs identificados como oponentes são estabelecidas

com potencial unitário, tratando-os como obstáculos para que não gere confrontos ou obstruções indesejadas.

A função Ataque Pesado visa garantir a condução da bola até o gol pelo agente robótico, para pontuar a favor. Para esse comportamento, o desenvolvido foi o CPO cuja taxa de influência utilizada será detalhada na seção de Resultados Obtidos. A angulação ou orientação para o CPO foi desenvolvida como sendo o ângulo entre a posição da bola e o gol de ataque, exatamente utilizando o mesmo método explicado na seção Ataque Leve.

A peculiaridade da função Ataque Pesado que a difere das demais, e, que faz com que essa se torne capaz de fazer o agente robótico se deslocar e ter grandes chances de pontuar, é a fixação da meta imposta pelo CPO. Caso o robô esteja longe da bola, a meta é a posição da bola na orientação dada pela direção tomada até o centroide do gol de ataque. Caso o robô chegue na bola, ou já esteja com ela, dado que sua distância até a bola seja mínima constatando sua posse de bola, a meta do CPO muda para o centroide do gol de ataque.

De maneira prática, na função AP o robô visa a bola seguindo a orientação que o direciona até o gol de ataque favorecendo o chute a gol. Quando o robô chega até a bola, a meta passa a ser o centro do gol de ataque. Essa peculiaridade faz com que, o robô procurando se movimentar até alcançar a meta, crie possivelmente uma oportunidade grande de vir a pontuar, ou seja, fazer gol.

6 RESULTADOS OBTIDOS

Este capítulo apresenta os resultados obtidos por este trabalho. Essa apresentação se baseia tanto nos resultados da implementação do sistema *fuzzy* para apoio à decisão quanto nos resultados obtidos pela implementação proposta para os sistemas de navegação utilizados.

6.1 SISTEMA *FUZZY* DE APOIO À DECISÃO

Com a Base de Regras criada, o sistema *fuzzy* então pode ser completamente implementado. O sistema é capaz de, identificando as posições dos jogadores a favor da equipe bem como seus ângulos de orientação, e as posições em campo dos jogadores oponentes, calcular automaticamente os fatores FD, FC e FA. Na etapa de fuzzyficação, as três entradas dos três fatores são dispostas e analisadas quanto aos graus de pertinência em relação aos grupos Baixo, Médio$_1$, Médio$_2$ e Alto, gerando assim, para cada um desses fatores os respectivos graus de pertinência.

Após a fuzzyficação, é feito a etapa da inferência utilizando o método *Mamdami*, ou seja, o Método Min-Máx, sendo o mínimo utilizado para a composição e o máximo para a agregação (*Figura 13*), com base no conjunto de regras criado apresentado anteriormente.

Na defuzzyficação, o método utilizado foi o do centroide ou centro de massa, gerando um valor numérico para a saída, e, a partir desse valor, é calculada o maior grau de pertinência para o mesmo, e assim, é gerada a decisão. Sendo a saída o grupo cujo grau de pertinência do valor numérico de saída for maior.

Com o sistema elaborado, foi testado todas as situações inicialmente criadas para a elaboração da base de regras. Os resultados dos testes são apresentados na Figura 35 que relata, para cada uma das situações, a decisão gerada pelo sistema. Os *status* em verde significam que a resposta do sistema foi de acordo com a reação esperada pelo sistema no momento da criação da base de regras.

Apenas quatro situações divergiram do esperado, representando assim 5% de erro. Entretanto, a reação não esperada pelo sistema não representou uma divergência tão considerável uma vez que, ao alterar minimamente o valor de quaisquer posições de bola ou do robô a reação passa a ser a esperada na criação do sistema. Isso representa, portanto, situações limítrofes nas quais o sistema não perde a robustez por diferir em sua resposta. As respostas geradas são muito próximas das respostas esperadas, conforme pode-se averiguar na Figura 35 que apresentam os resultados obtidos.

	Decisão Fuzzy	Status
Foto 1	Ataque Leve	
Foto 2	Defesa Leve	
Foto 3	Defesa Leve	
Foto 4	Defesa Leve	
Foto 5	Defesa Leve	
Foto 6	Ataque Leve	
Foto 7	Defesa Leve	
Foto 8	Ataque Leve	
Foto 9	Ataque Pesado	
Foto 10	Defesa Leve	
Foto 11	Ataque Pesado	
Foto 12	Defesa Leve	
Foto 13	Ataque Leve	
Foto 14	Defesa Leve	
Foto 15	Defesa Leve	
Foto 16	Ataque Leve	
Foto 17	Ataque Pesado	
Foto 18	Ataque Leve	
Foto 19	Defesa Leve	
Foto 20	Defesa Leve	
Foto 21	Defesa Leve	
Foto 22	Defesa Leve	
Foto 23	Defesa Leve	
Foto 24	Defesa Leve	
Foto 25	Ataque Leve	
Foto 26	Defesa Leve	

	Decisão Fuzzy	Status
Foto 27	Defesa Leve	
Foto 28	Ataque Leve	
Foto 29	Ataque Leve	
Foto 30	Defesa Leve	
Foto 31	Ataque Leve	
Foto 32	Ataque Pesado	
Foto 33	Defesa Leve	
Foto 34	Defesa Leve	
Foto 35	Defesa Pesada	
Foto 36	Defesa Pesada	
Foto 37	Defesa Leve	
Foto 38	Defesa Leve	
Foto 39	Ataque Leve	
Foto 40	Ataque Pesado	
Foto 41	Defesa Leve	
Foto 42	Ataque Leve	
Foto 43	Ataque Leve	
Foto 44	Defesa Leve	
Foto 45	Defesa Leve	
Foto 46	Defesa Leve	
Foto 47	Ataque Leve	
Foto 48	Ataque Leve	
Foto 49	Ataque Leve	
Foto 50	Ataque Leve	
Foto 51	Defesa Leve	
Foto 52	Ataque Leve	

	Decisão Fuzzy	Status
Foto 53	Ataque Pesado	
Foto 54	Ataque Leve	
Foto 55	Ataque Leve	
Foto 56	Ataque Leve	
Foto 57	Defesa Leve	
Foto 58	Defesa Leve	
Foto 59	Defesa Leve	
Foto 60	Ataque Leve	
Foto 61	Ataque Leve	
Foto 62	Ataque Leve	
Foto 63	Defesa Leve	
Foto 64	Defesa Leve	
Foto 65	Defesa Leve	
Foto 66	Defesa Leve	
Foto 67	Defesa Leve	
Foto 68	Defesa Leve	
Foto 69	Ataque Leve	
Foto 70	Ataque Leve	
Foto 71	Ataque Leve	
Foto 72	Ataque Leve	
Foto 73	Ataque Leve	

LEGENDA	
	Confere
	Não Confere

Figura 35 – Resultados obtidos em relação às entradas propostas na criação da base de regras *fuzzy*.

A título ilustrativo, apresenta-se nas Figuras 36 a 39 os resultados de algumas entradas para o sistema *fuzzy* criado. A título de testes, foi feito no simulador [24] o sistema criado, adicionando a mesma base de regras criada, utilizando os mesmos grupos e funções de pertinência previamente apresentados na Seção 3.3. Os resultados obtidos são expostos nas Figuras 36 até 39, podendo verificar as entradas do sistema, algumas regras disparadas de acordo com as entradas que geram o resultado da inferência máx-min e por sua vez um número *fuzzy* que, depois da defuzzyficação gera a saída numérica apresentada na variável "DECISÕES".

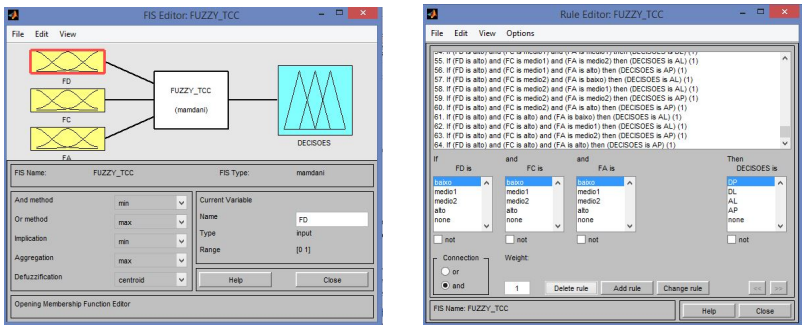

Figura 36 – Sistema *fuzzy* simulado em [24]

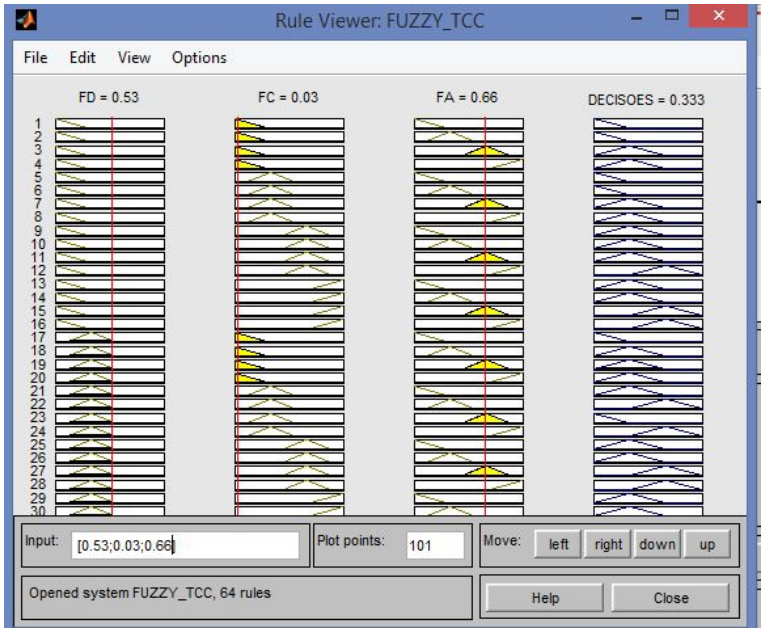

Figura 37 – Resultado obtido para uma entrada [0.53, 0.03, 0.66]

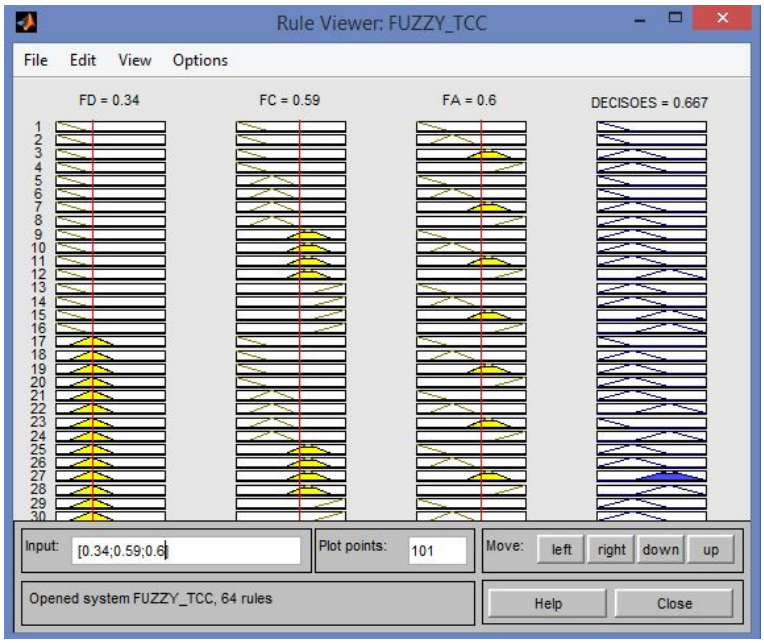

Figura 38 – Resultado obtido para uma entrada [0.34, 0.59, 0.06]

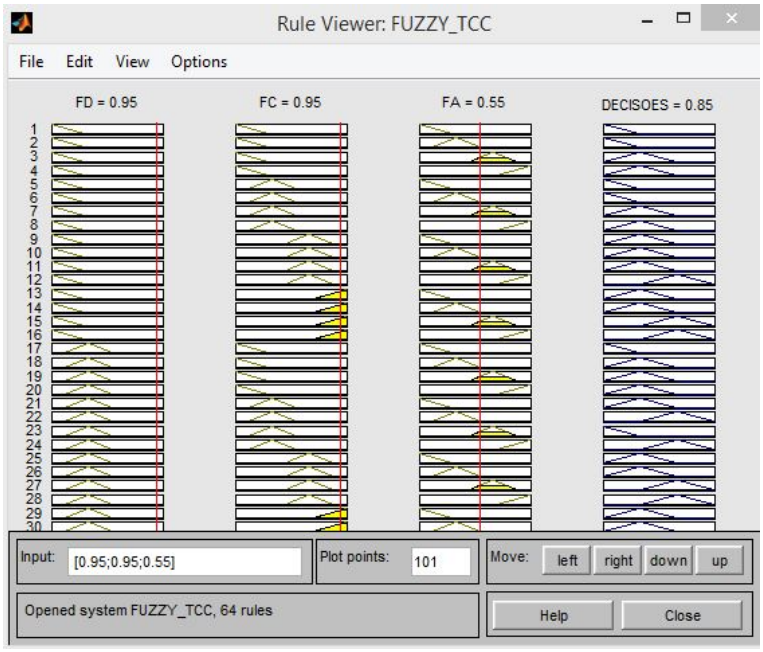

Figura 39 – Resultado obtido para uma entrada [0.95, 0.95, 0.55]

O sistema se comporta de acordo com o esperado. Sendo um bom aproximador das decisões tomadas no decorrer da partida tendo como entrada apenas os fatores, cujos cálculos dependem apenas de posições e orientações dos agentes em campo. A atualização dos fatores no tempo de captura da câmera faz com que várias decisões sejam geradas e a partida possa ser regida de acordo com esse sistema de apoio à decisão, assim, pode ser considerado satisfatoriamente robusto em termos computacionais e práticos.

6.2 SISTEMAS DE NAVEGAÇÃO APLICADOS ÀS FUNÇÕES DE JOGO

Esta Seção visa ilustrar e apresentar a eficácia da proposta de adotar o CPH e CPO como sistema de navegação para o ambiente do VSSS autônomo e sua integração com o sistema *fuzzy* de apoio à decisão. A seguir é apresentado a simulação de alguns movimentos pertinentes à cada função de jogo apresentada. As simulações foram feitas gerando os *grids* de potenciais e de orientações angulares nas implementações desses sistemas de navegação em C++ nos Apêndices C e D desse trabalho. Esses *grids* foram inseridos em MATLAB® e simulados a movimentação do robô para cada peculiaridade da função de movimento proposta.

Nas simulações a serem apresentadas, os obstáculos pintados de preto representam os robôs adversários, os obstáculos pintados de preto com círculos roxos representam os robôs aliados. O robô pintado de roxo é o robô cuja movimentação está se analisando. A estrela representa a posição da meta atual, o círculo verde representa a posição da bola. A sigla ATK é para referenciar o lado do campo para o qual o robô ataca.

A Figura 40 é representativa da função Defesa Pesada. Nota-se, nessa simulação, a movimentação do robô para buscar a bola no seu campo defensivo. É notada a barreira virtual comentada na Seção 5.1, a qual impossibilita o robô chegar até a bola de forma que, acidentalmente, poderia empurrá-la contra seu próprio gol. O quadrado azul representa um obstáculo virtual que não existe no campo real (barreira fictícia).

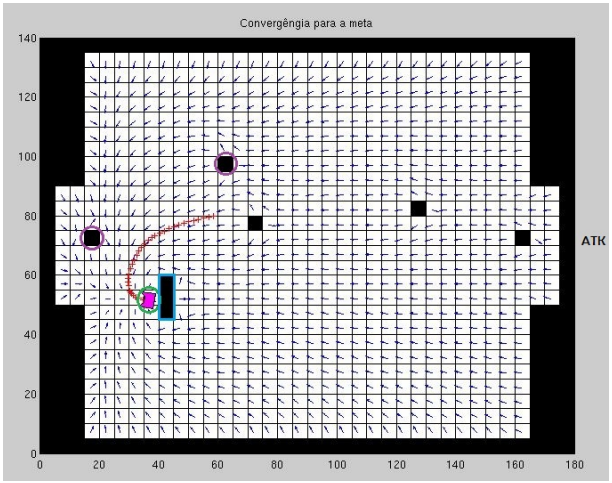

Figura 40 – Exemplo de movimentação relativa à função de jogo *Defesa Pesada*.

A Figura 41 a ser apresentada é representativa da função Defesa Leve. Nota-se, nessa simulação, a movimentação do robô para acompanhar a bola no seu campo defensivo. É notada a meta em seu campo defensivo como sendo à uma distância pré definida e calculada, de forma que o robô acompanha a linha da bola, em uma movimentação típica de característica defensiva.

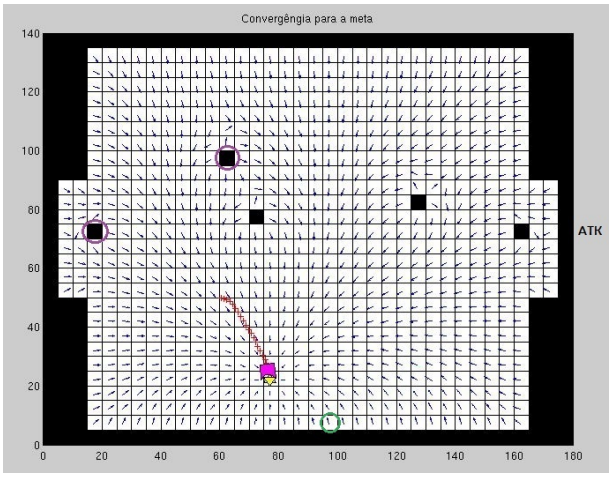

Figura 41 – Exemplo de movimentação relativa à função de jogo *Defesa Leve*.

A Figura 42 é representativa da função Ataque Leve. Nota-se, nessa simulação, a movimentação do robô para acompanhar a bola em seu campo de ataque. É notada

a meta no meio de campo. Estando a bola em seu campo defensivo, distante da área e campo de ataque a meta é atribuída ao meio do campo conforme explicitado na Seção 5.3.

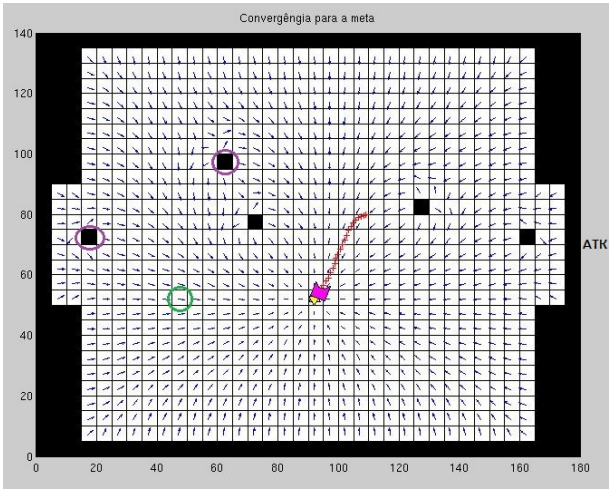

Figura 42 – Exemplo de movimentação relativa à função de jogo *Ataque Leve*.

A Figura 43 a ser apresentada é representativa da função Ataque Pesado. Nota-se, nessa simulação, a movimentação do robô para chegar até a bola de forma orientada em relação ao gol. Essa imagem retrata o primeiro momento da proposta da função Ataque Pesado, uma vez que observa-se a meta do CPO sendo a posição da bola. Conforme a Seção 5.4 foi explicado que, estando o robô com a posse de bola, ou seja, a uma distância mínima até a posição da bola, a próxima etapa na dinâmica de jogo seria a mudança da meta para uma posição no centro do gol de ataque.

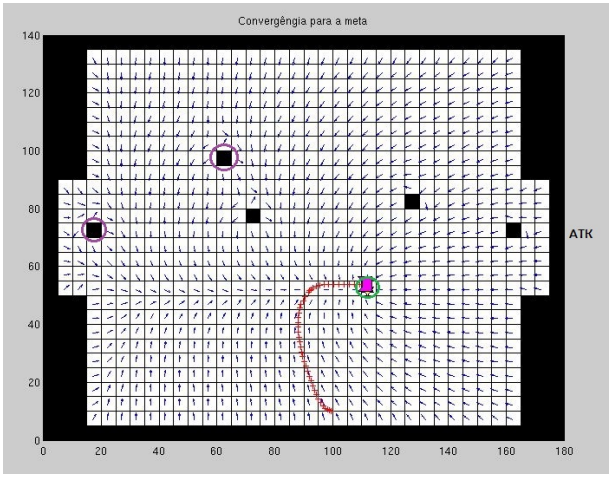

Figura 43 – Exemplo de movimentação relativa à função de jogo *Ataque Pesado*.

Para a função de jogo e movimentação relativa ao Ataque Leve e ao Ataque Pesado, em que ambos utilizam o CPO como sistema de navegação, a taxa de influência ϵ utilizada é função da distância entre as posições do robô e da bola. A proporção do quanto varia

o valor de ϵ com a variação da distância entre o robô e a bola é obtida empiricamente. Gera-se o fato de quanto maior a distância entre o robô e a bola, o valor numérico de ϵ aumenta, e vice versa. Isso garante que, quando o robô está longe da bola, o movimento obtido funciona como um CPO, o agente se movimenta de forma a orientar-se no sentido do vetor de direção do método. Conforme o robô vai se aproximando da bola, o valor de ϵ muda de forma que o movimento passa a se aproximar de um CPH tradicional, o qual não segue uma direção pré estabelecida para atingir a meta.

A adaptação proposta é motivada pelos testes em robô real. Uma vez que o controle de baixo nível e captação da orientação do agente robótico pela visão computacional é bem sensível e frágil. O robô real tem dificuldades para seguir reto quando atinge o vetor de direcionamento. Assim, a modificação visa obter o comportamento do CPO até que esse atinja a direção do vetor, e assim o movimento vai se aproximando de um CPH na medida em que se aproxima da meta (bola) com a diminuição de ϵ.

7 CONCLUSÕES

Este trabalho apresentou um método de tomada de decisão apoiado por um sistema *fuzzy*. O sistema de apoio à decisão se comporta de forma satisfatória e com ampla aplicabilidade em termos computacionais e práticos. Sendo um bom aproximador das decisões tomadas no decorrer da partida mediante os valores dos fatores FD, FC e FA. Tal sistema pode ser considerado satisfatoriamente robusto para o contexto envolvido.

Sobre o sistema de navegação implementado e adaptado de acordo com a função escolhida pelo sistema decisório, pode-se dizer que o CPH demonstra sua eficiência na medida em que é capaz de planejar caminhos livres de colisões, bem como ileso da ocorrência dos mínimos locais, ou seja, um caminho livre até a meta sempre será gerado caso ele exista. Segundo [3], o método CPH/CPO se expande também no contexto de exploração de ambientes. Sendo o custo computacional deste método crescente na proporção do tamanho do ambiente mapeado.

A aplicabilidade do sistema e estratégia tratados neste documento é satisfatória. Este trabalho é motivado pelo aprimoramento das técnicas e estratégias de jogo no contexto VSSS da *Rinobot Team* da UFJF. Visto esse cenário pode-se dizer que o trabalho cumpre com o objetivo esperado, já que supera o modelo estratégico até então utilizado pela Equipe, para a adoção de um sistema mais autônomo no âmbito de tomada de decisões instantâneas no decorrer da partida. A aplicabilidade do sistema criado pôde ser verificada no evento IWCA 2017 (*Inatel Week of Control and Automation*), onde, apesar das limitações e dificuldades encontradas (comentadas nas próximas seções), o sistema pôde ser apresentado.

7.1 PONTOS POSITIVOS

Alguns pontos positivos ou benéficos para o objetivo de criação do sistema abordado neste trabalho podem ser listados e apresentados a seguir:

- Modelagem e adaptação dos fatores FD, FC e FA, capazes de exprimir em termos das variações numéricas dos mesmos a real e instantânea condição de jogo no decorrer da partida VSSS.
- Divisão em quatro grupos *fuzzy* (Baixo, Medio 1, Medio 2 e Alto), cobrindo um leque maior de possibilidade de condições de jogo, ou seja, modelando de forma mais completa as situações possíveis de ocorrer durante a partida, evitando a ocorrência de sempre a mesma função de jogo, ficando esta sobrecarregada a cumprir com diversos objetivos durante o jogo.
- Criação de Base de Regras satisfatoriamente completa e robusta, baseada em setenta

situações típicas e atípicas de jogo, modelando de forma realística o sistema de interesse.

- Utilização de sistemas de navegação baseados em CPH e CPO, que retratam sistemas robustos e completos, gerando caminhos livre de colisões e não ocorrendo os mínimos locais, como pode ocorrer no tradicional método de CP de *Khatib*.

- Funções de jogo (DP, DL, AL, AP) bem definidas, sendo possível modelar o comportamento esperado para cada uma dessas funções a partir de adaptações no próprio sistema de navegação (CPH/CPO). O comportamento esperado para cada saída do sistema decisório foi obtido.

- Pequeno esforço computacional. Todo o cálculo do sistema *fuzzy* e bem como a solução numérica da navegação utilizada foi feito entre dois *frames* da câmera, sendo portanto em média entre vinte e trinta milissegundos.

7.2 PONTOS NEGATIVOS E LIMITAÇÕES DO SISTEMA

Para o sistema criado, também foi obtido algumas limitações, confrontos, pontos indesejáveis ou de certa forma prejudiciais à obtenção de um sistema ideal. Esta seção visa identificar quais foram os pontos negativos ou barreiras para implantação do sistema proposto.

Algumas limitações obtidas, foram, por exemplo, o confronto de posições pelo sistema decisório. O sistema criado analisa cada fator FD, FC e FA individualmente para cada agente robótico, exceto o goleiro que tem posição fixa. Assim, a decisão gerada pelo sistema *fuzzy* é gerada para cada jogador individualmente de acordo com os fatores obtidos. Assim sendo, uma decisão sobre qual função de jogo adotar para um jogador não influencia na decisão do outro, com isso, o sistema está sujeito à ocorrer confronto de posições e disputas entre os próprios jogadores do mesmo time.

Para evitar a ocorrência de confronto de posições, a decisão gerada individualmente foi tratada pelo sistema criado. Assim, se ambos os robôs gerarem decisões de ataque, o sistema identifica esse confronto, para não ocorrer de os dois disputarem a bola e se chocarem e não aproveitarem a oportunidade ofensiva. O sistema faz o robô que estiver mais próximo ao gol defensivo retornar à um comportamento de Defesa Leve.

Ainda para evitar a ocorrência de confronto de posições, se ambos os robôs gerarem decisões de defesa, o sistema identifica esse confronto e faz o robô que estiver mais distante do gol defensivo retornar à um comportamento de Ataque Leve, enquanto o outro permanece na decisão originalmente gerada. Isso é feito para não ocorrer de os dois robôs disputarem o mesmo posicionamento e assim a defesa não ser eficaz.

Pode-se citar também a sensibilidade do sistema geral para com a captação do ângulo de orientação dos agentes robóticos. As trepidações ou flutuações do ângulo de orientação obtidas pela visão computacional podem comprometer o sistema no âmbito da navegação e de decisão. A modelagem do fator FA é dependente da orientação do robô, caso essa orientação trepide, ou exista uma variação considerável, o valor de FA muda, podendo acarretar numa decisão indevida pelo sistema. Além disso, os *grids* de direção gerados pelo método de navegação CPO e CPH dependem também da captação correta do ângulo do agente, para garantir a navegação adequada. Para esse caso, é importante que o controle de baixo nível do agente seja robusto o suficiente para acompanhar a navegação proposta.

A navegação que utiliza CPO como método principal (Ataque Leve e ao Ataque Pesado) necessitou de ajustes. A taxa de influência ϵ utilizada é função da distância entre as posições do robô e da bola. Quanto maior a distância entre o robô e a bola, maior o valor numérico de ϵ. Fazendo que, quando o robô está longe da bola, o movimento obtido funciona como um CPO. Conforme o robô vai se aproximando da bola, o movimento passa a se aproximar de um CPH tradicional. Isso ocorre uma vez que, pelas trepidações da angulação pela visão computacional, bem como o controle de baixo nivel, o robô real tem dificuldades para seguir reto quando atinge o vetor de direcionamento. Assim, a modificação visa corrigir essa limitação real.

7.3 TRABALHOS FUTUROS E POSSÍVEIS MELHORIAS

Esta seção visa, a partir do sistema criado e desenvolvido neste trabalho, relatar possíveis melhorias e estudos para possíveis trabalhos futuros. Algumas das tais possibilidades serão listadas nos itens a seguir.

- Uma modificação possível é o tratamento dos robôs de um mesmo time como obstáculos. Neste trabalho os robôs adversários são vistos como barreiras para os robôs em questão, para evitar choque desnecessários. Da mesma forma, pode-se adotar a mesma medida para robôs que jogam a favor, evitando assim confrontos por posições e choques desnecessários.

- Neste trabalho, para o sistema criado, o algoritmo *fuzzy* é processado em paralelo, entre duas capturas consecutivas da câmera, juntamente com os métodos de navegação relativos às quatro funções de jogo. Feito isso, após a decisão *fuzzy*, é selecionado o método de navegação relativo à função de jogo adotada. Uma sugestão seria a implantação de um sistema sequencial, que executa a função de jogo após a decisão gerada pelo sistema.

- Uma possível melhoria seria a criação de um sistema autônomo supervisório utilizando algum tipo de inteligência artificial tais como redes neurais artificiais (RNA),

algoritmos evolutivos ou até mesmo um sistema ou controle *fuzzy* capaz de tratar o sistema de forma coletiva e não mais individual. Ou seja, o sistema reconheceria os três robôs jogadores e proporcionaria uma técnica de revezamento de funções ou posições de jogo para o time como um todo, abandonando o tratamento individual no qual o comportamento de um robô do time não influencia o outro.

- Separação de atuação dos robôs jogadores por regiões pré definidas de atuação. Como por exemplo, região direita, esquerda ou central do campo. Evitando o confronto de posições dos robôs jogadores e promovendo maior organização do time jogador. Essa técnica poderia ser utilizada juntamente com uma inteligência embarcada supervisória, conforme comentado no item anterior.

- Criação e modelagem de diversos outros fatores de indicação para modelagem das condições de jogo. Além dos fatores criados (FD, FC e FA) esses fatores contribuiriam para melhor modelagem do jogo e possibilitaria até o revezamento de funções incluindo a posição de goleiro, por exemplo, uma vez que tais fatores indicariam, no decorrer do jogo, o melhor atuante para assumir a posição de goleiro.

- Possibilidade de, baseado nos métodos de navegação adotados e em um sistema supervisório completo que se integra uma coletividade de robôs em atividade, utilizar tais técnicas em trabalhos de cooperação entre robôs, possivelmente também em trabalhos de exploração de ambientes desconhecidos uma vez que CPO e CPH são métodos de navegação completos.

- Melhorias na precisão angular na orientação dos agentes robóticos, obtendo maior confiabilidade em relação ao sistema decisório abordado e também em relação à navegação adotada.

- Uma sugestão de possível implementação é um comportamento nas situações de ataque em que, dado um horizonte de eventos para o robô jogador, logo que esse robô identifica a presença de um robô oponente a sua frente é mudada a orientação da direção a ser seguida por parte do robô jogador. Dessa forma o robô já pode desviar do robô oponente com antecedência e, possivelmente, evitando a defesa por parte do jogador adversário. Sem a implementação deste comportamento, o robô só desviará do robô oponente (já que o robô adversário é interpretado como obstáculo) quando estiver próximo do mesmo, facilitando a defesa por parte deste. Este comportamento, batizado de "Drible", é eficaz somente quando se tem um controle de baixo nível eficaz, garantindo a exatidão nesses movimentos propostos.

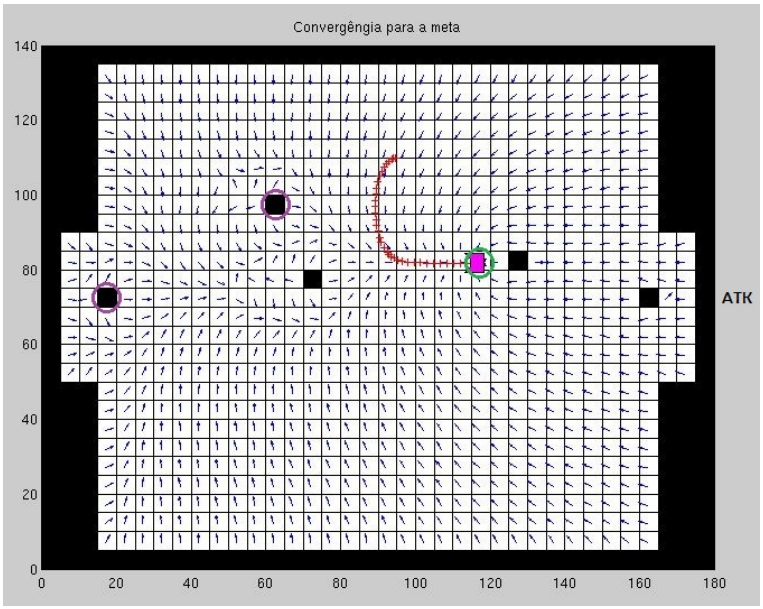

Figura 44 – Simulação de comportamento sem utilizar o drible.

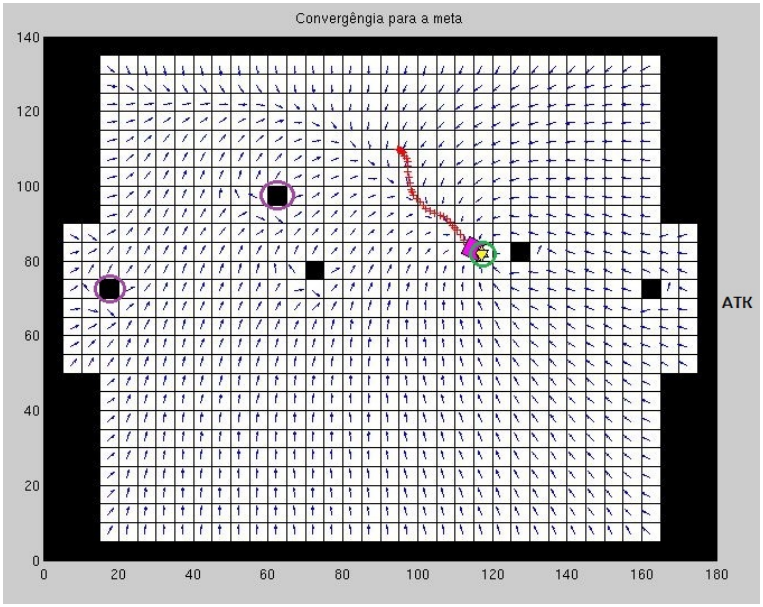

Figura 45 – Simulação de comportamento utilizando o drible.

REFERÊNCIAS

[1] SILVA, A.; SILVA, L. R.; ALMEIDA, P. L. O.; VIDAL, L. C. *Robótica na Indústria Atual*. Disponível em <*http://www.aedb.br/wp-content/uploads/2015/05/8202.pdf*>, acessado em Maio, 2017.

[2] MAIA, D. V. A. *Automação Industrial e Robótica*. UFRN – Universidade Federal do Rio Grande do Norte, Natal, RN. Disponível em <*http://www.aedb.br/wp-content/uploads/2015/05/8202.pdf*>, acessado em Maio, 2017.

[3] Alcântara Pires, Pedro Antônio. *Navegação de Robôs Móveis através de Campos Potenciais Baseados em Problemas de Valor de Contorno* / Pedro Antônio Alcântara Pires. – 2016.70 f. : il.

[4] *https://www.generationrobots.com/en/402395-robot-mobile-pioneer-3-dx.html*, acessado em Março, 2017.

[5] *http://www.cbr09.fei.edu.br/Regras/regras2009VerySmall.pdf*, acessado em Janeiro, 2017.

[6] ANDRADE, M. T. C. de. *Uma contribuição à pesquisa em inteligência computacional*. Tese (Livre Docência) - Escola Politécnica da Universidade de São Paulo, São Paulo, 2002.

[7] LEGASPE, E. P. *Controlador fuzzy de código aberto para uso em controladores programáveis*/E.P.Legaspe. - São Paulo, 2012. 144p.

[8] ZADEH, L. A. *Fuzzy Sets. Information and Control*, v. 8, p. 338–353, 1965.

[9] FARIA, L. T. *Sistema inteligente híbrido intercomunicativo para detecção de perdas comerciais*. 2012. 112 f. Dissertação (Mestrado em Automação) – Faculdade de Engenharia, Universidade Estadual Paulista, Ilha Solteira, 2012.

[10] REZENDE, S. O. et al. *Sistemas inteligentes: fundamentos e aplicações*. Barueri: Manole, 2005. 525 p.

[11] MINUSSI, C. R. *Lógica Nebulosa (Lógica Fuzzy)*. Ilha Solteira: Unesp/FE/DEEE, 2009. 119 p.

[12] SIMÕES, M. G.; SHAW I. S. *Controle e Modelagem Fuzzy*. São Paulo. Edgard Blucher. 2.Edição. 2007

[13] *http://www.pucsp.br/ logica/Fuzzy.htm*, acessado em Maio, 2017.

[14] *https://www.slideshare.net/HlioJovo/2-sistema-fuzzy*, acessado em Maio, 2017.

[15] *https://www.researchgate.net/figure/228778271_fig2_Fig-2-Esquema-simplificado-de-um-sistema-fuzzy*, acessado em Maio, 2017.

[16] NOGUEIRA, M. M.;*Aplicando lógica fuzzy no controle de robôs móveis usando dispositivos lógicos programáveis e a linguagem VHDL* /Maycon Mariano Nogueira. – Ilha Solteira: [s.n.], 2013 95 f. : il.

[17] REZENDE, S. O. *Sistemas Inteligentes - Fundamentos e Aplicações*. Editora Manole Ltda. Barueri, SP, 2003.

[18] MAMDANI, E. H. *Application of Fuzzy Logic to Approximate Reasoning using Linguistic Synthesis*, Queen Mary College, London, 1977.

[19] LEE, C. C. *Fuzzy logic in control system: fuzzy logic controller* Part II. IEEE Transactions on System, Man and Cybernetic, Vol. 20, nº 2, p. 419-435, 1990

[20] SUGENO, M. *An introductory survey of fuzzy control information*. Science, London, v. 36, 1974.

[21] MARRO, A. A.; SOUZA, A. M. C.; CAVALCANTE, E. R. S.; BEZERRA, G. S.; NUNES, R. O. *Lógica Fuzzy: Conceitos e Aplicações* Departamento de Informática e Matemática Aplicada (DIMAp), Universidade Federal do Rio Grande do Norte (UFRN), Natal, RN, Brasil.

[22] McNeill, F. Martin; Thro, Ellen. (1994) Fuzzy Logic: A practical approach. AP Professional/Academic Press.

[23] BATISTA, M. R.; SILVA, M. O.; ROMERO, R. A. F. *Sistema Fuzzy para Tomada de Decisão de Sistemas Multiagentes* Instituto de Ciências Matemáticas e de Computação (USP), São Paulo

[24] MATLAB, "Fuzzy logic toolbox," *http://www.mathworks.com.au/products/fuzzy-logic/index.html*, acessado em Janeiro, 2017.

[25] KHATIB, O. *Real-time obstacle avoidance for manipulators and mobile robots.* The International Journal of Robotics Research 5 (1): 90–98, 1986.

[26] FARIA, G. *Uma Arquitetura de Controle Inteligente para Múltiplos Robôs*. PhD thesis, ICMC-USP São Carlos, 2006.

[27] CONNOLLY, C. I. and GRUPEN, R. A. *On the application of harmonic functions to robotics*. Journal of Robotic Systems , 10, 913-946, 1993.

[28] EVANS, L. *Partial Differential Equations*, 2 ed. American Mathematical Society [S.I], 2010.

[29] AMES, W. F. *Numerical Methods for Partial Differential Equations*. Thomas Nelson and Sons Ltd., 1969.

[30] SMITH, G. D. *Numerical Solution of partial differential equations: finite difference methods*. Oxford University, 1992.

[31] PRESTES, E. *Navegação Explanatória Baseada em Problemas de Valores de Contorno.* PhD thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 2003.

[32] DA SILVA, M. O. *Campos Potenciais Modificados Aplicados ao Controle de Múltiplos Robôs*. Master's thesis, ICMC-USP São Carlos, 2011.

APÊNDICE A – Fotos dos Testes para Criação da Base de Regras

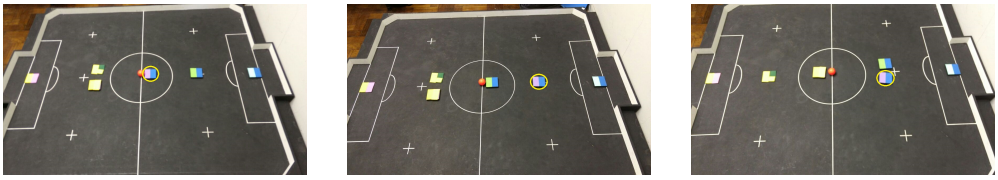

Figura 46 – Fotos 1, 2 e 3

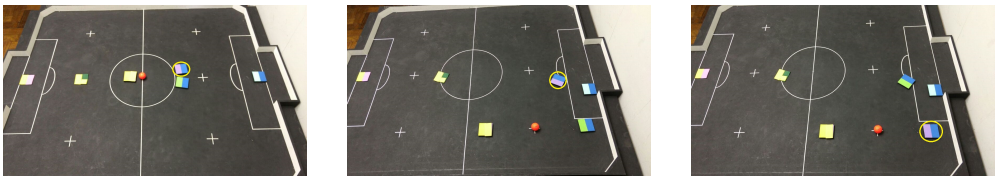

Figura 47 – Fotos 4, 5 e 6

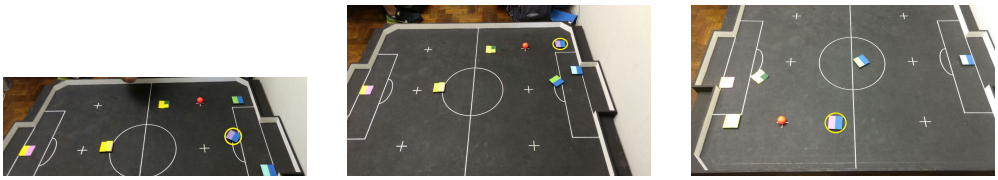

Figura 48 – Fotos 7, 8 e 9

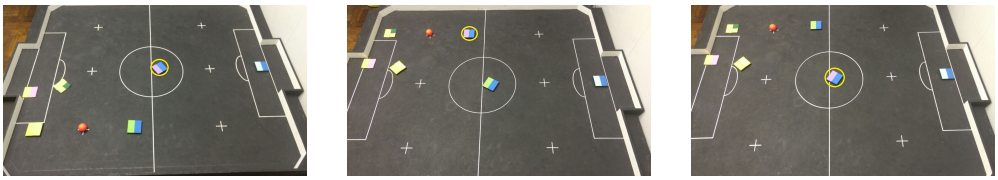

Figura 49 – Fotos 10, 11 e 12

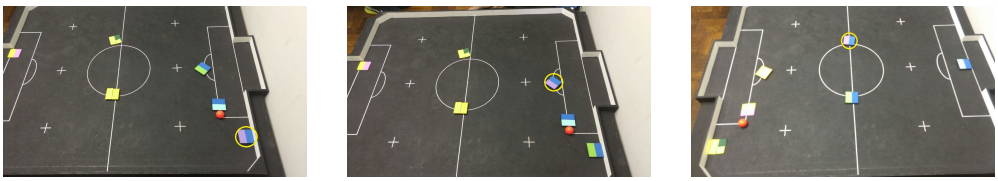

Figura 50 – Fotos 13, 14 e 15

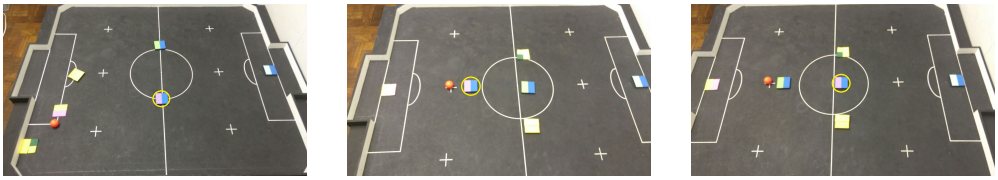

Figura 51 – Fotos 16, 17 e 18

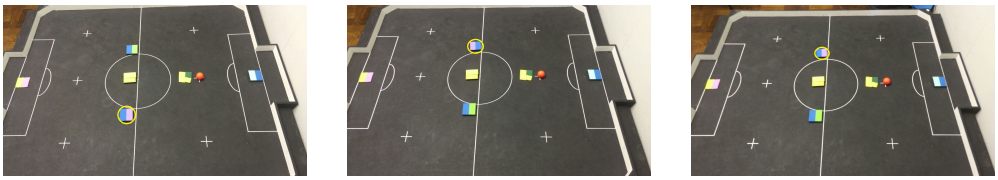

Figura 52 – Fotos 19, 20 e 21

Figura 53 – Fotos 22, 23 e 24

Figura 54 – Fotos 25, 26 e 27

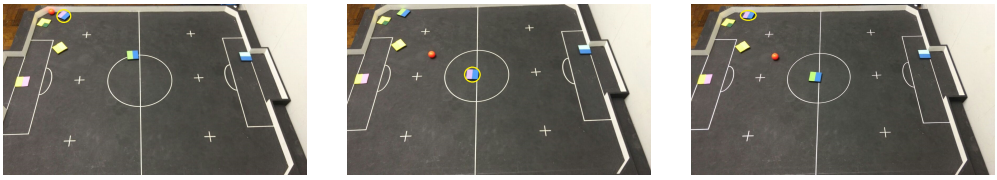

Figura 55 – Fotos 28, 29 e 30

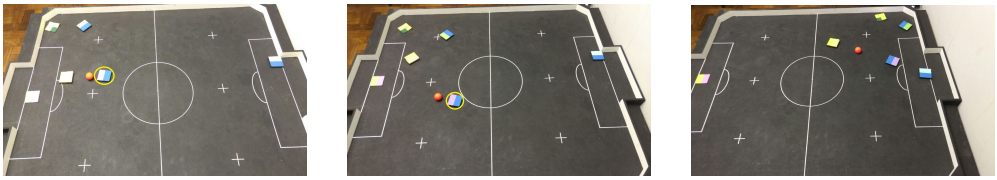

Figura 56 – Fotos 31, 32 e 33

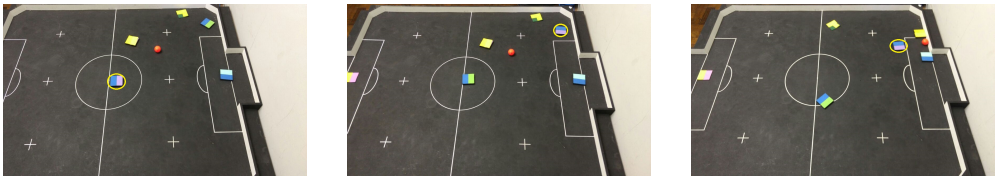

Figura 57 – Fotos 34, 35 e 36

Figura 58 – Fotos 37, 38 e 39

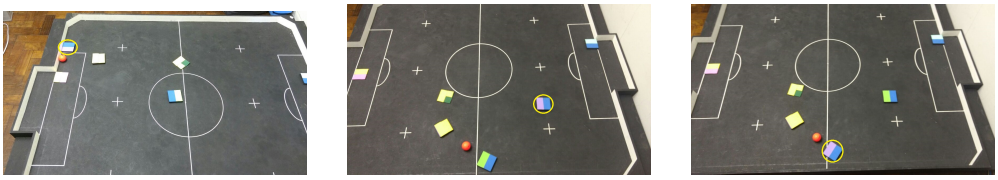

Figura 59 – Fotos 40, 41 e 42

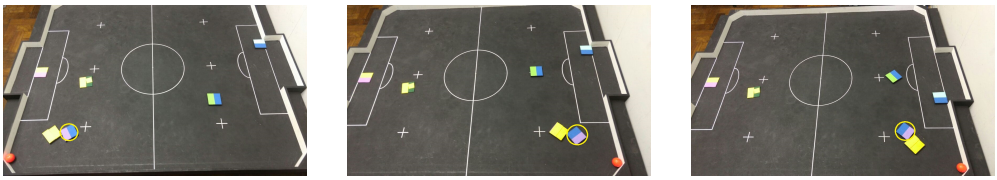

Figura 60 – Fotos 43, 44 e 45

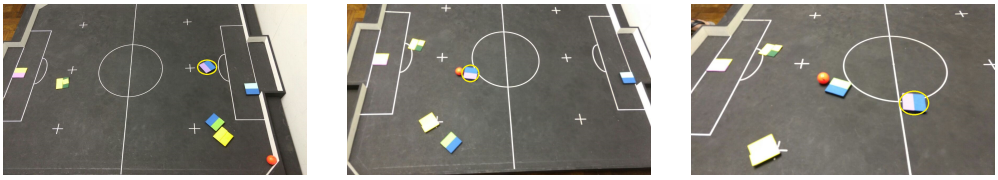

Figura 61 – Fotos 46, 47 e 48

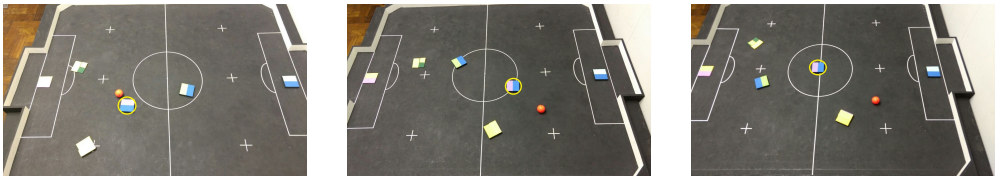

Figura 62 – Fotos 49, 50 e 51

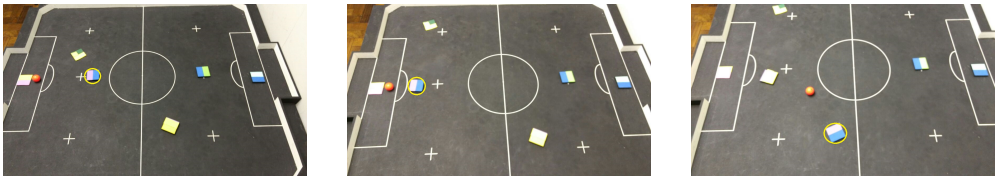

Figura 63 – Fotos 52, 53 e 54

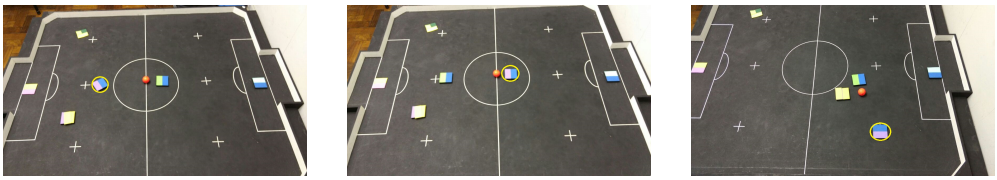

Figura 64 – Fotos 55, 56 e 57

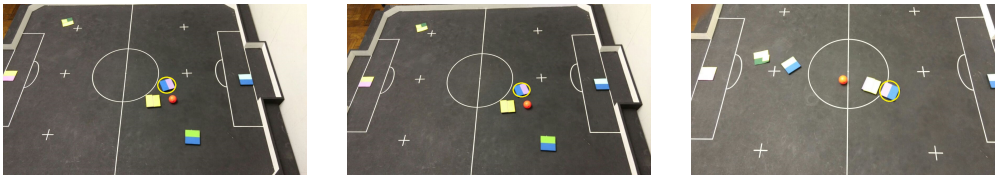

Figura 65 – Fotos 58, 59 e 60

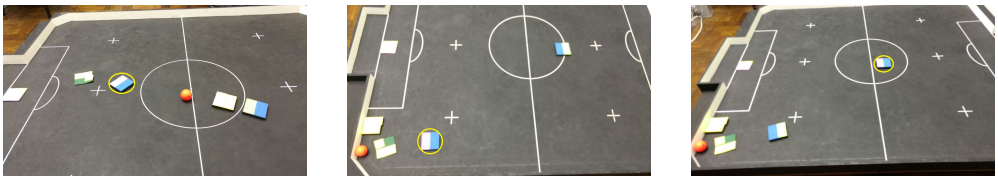

Figura 66 – Fotos 61, 62 e 63

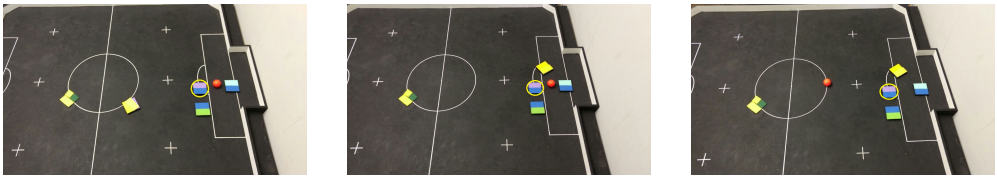

Figura 67 – Fotos 64, 65 e 66

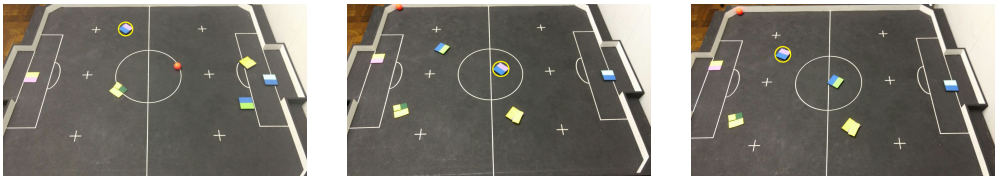

Figura 68 – Fotos 67, 68 e 69

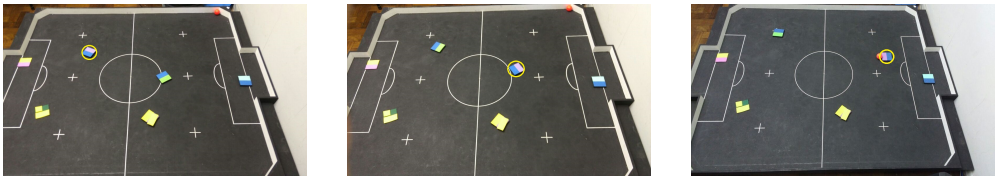

Figura 69 – Fotos 70, 71 e 72

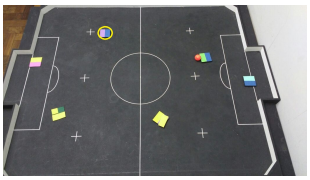

Figura 70 – Foto 73

APÊNDICE B – Código em C++ do Sistema Real VSSS - arquivo fuzzy.cpp

```
1  #include <iostream>
2  #include <iomanip>
3  #include <cmath>
4  #include "fuzzy.h"
5  #include "utils.h"
6  #include "robot.h"
7
8  using namespace std;
9
10 double parameters[][3] = {{-0.3, 0, 0.3},{0.111, 0.333,
       0.555},{0.444, 0.666, 0.888},{0.7, 1 , 1.3}};
11 Point2d eixo_x(1.0,0.0);
12
13
14 //no construtor define as entradas do sistema FD, FC e FA
15 Fuzzy::Fuzzy(){
16
17 stop = true;
18 duniverse_initialized = false;
19 output = 0.0;
20 enemy_pos_grid = pVector(3);
21 team_pos_grid = pVector(3);
22
23 pertinencia.resize(3);
24 D.resize(64);
25 y_output.resize(101);
26 y_baixo.resize(101);
27 y_medio1.resize(101);
28 y_medio2.resize(101);
29 y_alto.resize(101);
30 d_universe.resize(101);
31 decisao_robo.resize(3);
32 input.resize(3);
33 mi_output.resize(3);
34
35 mi = dMatrix(3, vector<double>(4, 0.0));
36 limite = dMatrix(64, vector<double>(101, 0.0));
37
38
```

```
39 }
40
41 Fuzzy::~Fuzzy(){
42
43 }
44
45 bool Fuzzy::isStopped() const
46 {
47 return this->stop;
48 }
49
50 void Fuzzy::Play(){
51 if(isStopped())
52 stop = false;
53 start();
54 }
55
56 void Fuzzy::Stop(){
57 stop = true;
58 }
59
60 bool Fuzzy::is_running(){
61 return isRunning();
62 }
63
64 void Fuzzy::msleep(int ms){
65 struct timespec ts = {ms / 1000, (ms % 1000) * 1000 * 1000};
66 nanosleep(&ts, NULL);
67 }
68
69 void Fuzzy::run(){
70 if(!duniverse_initialized){
71 init_duniverse();
72 init_funcao_pertinencia();
73 duniverse_initialized = true;
74 }
75 if(ball_pos.x > 0 && ball_pos.y > 0){
76 //Pro primeiro robo
77 calcula_input(selec_robot.r1);
78 fuzzification();
79 decisao_robo[0] = defuzzification();
80 double gandalf = input[0] + input[1] + input[2];
```

```
81
82 //Pro segundo robo
83 calcula_input(selec_robot.r2);
84 fuzzification();
85 decisao_robo[1] = defuzzification();
86 double presto = input[0] + input[1] + input[2];
87
88 if (decisao_robo[0] >= 2 && decisao_robo[1] >= 2){//TIRA O
      CONFRONTO POR ATK
89 if(fabs(selec_robot.r1.get_pos().x - centroid_def.x) > fabs(
      selec_robot.r2.get_pos().x - centroid_def.x)){
90 decisao_robo[1] = 1;
91 }
92 else {
93 decisao_robo[0] = 1;
94 }
95 }
96 else{
97 //tratar aqui
98 }
99 if (decisao_robo[0] <= 1 && decisao_robo[1] <= 1){//TIRA
      CONFRONTO DEFESA
100 if(fabs(selec_robot.r1.get_pos().x - centroid_def.x) > fabs(
      selec_robot.r2.get_pos().x - centroid_def.x)){
101 decisao_robo[0] = 2;
102 }
103 else {
104 decisao_robo[1] = 2;
105 }
106 }
107 else{
108 //tratar aqui
109 }
110 decisao_robo[2] = 4;
111
112 selec_robot.r1.set_flag_fuzzy(decisao_robo[0], centroid_atk,
      centroid_def, ball_pos);
113 selec_robot.r2.set_flag_fuzzy(decisao_robo[1], centroid_atk,
      centroid_def, ball_pos);
114 selec_robot.r3.set_flag_fuzzy(decisao_robo[2], centroid_atk,
      centroid_def, ball_pos);
115
```

```
116 }else{
117 //tratar bola aqui
118 }
119 emit emitRobots(selec_robot);
120 flag_finish_fuzzy = true;
121 }
122
123 void Fuzzy::set_to_select(Robot r1, Robot r2, Robot r3){
124 selec_robot.r1 = r1;
125 selec_robot.r2 = r2;
126 selec_robot.r3 = r3;
127 }
128
129 void Fuzzy::calcula_input(Robot r){
130
131 double aux1,aux2;
132
133 Point2d robot_pos = r.get_pos();
134 //Point2d robot2_pos = selec_robot.r2.get_pos();
135
136 double angle = r.get_angle();
137 if(angle != angle){
138 angle = r.get_last_angle();
139 if(angle != angle) angle = 0;
140 }
141 //double angle2 = selec_robot.r2.get_angle();
142
143 //Calculo do FD - distGOLDEF ate nosso player e distGOLATK ate
      nosso player
144 input[0] = pow(2.7183,-0.6931*(euclidean_dist(centroid_atk,
      robot_pos)/euclidean_dist(centroid_def,robot_pos)));
145 input[0] = (round(input[0]*100))/100;
146 //cout << "FD: "<< input[0] << endl;
147
148 //Calculo FC - distBallTeam, distBallEnemy
149
150 //Define Inimigo mais prox
151
152 Point2d enemy_prox;
153 if ((euclidean_dist(ball_pos,enemy_pos[0]) <= euclidean_dist(
      ball_pos,enemy_pos[1])) && (euclidean_dist(ball_pos,enemy_pos
      [0]) <= euclidean_dist(ball_pos,enemy_pos[2])))
```

```
154 enemy_prox = enemy_pos[0];
155 else if (euclidean_dist(ball_pos,enemy_pos[1]) <= euclidean_dist(
      ball_pos,enemy_pos[2]))
156 enemy_prox = enemy_pos[1];
157 else
158 enemy_prox = enemy_pos[2];
159
160 /*cout << "Distancia inimigo1: "<<euclidean_dist(ball_pos,
      enemy_pos[0]) << endl;
161 cout << "Distancia inimigo2: "<<euclidean_dist(ball_pos,enemy_pos
      [1]) << endl;
162 cout << "Distancia inimigo3 "<<euclidean_dist(ball_pos,enemy_pos
      [2]) << endl;
163
164 cout << "Inimigo mais prox bola: " << endl;
165 cout << "Em x: " << enemy_prox.x << " Em y: " << enemy_prox.y <<
      endl;*/
166
167 input[1] =pow(2.7183,-0.6931*(euclidean_dist(ball_pos,robot_pos)/
      euclidean_dist(ball_pos,enemy_prox)));
168 input[1] = (round(input[1]*100))/100;
169 //cout << "FC: "<< input[1] << endl;
170
171
172 //Calculo FA - AngBallAliado e AngAtkAliado
173
174 //Corrige Posicionamento
175 ball_pos.y = -ball_pos.y;
176 robot_pos.y = -robot_pos.y;
177 centroid_atk.y=-centroid_atk.y;
178
179 //Calcula angulo entre robo e bola
180 Point2d vec_ball_robot = ball_pos-robot_pos;
181 double ang_vec_ball_eixox = angle_two_points(vec_ball_robot,
      eixo_x);
182
183 //Corrige o angulo
184 if (vec_ball_robot.y < 0)
185 ang_vec_ball_eixox = -ang_vec_ball_eixox;
186
187 double ang_ball_robot = ang_vec_ball_eixox - angle;
188
```

```
189 //Calcula angulo entre robo e gol adversario
190 Point2d vec_atk_robot = centroid_atk-robot_pos;
191 double ang_vec_atk_eixox = angle_two_points(vec_atk_robot,eixo_x)
      ;
192
193 //Corrige o angulo
194 if (vec_atk_robot.y < 0)
195 ang_vec_atk_eixox = -ang_vec_atk_eixox;
196
197 double ang_atk_robot = ang_vec_atk_eixox - angle;
198
199 //ajusta angulos para menores que 180 e maiores que -180
200 if (ang_ball_robot>180) ang_ball_robot = ang_ball_robot - 360;
201 else if (ang_ball_robot<-180) ang_ball_robot = ang_ball_robot +
      360;
202 if (ang_atk_robot>180) ang_atk_robot = ang_atk_robot - 360;
203 else if (ang_atk_robot<-180) ang_atk_robot = ang_atk_robot + 360;
204
205 //cout << "Angulo entre bola e robo: "<< ang_ball_robot << endl;
206 //cout << "Angulo vetor bola robo: "<< ang_vec_ball_eixox << endl
      ;
207 //cout << "Angulo entre robo e atk " << ang_atk_robot << endl;
208 //cout << "Angulo vetor atk robo " << ang_vec_atk_eixox << endl;
209 //cout << "Angulo de orientacao robo: "<< angle << endl;
210
211 if (ang_ball_robot <= 90 && ang_ball_robot >= -90)
212 aux1 = (90 - fabs(ang_ball_robot))/90;
213 else
214 aux1 = (-90 + fabs(ang_ball_robot))/90;
215 if (ang_atk_robot <= 90 && ang_atk_robot >= -90)
216 aux2 = (90 - fabs(ang_atk_robot))/90;
217 else
218 aux2 = (-90 + fabs(ang_atk_robot))/90;
219
220 input[2]= 0.7*aux1+0.3*aux2;
221 input[2] = (round(input[2]*100))/100;
222 //cout << "FA: "<< input[2] << endl;
223
224 ball_pos.y = -ball_pos.y;
225 robot_pos.y = -robot_pos.y;
226 centroid_atk.y=-centroid_atk.y;
227 }
```

```
228
229 void Fuzzy::fuzzification(){
230 int i=0, j=0, k=0, cont = 0, aux1;
231 double aux2 = 0, aux3;
232
233
234 pertinencia[0] = input[0]/0.01;
235 pertinencia[1] = input[1]/0.01;
236 pertinencia[2] = input[2]/0.01;
237
238 //cout<<"\n Entradas:"<< endl;
239 //cout<<input[0]<<" "<<input[1]<<" "<<input[2]<<endl;
240
241 for(i=0;i<3;i++)
242 {
243 for(j=0;j<4;j++)
244 {
245 aux1 = pertinencia[i];
246 if(j == 0)
247 {
248 mi[i][j] = y_baixo[aux1];
249 }
250 if(j == 1)
251 {
252 mi[i][j] = y_medio1[aux1];
253 }
254 if(j == 2)
255 {
256 mi[i][j] = y_medio2[aux1];
257 }
258 if(j == 3)
259 {
260 mi[i][j] = y_alto[aux1];
261 }
262 }
263 }
264
265 for(i=0;i<4;i++)
266 {
267 for(j=0;j<4;j++)
268 {
269 for(k=0;k<4;k++)
```

```
{
aux3 = min_function(mi[0][i],mi[1][j]);
aux3 = min_function(aux3,mi[2][k]);
D[cont] = aux3;
cont++;
}
}

}

for(i=0;i<cont;i++)
{
if((i >= 0 && i <= 1) || (i >= 4 && i <= 5) || (i == 16) || (i ==
    22))
{
for(j=0;j<=100;j++)
{
limite[i][j] = min_function(D[i],y_baixo[j]);
}
}
else if((i >= 2 && i <= 3) || (i >= 6 && i <= 10) || (i >= 12 &&
   i <= 13) || (i >= 17 && i <= 20) || (i >= 24 && i <= 25) || (i
   >= 28 && i <= 29) || (i >= 32 && i <= 34) || (i >= 36 && i <=
   37) || (i >= 48 && i <= 49) || (i >= 52 && i <= 53))
{
for(j=0;j<=100;j++)
{
limite[i][j] = min_function(D[i],y_medio1[j]);
}
}
else if((i == 11) || (i >= 14 && i <= 15) || (i == 21) || (i ==
   23) || (i >= 26 && i <= 27) || (i >= 30 && i <= 31) || (i ==
   35) || (i == 38) || (i >= 40 && i <= 42) || (i >= 44 && i <=
   45) || (i >= 50 && i <= 51) || (i == 54) || (i >= 56 && i <=
   57) || (i >= 60 && i <= 61))
{
for(j=0;j<=100;j++)
{
limite[i][j] = min_function(D[i],y_medio2[j]);
}
}
```

```
303 else //if((i == 39) || (i == 43) || (i >= 46 && i <= 47) || (i ==
        55) || (i >= 58 && i <= 59) || (i >= 62 && i <= 63))
304 {
305 for(j=0;j<=100;j++)
306 {
307 limite[i][j] = min_function(D[i],y_alto[j]);
308 }
309 }
310
311 }
312
313 for(i=0;i<=100;i++)
314 {
315 for(k=0;k<cont;k++)
316 {
317 aux2 = max_function(limite[k][i],aux2);
318 }
319 y_output[i] = aux2;
320 aux2 = 0;
321 }
322 }
323
324 int Fuzzy::defuzzification(){
325 double sum1 = 0,sum2 = 0, aux2;
326 int i,j,aux1;
327 for(i=0;i<=100;i++)
328 {
329 sum1 = sum1 + d_universe[i]*y_output[i];
330 sum2 = sum2 + y_output[i];
331 }
332 output = sum1/sum2;
333 //cout << "Saida Fuzzy: "<< output << endl;
334
335 //Calcular a pertinencia da saida
336 aux1 = output/0.01;
337
338 for(j=0;j<4;j++)
339 {
340 if(j == 0)
341 {
342 mi_output[j] = y_baixo[aux1];
343 }
```

```
344 if(j == 1)
345 {
346 mi_output[j] = y_medio1[aux1];
347 }
348 if(j == 2)
349 {
350 mi_output[j] = y_medio2[aux1];
351 }
352 if(j == 3)
353 {
354 mi_output[j] = y_alto[aux1];
355 }
356 }
357
358 aux2 = max_function(mi_output[0],mi_output[1]);
359 aux2 = max_function(aux2,mi_output[2]);
360 aux2 = max_function(aux2,mi_output[3]);
361
362
363 if (aux2 == mi_output[0])
364 return 0;
365 else if (aux2 == mi_output[1])
366 return 1;
367 else if (aux2 == mi_output[2])
368 return 2;
369 else
370 return 3;
371
372 }
373
374 bool Fuzzy::get_flag_finish(){
375 return this->flag_finish_fuzzy;
376 }
377
378 void Fuzzy::zera_flag_finish(){
379 flag_finish_fuzzy = false;
380 }
381
382
383 double Fuzzy::min_function(double p, double q){
384 if(p <= q)
385 {
```

```
386 return p;
387 }
388 else
389 return q;
390 }
391
392 double Fuzzy::max_function(double p, double q){
393 if(p >= q)
394 {
395 return p;
396 }
397 else
398 return q;
399 }
400
401 void Fuzzy::init_duniverse(){
402 int i;
403 double aux = 0.01;
404 for(i=0;i<=100;i++)
405 {
406 d_universe[i] = i*aux;
407 }
408
409 }
410
411 void Fuzzy::init_funcao_pertinencia(){
412 int i;
413 for(i=0;i<=100;i++)
414 {
415 if(d_universe[i] < parameters[0][0] || d_universe[i] > parameters
      [0][2])
416 {
417 y_baixo[i] = 0;
418 }
419 else if(d_universe[i] < parameters[0][1])
420 {
421 y_baixo[i] = (d_universe[i] - parameters[0][0])/(parameters[0][1]
       - parameters[0][0]);
422 }
423 else if(d_universe[i] >= parameters[0][1])
424 {
```

```
425 y_baixo[i] = (d_universe[i] - parameters[0][2])/(parameters[0][1]
        - parameters[0][2]);
426 }
427 }
428 for(i=0;i<=100;i++)
429 {
430 if(d_universe[i] < parameters[1][0] || d_universe[i] > parameters
       [1][2])
431 {
432 y_medio1[i] = 0;
433 }
434 else if(d_universe[i] < parameters[1][1])
435 {
436 y_medio1[i] = (d_universe[i] - parameters[1][0])/(parameters
       [1][1] - parameters[1][0]);
437 }
438 else if(d_universe[i] >= parameters[1][1])
439 {
440 y_medio1[i] = (d_universe[i] - parameters[1][2])/(parameters
       [1][1] - parameters[1][2]);
441 }
442 }
443 for(i=0;i<=100;i++)
444 {
445 if(d_universe[i] < parameters[2][0] || d_universe[i] > parameters
       [2][2])
446 {
447 y_medio2[i] = 0;
448 }
449 else if(d_universe[i] < parameters[2][1])
450 {
451 y_medio2[i] = (d_universe[i] - parameters[2][0])/(parameters
       [2][1] - parameters[2][0]);
452 }
453 else if(d_universe[i] >= parameters[2][1])
454 {
455 y_medio2[i] = (d_universe[i] - parameters[2][2])/(parameters
       [2][1] - parameters[2][2]);
456 }
457 }
458 for(i=0;i<=100;i++)
459 {
```

```
460 if(d_universe[i] < parameters[3][0] || d_universe[i] > parameters
       [3][2])
461 {
462 y_alto[i] = 0;
463 }
464 else if(d_universe[i] < parameters[3][1])
465 {
466 y_alto[i] = (d_universe[i] - parameters[3][0])/(parameters[3][1]
       - parameters[3][0]);
467 }
468 else if(d_universe[i] >= parameters[3][1])
469 {
470 y_alto[i] = (d_universe[i] - parameters[3][2])/(parameters[3][1]
       - parameters[3][2]);
471 }
472 }
473 }
474
475 Point Fuzzy::convert_C_to_G(Point2d coord){
476 Point i;
477
478 coord.x = int(coord.x) + 5;
479 coord.y = int(coord.y) + 5;
480
481 if(coord.x / 5 != 35){
482 i.x = coord.x / 5;
483 }else{
484 i.x = coord.x / 5 - 1;
485 }
486
487 if(coord.y / 5 != 27){
488 i.y = coord.y / 5;
489 }else{
490 i.y = coord.y / 5 - 1;
491 }
492 return i;
493 }
494
495 void Fuzzy::set_enemy_pos(p2dVector enemy_pos){
496 this->enemy_pos = enemy_pos;
497 }
498
```

```
499 void Fuzzy::set_ball_pos(Point2d ball_pos){
500 this->ball_pos = ball_pos;
501 }
502
503 void Fuzzy::set_centroid_atk(Point2d centroid_atk){
504 this->centroid_atk = centroid_atk;
505 }
506
507 void Fuzzy::set_centroid_def(Point2d centroid_def){
508 this->centroid_def = centroid_def;
509 }
```

APÊNDICE C – Código em C++ do Método CPH

Arquivo *main.cpp*

```
1  //main.cpp
2
3  #include<iostream>
4  #include<fstream>
5  #include<ctime>
6
7  #include"Environment.h"
8  using namespace std;
9
10 int main()
11 {
12 srand(time(0));
13 int cont=0;
14 Environment obj(5,5);
15 obj.initGrid();
16 obj.setPotencial(1+(rand()%18),1+(rand()%18),0);
17 obj.setDirecoes();
18 obj.printGrid();
19 do
20 {
21 cont++;
22 }while(obj.iterador()>=0.000001);
23 cout<<"Numero de iteracoes igual a "<<cont<<"\n";;
24 obj.setDirecoes();
25 obj.printGrid();
26 getchar();
27 return 0;
28 }
```

Arquivo *Environment.h*

```
1  //Environment.h
2
3  #ifndef ENVIRONMENT_H
4  #define ENVIRONMENT_H
5  #include<string>
6
7
8  class Environment
9  {
10 public:
11 Environment(double,double);
12 double iterador();
13 void setPotencial(int, int,double);
14 void setDirecoes();
15
16 double getVizinho(int, int, int);//insere i, j e qual (1,2,3 ou
       4)
17 double getPotencial(int, int);//insere i e j
18 bool getOcupado(int, int);// i, j
19
20 void initGrid();
21 void printGrid();
22
23 protected:
24 double pGrid[20][20];
25 float tGrid[20][20];
26 double dx;
27 double dy;
28 };
29 #endif
```

Arquivo *Environment.cpp*

```
1  //Environment.cpp
2
3  #include "Environment.h"
4  #include <iostream>
5  #include<fstream>
6  #include<iomanip>
7  #include "math.h"
8  using std::cout;
9  using std::cin;
10 using std::endl;
11 using std::setw;
12
13 //inicializar construtor
14 Environment::Environment(double a,double b)
15 {
16 dx=a;
17 dy=b;
18 cout<<"Ambiente Criado"<<"\n";
19 }
20
21 //Funcao iterador
22 double Environment::iterador()
23 {
24 double erro = 0;
25 double top;
26 double botton;
27 double left;
28 double right;
29 double newP, oldP;
30 for(int i=0;i<20;i++)
31 {
32 for(int j=0;j<20;j++)
33 {
34 if(getOcupado(i,j))
35 {
36 oldP = getPotencial(i,j);
37 top = getVizinho(i,j,0);
38 botton = getVizinho(i,j,1);
39 left = getVizinho(i,j,2);
40 right = getVizinho(i,j,3);
41 newP = 0.25*(top+botton+left+right);
```

```
42 erro = erro + pow((newP - oldP),2);
43 setPotencial(i,j,newP);
44 }
45 }
46 }
47 return erro;
48 }
49
50 void Environment::setDirecoes()
51 {
52 for(int i=0;i<20;i++)
53 {
54 for(int j=0;j<20;j++)
55 {
56 if(getOcupado(i,j))
57 {
58 tGrid[i][j] = -atan2(getVizinho(i,j,0)-getVizinho(i,j,1),
      getVizinho(i,j,2)-getVizinho(i,j,3))*(180/3.1415);
59 }
60 else
61 {
62 tGrid[i][j] = getPotencial(i,j)*1000;
63 }
64 }
65 }
66 }
67
68 double Environment::getVizinho(int i,int j,int k)
69 {
70 double top,botton,left,right,aux;
71 if(i==0)
72 {
73 top=1;
74 botton=pGrid[i+1][j];
75 }
76 else if(i+1==20)
77 {
78 top=pGrid[i-1][j];
79 botton=1;
80 }
81 else
82 {
```

```
83 top=pGrid[i-1][j];
84 botton=pGrid[i+1][j];
85 }
86 if(j==0)
87 {
88 left=1;
89 right=pGrid[i][j+1];
90 }
91 else if(j+1==20)
92 {
93 left=pGrid[i][j-1];
94 right=1;
95 }
96 else
97 {
98 left=pGrid[i][j-1];
99 right=pGrid[i][j+1];
100 }
101 if(k==0)
102 {
103 aux=top;
104 }
105 else if(k==1)
106 {
107 aux=botton;
108 }
109 else if(k==2)
110 {
111 aux=left;
112 }
113 else if(k==3)
114 {
115 aux=right;
116 }
117 return aux;
118 }
119
120 void Environment::initGrid()
121 {
122 int i,j = 0;
123 for(i=0;i<20;i++)
124 {
```

```
125 for(j=0;j<20;j++)
126 {
127 if( i!=0 && i!=19 &&j!=0 && j!=19)
128 {
129 pGrid[i][j]=0.5;
130 }
131 else
132 {
133 pGrid[i][j]=1;
134 }
135 }
136 }
137 }
138
139 void Environment::setPotencial(int i,int j,double valor)
140 {
141 pGrid[i][j]=valor;
142 }
143
144 double Environment::getPotencial(int i,int j)
145 {
146 return pGrid[i][j];
147 }
148
149 bool Environment::getOcupado(int i,int j)
150 {
151 if(getPotencial(i,j)==1 || getPotencial(i,j)==0)
152 {
153 return 0;
154 }
155 else
156 {
157 return 1;
158 }
159 }
160
161 void Environment::printGrid()
162 {
163 cout<<"\n\nGrid dos potenciais\n";
164 for(int i=0;i<20;i++)
165 {
166 for(int j=0;j<20;j++)
```

```
167 {
168 std::cout<<std::setw(8);
169 cout<<pGrid[i][j]<<" ";
170 }
171 cout<<"\n";
172 }
173 cout<<"\n\nGrid dos angulos\n";
174 for(int i=0;i<20;i++)
175 {
176 for(int j=0;j<20;j++)
177 {
178 std::cout<<std::setw(4);
179 cout<<(int)tGrid[i][j]<<" ";
180 }
181 cout<<"\n";
182 }
183 }
```

APÊNDICE D – Código em C++ do Método CPO

Arquivo *main.cpp*

```
1  //main.cpp
2
3  #include<iostream>
4  #include<fstream>
5  #include<ctime>
6  #include <cstdio>
7
8  #include"Environment.h"
9  using namespace std;
10
11 int main()
12 {
13 std::clock_t start;
14 double tempo;
15 start=std::clock();
16
17
18 srand(time(0));
19 int cont=0;
20 Environment obj(5,5);
21 obj.initGrid();
22 obj.setPotencial(15,5,0);//seta a meta
23 //for(int y=7;y<=14;y++)
24 //{
25 //    obj.setPotencial(y,7,1);
26 //}
27
28 obj.setDirecoes();
29 obj.printGrid();
30 do
31 {
32 cont++;
33 }while(obj.iterador()>=0.000001);
34 cout<<"Numero de iteracoes igual a "<<cont<<"\n";;
35 obj.setDirecoes();
36 obj.printGrid();
37
38
```

```
39 tempo=(std::clock()-start)/(double) CLOCKS_PER_SEC;
40 cout<<"Tempo decorrido: "<<tempo<<" segundos";
41 return 0;
42 }
```

Arquivo *Environment.h*

```
1  //Environment.h
2
3  #ifndef ENVIRONMENT_H
4  #define ENVIRONMENT_H
5  #include<string>
6
7
8  class Environment
9  {
10 public:
11 Environment(double,double);
12 double iterador();
13 void setPotencial(int, int,double);
14 void setDirecoes();
15
16 double getVizinho(int, int, int);//insere i, j e qual (1,2,3 ou
       4)
17 double getPotencial(int, int);//insere i e j
18 bool getOcupado(int, int);// i, j
19
20 void initGrid();
21 void printGrid();
22
23 protected:
24 double pGrid[20][20];
25 float tGrid[20][20];
26 double dx;
27 double dy;
28 };
29 #endif
```

Arquivo *Environment.cpp*

```
1  //Environment.cpp
2
3  #include "Environment.h"
4  #include <iostream>
5  #include<fstream>
6  #include<iomanip>
7  #include "math.h"
8  using std::cout;
9  using std::cin;
10 using std::endl;
11 using std::setw;
12
13 //inicializar construtor
14 Environment::Environment(double a,double b)
15 {
16 dx=a;
17 dy=b;
18 cout<<"Ambiente Criado"<<"\n";
19 }
20
21 //Funcao iterador
22 double Environment::iterador()
23 {
24 double erro = 0;
25 double top;
26 double botton;
27 double left;
28 double right;
29 double newP, oldP;
30
31 double h=dx/dy;
32 double e=0.5;//taxa de influencia
33 double angulo=135;
34 double vy=sin(angulo*3.1415/180),vx=cos(angulo*3.1415/180),lambda
      =e*h/2;
35
36
37
38 for(int i=0;i<20;i++)
39 {
40 for(int j=0;j<20;j++)
```

```
41 {
42 if(getOcupado(i,j))
43 {
44 oldP = getPotencial(i,j);
45 top = getVizinho(i,j,0);
46 botton = getVizinho(i,j,1);
47 left = getVizinho(i,j,2);
48 right = getVizinho(i,j,3);
49 newP = 0.25*((1+lambda*vy)*top+(1-lambda*vy)*botton+(1+lambda*vx)
      *right+(1-lambda*vx)*left);
50 //newP=newP+0.8*(newP-oldP);
51 erro = erro + pow((newP - oldP),2);
52 setPotencial(i,j,newP);
53 }
54 }
55 }
56 return erro;
57 }
58
59 void Environment::setDirecoes()
60 {
61 for(int i=0;i<20;i++)
62 {
63 for(int j=0;j<20;j++)
64 {
65 if(getOcupado(i,j))
66 {
67 tGrid[i][j] = -atan2(getVizinho(i,j,0)-getVizinho(i,j,1),
      getVizinho(i,j,2)-getVizinho(i,j,3))*(180/3.1415);
68 }
69 else
70 {
71 tGrid[i][j] = getPotencial(i,j)*1000;
72 }
73 }
74 }
75 }
76
77 double Environment::getVizinho(int i,int j,int k)
78 {
79 double top,botton,left,right,aux;
80 if(i==0)
```

```
81 {
82 top=1;
83 botton=pGrid[i+1][j];
84 }
85 else if(i+1==20)
86 {
87 top=pGrid[i-1][j];
88 botton=1;
89 }
90 else
91 {
92 top=pGrid[i-1][j];
93 botton=pGrid[i+1][j];
94 }
95 if(j==0)
96 {
97 left=1;
98 right=pGrid[i][j+1];
99 }
100 else if(j+1==20)
101 {
102 left=pGrid[i][j-1];
103 right=1;
104 }
105 else
106 {
107 left=pGrid[i][j-1];
108 right=pGrid[i][j+1];
109 }
110 if(k==0)
111 {
112 aux=top;
113 }
114 else if(k==1)
115 {
116 aux=botton;
117 }
118 else if(k==2)
119 {
120 aux=left;
121 }
122 else if(k==3)
```

```
123 {
124 aux=right;
125 }
126 return aux;
127 }
128
129 void Environment::initGrid()
130 {
131 int i,j = 0;
132 for(i=0;i<20;i++)
133 {
134 for(j=0;j<20;j++)
135 {
136 if( i!=0 && i!=19 &&j!=0 && j!=19)
137 {
138 pGrid[i][j]=0.9;
139 }
140 else
141 {
142 pGrid[i][j]=1;
143 }
144 }
145 }
146 }
147
148 void Environment::setPotencial(int i,int j,double valor)
149 {
150 pGrid[i][j]=valor;
151 }
152
153 double Environment::getPotencial(int i,int j)
154 {
155 return pGrid[i][j];
156 }
157
158 bool Environment::getOcupado(int i,int j)
159 {
160 if(getPotencial(i,j)==1 || getPotencial(i,j)==0)
161 {
162 return 0;
163 }
164 else
```

```
165 {
166 return 1;
167 }
168 }
169
170 void Environment::printGrid()
171 {
172 cout<<"\n\nGrid dos potenciais\n";
173 for(int i=0;i<20;i++)
174 {
175 for(int j=0;j<20;j++)
176 {
177 std::cout<<std::setw(8);
178 cout<<pGrid[i][j]<<" ";
179 }
180 cout<<"\n";
181 }
182 cout<<"\n\nGrid dos angulos\n";
183 for(int i=0;i<20;i++)
184 {
185 for(int j=0;j<20;j++)
186 {
187 std::cout<<std::setw(4);
188 cout<<(int)tGrid[i][j]<<" ";
189 }
190 cout<<"\n";
191 }
192 }
```