UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

GABRIEL ROMIO

DESENVOLVIMENTO DE SISTEMA DE NAVEGAÇÃO AUTÔNOMA PARA MODELO DE VEÍCULO NÁUTICO

BENTO GONÇALVES
2020

GABRIEL ROMIO

DESENVOLVIMENTO DE SISTEMA DE NAVEGAÇÃO AUTÔNOMA PARA MODELO DE VEÍCULO NÁUTICO

Trabalho de Conclusão de Curso apresentado no Curso de Engenharia Elétrica do Campus Universitário da Região dos Vinhedos, da Universidade de Caxias do Sul, como requisito parcial à obtenção do título de Engenheiro Eletricista

Orientador:
Prof. Dr. Alexandre Mesquita

BENTO GONÇALVES

2020

GABRIEL ROMIO

DESENVOLVIMENTO DE SISTEMA DE NAVEGAÇÃO AUTÔNOMA PARA MODELO DE VEÍCULO NÁUTICO

Trabalho de Conclusão de Curso apresentado no Curso de Engenharia Elétrica do Campus Universitário da Região dos Vinhedos, da Universidade de Caxias do Sul, como requisito parcial à obtenção do título de Engenheiro Eletricista

Orientador:
Prof. Dr. Alexandre Mesquita

Aprovado em: _____/_____/_____.

Banca Examinadora

_____

Orientador Prof. Dr. Alexandre Mesquita
Universidade de Caxias do Sul – RS

_____

Prof. Me. Patric Janner Marques
Universidade de Caxias do Sul – RS

_____

Prof. Me. Felipe Augusto Tondo
Universidade de Caxias do Sul – RS

AGRADECIMENTOS

Agradeço à minha família por todo o apoio e paciência ao longo do período da graduação.

Ao Prof. Alexandre Mesquita pela orientação e auxílio durante a elaboração deste projeto.

Aos demais professores e colegas que de alguma forma me auxiliaram e compartilharam seus conhecimentos.

RESUMO

Este trabalho consiste no desenvolvimento de um sistema de navegação autônoma aplicado a um modelo de veículo náutico, para que o mesmo possa se deslocar de forma independente entre dois pontos. O barco, que possui as dimensões de 450 mm (comprimento) x 140 mm (largura) x 180 mm (altura), dispõe de uma câmera estéreo com infravermelho, visando detectar obstáculos à sua movimentação e estimar a distância até uma eventual colisão. Com base nesses dados são determinados quais objetos devem ser contornados e quais não oferecem riscos à integridade do veículo. O seu deslocamento se dá por meio de direções de navegação emitidas por uma bússola eletrônica, a fim de mantê-lo, sempre que possível, em uma orientação predeterminada. A tomada de decisões e o controle do barco são realizados por um sistema de controle inteligente baseado em lógica *fuzzy*, que busca definir as melhores rotas até a posição de destino e planejar o desvio dos obstáculos. Os resultados foram analisados a partir de ensaios que consistiram na movimentação autônoma do barco em uma piscina externa de 6,5 x 3,1 m, onde foram inseridos obstáculos em diferentes disposições. O veículo apresentou resposta instantânea ao surgimento de quaisquer obstruções e se mostrou, em todos os testes realizados, capaz de evitá-las.

Palavras-chave: Veículos Autônomos. Visão Estéreo. Processamento de Imagens. Sistemas de Posicionamento. Lógica *Fuzzy*.

ABSTRACT

This work consists in the development of an autonomous navigation system applied to a nautical vehicle model, so that it can move independently between two places. The boat, which has the dimensions of 450 mm (length) x 140 mm (width) x 180 mm (height), has an infrared stereo camera, aiming to detect obstacles and stipulate the distance to a possible collision. Based in these data, it is determined which objects must be contoured and which ones do not present risks to the vehicle integrity. Its movement takes place by means of navigation coordinates emitted by an electronic compass, in order to keep the boat, whenever possible, in a predetermined direction. The boat decision making and control are performed by an intelligent control system based on fuzzy logic, which attempt to define the best routes to the target position and plan the deflect of the obstacles. The results were analyzed from tests that consisted of the autonomous movement of the boat in an outdoor pool of 6,5 x 3,1 m, where obstacles were inserted in different arrangements. The vehicle showed instantaneous response to the appearance of any obstructions and proved, in all tests performed, be able to avoid them.

Keywords: Autonomous Vehicles. Stereo Vision. Image Processing. Positioning Systems. Fuzzy Logic.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABS	*Anti-lock Braking System*
AIS	*Automatic Identification System*
ASV	*Autonomous Surface Vehicle*
AUV	*Autonomous Underwater Vehicle*
CAD	*Computer Aided Design*
CoA	Centro da Área
CoM	Centro do Máximo
DARPA	*Defense Advanced Research Projects Agency*
FPGA	*Field Programmable Gate Array*
GPS	*Global Positioning Systems*
GSM	*Global System for Mobile Communications*
I^2C	*Inter-Integrated Circuit*
JPEG	*Joint Photographic Experts Group*
LED	*Light-Emitting Diode*
LIDAR	*Light Detection and Ranging*
MBR	*Maritime Broadband Radio*
MoM	Média dos Máximos
ONR	*Office of Naval Research*
OpenCV	*Open Computer Vision*
PWM	*Pulse Width Modulation*
RADAR	*Radio Detection and Ranging*
RNA	Rede Neural Artificial
SQUID	*Superconducting Quantum Interference Device*
SVA	Sistema de Visão Artificial
UDP	*User Datagram Protocol*
VANT	Veículo Aéreo Não Tripulado

SUMÁRIO

1 INTRODUÇÃO

Veículos autônomos consistem em meios de transporte capazes de deslocar pessoas ou objetos até um destino desejado de forma independente e sem a necessidade de intervenção humana. Para isso podem contar com diversos sensores para determinar a sua posição, desviar de obstáculos e tomar decisões da melhor forma possível. Entre os benefícios de um veículo autônomo em relação aos tradicionais destacam-se a inibição de acidentes causados por erros humanos, a possibilidade de deslocamento de deficientes físicos ou visuais sem o auxílio de terceiros, a otimização do controle dos recursos veiculares, entre outros. (UDACITY, 2018)

Apesar das diversas pesquisas relacionadas aos veículos autônomos que se destinam aos carros, algumas empresas têm se dedicado à elaboração de outros meios de transportes, como é o caso da Boeing com aeronaves e espaçonaves não tripuladas ou operadas remotamente (BOEING, 2020) e da Volvo com caminhões autônomos para transporte de cargas (VOLVO, 2020). Outras pesquisas envolvem Veículos Aéreos Não Tripulados (VANTs) que, com um crescente mercado nos últimos anos devido à sua versatilidade, têm sido empregados nas mais diversas áreas, seja para monitoração de recursos agrícolas, gerenciamento de queimadas e poluição, pesquisas ambientais, vigilância ou até mesmo para missões em áreas de risco ou de difícil acesso (FURTADO et al., 2008).

Uma área de grande potencial ainda pouco explorada compreende o desenvolvimento de veículos navais autônomos. Atualmente grande parte dos projetos que a abrange encontram-se em fases de testes, como o navio de carga Yara Birkeland, em desenvolvimento pela empresa de fertilizantes norueguesa Yara (YARA, 2020). Além disso, há pesquisas em andamento ao redor do mundo visando transporte de pessoas e mercadorias, controle de pontes móveis e coleta de dados aquáticos de forma autônoma. Por outro lado, um exemplo de veículo naval autônomo que já opera com sucesso é o antissubmarino Sea Hunter, elaborado pelo departamento de defesa americano para fins militares (DARPA, 2018).

O presente trabalho tem como proposta o desenvolvimento de um modelo de veículo náutico autônomo. Contando com um sistema de posicionamento baseado em sinais emitidos por uma bússola eletrônica, o barco se desloca até um local predefinido desviando de possíveis obstáculos, cuja detecção é feita por meio de uma câmera

estéreo que registra fotografias periódicas do ambiente. A aquisição e o processamento das imagens são realizados por um computador externo, enquanto um sistema embarcado presente no interior do barco é o responsável pela coleta de dados e controle dos motores.

1.1 OBJETIVO GERAL

O objetivo geral deste trabalho é o desenvolvimento de um modelo de veículo náutico com navegação autônoma. Para isso, dispõe de sistema de posicionamento, identificação de obstáculos e tomada de decisões para contornar eventuais obstruções a fim de chegar a um destino predeterminado.

1.2 OBJETIVOS ESPECÍFICOS

Para atingir o objetivo geral, elaborou-se os seguintes objetivos específicos:

- Dimensionar os componentes mecânicos e eletrônicos que irão compor o protótipo;
- Determinar protocolo para troca de dados entre os dispositivos presentes no barco e um computador externo;
- Definir o sistema de posicionamento;
- Configurar um sistema estéreo de processamento de imagem para reconhecimento de obstáculos e determinação da distância;
- Implementar lógica *fuzzy* para controle do sistema;
- Avaliar o comportamento do sistema em um ambiente controlado, onde o barco se deslocará entre dois pontos desviando de obstáculos.

1.3 ESCOPO E RESTRIÇÕES

Este trabalho visa o desenvolvimento de um modelo de barco autônomo, com deslocamento e reconhecimento de obstáculos restringido a um ambiente controlado. Dessa forma, não são consideradas todas as situações pelas quais um veículo náutico real estaria sujeito.

1.4 APRESENTAÇÃO DO TRABALHO

Este trabalho está dividido em cinco capítulos. O primeiro apresentou uma breve introdução ao projeto e expôs os objetivos e restrições. O segundo capítulo expressa o referencial teórico que dá base bibliográfica aos temas abordados nas demais etapas do trabalho. No capítulo 3 está demonstrada a metodologia utilizada no desenvolvimento dos objetivos propostos. O quarto capítulo aborda os resultados obtidos a partir de testes práticos. O capítulo 5 expõe as considerações finais e algumas sugestões para projetos futuros. Por fim, são apresentadas as referências bibliográficas citadas no decorrer deste documento e os algoritmos desenvolvidos.

2 REVISÃO BIBLIOGRÁFICA

Com a finalidade de fundamentar a metodologia utilizada, são apresentados conceitos teóricos e práticos para melhor compreensão dos assuntos relacionados à elaboração deste projeto.

2.1 VEÍCULOS AUTÔNOMOS

De acordo com Ozguner, Stiller e Redmill (2007), o que define a autonomia de um veículo é a sua capacidade de tomadas de decisões sem a intervenção humana. Alguns exemplos já estão presentes nos modelos de carros atuais, como o freio *Anti-lock Braking System* (ABS) e sistemas de *cruise control*, ou piloto automático, que mantém o veículo em uma determinada velocidade.

Alguns fabricantes de automóveis, como a Audi, Tesla e Renault, têm trabalhado em projetos de carros com um maior nível de autonomia, almejando automóveis que possam dirigir de forma independente sem a necessidade de um condutor. A partir da coleta e análise de dados dos sensores e câmeras que informam ao sistema situações sobre o ambiente em que se encontra, o veículo é capaz de uma tomada de decisões semelhante à realizada pelo cérebro humano. Dessa forma, poderá realizar tarefas como parar em um sinal vermelho ou uma faixa de pedestres, obedecer a sinalizações da pista e dirigir-se autonomamente a um local específico analisando mapas e coordenadas GPS. (UDACITY, 2018b)

A principal motivação para o desenvolvimento de veículos autônomos consiste na redução dos números de acidentes de trânsito que, no Brasil, foram responsáveis por aproximadamente um milhão de óbitos no período entre 1980 e 2011, havendo uma estimativa de que esse número permaneça crescendo em um ritmo de 3,7% ao ano (WAISELFISZ, 2013). A maior parte desses acidentes são ocasionados por fatores humanos como excesso de velocidade, motoristas embriagados, utilização do celular durante a condução, falta de uso das setas na realização de manobras e o não cumprimento da distância mínima em relação ao veículo frontal ao trafegar por vias (DETRAN, 2016). Assim, considera-se que um carro completamente autônomo possa realizar tomadas de decisões fazendo uso de todos os recursos veiculares e condições da pista e tráfego, agindo sempre da melhor forma possível em cada

situação e inibindo acidentes causados por falhas humanas (PISSARDINI, WEI e JÚNIOR, 2013).

2.1.1 Níveis de automação

Conforme estabelecido pela SAE (2014), os veículos podem ser classificados em seis níveis de autonomia. Os tópicos a seguir descrevem detalhadamente cada um deles.

- Nível 0: Representa os veículos onde a automação é inexistente, englobando a grande maioria dos automóveis do mercado. Dessa forma, um motorista humano é responsável por todos os aspectos da condução, desde a direção até a sinalização, sem o auxílio de nenhum dispositivo automático.
- Nível 1: Os veículos que se encaixam nessa categoria apresentam alguma ferramenta específica de apoio ao condutor, como auxílio de direção ou assistência na aceleração e frenagem a partir de informações coletadas do ambiente em que se encontra.
- Nível 2: Engloba os meios de transporte com sistemas combinados de automação. Assim, necessita que os veículos possuam ao menos duas ferramentas de auxílio ao condutor trabalhando em conjunto. As demais operações devem ser realizadas por um motorista humano.
- Nível 3: Nesse nível o veículo deve apresentar autonomia para assumir todos os aspectos de direção em um ambiente controlado ou situações específicas. O sistema sempre considerará que há um condutor humano pronto para assumir a direção.
- Nível 4: Assim como no Nível 3, o veículo tem autonomia para assumir completamente a direção em locais controlados. Não há a necessidade de um condutor humano, de forma que o sistema sempre deverá saber como agir.
- Nível 5: Para um veículo ser considerado desse nível, o mesmo deverá ter a capacidade de dirigir sobre qualquer situação e qualquer ambiente, realizando tomadas de decisões da mesma forma que um condutor humano é capaz de fazer.

2.1.2 Detecção do ambiente

A fim de coletar informações sobre o ambiente no qual está situado e detectar obstáculos ao seu deslocamento, um veículo autônomo pode contar com diversos sensores para auxiliá-lo nesse processo. Alguns exemplos serão descritos a seguir.

2.1.2.1 RADAR e LIDAR

Um *Radio Detection and Ranging* (RADAR) consiste em um sensor que utiliza ondas de rádio, em frequências de pelo menos 300Mhz, para detectar obstruções ou outros veículos. Uma antena é responsável pela transmissão das ondas a uma velocidade conhecida, onde qualquer obstáculo encontrado gerará um eco que regressará ao RADAR. A partir de uma análise das características desse sinal, poderá ser feito um mapeamento do local de forma a informar as distâncias e propriedades de cada um dos alvos detectados. (LEITÃO, 2020)

Através do Efeito Doppler, o RADAR pode detectar alvos em movimento e calcular a sua velocidade. Isso ocorre devido ao deslocamento de um objeto refletir ondas eletromagnéticas com frequências diferentes das que recebe, de forma que ao se aproximar da antena receptora elas regressarão em um comprimento de onda menor, enquanto ao se afastar as mesmas possuirão um comprimento de onda maior. (UFRGS, 2020)

O *Light Detection and Ranging* (LIDAR) difere-se do RADAR devido à utilização de pulsos laser no lugar das ondas de rádio. Seu funcionamento consiste na emissão de sinais óticos em uma elevada frequência de repetição, que serão refletidos por obstáculos. O tempo de retorno de cada um desses pulsos é medido por um sensor, realizando uma varredura do ambiente e, ao associar esses valores a um sistema de posicionamento como o GPS, pode-se obter um mapeamento 3D do local. (MACIEL, 2011)

2.1.2.2 Câmeras

Veículos autônomos também podem dispor de câmeras para o registro de informações sobre o ambiente, realizando tarefas como a detecção de pedestres,

semáforos, delimitações da pista, outros veículos e demais obstáculos. Entre as câmeras utilizadas estão as monoculares, onde somente é registrada uma imagem comum do local; estéreo, no qual um sistema de duas câmeras calibradas realiza a detecção da distância entre os obstáculos; ou omnidirecional, detectando um ângulo de 360º ao seu entorno. Além das tradicionais câmeras coloridas, diversas aplicações utilizam câmeras monocromáticas, ou seja, a imagem será captada em tons de cinza, exigindo uma menor capacidade de processamento do *hardware*. (JUNG et al., 2005)

2.1.3 Controle do veículo

O desenvolvimento de um sistema para controle de um veículo autônomo é uma tarefa essencial e complexa, devido à grande quantidade de informações que este deverá processar de forma a estar apto para agir e planejar soluções em diversas situações. Dessa forma, o sistema computacional deverá apresentar capacidades para ler e interpretar sinais provindos dos diversos sensores do veículo, evitar obstáculos em seu caminho, planejar trajetórias e reagir a eventos inesperados, além de gerenciar da melhor forma possível todos os dispositivos presentes no sistema. Para executar todas essas tarefas faz-se necessária a utilização de uma arquitetura de controle, geralmente implementadas de acordo com as características da aplicação. Algumas das técnicas mais utilizadas são baseadas nos controles reativo, deliberativo ou híbrido. (JUNG et al., 2005)

Geralmente a arquitetura de implementação mais simples devido à baixa necessidade de recursos computacionais, o sistema reativo caracteriza-se por ser utilizado em funções onde não são utilizados mapas, memórias ou informações adicionais, dependendo exclusivamente dos sinais obtidos em tempo real pelos sensores para a execução das tarefas. Dessa forma, o controle é realizado a partir de um laço constituído por leitura dos sensores, processamento instantâneo das informações e geração de um sinal de resposta aos atuadores. Esse sistema normalmente é utilizado em dispositivos cujo movimento tem a necessidade de desviar de obstáculos ou seguir um objeto. (JUNG et al., 2005)

No controle deliberativo as ações são realizadas de forma planejada e organizada. Para isso ser possível, há a necessidade de um conhecimento prévio do sistema com informações referentes ao ambiente em que se encontra, como um mapeamento do local, rotas disponíveis e dados sobre possíveis obstáculos. Tende a

realizar processos de alto nível de raciocínio e tomada de decisões, procurando atuar sempre da melhor forma possível, ou seja, um sistema com essa arquitetura de controle pode ser capaz de planejar a melhor rota para um determinado deslocamento entre dois pontos quaisquer a partir da análise de um mapa do local. (JUNG et al., 2005)

A desvantagem do sistema deliberativo é que pode ser incapaz de atuar fora de um ambiente controlado, havendo sempre a necessidade de um modelo exato da área em que se encontra para executar suas ações. Isso acarreta em um controle extremamente frágil devido à erros causados por ruídos nos sensores ou imprecisões ocorridas no mapeamento do local. (HEINEN, 2002)

Um controle híbrido buscará combinar os sistemas de controle reativo e deliberativo. Dessa forma, o controlador trabalhará com dois módulos, sendo que o primeiro será o responsável pelo planejamento e execução de tarefas em longo prazo enquanto um segundo módulo atuará em situações que exigem resposta imediata, como manter o veículo em sua trajetória original ou desviar de obstáculos inesperados. Devido à complexidade do sistema, em algumas aplicações pode se fazer necessário o uso de um módulo adicional que gerenciará a comunicação entre ambos os controladores, a fim de resolver possíveis conflitos resultantes dessa integração. (HEINEN, 2002)

2.1.4 Veículos navais autônomos

Assim como fabricantes de automóveis estão competindo para remover o elemento humano da condução de carros, há vários estudos com objetivo de desenvolver veículos autônomos que possam se deslocar pela água. Pesquisadores holandeses têm buscado projetar embarcações autônomas para transporte de cargas e pessoas, enquanto a marinha americana desenvolveu patrulhas portuárias utilizando barcos de pesca não tripulados. (NEWATLAS, 2018)

Enquanto veículos terrestres devem ser capazes de se orientarem por estradas lotadas de motoristas humanos e pedestres, os navios autônomos possuem suas próprias dificuldades. Algumas embarcações podem necessitar de centenas de câmeras para ter toda sua extensão coberta, de forma que será requerido um poder de *hardware* expressivo para que todos os dados possam ser processados em tempo real, além da consequente necessidade de um enorme espaço disponível para

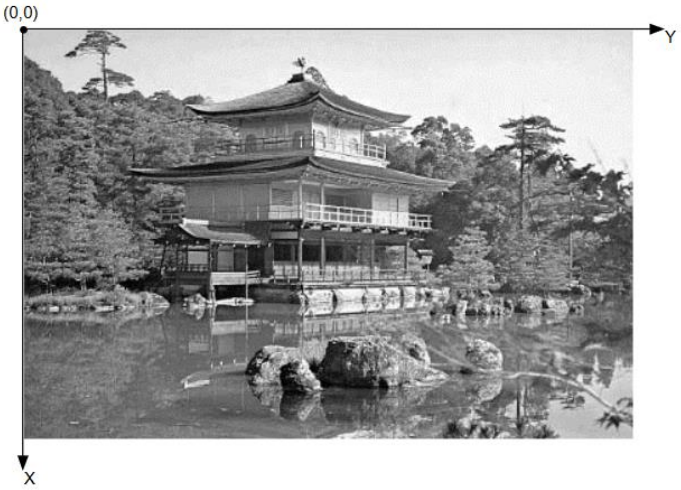
alocação de servidores. Não obstante as diversas câmeras, podem ainda contar com outros sensores como radares e sonares, além de algumas embarcações possuírem placas fotovoltaicas para auxiliar na recarga das baterias de alimentação do sistema. (WIRED, 2018)

Os veículos navais autônomos se dividem em duas categorias, *Autonomous Underwater Vehicles* (AUVs) e *Autonomous Surface Vehicles* (ASVs). Os AUVs consistem nos veículos que operam completamente submersos em aplicações que incluem funções de pesquisa arqueológica subaquática, coleta de dados hidrográficos e auxílio na navegação submarina de outras embarcações. Os veículos que realizam suas operações sobre a água são chamados de ASVs. Geralmente são utilizados para transporte e determinação de características da água a partir de sensores específicos. Eles se destacam por possuírem captação de sinal GPS, o que possibilita uma determinação precisa da sua localização e, consequentemente, podem operar como guias para embarcações submarinas. (FERREIRA et al., 2006)

2.2 PROCESSAMENTO DIGITAL DE IMAGENS

Uma imagem pode ser interpretada como um função bidimensional $f(x,y)$, onde x e y representam as coordenadas de um determinado ponto e f indica a intensidade da cor nesse local, também chamado de pixel. Quando os valores de x, y e f forem finitos e discretos, obtém-se uma imagem digital, conforme demonstrado na Figura 1. A partir de processos computacionais e matemáticos pode-se aperfeiçoar informações presentes em uma figura para melhor compreensão humana ou extrair informações para uma percepção automática desempenhada por máquinas. A essa edição dá-se o nome de processamento digital imagens. (GONZALEZ e WOODS, 2010)

Figura 1 – Representação de imagem monocromática em função de *f(x,y)*



Fonte: FILHO e NETO (1999).

2.2.1 Etapas do processamento digital de imagens

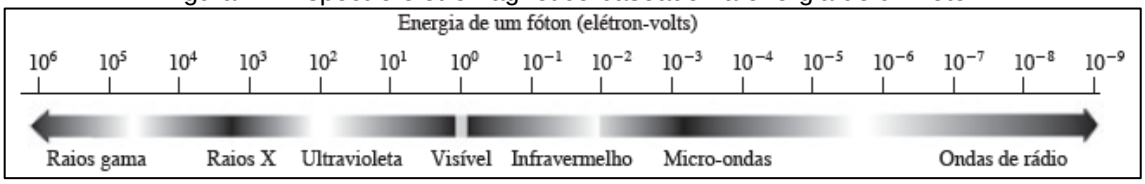
Para garantir que a imagem de saída do processo possua as características desejadas, diversas etapas se fazem necessárias. Os tópicos seguintes têm como finalidade descrever as principais funções de um sistema para aquisição e processamento de imagens, destinadas a aplicações nos campos de engenharia e sistemas de informação.

2.2.1.1 Aquisição

A aquisição consiste na conversão de uma imagem para uma representação numérica na qual será realizado o processamento digital. Inicialmente um dispositivo físico sensível a uma faixa de energia deve captar o espectro eletromagnético a ser analisado. Na sequência será necessário converter esse sinal analógico à informação digital, ou seja, representá-lo de forma binária. (FILHO e NETO, 1999)

As ondas eletromagnéticas capturadas podem ser tratadas como senoides de variados comprimentos, que representam a propagação de partículas sem massa na velocidade da luz e são constituídas por certas quantidades de energia chamadas de fótons. Essa energia determina a banda espectral do sinal, a partir das quais compõe-se o espectro eletromagnético mostrado na Figura 2. (GONZALEZ e WOODS, 2010)

Figura 2 – Espectro eletromagnético baseado na energia de um fóton



Fonte: GONZALES e WOODS (2010).

Por ser a faixa mais comum na vida cotidiana, a banda de luz visível geralmente é a empregada no processo de formação de imagens. Apesar disso, muitas vezes é utilizada em conjunto com a banda infravermelha que, de acordo com o intervalo do comprimento de onda registrado, possibilita destacar determinadas características da imagem, realçando atributos desejadas e reduzindo a saturação muitas vezes ocasionada pela luz visível. Dessa forma, seu emprego pode se dar a aplicações de sensoriamento remoto como mapeamento térmico, mineral, de biomassa e de linhas costeiras, além de estimar a umidade do solo e características de vegetação. Para outras áreas, pode auxiliar em processos de microscopia ótica, astronomia, policiamento, além de aplicações envolvendo processos industriais. (GONZALEZ e WOODS, 2010)

2.2.1.2 Processamento

Nessa etapa é realizado o processamento digital da imagem. Com o auxílio de *softwares*, pode-se realizar procedimentos expressos sob forma algorítmica, visando promover melhorias ou realçar determinados pontos da imagem. Algumas situações podem exigir o uso de *hardware* especializado, como placas gráficas específicas, devido às limitações do computador principal (FILHO e NETO, 1999). Algumas metodologias comumente aplicadas durante o processamento serão brevemente descritas abaixo.

- Realce de imagens: Consiste no processo de editar uma imagem para deixá-la mais adequado a uma aplicação específica. Dessa forma, as técnicas de realce geralmente são empregadas a fim de gerar uma melhor qualidade visual da imagem para facilitar a interpretação de um observador. (GONZALEZ e WOODS, 2010)
- Restauração de imagens: Assim como o realce, também se destina à obtenção de melhorias na qualidade da imagem. A diferença é que se trata de um

processo objetivo, ou seja, as técnicas de restauração baseiam-se em modelos matemáticos ou probabilísticos com fins de redução da degradação da imagem causada durante a aquisição. (GONZALEZ e WOODS, 2010)

- Compressão: Emprega técnicas de redução do armazenamento requerido para salvar uma imagem ou transmiti-la. Apesar dos sistemas de armazenamento terem evoluído, o mesmo não se aplica à capacidade de transmissão entre dispositivos, principalmente em relação à protocolos de internet. Uma das formas mais conhecidas de compressão é o padrão *Joint Photographic Experts Group* (JPEG), que gera arquivos com a extensão *.jpg*. (GONZALEZ e WOODS, 2010)
- Processamento morfológico: Utiliza técnicas e ferramentas com a finalidade de facilitar a representação e descrição de contornos e forma dos objetos, permitindo a extração de seus componentes. (GONZALEZ e WOODS, 2010)
- Segmentação: Consiste em separar uma imagem em partes ou objetos constituintes. A segmentação autônoma é uma das técnicas mais complicadas no processamento de imagens, necessitando de algoritmos complexos e bem desenvolvidos para evitar possíveis erros. Consequentemente, quanto melhor for a segmentação maiores serão as chances de um reconhecimento de objetos bem-sucedido. (GONZALEZ e WOODS, 2010)
- Representação: Responsável por converter os dados gerados pela segmentação a uma forma adequada para o seu subsequente reconhecimento ou processamento computacional. O método empregado deve descrever os dados de forma a realçar as características de interesse, como informações que possam tornar possível a diferenciação entre as classes de objetos que serão trabalhadas. (GONZALEZ e WOODS, 2010)
- Reconhecimento: É nesse processo que o objeto será reconhecido e rotulado, ou seja, a partir de uma análise dos padrões encontrados o sistema deverá indicar se a área selecionada compreende um pedestre, um veículo ou outro objeto específico. (GONZALEZ e WOODS, 2010)

2.2.1.3 Transmissão

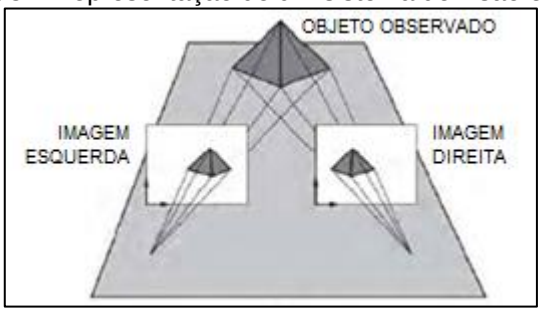
A partir de redes de computadores e protocolos de comunicação, pode-se realizar a transmissão de imagens do dispositivo que realizará a captura ao equipamento responsável pelo processamento digital, ou o envio de imagens processadas aos instrumentos que realizarão a sua exibição. A maior dificuldade nessa etapa é a necessidade de transmissão de grandes quantidades de informação no menor intervalo de tempo possível, indispensável quando deseja-se fazer o envio em tempo real. Para a transferência de vídeos o problema pode ser ainda pior, onde frequentemente empregam-se técnicas de compressão e descompressão de arquivos. (FILHO e NETO, 1999)

2.2.2 Sistemas de visão estéreo

Apesar da possibilidade de obtenção de informações detalhadas sobre objetos e cores presentes no ambiente de aquisição da imagem, uma única captura não é capaz de fornecer dados relacionados às suas distâncias, algo essencial para dispositivos móveis ou robôs que desejam mover-se livremente em um local desconhecido. A partir da análise de múltiplas imagens de um objeto, capturadas simultaneamente e em ângulos diferentes, é possível representá-lo tridimensionalmente e, consequentemente, obter sua localização exata. (RIOS, 2010)

Um sistema com duas câmeras buscará observar o ambiente de forma análoga à realizada pela visão humana, ou seja, cada um dos pontos presentes nas imagens capturadas possuirá um pequeno deslocamento entre si. Uma vez conhecida essa distância, a posição de cada objeto poderá ser calculada (MENDES, 2012). A Figura 3 representa um sistema de visão estéreo, onde duas câmeras dispostas em ângulos diferentes capturam imagens de um mesmo item.

Figura 3 – Representação de um sistema de visão estéreo

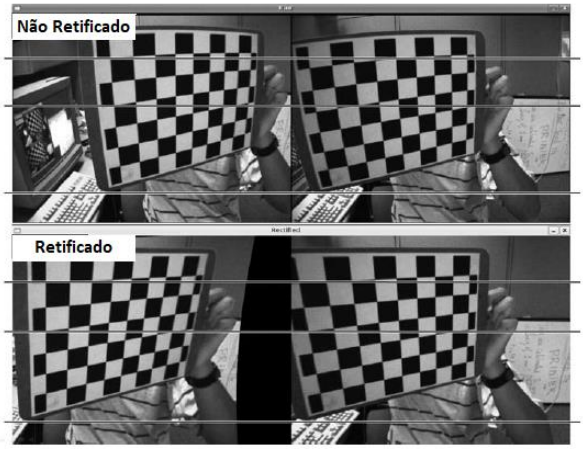


Fonte: Adaptado de RIOS (2010).

Para obter um mapeamento 3D do ambiente é necessária uma etapa de calibração das câmeras. A maioria dos sistemas utilizam uma geometria semelhante a um tabuleiro de xadrez para realizar esse processo. Isso se deve a simplicidade com que os vértices dos quadrados podem ser reconhecidos de forma automática por um *software*, possibilitando estabelecer uma relação das coordenadas de cada um deles nas fotos geradas simultaneamente pelas duas câmeras. (PINTO e PARRO, 2013)

Mesmo após o processo de calibração ambas as imagens obtidas podem estar desalinhadas, decorrência da dificuldade de se obter um alinhamento físico perfeito entre as duas câmeras. Dessa forma, pode-se fazer necessária a implementação de um algorítimo para tornar as imagens adequadas à sua aplicação final. A Figura 4 demonstra um exemplo de imagens sem retificação e o resultado depois da retificação, onde a posição vertical dos vértices do tabuleiro de xadrez, ao serem comparadas por meio de linhas horizontais, estão desalinhados nas imagens originais e, após esse processo, passam a estar perfeitamente alinhados. (BRADSKI e KAEHLER, 2008).

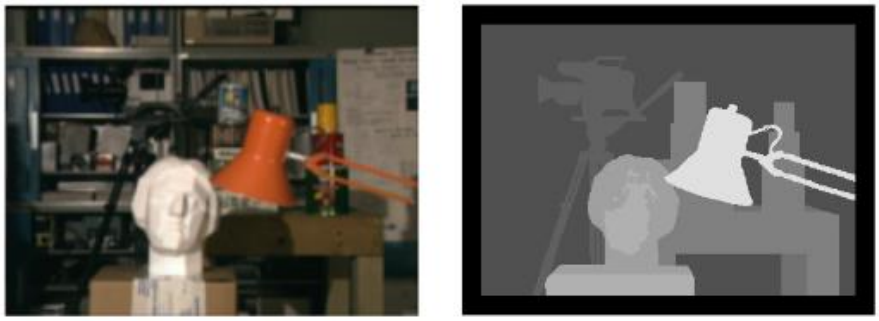
Figura 4 – Demonstração dos resultados de um processo de retificação



Fonte: Adaptado de BRADSKI e KAEHLER (2008).

Após realizados esses procedimentos, é possível se obter um mapeamento 3D do ambiente automaticamente com o auxílio de *softwares* específicos. Um desses meios consiste em um mapeamento monocromático, onde a distância dos objetos na imagem de saída do sistema é representada em escalas de cinza, ou seja, o tom da cor variará conforme o afastamento do objeto. A Figura 5 expressa a imagem de um determinado ambiente em comparação ao seu respectivo mapa de profundidade monocromático. (SCHARSTEIN e SZELISKI, 2002)

Figura 5 – Mapa de profundidade monocromático de um determinado ambiente



Fonte: SCHARSTEIN e SZELISKI (2002).

2.2.3 Sistemas de visão artificial

Um Sistema de Visão Artificial, ou SVA, consiste em um sistema computadorizado instruído a realizar todas as tarefas de processamento de imagem, desde a sua captura até a interpretação e reconhecimento dos objetos presentes. Com os avanços nas áreas da tecnologia e computação desempenhados nos últimos anos, obtiveram-se sistemas mais rápidos, com uma maior capacidade de armazenamento e possibilidade de execução de diversas tarefas paralelamente. Isso tornou viável a implementação de algoritmos automáticos para realizar o processamento de imagens. Porém um grande desafio continua sendo conceber um sistema de visão artificial que opere normalmente em diferentes condições ambientais, principalmente em relação à luminosidade e posições variáveis dos objetos a serem analisados. (FILHO e NETO, 1999)

De forma análoga à visão humana, um sistema ideal deve ser capaz de reconhecer e diferenciar pequenas variações que não comprometem a identidade do objeto, da mesma forma que o ser humano é capaz de reconhecer outra pessoa independente desta estar usando óculos ou não, ou se fez mudanças no cabelo e na

barba. Para isso ser possível, o sistema deve possuir uma base de dados para consulta, alta velocidade de processamento e capacidade de trabalhar sob variadas condições e ambientes. A Tabela 1 relaciona algumas divergências entre o sistema de visão humano e um sistema de visão artificial. (FILHO e NETO, 1999)

Tabela 1 – Sistema visual humano vs. sistema de visão artificial

	Sistema Visual Humano	**Sistema de Visão Artificial**
Espectro	Limitado à faixa de luz visível (300 nm a 700 nm) do espectro e ondas eletromagnéticas.	Pode operar em praticamente todo o espectro de radiações eletromagnéticas, dos raios X ao infravermelho.
Flexibilidade	Extremamente flexível, capaz de se adaptar a diferentes tarefas e condições de trabalho.	Normalmente inflexível, apresenta bom desempenho somente na tarefa para a qual foi projetado.
Habilidade	Pode estabelecer estimativas relativamente precisas em assuntos subjetivos.	Pode efetuar medições exatas, baseadas em contagem de pixels e, portanto, dependentes da resolução da imagem digitalizada.
Cor	Possui capacidade de interpretação subjetiva de cores.	Mede objetivamente os valores das componentes R, G e B para determinação de cor.
Sensibilidade	Capaz de se adaptar a diferentes condições de luminosidade, características físicas da superfície do objeto e distância ao objeto. Limitado na distinção de muitos níveis diferentes de cinza, simultaneamente.	Sensível ao nível e padrão de iluminação, bem como à distância em relação ao objeto e suas características físicas. Pode trabalhar com centenas de tons de cinza, conforme projeto do digitalizador.
Tempo de Resposta	Elevado, da ordem de 0,1 s.	Dependente de aspectos de *hardware*, podendo ser tão baixo quanto 0,001 s.
2D e 3D	Pode executar tarefas 3-D e com múltiplos comprimentos de onda (dentro do espectro de luz visível) facilmente.	Executa tarefas 2-D com relativa facilidade, mas é lento e limitado em tarefas 3-D.
Percepção	Percebe variações de brilho em escala logarítmica. A interpretação subjetiva de brilho depende da área ao redor do objeto considerado.	Pode perceber brilho em escala linear ou logarítmica.

Fonte: Adaptado de FILHO e NETO (1999).

2.3 SISTEMAS DE CONTROLE INTELIGENTE

Entender como o pensamento humano funciona tem sido um objeto de pesquisas durante milhares de anos. Diversas discussões têm sido geradas sobre como um conjunto de matéria é capaz de reconhecer padrões, aprender, prever e manipular o ambiente a sua volta. A área da inteligência artificial vai além, pois não busca apenas compreender o funcionamento do cérebro humano, mas também criar novos sistemas inteligentes capazes de resolver problemas e tomar decisões de forma complexa e autônoma. (RUSSEL e NORVIG, 2004)

Devido aos avanços de *hardware* e *software*, foi possível o desenvolvimento de sistemas inteligentes atuando nas mais diversas áreas da tecnologia, realizando tarefas como armazenar, organizar e recuperar grandes quantidades de informação, processar dados e analisar situações. Com base nas informações disponíveis, o sistema deverá ser capaz de se adaptar ou moldar seu comportamento de acordo com a ocasião. (REZENDE, 2003)

Algumas técnicas foram desenvolvidas com esses fins, sendo hoje amplamente utilizadas de forma individual ou operando em conjunto (REZENDE, 2003). Dentre elas, destacam-se as chamadas Aprendizado de Máquina, Redes Neurais Artificiais e Sistemas *fuzzy*, que serão brevemente apresentadas a seguir.

2.3.1 Aprendizado de Máquina

Com o objetivo de desenvolver sistemas para a realização de tomada de decisões, o aprendizado de máquina consiste em uma técnica de inteligência artificial que utiliza como base experiências anteriores adquiridas por meio da correta solução de problemas. Através de uma tarefa conhecida como indução, o sistema chegará a um resultado analisando o processo e comparando-o com situações anteriores, de forma a utilizar como base o exemplo armazenado mais próximo à ocasião em que o controlador se encontra. Porém esse sistema não está imune à erros, devido à possibilidade das hipóteses geradas terem como modelo poucas informações ou então não possuir nenhum dado armazenado que corresponda à determinada situação. (MONARD e BARANAUSKAS, 2003)

Assim, a realização do aprendizado indutivo se dá através de exemplos fornecidos ao sistema, podendo ocorrer a partir de um algoritmo de aprendizado que

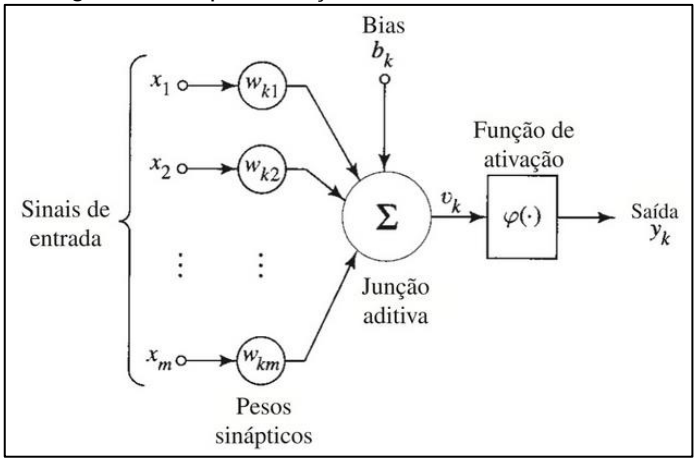
ajudará na classificação dos eventos ou então provendo-se da análise desses exemplos para verificar automaticamente quais dados serão armazenados e classificados. Dessa forma, ao demandar de uma consulta para execução de um processo, os dados registrados poderão ser acessados rapidamente de acordo com a sua distribuição. (MONARD e BARANAUSKAS, 2003)

2.3.2 Redes Neurais Artificiais

Uma Rede Neural Artificial (RNA) consiste em um modelo matemático baseado em estruturas neurais biológicas que apresenta funções computacionais por meio de aprendizado e generalização. Dessa forma, demonstram capacidade para adaptar seus parâmetros de acordo com as informações lidas sobre o meio em que está inserida. Como o processo de aprendizado de um sistema RNA consiste na análise de dados de situações reais pelas quais já passou, seu desempenho tende a melhorar gradativamente ao longo do tempo que interage com o ambiente, enquanto que para funções básicas o sistema poderá passar por um processo inicial de treinamento de modo a se preestabelecer parâmetros de controle essenciais para o modelo. (BRAGA, CARVALHO e LUDERMIR, 2003)

O processamento de dados em uma RNA é feito a partir de estruturas neurais artificiais, onde o armazenamento de informações e o seu processamento são executados paralelamente (BRAGA, CARVALHO e LUDERMIR, 2003). Assim, cada processador equivale a um neurônio artificial que receberá dados, os analisará e executará uma ação, conforme representado pela Figura 6.

Figura 6 – Representação de um neurônio artificial

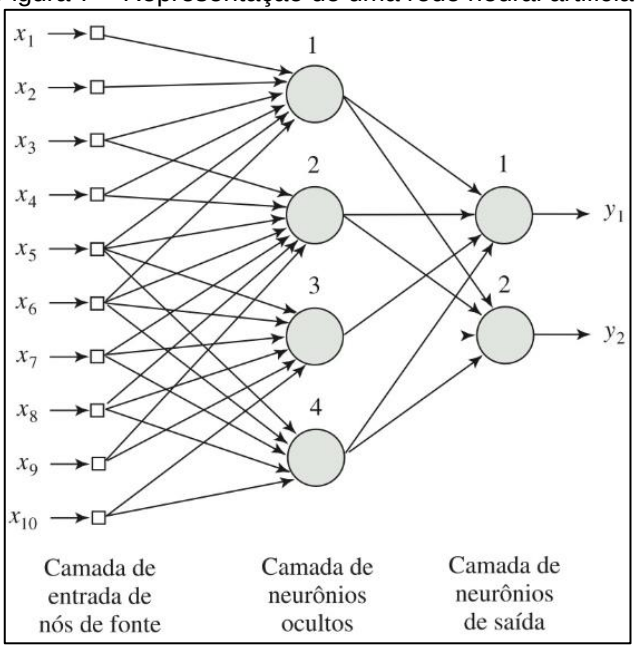


Fonte: HAYKIN (2008).

Consequentemente, uma rede neural é formada por um conjunto de elementos processadores, capazes de execução de tarefas complexas. A sua grande vantagem é a capacidade de execução de diversas funções paralelamente, independente da complexidade do sistema, e sem perdas no processamento, devido à cada função ser executada por um dispositivo individual. Dessa forma, caso se faça necessária a implementação de novas etapas ao processo, bastará acrescentar novos neurônios artificiais à rede. (BRAGA, CARVALHO e LUDERMIR, 2003).

A Figura 7 demonstra esse agrupamento de processadores atuando como uma rede. Nota-se a presença de diversos sinais de entradas que são processados por múltiplos neurônios artificiais para gerar os dados que comporão as saídas do sistema.

Figura 7 – Representação de uma rede neural artificial



Fonte: HAYKIN (2008).

2.3.3 Sistemas *Fuzzy*

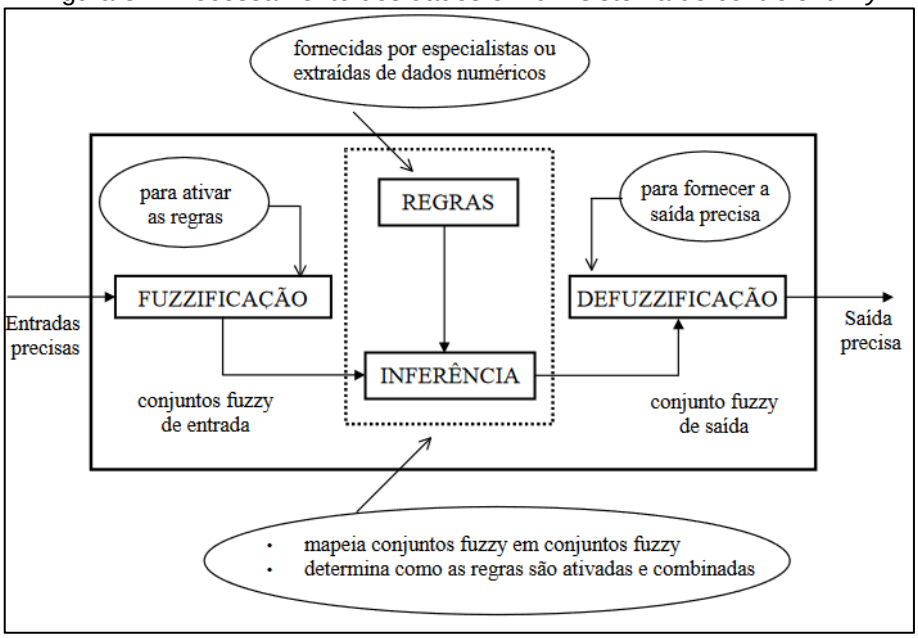
A principal característica de um sistema *fuzzy*, ou lógica nebulosa, é representar e lidar com informações imprecisas, ou seja, essa lógica procura demonstrar, em valores numéricos, os dados vagos e qualitativos geralmente empregados na comunicação humana. Dessa forma, pode-se implementar um controlador computadorizado capaz de tomar decisões estratégicas em problemas

complexos que, utilizando a lógica binária tradicional, seriam difíceis de solucionar. (SHAW e SIMÕES, 1999)

Na lógica *fuzzy* os dados são representados e classificados por conjuntos. Assim, um determinado valor pode corresponder parcialmente a diversos grupos e, ao invés de pertencer ou não à cada um deles, o dado terá um grau de pertinência. Essa classificação pode ser vista de forma análoga à perspectiva humana, onde o cérebro processa dados vagos e imprecisos devido à natureza incerta das informações presentes no mundo real. Como computadores operam de forma binária, onde uma determinada informação será sempre verdadeira ou falsa e sem valores intermediários, um controlador *fuzzy* busca aproximar a lógica computacional ao raciocínio humano. (SHAW e SIMÕES, 1999)

A Figura 8 apresenta, através de um diagrama de blocos, uma visão geral do processo pelo qual um dado de entrada passa para gerar um sinal de saída em um controlador *fuzzy*.

Figura 8 – Processamento dos dados em um sistema de controle *fuzzy*

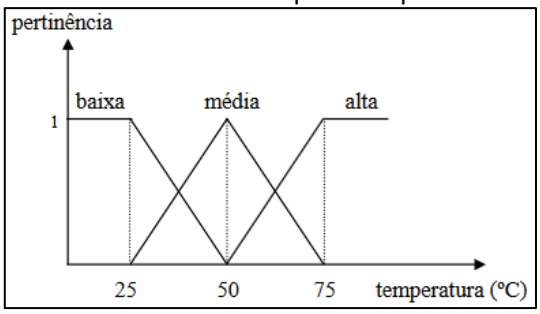


Fonte: TANSCHEIT (2004).

Em um processo chamado fuzzificação, os dados devem ser convertidos para variáveis linguística de forma a caracterizar esse valor numérico conforme as suas propriedades. A Figura 9 apresenta um exemplo de controle de temperatura em determinado processo. A temperatura lida será classificada em baixa, média ou alta e, para valores intermediários, pode-se determinar o quão baixa ou alta está. As

variáveis linguísticas têm como objetivo descrever valores complexos ou vagos para tratá-los futuramente em um sistema *fuzzy*. (TANSCHEIT, 2004)

Figura 9 – Conversão de um dado de temperatura para uma variável linguística



Fonte: TANSCHEIT (2004).

Na sequência esses valores serão processados em uma lógica computacional a partir de um modelo de regras. Para isso serão executadas funções de acordo com o conjunto do qual o valor lido foi classificado. Esse modelo pode ser implementado por uma sequência de verificações e execuções, gerando um sinal de saída que influenciará no processo. Assim, um controlador *fuzzy* contará com diversas regras, ou inferências, que serão executadas simultaneamente garantindo ao sistema alta velocidade de processamento. Ao invés de utilizarem a tradicional lógica binária, as inferências *fuzzy* são definidas em conjuntos linguísticos, comumente seguindo o modelo definido por “SE <condições> ENTÃO <conclusão>”. (SHAW e SIMÕES, 1999)

Com os sinais gerados pelo conjunto de regras deve-se realizar uma etapa de defuzzificação, onde essas variáveis linguísticas serão transformadas em um dado discreto que representa os valores *fuzzy* resultantes da etapa anterior (SHAW e SIMÕES, 1999). Esse processo tem como objetivo conceber um sinal de saída para que o dispositivo que o recebe possa compreender a informação gerada pelo sistema e agir de acordo com o que foi especificado pelo controlador (TANSCHEIT, 2004).

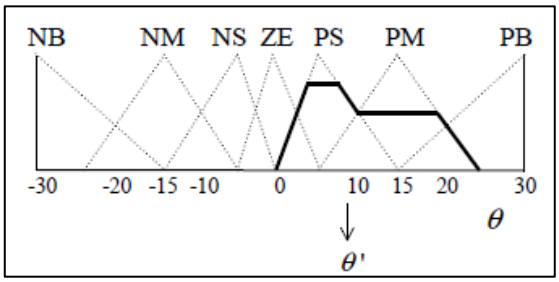
O método de defuzzificação deverá ser definido de acordo com o processo à que será aplicado, levando em consideração centroides ou valores máximos resultantes de cada função de pertinência correspondente às saídas. Alguns dos métodos mais empregados são os denominados Centro da Área (CoA), Centro do Máximo (CoM) e Média dos Máximos (MoM). O Centro da Área define o valor central da região composta pela combinação de todos os conjuntos resultantes do processo

de inferência *fuzzy,* por meio de uma média ponderada. Tem como base a Equação 1, onde $u_{OUT}(u_i)$ representa os resultados da inferência *fuzzy* das funções de pertinência e u_i indica as suas posições. (SHAW e SIMÕES, 1999)

$$CoA = \frac{\sum_{i=1}^{N} u_i . u_{OUT}(u_i)}{\sum_{i=1}^{N} u_{OUT}(u_i)} \quad (1)$$

A Figura 10 demonstra o valor resultante da aplicação do método CoA por meio de um exemplo prático onde a saída, definida como θ', aponta a região central da área somada dos conjuntos PS e PM.

Figura 10 – Processamento dos dados em um sistema de controle *fuzzy*



Fonte: TANSCHEIT (2004).

Ao se utilizar o método Centro do Máximo considera-se apenas o local onde ocorrem os picos das funções de pertinência e, ao contrário do modelo CoA, ignora-se a área total de cada conjunto. Apesar disso, a utilização de qualquer um desses dois métodos acarretará em respostas semelhantes. A Equação 2, cuja estrutura assemelha-se à Equação 1, demonstra o cálculo do valor defuzzificado por meio do critério do CoM, onde $u_{O,k}(u_i)$ representa os locais de ocorrência dos máximos. (SHAW e SIMÕES, 1999)

$$CoM = \frac{\sum_{i=1}^{N} u_i . \sum_{k=1}^{n} u_{O,k}(u_i)}{\sum_{i=1}^{N} \sum_{k=1}^{n} u_{O,k}(u_i)} \quad (2)$$

Por fim, a defuzzificação pela Média do Máximo considera como saída o conjunto que apresentar o maior grau de pertinência. Devido às limitações do método ao operar em sistemas que podem apresentar mais de uma função com o valor máximo, sua aplicação é dada principalmente em processos onde objetiva-se a

classificação ou reconhecimento de padrões. Esse critério utiliza como referência a Equação 3, onde M equivale a uma quantidade m de elementos referentes à função do máximo e u_m representa seus respectivos valores de pertinência. (SHAW e SIMÕES, 1999)

$$MoM = \sum_{m=1}^{M} \frac{u_m}{M} \tag{3}$$

2.4 SISTEMAS DE POSICIONAMENTO

Desde os primórdios da civilização, o homem necessitou saber a sua localização exata sobre a superfície terrestre para melhor desenvolver suas atividades. Dessa forma, sempre buscou por técnicas e ferramentas que auxiliassem em seu deslocamento. Devido aos avanços tecnológicos, sistemas extremamente precisos têm surgido utilizando como base sinais de referência transmitidos por satélites ou outros dispositivos emissores, a partir dos quais um receptor pode determinar sua localização precisamente. (CARVALHO e ARAÚJO, 2009)

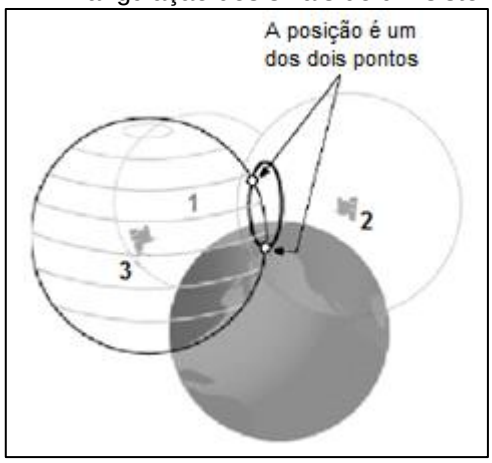
2.4.1 *Global Positioning Systems* (GPS)

Projetado inicialmente para fins de defesa, atualmente o GPS é usado em diversas aplicações não militares. Sua infraestrutura consiste em satélites que orbitam a terra em uma altitude de aproximadamente 20.000km e, com base em relógios atômicos extremamente precisos, são transmitidos fluxos de dados em forma de um sinal de rádio frequência. Esses satélites circundam o planeta duas vezes por dia e, a partir desses dados transmitidos, um dispositivo receptor pode analisá-los para determinar a sua localização exata através de um processo chamado triangulação. (LIMA, 2001)

A triangulação de um sistema GPS consiste na utilização de ao menos quatro satélites. Dessa forma, a partir do sinal do primeiro pode-se gerar uma esfera cujo raio é a distância calculada entre o satélite e o dispositivo receptor. Com a distância entre o receptor e o segundo satélite obtém-se uma nova esfera, que intersectará o primeiro e reduzirá a incerteza à apenas um círculo, enquanto o sinal recebido do terceiro

emissor irá cruzar o círculo anterior em dois pontos. Devido a um desses pontos encontrar-se muito distante da Terra, pode-se determinar a posição exata por meio de um processo de exclusão. O quarto satélite envia um sinal para auxiliar na determinação do tempo em que ocorreram as emissões, evitando a necessidade do receptor utilizar um relógio atômico com esse fim. A Figura 11 representa graficamente esse processo. (CORREIA, 2003)

Figura 11 – Triangulação dos sinais de um sistema GPS

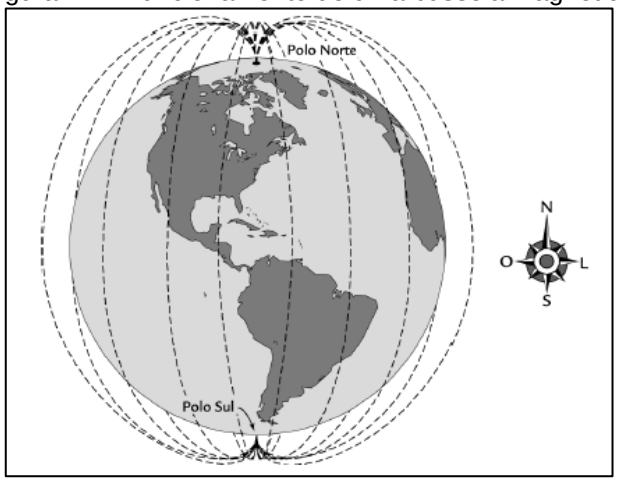


Fonte: Adaptado de CORREIA (2003).

2.4.2 Magnetômetro e bússola eletrônica

Historicamente, o desenvolvimento da bússola magnética foi de grande importância para a expansão do comércio no final do século XIII, sendo o primeiro instrumento a possibilitar a navegação por mar e por terra de forma precisa independentemente das condições climáticas e ambientais. Seu funcionamento se dá devido às características magnéticas da Terra, que possui linhas eletromagnéticas correndo entre seus polos norte e sul conforme demonstrado na Figura 12. Esse campo magnético é capaz de exercer influência sobre o ferro, atraindo ou repelindo magnetos dependendo de sua orientação. Assim sendo, a agulha presente em uma bússola é constituída por um material magnético, suspenso sobre o ar ou a água, que reage e se alinha ao campo magnético terrestre. (ACZEL, 2002)

Figura 12 – Funcionamento de uma bússola magnética



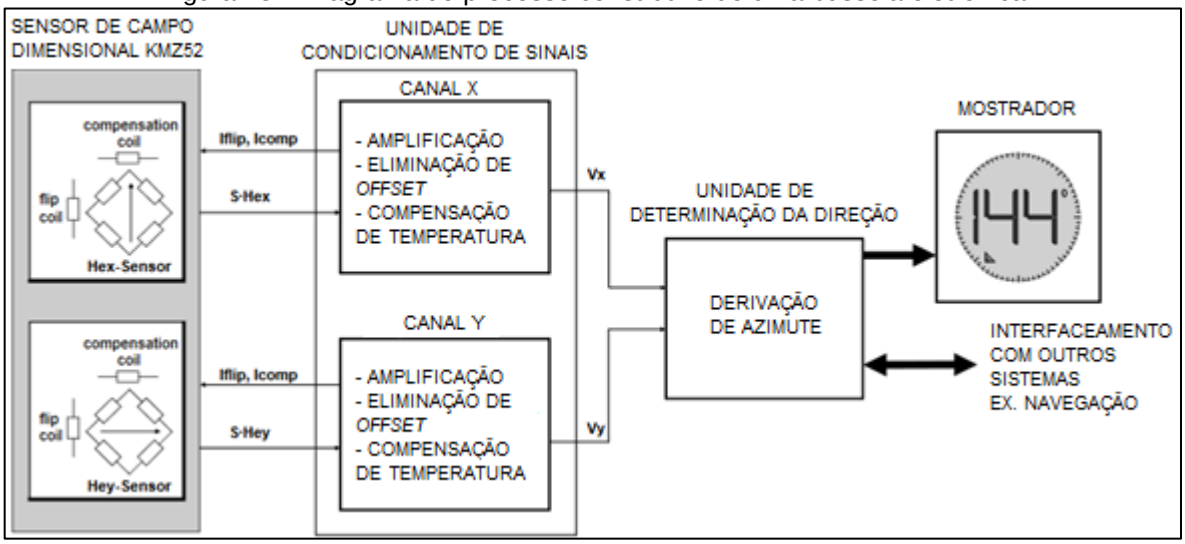
Fonte: ACZEL (2002).

Com premissa semelhante, os magnetômetros dispõem de meios de sensoriamento mais sensíveis e precisos para detecção de campos magnéticos. São classificados de acordo com sua utilização e podem se dividir em dois grupos, os escalares, capazes apenas de medir o módulo do campo, e os vetoriais, que além da intensidade identificam também as componentes referentes às direções do vetor. (LENZ, 1990)

A sensibilidade de um magnetômetro varia de acordo com suas características construtivas e materiais utilizados, dentre os quais estão o *Superconducting Quantum Interference Device* (SQUID), que é o mais sensível modelo de magnetômetro existente por prover da interação de campos magnéticos sobre supercondutores; os sensores magnetoressistivos, cujas resistências variam de acordo com a intensidade do campo incidente; as bobinas sensoras que, com base na lei da indução de Faraday, empregam enrolamentos de fios condutores para medição da variação do fluxo magnético através da corrente induzida sob suas espirais; o efeito Hall, no qual a incidência eletromagnética sobre uma superfície de material semicondutor gera diferenças de potencial; os *fluxgates*, onde duas bobinas são enroladas em um núcleo ferromagnético que, devido à sua histerese, causa defasagens das variações do fluxo magnético sobre o mesmo, resultando na detecção do campo e de sua excitação pelas bobinas; entre outros. (LENZ, 1990)

Uma bússola eletrônica baseia-se na associação desses sensores de campo magnético. A Figura 13 demonstra um exemplo prático do seu processo construtivo por meio de um diagrama de blocos.

Figura 13 – Diagrama do processo construtivo de uma bússola eletrônica



Fonte: Adaptado de STORK (2001).

No primeiro bloco utiliza-se dois conjuntos de sensores magnetorresistivos agrupados, formando pontes de Wheatstone, onde a medição do campo magnético pode ser determinada por meio da diferença de potencial entre seus terminais. De forma a medir a intensidade de duas componentes magnéticas horizontais, sendo uma frontal e a outra lateral, esses sensores são dispostos perpendicularmente. Na sequência um circuito condicionador de sinais amplifica as tensões geradas pelos sensores, além de aplicar filtros para compensação de sensibilidade, temperatura e remoção de *offset*. Seu resultado corresponde a saídas de tensão diretamente proporcionais aos campos magnéticos incidentes em cada direção. O próximo bloco tem como função calcular a direção angular da bússola a partir dos sinais provenientes das etapas anteriores. Esses dados podem ser então fornecidos a um display para exibição de seu posicionamento ou para interfaceamento com dispositivos de navegação. (STORK, 2001)

2.5 TRABALHOS RELACIONADOS

Considerando o crescente interesse em veículos autônomos por parte de pesquisadores e empresas nas últimas décadas, diversos autores têm proposto métodos para controle, detecção de obstáculos e posicionamento, dos quais muitos têm sido desenvolvidos, aperfeiçoados e aplicados em modelos reais. Os próximos

parágrafos apresentam alguns desses métodos, buscando evidenciar o que há de mais avançado na área de veículos náuticos autônomos.

O trabalho de Mendes (2012) propõe um sistema de câmeras estéreo para detecção de obstáculos ao deslocamento de um veículo autônomo, onde a sua calibração e a retificação das imagens foram realizadas com base em bibliotecas *Open Computer Vision* (*OpenCV*). O autor testou dois métodos para a verificação de obstáculos, a detecção baseada em plano e a detecção baseada em cone. O primeiro método consiste em determinar a área sobre a qual o veículo circula, de forma que pontos logo acima desse plano representarão obstáculos a navegação do veículo e deverão ser desviados, enquanto aos demais pontos considera-se que não estarão em sua rota de colisão. A segunda alternativa seleciona possíveis obstáculos através da posição dos pontos de um determinado objeto, comparando o ângulo formado entre eles e a sua altura total. Dessa forma, é possível diferenciar obstruções à trajetória do veículo e o plano pelo qual ele circulará. Apesar da necessidade de maior nível de *hardware* para o processamento dos dados, Mendes (2012) obteve melhores resultados com a segunda opção. As Figuras 14 e 15 representam as imagens geradas em cada um dos dois métodos, onde os pixels vermelhos representam os obstáculos encontrados.

Figura 14 – Imagem gerada a partir da detecção baseada em plano

Fonte: MENDES (2012).

Figura 15 – Imagem gerada a partir da detecção baseada em cone

Fonte: MENDES (2012).

Pereira (2015) propõe uma lógica baseada em *fuzzy* para controlar o deslocamento de um veleiro autônomo. Seu projeto utiliza dados de entrada recebidos por sensor GPS, bússola e cata-vento, visando determinar a sua posição, coordenada de destino e orientação do vento. A partir dessas informações, o controlador *fuzzy* será responsável por determinar a direção que o veleiro deverá virar para chegar ao local designado. Para isso, o autor definiu sete posições de virada do leme, conforme demonstrado na Tabela 2.

Tabela 2 – Direção do veleiro de acordo com as saídas do controlador *fuzzy*

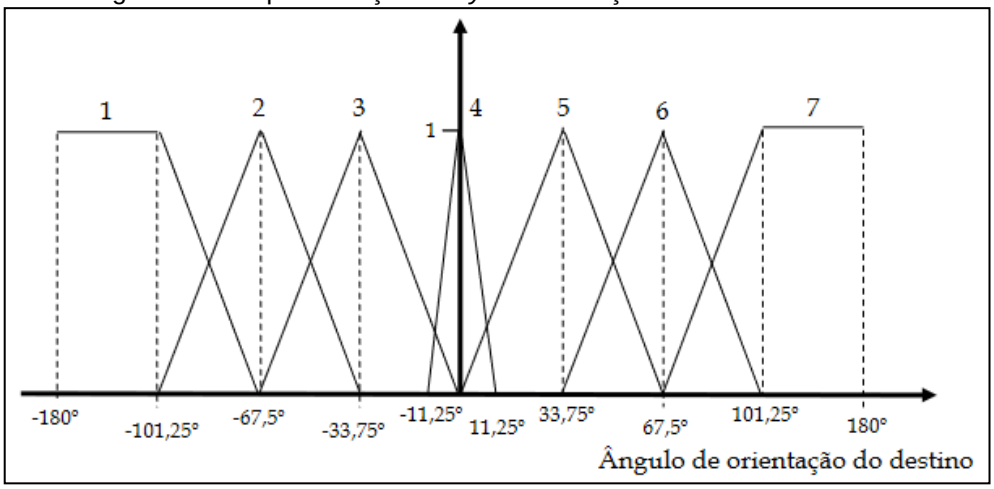
Funções de pertinência	**Orientação do destino em relação à proa**
1	Tudo para a esquerda
2	Esquerda
3	Um pouco à esquerda
4	Centrado
5	Um pouco à direita
6	Direita
7	Tudo para a direita

Fonte: Adaptado de PEREIRA (2015).

Para obter a direção do deslocamento do barco, Pereira (2015) utilizou cada um dos conjuntos presentes na Tabela 2 como possíveis dados de saída de um sistema *fuzzy*, selecionados de acordo com o ângulo necessário para que o barco

chegue ao seu destino. Assim, a Figura 16 representa graficamente os valores angulares possíveis para essa rotação.

Figura 16 – Representação *fuzzy* da orientação do destino do veleiro

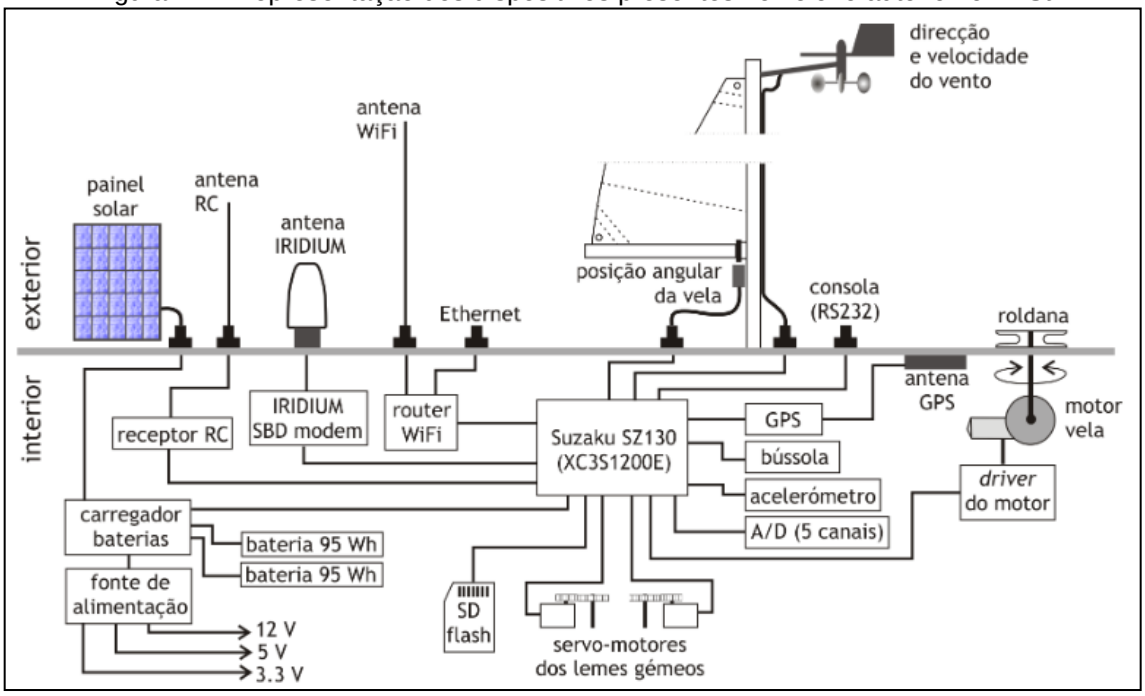


Fonte: PEREIRA (2015).

O veleiro autônomo FASt foi desenvolvido pelo Departamento de Engenharia Elétrica e de Computação da Universidade do Porto. O projeto foi iniciado em 2007 para participar de uma série de competições internacionais e ao longo dos anos tem passado por diversas melhorias e otimizações. (FEUP, 2014)

A embarcação é controlada por um sistema *Field Programmable Gate Array* (FPGA) e conta com GPS, bússola eletrônica, inclinômetro, sensores de direção, velocidade e ângulo das velas, além de um painel solar para auxiliar no carregamento das baterias que alimentam o sistema. A posição das velas é comandada por um motor DC acionado por *Pulse Width Modulation* (PWM), enquanto os lemes presentes no veículo são controlados por dois servomotores independentes. A Figura 17 demonstra esses e outros dispositivos embutidos no veleiro. (ALVES e CRUZ, 2009)

Figura 17 – Representação dos dispositivos presentes no veleiro autônomo FASt



Fonte: ALVEZ e CRUZ (2009).

O navio de carga norueguês Yara Birkeland, em construção pelas empresas Yara e Kongsberg, teve seu desenvolvimento iniciado em 2017 e foi lançado ao mar pela primeira vez em fevereiro de 2020. A embarcação é completamente elétrica, contando com um conjunto de baterias com capacidade de 7 MWh e possui as dimensões de 80 x 15 metros, podendo alcançar velocidades de 13 nós (aproximadamente 24 km/h). (YARA, 2020)

Para evitar obstáculos e se guiar pelo percurso, o Yara Birkeland conta com sensores RADAR, LIDAR e *Automatic Identification System* (AIS), além de um sistema de visão composto por câmeras tradicionais e câmeras de infravermelho. A comunicação com o centro de controle é feita através de processos de troca de dados por satélite, do recurso *Global System for Mobile Communications* (GSM), geralmente empregado em telefones celulares, e do sistema de comunicação marítima por ondas de rádio *Maritime Broadband Radio* (MBR). O navio realizará carregamentos e descarregamentos de containers em três portos, localizados nas cidades de Larvik, Brevik e Herøya, todas na Noruega, cobrindo uma distância de 12 milhas náuticas (aproximadamente 22,2 km) com capacidade de transporte de até 120 containers de tamanho padrão. Esses carregamentos serão feitos de forma completamente independente, onde a embarcação utilizará guindastes e receberá auxílio de

equipamentos elétricos, além de empregar um sistema para ancoragem autônoma. (KONGSBERG, 2020)

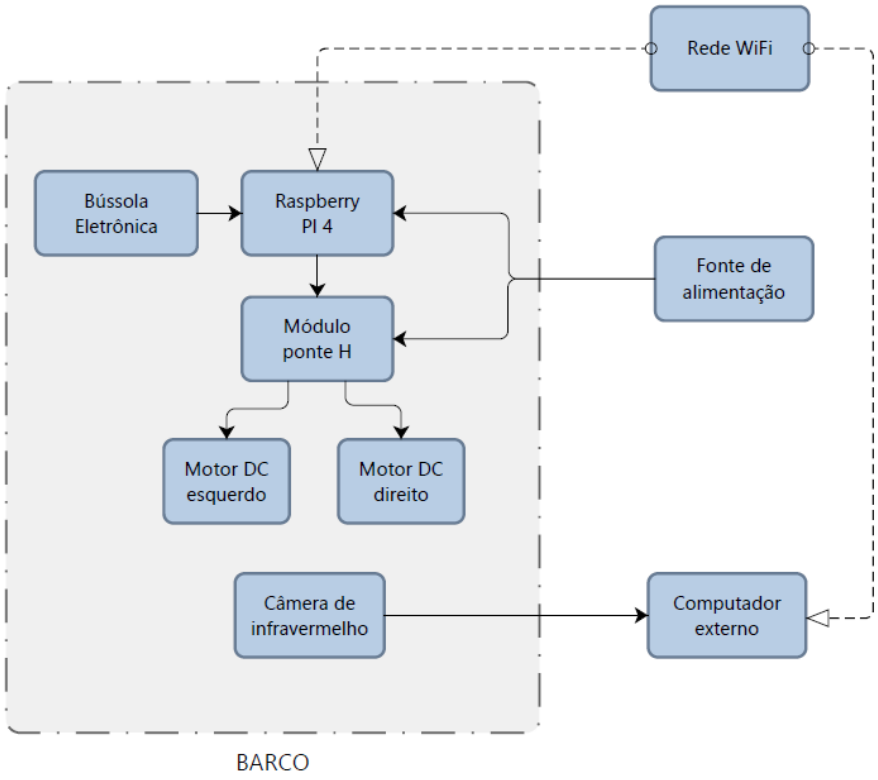
Desenvolvido pela agência governamental americana *Defense Advanced Research Projects Agency* (DARPA), o navio autônomo Sea Hunter foi criado para utilização militar, com o objetivo de detectar a presença de submarinos inimigos. Para identificar outras embarcações e determinar informações sobre seus arredores, utiliza sistemas de RADAR e SONAR. O navio mede aproximadamente 41 metros de comprimento e pesa 135 toneladas. Também está capacitado para a troca de dados com uma estação de controle, responsável por determinar suas condições e localização no oceano e, embora tenha sido projetado para operar sem qualquer intervenção humana, possibilitar o seu controle remotamente. (DEFENSE SYSTEMS, 2017)

O projeto teve início em 2014 e em 2016 foram realizados os primeiros testes envolvendo os sensores e recursos de autonomia da embarcação em mar aberto. Em 2017 foram efetuados experimentos envolvendo contramedidas de minas, que consistem na localização e destruição de minas navais. No início de 2018 a DARPA entregou o projeto à *Office of Naval Research* (ONR), que deu sequência aos testes da embarcação visando aperfeiçoar o processamento de dados dos sensores, melhorar seus sistemas de controle e autonomia, além de integrar sua operação e comunicação com outras embarcações semelhantes. (DARPA, 2018)

3 METODOLOGIA

A proposta deste trabalho consiste no desenvolvimento de um veículo náutico com deslocamento autônomo. Para isso, o dispositivo conta com um sistema de visão estéreo utilizando uma câmera de infravermelho com duas lentes alinhadas e bússola eletrônica para determinar o seu posicionamento. Embutido no barco, há um sistema embarcado Raspberry PI 4 para controle dos equipamentos que, através de uma rede Wi-Fi, comunica-se com o computador externo que efetua a aquisição e o processamento das imagens. Por fim, o deslocamento do barco é realizado por meio do acionamento de dois motores de propulsão. A Figura 18 demonstra, a partir de um diagrama de blocos, uma visão geral dos recursos utilizados no processo.

Figura 18 – Diagrama de blocos dos componentes do sistema



Fonte: O autor (2020).

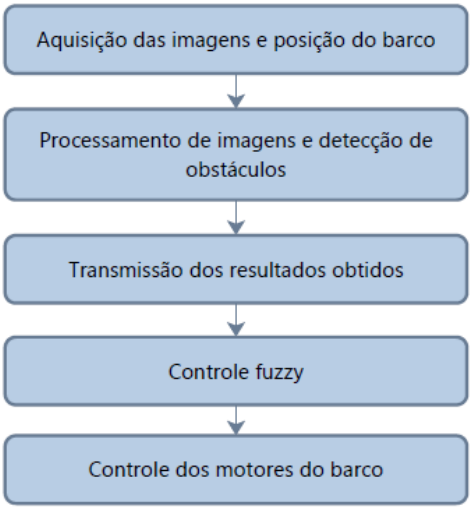
3.1 ARQUITETURA DO SISTEMA

Inicialmente realiza-se a aquisição das imagens do ambiente no qual o barco se encontra por meio uma câmera de infravermelho conectada a um computador externo. Esse computador é o responsável pelo processamento das imagens e pela

determinação de quais obstáculos devem ser desviados a partir de estimativas de seus tamanhos e distâncias. Simultaneamente, uma bússola eletrônica analisa a orientação do barco possibilitando determinar a diferença entre a sua direção atual e o sentido no qual deve se mover. Essas informações são transmitidas ao sistema embarcado presente no interior do veículo que, por meio de um controlador baseado em lógica *fuzzy,* define os movimentos dos motores necessários para a navegação do veículo.

Todos esses processos são realizados durante o deslocamento do barco pelo ambiente. A Figura 19 representa essas etapas através de um fluxograma.

Figura 19 – Arquitetura do sistema a ser implementado



Fonte: O autor (2020).

3.2 *HARDWARE*

Para realizar as etapas descritas no tópico anterior, foi necessário o dimensionamento de sensores e dispositivos responsáveis pela monitoração e análise dos dados referentes ao ambiente de navegação, conforme expresso na Figura 18. Dessa forma, a aquisição e o processamento das imagens do conjunto são realizados por meio de um computador disposto externamente, o que se fez necessário devido ao tamanho reduzido do barco não permitir comportá-lo. Através de uma rede sem fio, comunica-se com um sistema embarcado Raspberry PI 4, embutido no veículo, que faz a monitoração dos sensores e acionamento dos dispositivos presentes no protótipo.

A plataforma Raspberry PI 4 Model B consiste em um computador alocado a uma única placa, contando com conexões Bluetooth e antena Wi-Fi. Possui um processador Quad Core que opera a uma frequência de 1,5 GHz, memória RAM de 4 GB, quatro portas USB, duas portas micro-HDMI para comunicação de vídeo e 40 pinos para conexões de entradas e saídas. Também apresenta portas para acoplamento de câmera e display touchscreen, enquanto o sistema operacional e demais dados do sistema são armazenados em um cartão SD. (RASPBERRYPI, 2020)

O Raspberry PI 4 é o responsável pelo deslocamento da embarcação. Para isso, conta com um algoritmo baseado em lógica *fuzzy* que analisa os dados referentes aos obstáculos detectados, realizado pelo computador externo, e à posição do barco, com base nas informações obtidas por meio de comunicação serial *Inter-Integrated Circuit* (I^2C) com uma bússola eletrônica. A saída do sistema resulta em comandos de navegação, que são transmitidos a um módulo de acionamento com Ponte H que atua como uma interface para o acionamento dos motores.

As fotografias são registradas por uma câmera estéreo DUO MLX, fabricada pela DUO3D. Essa câmera captura pares de imagens monocromáticas a partir de iluminação infravermelho, emitida por 3 *Light-Emitting Diodes* (LEDs) programáveis presentes no equipamento. Suas lentes possuem um campo de visão horizontal de 165º, abertura focal de F/2.0 a F/2.1 e resolução fotográfica de até 752 x 480 pixels a 45 FPS. (DUO3D, 2020). Utilizando bibliotecas disponibilizadas pelo fabricante da câmera, esses pares de imagens são comparados para se gerar um mapa de profundidade, o que possibilita determinar as características de eventuais obstáculos à navegação do veículo.

Todos os recursos presentes no protótipo são energizados por uma fonte externa. Para a instalação dos dispositivos fez-se necessária a utilização de um modelo naval elétrico em dimensões reduzidas, do qual foram reaproveitadas sua estrutura mecânica e seus dois motores de propulsão, integrados ao controlador desenvolvido. O seu interior dispõe de isolação à prova de água, possibilitando a alocação das placas eletrônicas e fazendo com que apenas a câmera esteja exposta ao ambiente. A Figura 20 apresenta o modelo selecionado, que possui dimensões de aproximadamente 450 mm (comprimento) x 140 mm (largura) x 180 mm (altura).

Figura 20 – Modelo sobre o qual os dispositivos serão montados

Fonte: AMAZON (2020).

3.3 IMPLEMENTAÇÃO

O processo de implementação do *software* e controle do sistema consiste na execução de três operações, sendo elas a aquisição e o processamento de imagens para uma conseguinte detecção de obstáculos, a determinação da orientação do veículo utilizando uma bússola eletrônica o e o controle baseado em lógica *fuzzy*, responsável por analisar os dados obtidos nas duas etapas anteriores e determinar os comandos necessários para a sua navegação. Todas essas operações são executadas simultaneamente e trocam dados entre si. Os próximos tópicos abordam a realização dessas tarefas.

3.3.1 Posicionamento

Inicialmente foram realizados testes utilizando um módulo GPS GY-NEO6MV2 e verificou-se uma imprecisão significativa entre as leituras que, durante um dia ensolarado, causaram oscilações de até 1 m entre cada medição. Em dias nublados o erro aumentou para até 2 m. Esses resultados inviabilizaram sua aplicação devido à pequena escala do protótipo, tornando a utilização do GPS adequada somente em um sistema com as dimensões reais, onde essa variação é insignificante em relação ao tamanho do barco.

Assim, o controle da posição do veículo foi realizado apenas pela monitoração da direção na qual ele se desloca, por meio de uma bússola eletrônica HMC5883L. Esse módulo consiste no interfaceamento de sensores magnetoresistivos suscetíveis

à incidência de campos magnéticos de baixa intensidade, comumente utilizado para medição da direção e magnitude dos campos magnéticos terrestres. Dessa forma, ao serem energizados, os sensores convertem as variações eletromagnéticas em oscilações de tensão referentes a três eixos distintos, X, Y e Z, demonstrando a rotação do módulo em cada sentido. Também possui incluso em seu encapsulamento uma porta serial I²C, que possibilita a transmissão desses sinais após conversão analógico-digital por um ADC de 12 bits. (DIGIKEY, 2013)

No Raspberry PI 4, uma rotina escrita em linguagem de programação C realiza a comunicação por protocolo I²C com o módulo HMC5883L e recebe a rotação do eixo X que, no barco, representa o ângulo formado entre a sua proa e um polo magnético terrestre de referência. Na sequência, esse resultado é enviado à aplicação responsável pelo controlador *fuzzy* por meio de um endereço de memória compartilhado entre ambos.

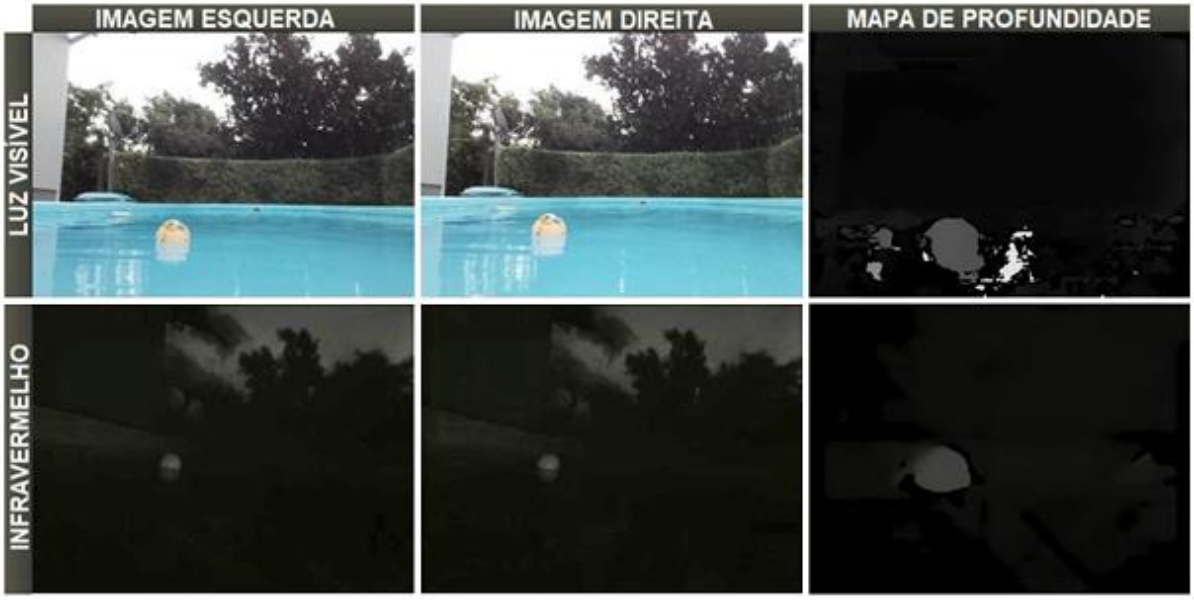
3.3.2 Detecção de obstáculos

A identificação de obstáculos à navegação do veículo se dá por meio de um sistema de visão estéreo. Dessa forma, além de detectar possíveis obstruções à trajetória do veículo, o *software* também é capaz de estimar a distância de cada uma delas em relação ao barco. A configuração da câmera e o conseguinte processamento automático das imagens foi realizado através de linguagens C/C++ e *OpenCV*, provendo-se de recursos exclusivos para esse tipo de aplicação.

Com a utilização de uma câmera de luz visível, o equipamento estaria sujeito a todo o tipo de influências externas como luz solar, oscilações luminosas, saturação, entre outros. Para a implementação de um mapa de profundidade, esses efeitos seriam agravados devido à comparação ponto a ponto entre duas imagens registradas, onde os efeitos da iluminação externa podem variar para cada câmera e, no caso de um barco em ambiente aquático, deve-se considerar também reflexos do sol ou de outros objetos sobre a água, dificultando, ou até mesmo inviabilizando, a identificação de obstáculos reais. Dessa forma optou-se pela utilização de infravermelho, onde a luz é emitida por LEDs embutidos na própria câmera, o que reduz consideravelmente os efeitos da iluminação externa e também possibilita a navegação do barco em ambientes escuros. A Figura 21 ressalta algumas das limitações de um mapa de profundidade gerado a partir de câmeras de luz visível em

uma superfície aquática, comparando-o com um modelo gerado por imagens infravermelhas.

Figura 21 – Mapas de profundidade com luz visível e infravermelho



Fonte: O autor (2020).

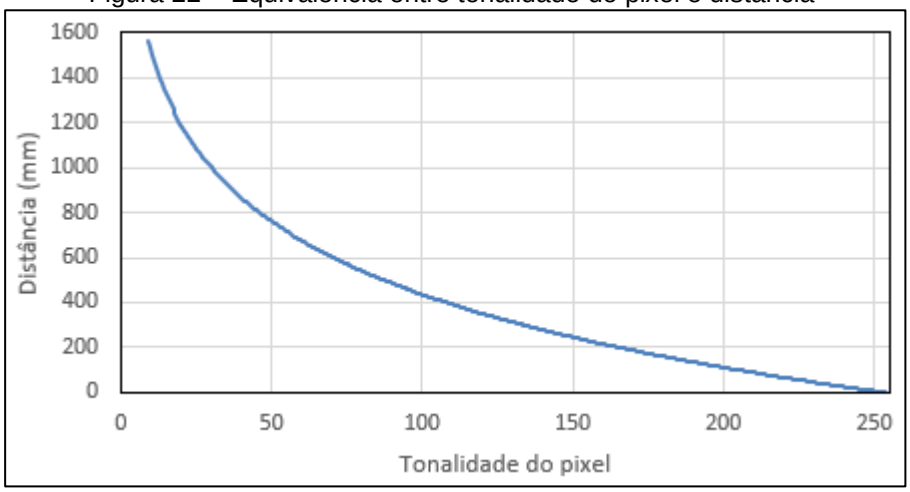
A câmera DUO MLX utilizada transmite as fotografias diretamente ao computador externo. Essa configuração fez-se necessária devido a problemas de compatibilidade de algumas bibliotecas da câmera com o Raspberry PI. Assim, o computador é o responsável pela aquisição das imagens, controle das configurações da DUO MLX, geração do mapa de profundidade, aplicação de filtros para adequar as figuras à aplicação e facilitar a identificação e classificação dos obstáculos, mapeamento das obstruções à navegação do barco e transmissão desses resultados, por meio de uma rede Wi-Fi, aos dispositivos presentes no veículo.

Inicialmente a câmera passou por um processo de calibração através do *software* DUO Calibration, disponibilizado pelo fabricante da câmera. Essa calibração é feita de forma automática a partir de pares de imagens registrados simultaneamente por ambas as lentes detectando um padrão, semelhante a um tabuleiro de xadrez, formado por diversos círculos de tamanhos conhecidos. O *software* calibra a câmera mapeando e comparando a posição de cada um desses círculos nas duas imagens, para em seguida registrar os resultados em um arquivo que é utilizado como base durante a aplicação.

Em seguida deve-se definir os parâmetros mais adequados da câmera para o processo. Essas configurações estão presentes no código escrito em C/C++, inicializando-os e os transferindo à câmera toda vez que se inicia a aplicação. Considerando-se as suas características padrões, fez-se necessário realizar alterações na intensidade luminosa dos LEDs de infravermelho, além do alcance e tempo de exposição da câmera de forma a otimizar sua operação em ambiente externo e a se adequar as dimensões e distâncias dos obstáculos.

De forma a se obter um desempenho satisfatório do sistema sem comprometer a sua confiabilidade na identificação dos obstáculos, estabeleceu-se a resolução das fotografias em 320 x 240 pixels sendo capturadas a uma taxa de 30 quadros por segundo. Durante a navegação do veículo, o mapa de profundidade é gerado a cada figura recebida, resultando em imagens em tons de cinza onde quanto mais clara for a cor do objeto mais próximo ele está do barco. Para a etapa do processamento digital das imagens a figura é tratada como uma matriz de 320 x 240 posições e cada uma delas possui valores de 0 (preto) a 255 (branco), de acordo com o tom de cada pixel. A Figura 22 relaciona tonalidade de cinza com a distância estimada, a partir de valores obtidos por medições práticas.

Figura 22 – Equivalência entre tonalidade do pixel e distância



Fonte: O autor (2020).

A cada cinco mapas de profundidade registrados é aplicado um filtro que busca remover ruídos e inconsistências. Para isso, desenvolveu-se uma rotina que realiza a comparação entre os pixels de todas essas cinco imagens referentes a cada posição matricial, sendo que seu valor será considerado apenas se 80%, ou 4/5, das figuras

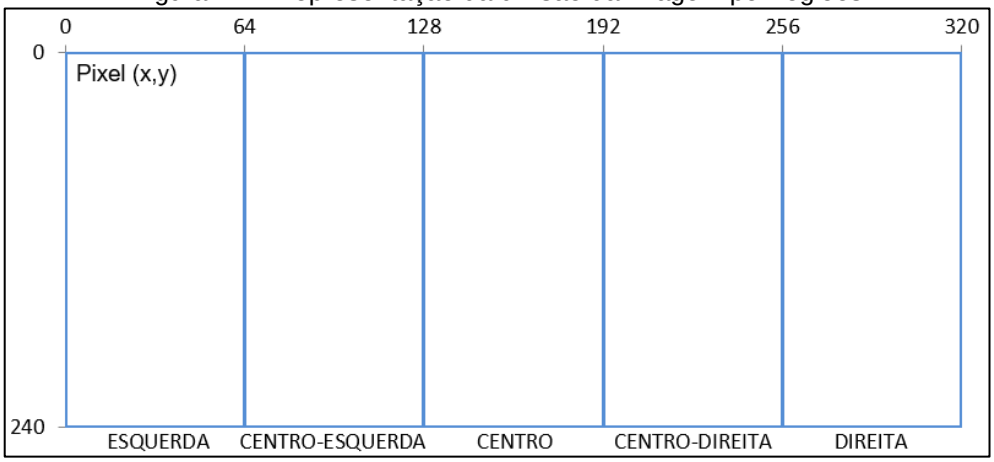
analisadas possuírem tonalidades semelhantes de cinza naquela coordenada. Consequentemente é gerada uma nova imagem derivada das anteriores, onde a cor do novo pixel, se validado, possuirá o tom médio dentre os cinco analisados. Caso o pixel seja desprezado pelo filtro aquela posição assumirá a cor preta, ou seja, o mesmo será considerado ruído e nenhum obstáculo será identificado naquela posição. Na Figura 23 compara-se o mapa de profundidade antes e após a aplicação do filtro.

Figura 23 – Comparação entre mapa de profundidade com e sem filtro

Fonte: O autor (2020).

A imagem é então dividida em cinco regiões, conforme demonstrado na Figura 24. Cada uma delas engloba 1/5 da área total, nas quais se identifica individualmente a presença e a distância geral dos obstáculos.

Figura 24 – Representação da divisão da imagem por regiões



Fonte: O autor (2020).

Assim, se busca o pixel mais claro de cada região, a partir dos quais é gerado um vetor de cinco posições que contém as informações sobre o ponto mais próximo da câmera em cada área da figura. Esse vetor é então enviado ao Raspberry PI 4 presente no barco através de uma rede Wi-Fi com o auxílio de *sockets* em protocolo *User Datagram Protocol* (UDP). Devido à taxa de 30 FPS das fotografias e ao processamento ser realizado uma vez a cada cinco frames, são enviados ao veículo seis vetores a cada segundo de navegação.

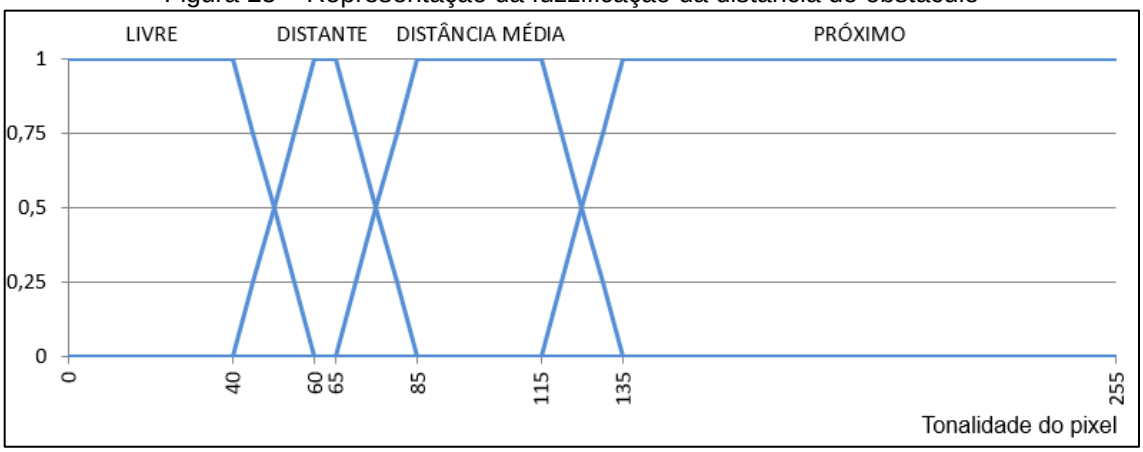
3.3.3 Controle de deslocamento

O controle de deslocamento do barco foi feito através de lógica *fuzzy*, por meio de um código escrito em linguagem C/C++. O resultado determina a direção de navegação do barco, através de diferentes formas de acionamento dos dois motores de propulsão. Para isso, são consideradas como variáveis de entrada do sistema a distância dos obstáculos em relação ao veículo e a diferença angular entre a sua orientação momentânea e o sentido designado para o seu deslocamento, ambos resultantes dos estágios anteriores.

Inicialmente ocorre o processo de fuzzificação das variáveis de entrada, que consiste em classificá-las em conjuntos de acordo com o seu grau de pertinência. Cada uma das cinco variáveis do vetor proveniente da etapa de processamento de imagens é classificada conforme a sua proximidade do barco, representado por valores de 0-255, e divididas em quatro conjuntos. O conjunto “LIVRE” engloba os obstáculos com afastamento superior à 770 mm, inicialmente não oferecendo nenhum

risco à sua navegação. Obstruções pertinentes à “DISTANTE” estão a uma distância entre 560 a 770 mm e começam a ser consideradas pelo processo, enquanto o grupo “DISTÂNCIA MÉDIA” representa aqueles que estão entre 330 a 560 mm. O conjunto “PRÓXIMO” indica que há bloqueios a menos de 330 mm, requerendo uma ação imediata por oferecer riscos eminentes à integridade da embarcação. A Figura 25 apresenta o processo de fuzzificação das variáveis referentes às distâncias dos obstáculos. As delimitações de cada conjunto foram estimadas e definidas a partir de ensaios práticos, nos quais se verificou a velocidade de reação e tempo necessário para a execução dos movimentos do veículo evitando qualquer tipo de colisão.

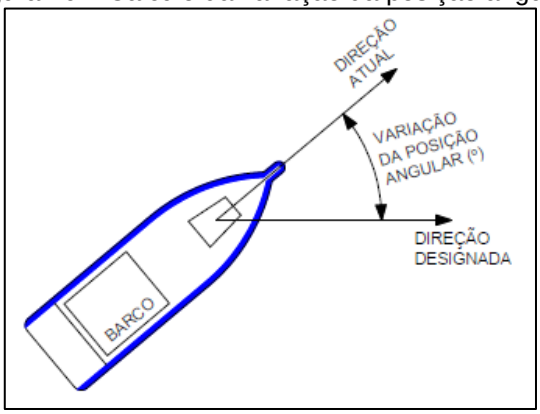
Figura 25 – Representação da fuzzificação da distância do obstáculo



Fonte: O autor (2020).

A variável seguinte refere-se ao direcionamento do barco, a partir do resultado da etapa de posicionamento e leitura da bússola eletrônica. Assim, foi definido um valor angular para a posição do veículo, utilizado como referência na navegação para tornar seu deslocamento, sempre que possível, uma linha reta. A entrada do sistema *fuzzy* corresponde à direção atual da embarcação subtraído da direção designada, resultando na divergência angular a ser corrigida. O diagrama apresentado na Figura 26 ilustra como é determinado do valor dessa variável através de um exemplo prático.

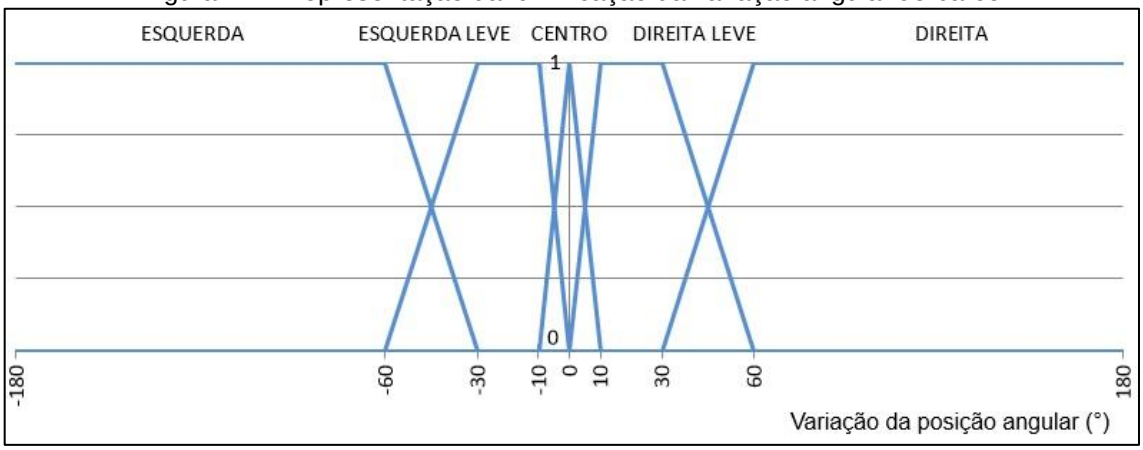
Figura 26 – Cálculo da variação da posição angular



Fonte: O autor (2020).

Em seguida o resultado obtido passa pelo processo de fuzzificação a partir do gráfico expresso na Figura 27. O controlador sempre buscará manter essa diferença em 0º, executando movimentos para o lado contrário ao valor da variável.

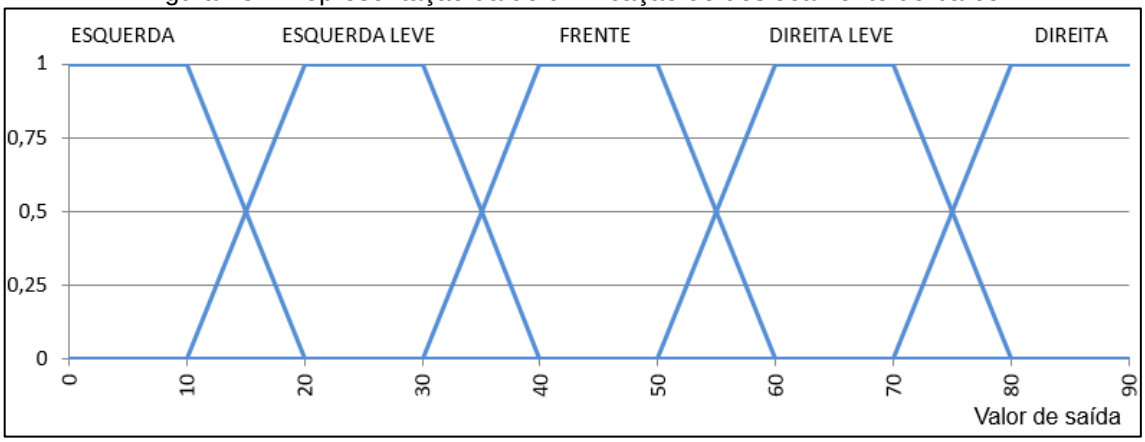
Figura 27 – Representação da fuzzificação da variação angular do barco



Fonte: O autor (2020).

O controle resulta em uma saída que determina a posição de navegação do barco para que o mesmo desvie de qualquer obstrução, procurando também manter o posicionamento angular definido anteriormente. Dessa forma, a defuzzificação consiste na classificação dos valores resultantes com base nos conjuntos demonstrados na Figura 28. Além desses grupos também é possível que o protótipo se desloque para trás, se afastando dos obstáculos, quando não houver nenhuma forma de contorná-los.

Figura 28 – Representação da defuzzificação do deslocamento do barco



Fonte: O autor (2020).

Após passarem pelo processo de fuzzificação, as variáveis de entrada são submetidas a um conjunto de regras nebulosas, definidas de acordo com as necessidades do projeto, que resultam na magnitude de cada um dos grupos de saída. Para facilitar a compreensão, as variáveis representadas nas regras estão abreviadas conforme demonstrado na Tabela 3.

Tabela 3 – Abreviação das variáveis apresentadas nas regras nebulosas

Variável	**Descrição**
u0	Distância dos obstáculos presentes na região esquerda
u1	Distância dos obstáculos presentes na região centro-esquerda
u2	Distância dos obstáculos presentes na região central
u3	Distância dos obstáculos presentes na região centro-direita
u4	Distância dos obstáculos presentes na região direita
VPA	Variação entre a posição angular atual e a designada

Fonte: O autor (2020).

Os tópicos a seguir expressam as regras estabelecidas, seguindo o modelo definido por “SE <condições> ENTÃO <conclusão>”. Os conjuntos resultantes foram designados como “ESQUERDA”, “ESQUERDA LEVE”, “FRENTE”, “DIREITA LEVE”, “DIREITA” e “RÉ”.

1- Se (u_3 é Distante ou u_3 é Livre) e (u_4 é Distante ou u_4 é Livre) então Direita é $VPA_{ESQUERDA}$
2- Se (u_3 é Distante ou u_3 é Livre) e (u_4 é Distante ou u_4 é Livre) então Direita Leve é $VPA_{ESQUERDA\ LEVE}$
3- Se (u_0 é Distante ou u_0 é Livre) e (u_1 é Distante ou u_1 é Livre) então Esquerda é $VPA_{DIREITA}$
4- Se (u_0 é Distante ou u_0 é Livre) e (u_1 é Distante ou u_1 é Livre) então Esquerda Leve é $VPA_{DIREITA\ LEVE}$
5- Se VPA é Centro e u_2 é Livre e (u_1 é Distante ou u_1 é Livre) e (u_3 é Distante ou u_3 é Livre) então Frente é $u_{2\ LIVRE}$
6- Se máx$[u_0, u_1] \leq$ máx$[u_3, u_4]$ e (u_3 é Distante ou u_4 é Distante) então Esquerda Leve é máx$[u_{3\ DISTANTE}, u_{4\ DISTANTE}]$
7- Se máx$[u_0, u_1] \leq$ máx$[u_3, u_4]$ e (u_3 é Distância Média ou u_4 é Distância Média) então Esquerda é máx$[u_{3\ DISTÂNCIA\ MÉDIA}, u_{4\ DISTÂNCIA\ MÉDIA}]$
8- Se máx$[u_0, u_1] \leq$ máx$[u_3, u_4]$ e (u_3 é Próximo ou u_4 é Próximo) então Esquerda é 1
9- Se máx$[u_0, u_1] \leq$ máx$[u_3, u_4]$ e (u_2 é Distante) então Esquerda Leve é $u_{2\ DISTANTE}$
10- Se máx$[u_0, u_1] \leq$ máx$[u_3, u_4]$ e (u_2 é Distância Média) então Esquerda é $u_{2\ DISTÂNCIA\ MÉDIA}$
11- Se máx$[u_3, u_4] \leq$ máx$[u_0, u_1]$ e (u_0 é Distante ou u_1 é Distante) então Direita Leve é máx$[u_{0\ DISTANTE}, u_{1\ DISTANTE}]$
12- Se máx$[u_3, u_4] \leq$ máx$[u_0, u_1]$ e (u_0 é Distância Média ou u_1 é Distância Média) então Direita é máx$[u_{0\ DISTÂNCIA\ MÉDIA}, u_{1\ DISTÂNCIA\ MÉDIA}]$
13- Se máx$[u_3, u_4] \leq$ máx$[u_0, u_1]$ e (u_0 é Próximo ou u_1 é Próximo) então Direita é 1
14- Se máx$[u_3, u_4] \leq$ máx$[u_0, u_1]$ e (u_2 é Distante) então Direita Leve é $u_{2\ DISTANTE}$
15- Se máx$[u_3, u_4] \leq$ máx$[u_0, u_1]$ e (u_2 é Distância Média) então Direita é $u_{2\ DISTÂNCIA\ MÉDIA}$
16- Se u_2 é Próximo ou [(u_0 é Próximo ou u_1 é Próximo) e (u_3 é Próximo ou u_4 é Próximo)] então Ré é 1

Dessa forma, as regras 1, 2, 3 e 4 analisam o deslocamento angular necessário para que o barco navegue na posição determinada, definindo as saídas de acordo com a distância dos obstáculos presentes naquela posição. Já a número 5 considera o veículo no alinhamento correto e o mantém naquela direção enquanto não houverem objetos próximos. As regras 6, 7, 8, 9 e 10 ignoram a sua orientação e comparam a proximidade das obstruções à esquerda e à direita para que, caso estejam mais próximos à direita, resulte em movimentos no sentido esquerdo. Quanto maior for a sua proximidade, maior será a necessidade de ação e mais bruscos serão os seus movimentos.

Assim como nos anteriores, os itens 11, 12, 13, 14 e 15 comparam o sentido dos obstáculos, porém essas condições serão executadas quando eles estiverem mais próximos à esquerda, o que solicitará o deslocamento do barco para a direita. A regra de número 16 determina que não há nenhuma direção possível para evitar colisões, ordenando o recuo imediato do veículo.

Obtidos os resultados do processo de aplicação das regras inicia-se a etapa de defuzzificação, que consiste em convertê-los para dados numéricos. Para isso, devido à utilização das mesmas classes de saída em diversas regras, se tomou como base apenas o maior valor obtido para cada uma. Caso o conjunto “RÉ” apresente valor 1, será a direção selecionada. Se não, se dá sequência ao processo de defuzzificação, utilizando como referência o método do Centro do Máximo que define um ponto intermediário entre os conjuntos de saída a partir dos valores máximos de cada um. Sua escolha deve-se ao sistema relacionar todos os conjuntos, diferente do critério pela Média dos Máximos que considera apenas os de maior valor, e por apresentar resposta semelhante ao modelo do Centro de Área, porém com *software* reduzido. Dessa forma, a saída foi calculada tendo como base a Equação 2.

Tomando como referência as curvas de defuzzificação representadas anteriormente na Figura 24, os valores numéricos presentes no numerador representam o somatório dos pontos intermediários sobre o eixo horizontal de cada conjunto multiplicado pelas suas respectivas magnitudes. No denominador há o somatório do valor máximo de saída de cada um dos conjuntos. Substituindo-se esses valores na Equação 2, obtém-se a Equação 4, que foi usada no processo para gerar um resultado numérico correspondente à direção de deslocamento do barco. Essa relação está expressa na Tabela 4, a qual se baseia nas delimitações de cada conjunto extraído da Figura 28.

$$Saída_{MOM} = \frac{5.Esq + 25.Esq\ Leve + 45.Frente + 65.Dir\ Leve + 85.Dir}{Esq + Esq\ Leve + Frente + Dir\ Leve + Dir} \quad (4)$$

Tabela 4 – Delimitação dos conjuntos de saída da lógica *fuzzy*

Resultado Numérico	Direção de Navegação
0-15	Esquerda
15-35	Esquerda Leve
35-55	Frente
55-75	Direita Leve
75-90	Direita

Fonte: O autor (2020).

Por fim, o controlador envia comandos para manejar quatro saídas do grupo de I/O presente no Raspberry PI 4 que, através do interfaceamento por um módulo de Ponte H, são responsáveis pelo acionamento e sentido de rotação de cada um dos dois motores de propulsão. A Tabela 5 relaciona cada comando possível com a direção do veículo.

Tabela 5 – Comandos de direção dos motores em relação às variáveis de saída

Direção	Motor Esquerdo	Motor Direito
Esquerda	Sentido inverso	Sentido normal
Esquerda Leve	Desligado	Sentido normal
Frente	Sentido normal	Sentido normal
Direita Leve	Sentido normal	Desligado
Direita	Sentido normal	Sentido inverso
Ré	Sentido inverso	Sentido inverso

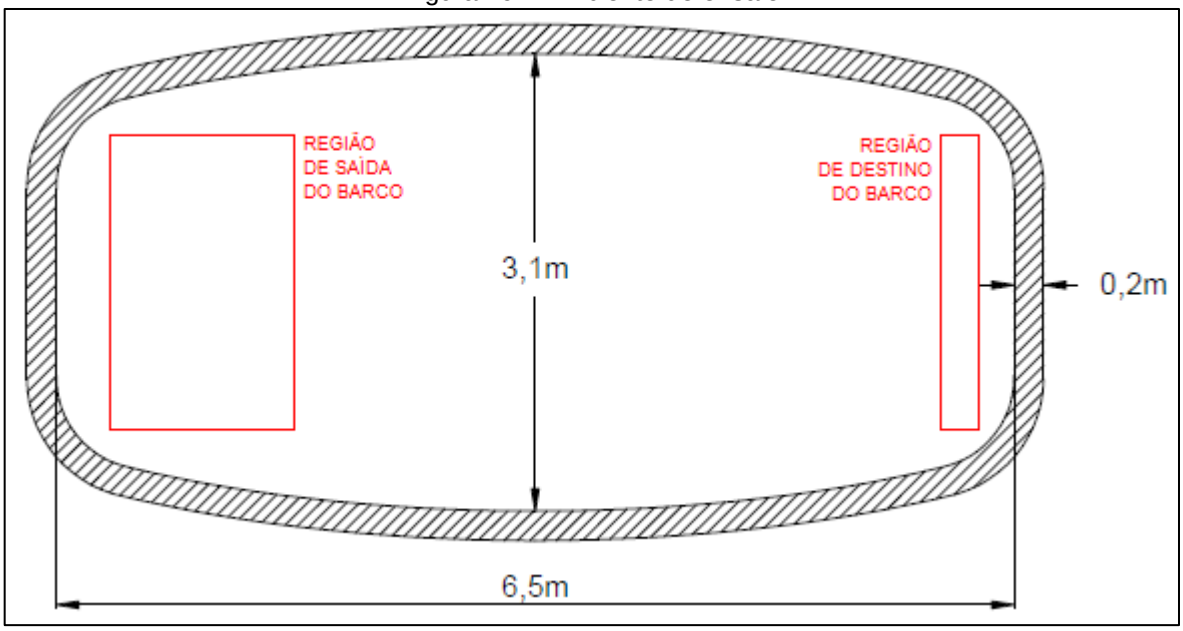
Fonte: O autor (2020).

3.4 ANÁLISE COMPORTAMENTAL DO PROJETO

A análise do projeto desenvolvido consistiu na movimentação autônoma da embarcação a partir de posições predefinidas, devendo desviar de quaisquer obstáculos ao seu deslocamento. Esses ensaios foram efetuados em uma piscina localizada em ambiente externo, cujas especificações estão demonstradas na Figura

29, com o barco partindo de pontos próximos à sua região lateral esquerda e se deslocando até a borda direita. Como o veículo se move continuamente em uma direção predeterminada e sem um ponto final específico, sua parada se deu a partir de comandos manuais para parada dos motores no momento em que o mesmo se encontrava próximo a borda direita, após já ter percorrido todo o trajeto ao longo da piscina.

Figura 29 – Ambiente de ensaio



Fonte: O autor (2020).

Foram então realizadas análises sob três premissas: a primeira onde somente o posicionamento do veículo foi avaliado, com seu deslocamento sendo limitado apenas pelas bordas da piscina. Na segunda foram elaborados ensaios com objetos estáticos sendo colocados previamente sobre a água, enquanto a terceira contou com obstáculos em constante movimento ou surgindo repentinamente em meio à trajetória do protótipo. Com base nessas propostas, foram analisados o comportamento e a tomada de decisão do veículo para cada situação.

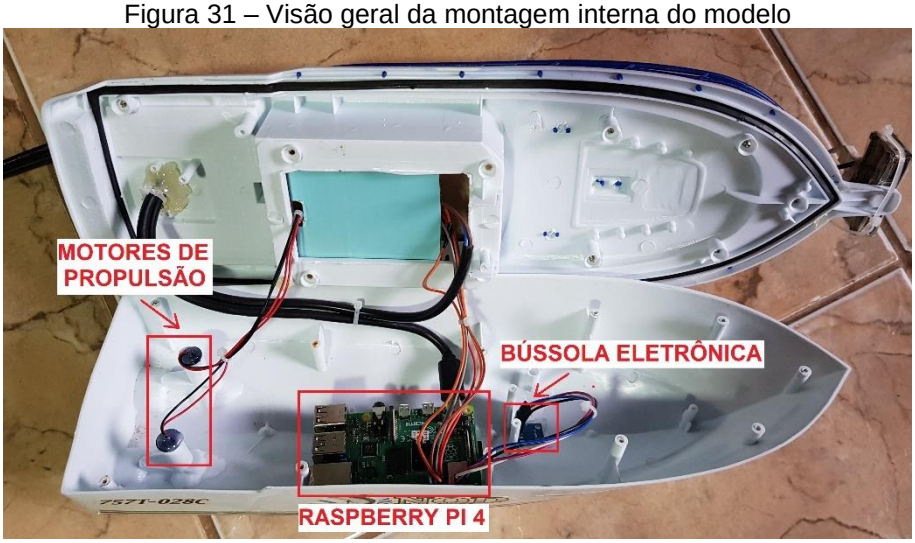
4 RESULTADOS

Nesta seção são abordados os resultados obtidos experimentalmente a partir da metodologia descrita no capítulo 4. Primeiramente realizou-se a montagem do protótipo, onde o Raspberry PI 4, a bússola eletrônica e o módulo de Ponte H foram alocados no interior do barco enquanto a câmera foi disposta externamente sobre a sua superfície frontal. Também foram removidas as partes desnecessárias da estrutura, deixando o modelo mais leve a fim de compensar o peso adicional dos equipamentos instalados. As Figuras 30 e 31 expõem uma visão geral do veículo após o processo de montagem.

Figura 30 – Visão geral da montagem externa do modelo

Fonte: O autor (2020).

Figura 31 – Visão geral da montagem interna do modelo



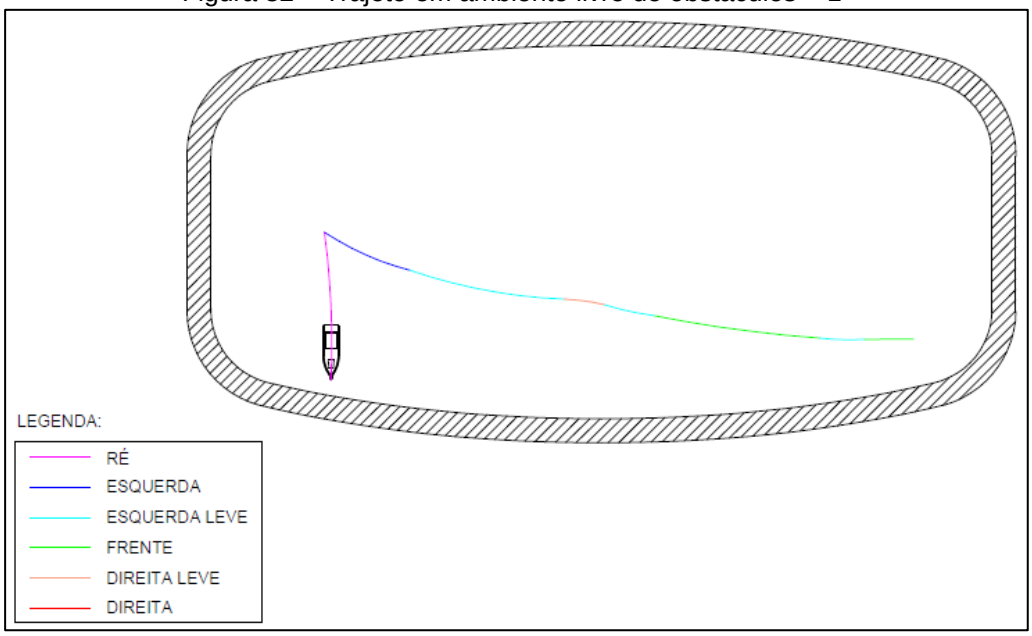
Fonte: O autor (2020).

O barco foi então submetido a ensaios práticos, dos quais serão apresentados mapeamentos detalhados da sua trajetória em uma piscina retangular de dimensões 6,5 x 3,1 m, conforme Figura 29. Para isso, foram traçados diagramas que representam o deslocamento do veículo através de um *software* de *Computer Aided Design* (CAD), onde sobrepôs-se linhas sobre vídeos registrados por uma câmera superior que filmou o trajeto do barco ao longo de todos os ensaios realizados.

4.1 DESLOCAMENTO EM AMBIENTE SEM OBSTÁCULOS ADICIONAIS

Inicialmente foi analisada a correção da variação angular do posicionamento do barco, onde o mesmo deveria se deslocar no sentido estabelecido evitando apenas as bordas da piscina. Para isso, definiu-se sua posição de destino como sendo a lateral direita da piscina, enquanto seu ponto de partida foi no lado oposto. A Figura 32 demonstra o seu comportamento ao sair de um ponto que faz ângulo de 90º com o designado. Nota-se que ao iniciar o processo o sistema já detectou a borda da piscina à sua frente, impossibilitando qualquer movimento. Assim, o controlador determinou o recuo do protótipo e, uma vez livre de quaisquer obstruções, o seu posicionamento passou a ser corrigido.

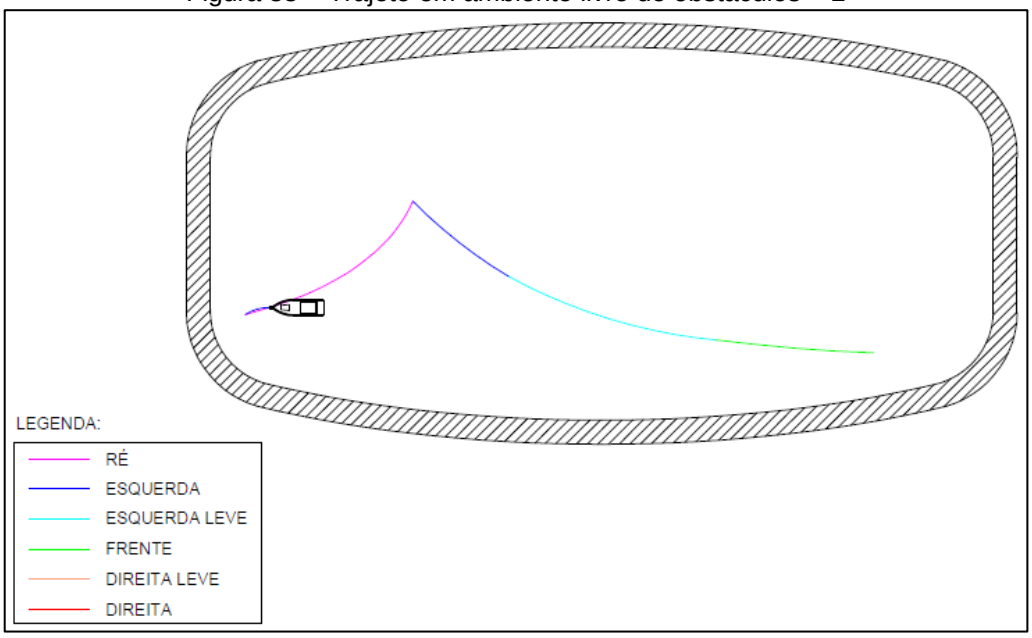
Figura 32 – Trajeto em ambiente livre de obstáculos – 1



Fonte: O autor (2020).

Em um segundo momento, a posição de partida do barco foi definida estando a 180º do destino designado. A Figura 33 detalha a sua trajetória, onde inicialmente o veículo tentou mover-se na direção esquerda, buscando corrigir o seu posicionamento e, ao constatar a impossibilidade de concluir o movimento antes de uma possível colisão com a borda, passou a se locomover para trás. Uma vez livre de obstáculos, a variação angular foi corrigida ao longo do deslocamento até a lateral direita da piscina.

Figura 33 – Trajeto em ambiente livre de obstáculos – 2

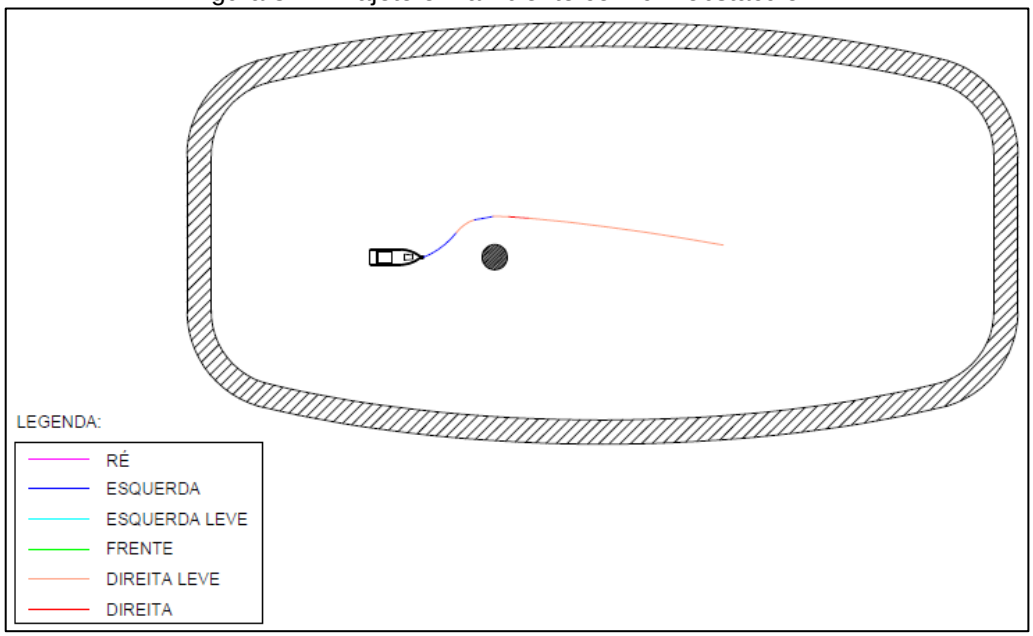


Fonte: O autor (2020).

4.2 DESLOCAMENTO COM OBSTÁCULOS ESTÁTICOS

Nesta etapa foi avaliada a tomada de decisão do barco com a adição de obstáculos estáticos em meio à sua trajetória. Primeiramente foi inserido um objeto esférico de diâmetro 210 mm, logo à frente do ponto de partida do veículo. Devido à sua proximidade, o desvio foi realizado tão logo quanto iniciaram seus movimentos. Ao determinar que a obstrução estava posicionada levemente à sua direita, o contorno deu-se pelo flanco esquerdo, conforme demonstrado na Figura 34.

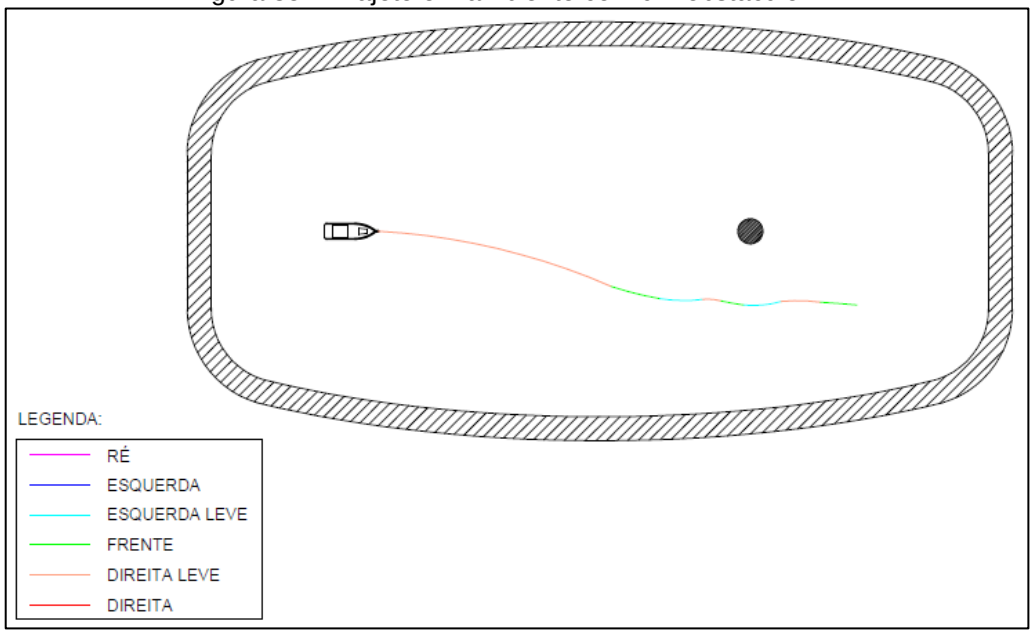
Figura 34 – Trajeto em ambiente com um obstáculo – 1



Fonte: O autor (2020).

Em seguida o mesmo obstáculo foi distanciado do barco, provendo maior tempo para o seu desvio. O resultado, expresso na Figura 35, demonstra movimentos menos bruscos se comparados ao ensaio anterior, devido ao trajeto do veículo ser corrigido antecipadamente. Porém, nota-se uma inconsistência na sua direção no momento em que a embarcação se encontrava próxima à lateral do objeto, executando movimentos à esquerda e à direita sequencialmente. Isso se deve ao fato do sistema considerar apenas obstruções visíveis, ou seja, no momento em que se deslocava para a direita, o obstáculo tornava-se invisível para a câmera, de forma que apenas a correção do posicionamento angular era considerada, ordenando movimentos no sentido esquerdo. Ao avistar novamente o obstáculo, o deslocamento era invertido a fim de desviá-lo mais uma vez. Esse processo repetiu-se até o momento em que o veículo ultrapassou a obstrução.

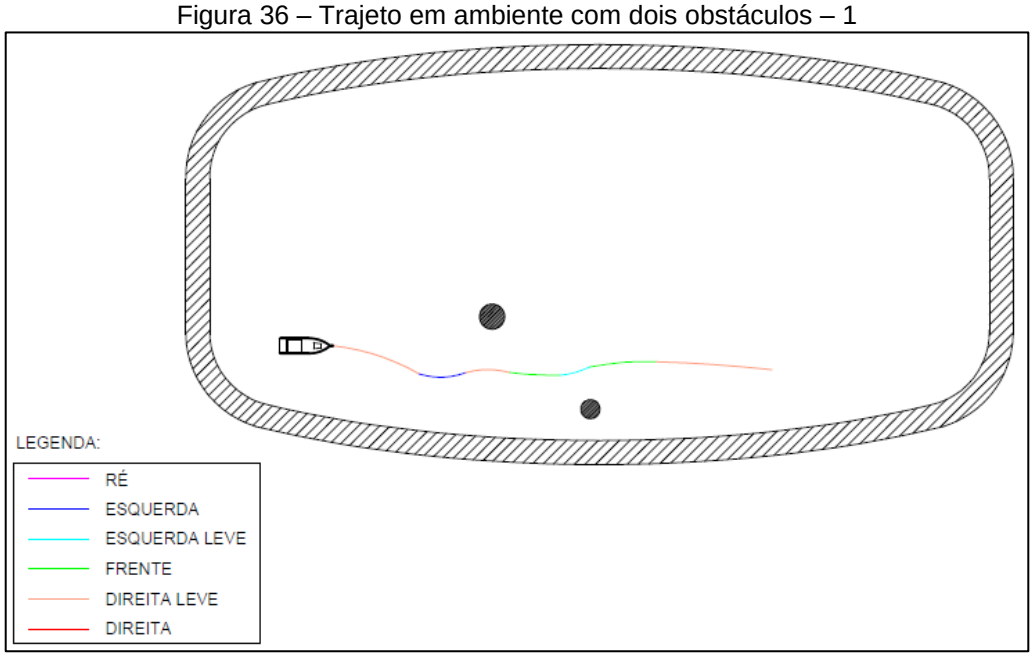
Figura 35 – Trajeto em ambiente com um obstáculo – 2



Fonte: O autor (2020).

Um segundo objeto, também esférico, porém com 160 mm de diâmetro, foi acrescentado ao ambiente posicionado logo após o primeiro, obrigando o barco a desviá-lo assim que ultrapassar o anterior. Seu trajeto está indicado na Figura 36, onde o veículo foi capaz de iniciar antecipadamente o desvio do primeiro e, ao se aproximar do segundo, corrigiu levemente sua posição à esquerda evitando uma possível colisão.

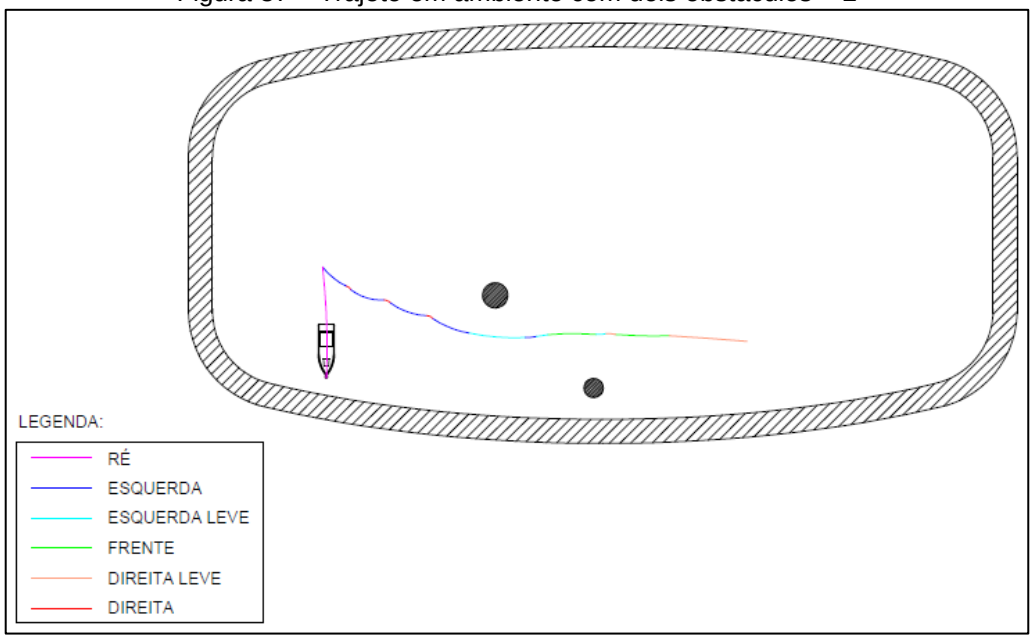
Figura 36 – Trajeto em ambiente com dois obstáculos – 1



Fonte: O autor (2020).

Considerando a mesma disposição de obstáculos utilizada no ensaio anterior, alterou-se o ponto de partida do barco, aproximando-o da borda inferior da piscina de modo a formar um ângulo de 90º em relação ao destino. Dessa forma, além de desviar dos obstáculos, o veículo precisou ter o seu posicionamento corrigido. Conforme demonstrado na Figura 37, o movimento inicial do protótipo foi afastar-se da barreira formada pela borda da piscina e, uma vez livre dela, buscou corrigir sua posição para a esquerda. Assim que detectado o primeiro objeto, teve seus movimentos alternados entre esquerda e direita, por desviar do obstáculo somente enquanto este estava visível, até ultrapassá-lo. Devido aos movimentos anteriores, bastou um leve desvio à esquerda para o barco ver-se livre da segunda obstrução.

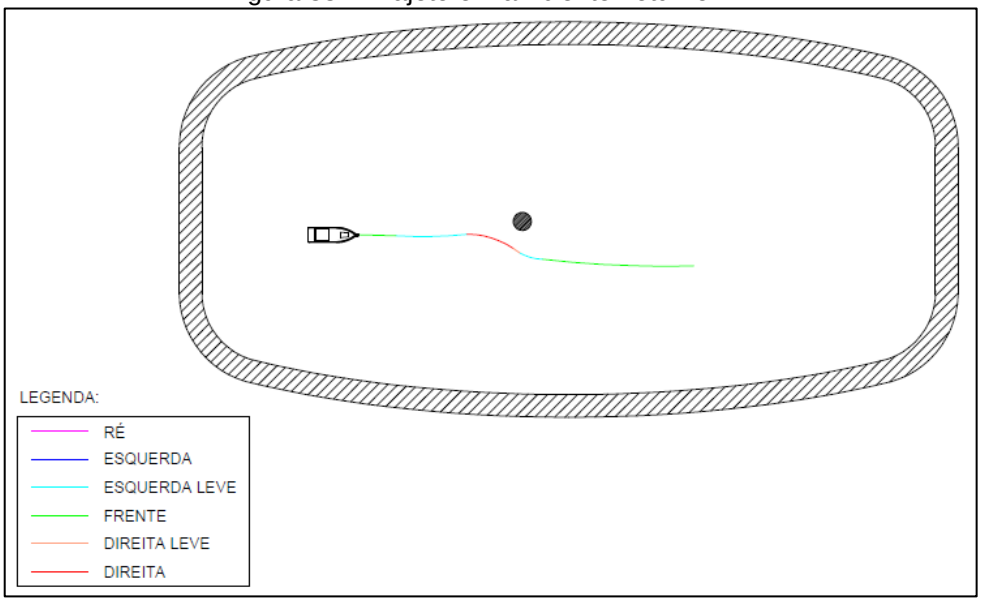
Figura 37 – Trajeto em ambiente com dois obstáculos – 2



Fonte: O autor (2020).

O ensaio seguinte foi realizado com apenas um obstáculo em ambiente noturno, a fim de se verificar os resultados da câmera de infravermelho em um local desprovido de qualquer iluminação externa. Analisando a Figura 38, nota-se que o veículo não foi capaz de antecipar seus movimentos, como ocorreu em ambientes diurnos, o que resultou no desvio do obstáculo apenas diante de uma colisão eminente. Isso se deu devido à configuração da intensidade luminosa dos LEDs emissores de infravermelho da câmera, que foi reduzida para se obter melhor performance em operações sob incidência de luz solar.

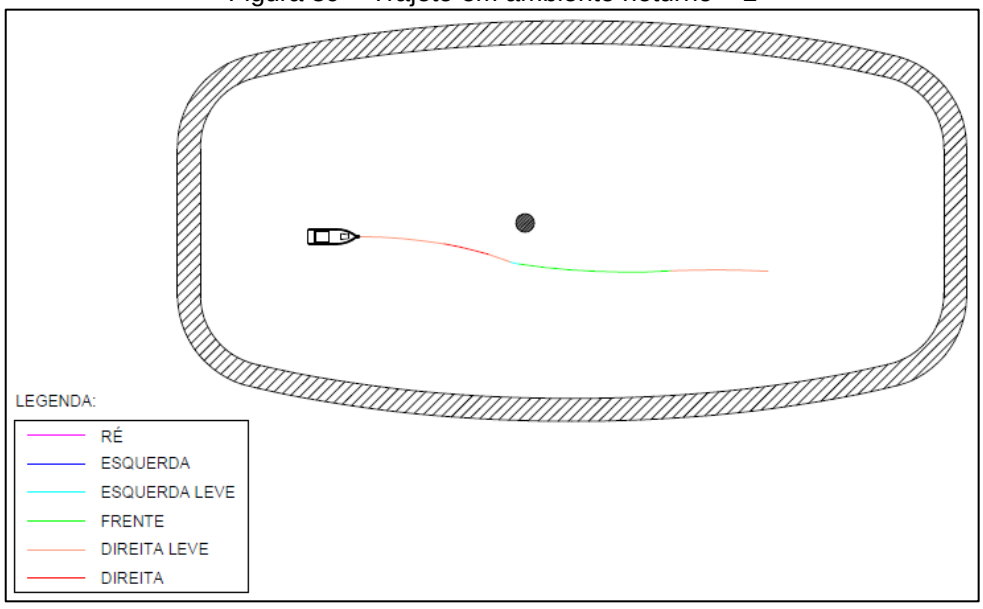
Figura 38 – Trajeto em ambiente noturno – 1



Fonte: O autor (2020).

Um segundo ensaio em ambiente noturno foi realizado, mantendo-se o barco e o obstáculo dispostos nas mesmas posições iniciais do ensaio anterior, porém com a câmera ajustada para melhor operar nessas condições. Para isso, a intensidade luminosa dos LEDs emissores de infravermelho foi aumentada em 40%. A Figura 39 mostra os resultados deste experimento, onde o veículo foi capaz de detectar os obstáculos a maiores distâncias, apresentando desempenho equivalente ao visto em ambientes diurnos.

Figura 39 – Trajeto em ambiente noturno – 2

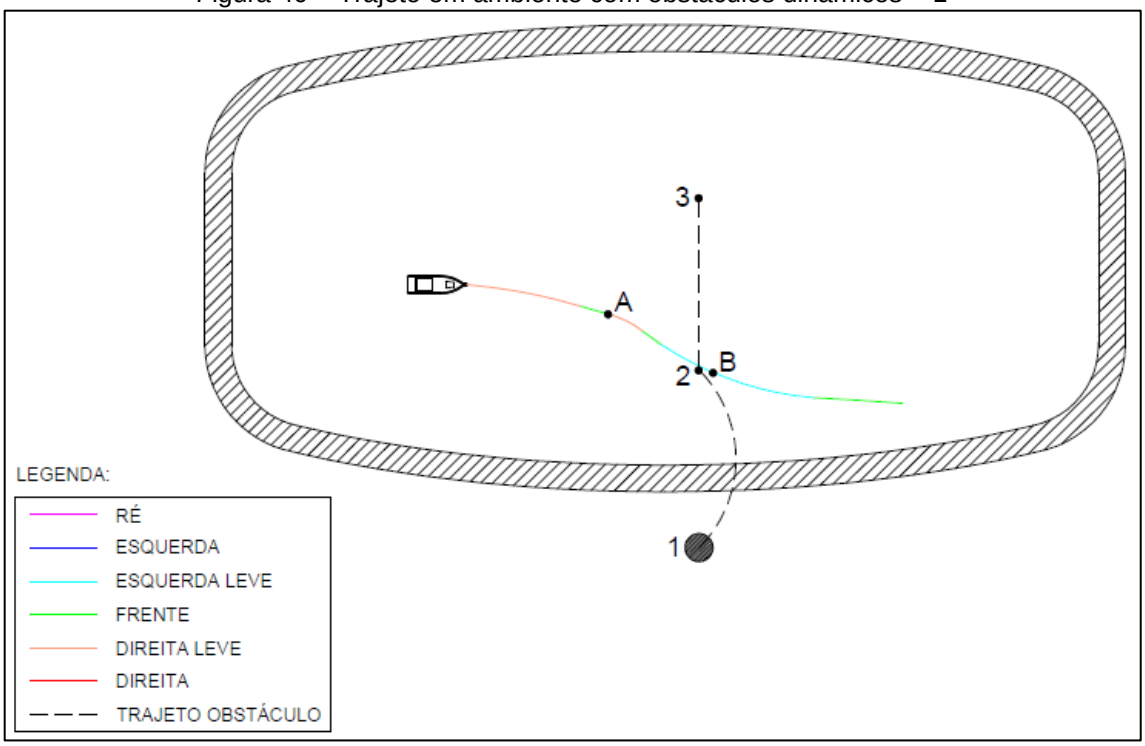


Fonte: O autor (2020).

4.3 DESLOCAMENTO COM OBSTÁCULOS DINÂMICOS

Os ensaios seguintes consistiram em analisar o comportamento do barco para obstáculos dinâmicos, ou seja, aqueles que surgem repentinamente em meio à sua trajetória. A Figura 40 ilustra o primeiro deles, onde o veículo parte de uma posição de repouso sem nenhuma obstrução no ambiente. Durante seu deslocamento, um objeto esférico de 210 mm de diâmetro foi lançado à sua frente, partindo do Ponto 1, fora da piscina, e chocou-se com a água no Ponto 2, ao mesmo tempo em que o barco passou pelo Ponto A. A embarcação então começou a desviar levemente à direita, a fim de evitar uma possível colisão com o novo obstáculo. A esfera continuou se movendo conforme trajeto representado pela linha tracejada até o Ponto 3, cruzando-o no momento em que o veículo passou pelo Ponto B. Assim, após contornar o objeto e estar livre de obstruções, o barco corrigiu sua posição e seguiu em linha reta com destino à lateral direita da piscina.

Figura 40 – Trajeto em ambiente com obstáculos dinâmicos – 1



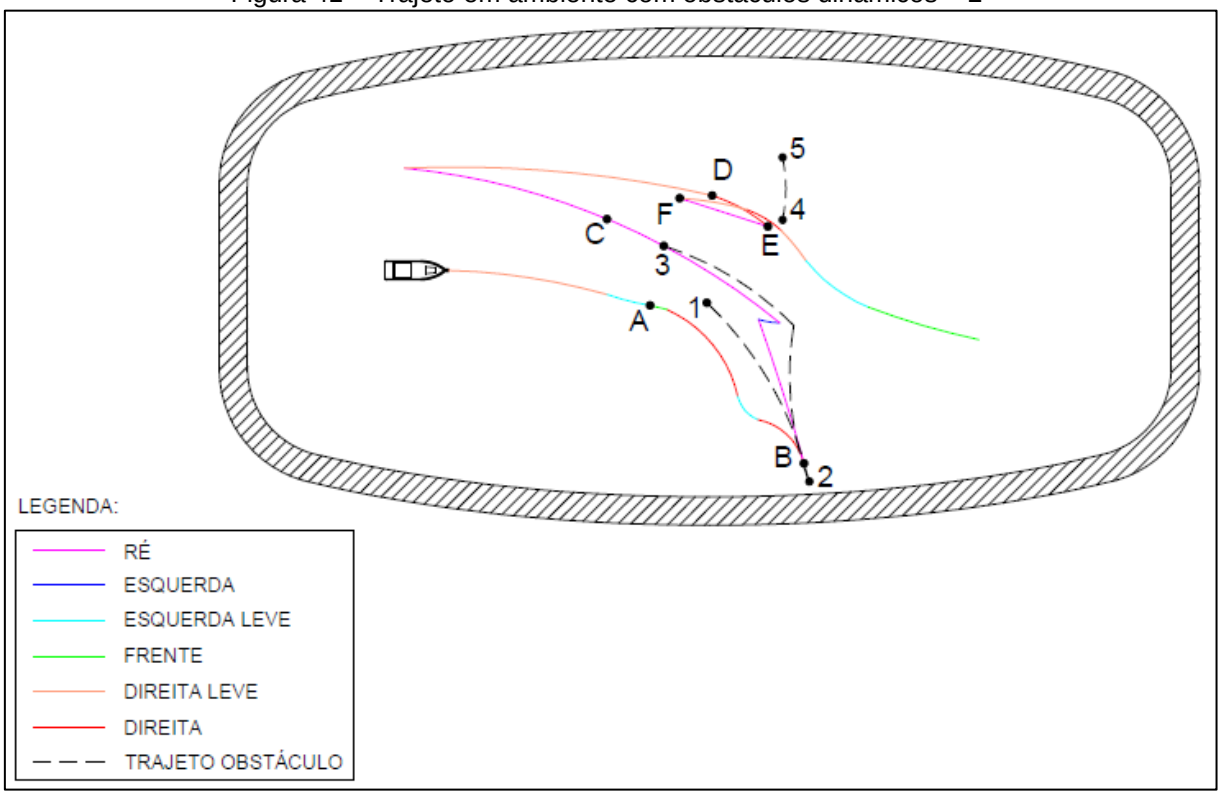
Fonte: O autor (2020).

O segundo ensaio envolvendo obstáculos em movimento baseou-se na utilização de uma haste com um retângulo de 400 x 300 mm na ponta, manuseada para bloquear o trajeto do veículo em determinados pontos e avançar contra seu

deslocamento em outros. Assim, conforme indicado na Figura 41, o barco partiu de uma posição inicial e ao se aproximar do ponto A inseriu-se o obstáculo no Ponto 1, deslocando-o linearmente até o Ponto 2. O veículo seguiu evitando-o enquanto navegava em direção ao Ponto B, onde o sistema determinou que não era mais possível evitar uma colisão e ordenou o seu recuo. Enquanto o barco retornou até o Ponto C, a barreira foi movida para o Ponto 3, conforme indicado pela linha tracejada, de forma a forçar seu retorno para a lateral esquerda da piscina.

O barco então retomou seu trajeto deslocando-se com destino à lateral direita e, ao cruzar o Ponto D, a barreira foi novamente inserida no ambiente, no local indicado pelo Ponto 4. A navegação seguiu até o Ponto E e, ao não conseguir contornar o obstáculo, recuou. Durante seu retorno até o Ponto F a barreira foi movida linearmente ao Ponto 5, permanecendo naquele local. Assim, a embarcação seguiu seu deslocamento movendo-se a direita para evitar o obstáculo. Após ultrapassá-lo, seu trajeto seguiu livremente até a extremidade oposta da piscina.[1]

Figura 41 – Trajeto em ambiente com obstáculos dinâmicos – 2



Fonte: O autor (2020).

[1] Partes desse e de outros testes de desempenho do barco podem ser conferidos em https://youtu.be/-xJN1b3Md4E.

5 CONCLUSÕES

Este trabalho apresentou como proposta um modelo de navegação de barco autônomo, no qual foram utilizados conceitos de sistemas de visão estéreo, controlador *fuzzy* e posicionamento através de bússola eletrônica. Embora o processo desenvolvido ainda não esteja pronto para ser aplicado em um modelo real, os resultados demonstraram uma precisa capacidade de detecção de obstáculos onde, durante os ensaios realizadas, o barco executou todos os movimentos necessários para contorná-los.

Os objetivos propostos, que consistiam principalmente na navegação em ambiente controlado, nos quais o barco deveria se deslocar no local em relação à direção designada desviando de quaisquer obstáculos, foram atingidos. Entretanto, ainda existem diversos aperfeiçoamentos e melhorias que podem ser realizados, de forma a adaptar sua aplicação a um barco autônomo real, sujeito a fatores adversos desconsiderados neste trabalho. Alguns exemplos disso são os efeitos causados por variações climáticas e tempestades, incidência de fortes ventos, instabilidade da embarcação, entre outros.

Dentre os pontos que obtiveram destaque na implementação do trabalho, podem-se citar o seu desempenho ao detectar obstáculos, com um sistema pouco suscetível à iluminação e demais influências externas, operando conjuntamente a um controlador *fuzzy* de baixa complexidade com atuação satisfatória, capaz de antecipar movimentos considerando obstruções em diferentes posições e distâncias para melhor se deslocar pelo ambiente. Também apresentou resposta rápida para obstruções com surgimento inesperado em meio à trajetória do veículo, provando-se sempre apto para contorná-las.

5.1 SUGESTÕES PARA TRABALHOS FUTUROS

Considerando os resultados obtidos, observaram-se fatores que poderiam ser melhorados em trabalhos futuros, visando aproximar-se de um modelo real de barco autônomo e a se adaptar melhor às condições do ambiente à que estaria sujeito. Assim, sugere-se principalmente acrescentar câmeras ou sensores nas regiões laterais e traseira do veículo, a fim de detectar obstáculos em atuais regiões cegas do protótipo. Também se recomenda a utilização de técnicas de mapeamento do local

para definir a melhor trajetória ao destino, de forma a evitar obstáculos já detectados anteriormente evitando o que, em certas ocasiões, resultou em movimentos dessincronizados do barco.

Devido às variações no alcance da detecção de obstáculos observada entre ambientes noturnos e diurnos, sugere-se a implementação de um sensor de luminosidade, buscando adaptar a intensidade dos LEDs emissores de infravermelho presentes na câmera de forma automática a partir da incidência luminosa detectada no ambiente. Durante os ensaios também se verificou algumas limitações no deslocamento do barco ocasionadas pelos cabos que conectam a câmera ao computador externo e alimentam os dispositivos presentes no barco. Dessa forma, aconselha-se sua remoção, por meio da utilização de um protótipo em escala maior a fim de comportar todo o *hardware* adequado à aplicação, bem como a sua energização por meio de um conjunto interno de baterias.

REFERÊNCIAS

ACZEL, Amir D. **Bússola: a invenção que mudou o mundo**. Rio de Janeiro: Zahar, 2002.

ALVES, José C.; CRUZ, Nuno A. **Um sistema computacional reconfigurável embarcado num veleiro autónomo**. Porto: FEUP, 2009.

AMAZON. **Lancha Barco De Controle Remoto Bateria Recarregável**. Disponível em: <https://www.amazon.com.br/Lancha-Controle-Remoto-Bateria-Recarreg%C3%A1vel/dp/B07VNTW3VJ>. Acesso em: 14 abr. 2020.

BOEING. **Boeing: Autonomous Systems**, 2020. Disponível em: <https://www.boeing.com/defense/autonomous-systems/index.page>. Acesso em: 29 maio 2020.

BRADSKI, Gary; KAEHLER, Adrian. **Learning OpenCV**. 1. ed. O'Reilly Media, 2008.

BRAGA, Antônio de Pádua; CARVALHO, André Carlos Ponce de Leon Ferreira de; LUDERMIR, Teresa Bernarda. **Redes Neurais Artificiais**. In: REZENDE, Solange Oliveira. (Org.). **Sistemas Inteligentes:** fundamentos e aplicações. 1. ed. Barueri, SP: Manole, 2003. p. 141-168.

CARVALHO, Edilson Alves de; ARAÚJO, Paulo César de. **Noções básicas de sistema de posicionamento global GPS**. Natal: EDUFRN, 2002.

CORREIA, Carlos. **Sistemas de Telecomunicações II**, 2003. Disponível em: < https://www.azoresuperyachtservices.pt/images/Download_pt/NAVEGACAO/GPS/GPS%20-%20FEUP.pdf>. Acesso em: 31 maio 2020.

DARPA. **ACTUV "Sea Hunter" Prototype Transitions to Office of Naval Research for Further Development**, 2018. Disponível em: <https://www.darpa.mil/news-events/2018-01-30a>. Acesso em: 4 jun. 2020.

DEFENSE SYSTEMS. **Navy anti-submarine drone-ship conducts minehunting testing**, 2017. Disponível em: <https://defensesystems.com/articles/2017/05/04/seahunter.aspx>. Acesso em: 4 jun. 2020.

DETRAN. **Veja as principais causas de acidentes nas vias e rodovias**, 2016. Disponível em: <http://www.detran.ms.gov.br/veja-as-principais-causas-de-acidentes-nas-vias-e-rodovias/>. Acesso em: 29 maio 2020.

DIGIKEY. **3-Axis Digital Compass IC HMC5883L**, 2013. Disponível em: <https://media.digikey.com/pdf/Data%20Sheets/Honeywell%20PDFs/HMC5883L.pdf>. Acesso em: 01 abr. 2020.

DUO3D. **DUO MLX**. Disponível em: < https://duo3d.com/docs/duo-mlx/>. Acesso em: 27 abr. 2020.

FEUP. **FASt Autonomous Sailboat**, 2014. Disponível em: < https://web.fe.up.pt/~jca/roboticsailing.pt/>. Acesso em: 3 jun. 2020.

FILHO, Ogê Marques; NETO, Hugo Vieira. **Processamento digital de imagens**. Rio de Janeiro: Brasport, 1999.

FURTADO, Vitor Hugo et al. **Aspectos de segurança na integração de veículos aéreos não tripulados (VANT) no espaço aéreo brasileiro**. São Paulo: Sitraer, 2008.

GONZALEZ, Rafael C.; WOODS, Richard C. **Processamento digital de imagens**. 3. ed. São Paulo: Pearson Prentice Hall, 2010.

HAYKIN, Simon. **Redes Neurais: Princípios e práticas**. 2. ed. Porto Alegre: Bookman, 2008.

HEINEN, Farlei José. **Sistema de Controle Híbrido para Robôs Móveis Autônomos**. 2002. 130f. Dissertação (Mestrado) – Universidade do Vale do Rio dos Sinos, São Leopoldo, 2002.

JUNG, Cláudio Rosito et al. **Computação Embarcada: Projeto e Implementação de Veículos Autônomos Inteligentes**. São Leopoldo, RS: SBC, 2005.

KONGSBERG. **Autonomous ship project, key facts about YARA Birkeland**, 2020. Disponível em: <https://www.kongsberg.com/maritime/support/themes/autonomous-ship-project-key-facts-about-yara-birkeland/>. Acesso em: 4 jun. 2020.

LEITÃO, Mário Jorge M. **Sistemas de Telecomunicações II:** Sistemas de Radar. Disponível em: <https://web.fe.up.pt/~mleitao/SRCO/Teoricas/SRCO_RAD.pdf>. Acesso em: 31 maio 2020.

LENZ, James E. **A Review of Magnetic Sensors**. IEEE, 1990.

LIMA, Eliomar Araújo de. **Sistemas para Localização de Pessoas e Objetos em Ambientes Indoor**. Rio de Janeiro: PUC-Rio, 2001.

MACIEL, Ariana de Oliveira. **Aplicações: Mapeamento Móvel utilizando tecnologia LIDAR**. Curitiba: SBSR, 2011.

MENDES, Caio César Teodoro. **Navegação de robôs móveis utilizando visão estéreo**. 2012. 86f. Dissertação (Mestrado) – Universidade de São Paulo, Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional, São Paulo, 2012.

MONARD, Maria Carolina; BARANAUSKAS, José Augusto. **Conceitos sobre Aprendizado de Máquina**. In: REZENDE, Solange Oliveira. (Org.). **Sistemas Inteligentes:** fundamentos e aplicações. 1. ed. Barueri, SP: Manole, 2003. p. 89-114.

NEWATLAS. **Autonomous boat gets around under wave Power**, 2018. Disponível em: <https://newatlas.com/autonaut-autonomous-unmanned-surface-vessel/53949/>. Acesso em: 30 maio 2020.

OZGUNER, Umit; STILLER, Christoph; REDMILL, Keith. **Systems for Safety and Autonomous Behavior in Cars: The DARPA Grand Challenge Experience**. IEEE, 2007.

PEREIRA, Thiago José de Almeida. **eVentos3 – Desenvolvimento de controlador difuso para navegação autónoma de veleiro**. 2015. 95f. Dissertação (Mestrado) – Universidade Nova de Lisboa, Lisboa, 2015.

PINTO, Juliano Tusi Amaral Laganá; PARRO, Vanderlei Cunha. **Sistema de visão estéreo em tempo real**. EEM, 2013.

PISSARDINI, Rodrigo de Souza; WEI, Daniel Chin Min; JÚNIOR, Edvaldo Simões da Fonseca. **Veículos autônomos: conceitos, histórico e estado-da-arte**. Belém: ANPET, 2013.

RASPBERRYPI. **Raspberry PI 4**. Disponível em: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>. Acesso em: 19 abr. 2020.

REZENDE, Solange Oliveira. **Sistemas Inteligentes:** fundamentos e aplicações. 1. ed. Barueri, SP: Manole, 2003.

RIOS, Luiz Romário Santana. **Visão Computacional**. Salvador: UFBA, 2010.

RUSSEL, Stuart; NORVIG, Peter. **Inteligência Artificial**. 1. ed. Rio de Janeiro: Elsevier, 2004.

SAE. **Levels of driving automation are defined in new SAE International standard J3016**, 2014. Disponível em: <https://www.sae.org/binaries/content/assets/cm/content/news/press-releases/pathway-to-autonomy/automated_driving.pdf>. Acesso em: 30 maio 2020.

SCHARSTEIN, Daniel; SZELISKI, Richard. **A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms**. IEEE, 2002.

SHAW, Ian S.; SIMÕES, Marcelo Godoy. **Controle e modelagem fuzzy**. 1. ed. São Paulo: Edgard Blucher, 1999.

STORK, Thomas. **Electronic Compass Design using KMZ51 and KMZ52**, 2001. Disponível em: <https://pdfs.semanticscholar.org/ad20/e5c06b4524fdef0f1dee5b83641822abd609.pdf>. Acesso em: 31 maio 2020.

TANSCHEIT, Ricardo. **Sistemas fuzzy**. Rio de Janeiro: PUC-Rio, 2004.

UDACITY. **Carros autônomos: entenda o funcionamento e a construção do transporte do futuro**, 2018a. Disponível em: <https://br.udacity.com/blog/post/carros-autonomos-funcionamento>. Acesso em: 15 out. 2018.

UDACITY. **Como veículos autônomos e cidades inteligentes se conectam**, 2018b. Disponível em: <https://br.udacity.com/blog/post/cidades-inteligentes-e-carros-autonomos>. Acesso em: 20 set. 2018.

UFRGS. **Medindo velocidades**. Disponível em: <http://www.if.ufrgs.br/oei/cgu/doppler.htm>. Acesso em: 30 maio 2020.

VOLVO. **Automated Trucks**, 2020. Disponível em: <https://www.volvotrucks.com/en-en/about-us/automation.html>. Acesso em: 3 maio 2020.

WAISELFISZ, Julio Jacobo. **Mapa da Violência 2013:** Acidentes de Trânsito e Motocicletas. CEBELA, 2013.

WIRED. **Forget Robo-Cars and Hit the Water on an Autonomous Boat**, 2018. Disponível em: <https://www.wired.com/story/self-driving-ships-boats/>. Acesso em: 30 maio 2020.

YARA. **Yara Birkeland press kit**, 2020. Disponível em: <https://www.yara.com/news-and-media/press-kits/yara-birkeland-press-kit/>. Acesso em: 4 jun. 2020.

APÊNDICE A – ALGORITMO PARA DETECÇÃO E CLASSIFICAÇÃO DOS OBSTÁCULOS

O algoritmo a seguir é o único a rodar no computador externo. Tem como objetivo capturar as imagens da câmera de infravermelho, gerar o mapa de profundidade e aplicar filtros para localizar e classificar obstáculos à navegação do veículo. Assim, gera um vetor de cinco posições, cada um deles referente à proximidade dos obstáculos em uma das regiões da imagem, e os envia ao Raspberry PI por meio de *sockets* UDP.

```
1.  #include "Sample.h"
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <unistd.h>
5.  #include <arpa/inet.h>
6.  #include <string.h>
7.
8.  //Definição da resolução das imagens e frames por segundo
9.  #define WIDTH   320
10. #define HEIGHT  240
11. #define FPS     30
12.
13. //Endereço para comunicação através de UDP
14. #define PORTSUPERVISORIO 2302
15.
16. //Estrutura para geração de mapa de profundidade colorido, onde cores
17. //quentes representam objetos próximos e cores frias objetos distantes
18. //Não será usado no projeto
19. Vec3b HSV2RGB(float hue, float sat, float val)
20. {
21.     float x, y, z;
22.     if(hue == 1) hue = 0;
23.     else         hue *= 6;
24.
25.     int i = static_cast<int>(floorf(hue));
26.     float f = hue - i;
27.     float p = val * (1 - sat);
28.     float q = val * (1 - (sat * f));
29.     float t = val * (1 - (sat * (1 - f)));
30.
31.     switch(i)
32.     {
33.         case 0: x = val; y = t; z = p; break;
34.         case 1: x = q; y = val; z = p; break;
35.         case 2: x = p; y = val; z = t; break;
36.         case 3: x = p; y = q; z = val; break;
37.         case 4: x = t; y = p; z = val; break;
38.         case 5: x = val; y = p; z = q; break;
39.     }
40.     return Vec3b((uchar)(x * 255), (uchar)(y * 255), (uchar)(z * 255));
41. }
42.
43. int main(int argc, char* argv[])
44. {
45.     //Variaveis referentes aos sockets
46.     socklen_t socketsize; //Recebe o tamanho da estrutura sockaddr_in
```

```
47.     int socket_supervisorio;
48.     struct sockaddr_in endereco_supervisorio;
49.
50.     //Demais variáveis referentes à captura de imagens e aplicação de filtros
51.     int cont=0;
52.     int contFrames=0;
53.     char saida[5];
54.     unsigned char pixel[5];
55.     Mat filter[5] = Mat(Size(WIDTH, HEIGHT), CV_8UC1);
56.
57.     //Variaveis para geração de mapa de profundidade colorido - Não será utilizado
58.     Mat colorLut = Mat(cv::Size(256, 1), CV_8UC3);
59.     for(int i = 0; i < 256; i++)
60.         colorLut.at<Vec3b>(i) = (i==0) ? Vec3b(0, 0, 0) : HSV2RGB(i/256.0f, 1, 1);
61.
62.     //Inicia câmera e captura de imagens
63.     if(!OpenDUOCamera(WIDTH, HEIGHT, FPS))
64.     {
65.         printf("Could not open DUO camera\n");
66.         return 1;
67.     }
68.
69.     printf("DUOLib Version:       v%s\n", GetDUOLibVersion());
70.     printf("Dense3D Version:      v%s\n", Dense3DGetLibVersion());
71.
72.     //Inicia bibliotecas para gerar mapa de profundidade
73.     Dense3DInstance dense3d;
74.     if(!Dense3DOpen(&dense3d))
75.     {
76.         printf("Could not open Dense3D library\n");
77.         CloseDUOCamera();
78.         return 1;
79.     }
80.
81.     //Insere licença para utilização das bibliotecas
82.     if(!SetDense3DLicense(dense3d, "XXXX-XXXX-XXXX-XXXX-XXXX"))
83.     {
84.         printf("Invalid or missing Dense3D license. To get your license visit https:
    //duo3d.com/account\n");
85.         CloseDUOCamera();
86.         Dense3DClose(dense3d);
87.         return 1;
88.     }
89.
90.     //Define dimensões das imagens
91.     if(!SetDense3DImageSize(dense3d, WIDTH, HEIGHT))
92.     {
93.         printf("Invalid image size\n");
94.         CloseDUOCamera();
95.         Dense3DClose(dense3d);
96.         return 1;
97.     }
98.
99.     //Carrega parâmetros da calibração realizada
100.    DUO_STEREO params;
101.    if(!GetCameraStereoParameters(&params))
102.    {
103.        printf("Could not get DUO camera calibration data\n");
104.        CloseDUOCamera();
105.        Dense3DClose(dense3d);
106.        return 1;
107.    }
108.
109.    //Define parâmetros da câmera
110.    SetDense3DScale(dense3d, 3);//3
111.    SetDense3DMode(dense3d, 1);//0 //1
```

```
112.     SetDense3DCalibration(dense3d, &params);
113.     SetDense3DNumDisparities(dense3d, 2);//4
114.     SetDense3DSADWindowSize(dense3d, 6);//6
115.     SetDense3DPreFilterCap(dense3d, 28);//28
116.     SetDense3DUniquenessRatio(dense3d, 27);//27
117.     SetDense3DSpeckleWindowSize(dense3d, 52);//52
118.     SetDense3DSpeckleRange(dense3d, 14);//14
119.     //Define exposição, intensidade dos LEDs de infravermelho e orientação da câmera

120.     SetExposure(10);//Valores mais altos melhoram as imagens em abientes escuros
121.     SetLed(20);//Ambiente escuro 28 / Ambiente claro 20
122.     SetVFlip(false);
123.     SetUndistort(true);
124.
125.     //Cria matrizes para armazenar imagens e mapas de profundidade
126.     Mat1f disparity = Mat(Size(WIDTH, HEIGHT), CV_32FC1);
127.     Mat3f depth3d = Mat(Size(WIDTH, HEIGHT), CV_32FC3);
128.
129.     //Inicializa sockets
130.     socket_supervisorio = socket(AF_INET, SOCK_DGRAM,0);
131.     socketsize = sizeof(struct sockaddr_in); //Guarda o tamanho da estrutura sockadd
    r_in
132.     endereco_supervisorio.sin_family = AF_INET;
133.     endereco_supervisorio.sin_addr.s_addr = inet_addr("192.168.1.199"); //IP do serv
    idor para o qual enviara pacotes
134.     endereco_supervisorio.sin_port=htons(PORTSUPERVISORIO); //Define a porta de escu
    ta da conexão
135.     memset(&(endereco_supervisorio.sin_zero),'\0',sizeof(endereco_supervisorio.sin_z
    ero)); //Zera o resto da estrutura
136.
137.     //Inicia loop de captura das imagens
138.     while((cvWaitKey(1) & 0xff) != 27)
139.     {
140.         //Captura frame
141.         PDUOFrame pFrameData = GetDUOFrame();
142.         if(pFrameData == NULL) continue;
143.
144.         //Cria matrizes para receber imagens da câmera
145.         Mat left = Mat(Size(WIDTH, HEIGHT), CV_8UC1, pFrameData->leftData);
146.         Mat right = Mat(Size(WIDTH, HEIGHT), CV_8UC1, pFrameData->rightData);
147.
148.         //Gera mapa de profundidade em tons de cinza
149.         if(Dense3DGetDepth(dense3d, pFrameData->leftData, pFrameData-
    >rightData, (float*)disparity.data, (PDense3DDepth)depth3d.data))
150.         {
151.             uint32_t disparities;
152.             GetDense3DNumDisparities(dense3d, &disparities);
153.             Mat disp8;
154.             disparity.convertTo(disp8, CV_8UC1, 255.0/(disparities*16));
155.
156.             //Converte mapa de profundidade em tons de cinza para mapa de profundida
    de colorido
157.             //Mat mRGBDepth;
158.             //cvtColor(disp8, mRGBDepth, COLOR_GRAY2BGR);
159.             //LUT(mRGBDepth, colorLut, mRGBDepth);
160.
161.             //Exibe janela com mapa de profundidade gerado
162.             imshow("Dense3D Disparity Map Greyscale", disp8);
163.             //Exibe janelas com imagens capturada pelas câmeras esquerda e direita
164.             imshow("Left", left);
165.             imshow("Right", right);
166.
167.
168.             // === Inicia processamento de imagem ===
169.
170.             //Armazena os 5 últimos frames para aplicação de filtros
```

```
171.            filter[4] = filter[3];
172.            filter[3] = filter[2];
173.            filter[2] = filter[1];
174.            filter[1] = filter[0];
175.            filter[0] = disp8;
176.            contFrames = contFrames+1;
177.
178.            //Define matriz para receber dados do mapa de profundidade pós-filtro
179.            Mat posfilter = Mat(Size(WIDTH, HEIGHT), CV_8UC1);
180.            int NivelFiltro = 4; //Nível do filtro de 1 a 5
181.            unsigned char auxpixel;
182.
183.            //Variável que indica o conjunto de pertinência de cada posição
184.            //apenas para fins de monitoração
185.            saida[0]=0;
186.            saida[1]=0;
187.            saida[2]=0;
188.            saida[3]=0;
189.            saida[4]=0;
190.
191.            //Variável que indica o tom de cinza referente à proximidade dos
192.            //obstáculos de cada região da imagem
193.            pixel[0]=0;
194.            pixel[1]=0;
195.            pixel[2]=0;
196.            pixel[3]=0;
197.            pixel[4]=0;
198.
199.            if (contFrames>=5){ //A cada 5 frames realiza o processamento da imagem

200.                contFrames = 0;
201.
202.                //Analisa a imagem pixel a pixel para localizar e classificar possív
    eis obstáculos
203.                for(int y=0; y<HEIGHT; y++){ //Percorre a imagem linha a linha
204.                    for(int x=0; x<WIDTH; x++){ //Percorre a imagem coluna a coluna

205.                        //Remove fundo e objetos muito distantes
206.                        if (filter[0].at<uchar>(y,x)<50){
207.                            posfilter.at<uchar>(y,x) = 0;
208.
209.                            auxpixel = filter[0].at<uchar>(y,x);
210.                            //Grava cor do pixel referente à distância do objeto em
    variavel que será enviada ao controlador fuzzy
211.                            if (x<=WIDTH*1/5 and pixel[0]<auxpixel)
212.                                pixel[0]=auxpixel;
213.                            else if (x>WIDTH*1/5 and x<=WIDTH*2/5 and pixel[1]<auxpi
    xel)
214.                                pixel[1]=auxpixel;
215.                            else if (x>WIDTH*2/5 and x<=WIDTH*3/5 and pixel[2]<auxpi
    xel)
216.                                pixel[2]=auxpixel;
217.                            else if (x>WIDTH*3/5 and x<=WIDTH*4/5 and pixel[3]<auxpi
    xel)
218.                                pixel[3]=auxpixel;
219.                            else if (x>WIDTH*4/5 and x<=WIDTH*5/5 and pixel[4]<auxpi
    xel)
220.                                pixel[4]=auxpixel;
221.                        }
222.                        //Seleciona objetos muito distantes
223.                        else if (((filter[0].at<uchar>(y,x)>=50 and filter[0].at<uch
    ar>(y,x)<75) + (filter[1].at<uchar>(y,x)>=50 and filter[1].at<uchar>(y,x)<75) + (fil
    ter[2].at<uchar>(y,x)>=50 and filter[2].at<uchar>(y,x)<75) + (filter[3].at<uchar>(y,
    x)>=50 and filter[3].at<uchar>(y,x)<75) + (filter[4].at<uchar>(y,x)>=50 and filter[4
    ].at<uchar>(y,x)<75))>=NivelFiltro){
224.                            posfilter.at<uchar>(y,x) = 75;
```

```
225.                            //Grava distância do objeto em variavel para monitoração

226.                            if (x<=WIDTH*1/5 and saida[0]<1)
227.                                saida[0]=1;
228.                            else if (x>WIDTH*1/5 and x<=WIDTH*2/5 and saida[1]<1)
229.                                saida[1]=1;
230.                            else if (x>WIDTH*2/5 and x<=WIDTH*3/5 and saida[2]<1)
231.                                saida[2]=1;
232.                            else if (x>WIDTH*3/5 and x<=WIDTH*4/5 and saida[3]<1)
233.                                saida[3]=1;
234.                            else if (x>WIDTH*4/5 and x<=WIDTH*5/5 and saida[4]<1)
235.                                saida[4]=1;
236.
237.                            auxpixel = (filter[0].at<uchar>(y,x) + filter[1].at<ucha
    r>(y,x) + filter[2].at<uchar>(y,x) + filter[3].at<uchar>(y,x) + filter[4].at<uchar>(
    y,x))/5;
238.                            //Grava cor do pixel referente à distância do objeto em
    variavel que será enviada ao controlador fuzzy
239.                            if (x<=WIDTH*1/5 and pixel[0]<auxpixel)
240.                                pixel[0]=auxpixel;
241.                            else if (x>WIDTH*1/5 and x<=WIDTH*2/5 and pixel[1]<auxpi
    xel)
242.                                pixel[1]=auxpixel;
243.                            else if (x>WIDTH*2/5 and x<=WIDTH*3/5 and pixel[2]<auxpi
    xel)
244.                                pixel[2]=auxpixel;
245.                            else if (x>WIDTH*3/5 and x<=WIDTH*4/5 and pixel[3]<auxpi
    xel)
246.                                pixel[3]=auxpixel;
247.                            else if (x>WIDTH*4/5 and x<=WIDTH*5/5 and pixel[4]<auxpi
    xel)
248.                                pixel[4]=auxpixel;
249.                        }
250.                        //Seleciona objetos de distância média
251.                        else if (((filter[0].at<uchar>(y,x)>=75 and filter[0].at<uch
    ar>(y,x)<125) + (filter[1].at<uchar>(y,x)>=75 and filter[1].at<uchar>(y,x)<125) + (f
    ilter[2].at<uchar>(y,x)>=75 and filter[2].at<uchar>(y,x)<125) + (filter[3].at<uchar>
    (y,x)>=75 and filter[3].at<uchar>(y,x)<125) + (filter[4].at<uchar>(y,x)>=75 and filt
    er[4].at<uchar>(y,x)<125))>=NivelFiltro){
252.                            posfilter.at<uchar>(y,x) = 170;
253.                            //Grava distância do objeto em variavel para monitoração

254.                            if (x<=WIDTH*1/5 and saida[0]<2)
255.                                saida[0]=2;
256.                            else if (x>WIDTH*1/5 and x<=WIDTH*2/5 and saida[1]<2)
257.                                saida[1]=2;
258.                            else if (x>WIDTH*2/5 and x<=WIDTH*3/5 and saida[2]<2)
259.                                saida[2]=2;
260.                            else if (x>WIDTH*3/5 and x<=WIDTH*4/5 and saida[3]<2)
261.                                saida[3]=2;
262.                            else if (x>WIDTH*4/5 and x<=WIDTH*5/5 and saida[4]<2)
263.                                saida[4]=2;
264.
265.                            auxpixel = (filter[0].at<uchar>(y,x) + filter[1].at<ucha
    r>(y,x) + filter[2].at<uchar>(y,x) + filter[3].at<uchar>(y,x) + filter[4].at<uchar>(
    y,x))/5;
266.                            //Grava cor do pixel referente à distância do objeto em
    variavel que será enviada ao controlador fuzzy
267.                            if (x<=WIDTH*1/5 and pixel[0]<auxpixel)
268.                                pixel[0]=auxpixel;
269.                            else if (x>WIDTH*1/5 and x<=WIDTH*2/5 and pixel[1]<auxpi
    xel)
270.                                pixel[1]=auxpixel;
271.                            else if (x>WIDTH*2/5 and x<=WIDTH*3/5 and pixel[2]<auxpi
    xel)
272.                                pixel[2]=auxpixel;
```

```
273.                             else if (x>WIDTH*3/5 and x<=WIDTH*4/5 and pixel[3]<auxpi
    xel)
274.                                 pixel[3]=auxpixel;
275.                             else if (x>WIDTH*4/5 and x<=WIDTH*5/5 and pixel[4]<auxpi
    xel)
276.                                 pixel[4]=auxpixel;
277.                         }
278.                         //Seleciona objetos muito próximos
279.                         else if (((filter[0].at<uchar>(y,x)>=125) + (filter[1].at<uc
    har>(y,x)>=125) + (filter[2].at<uchar>(y,x)>=125) + (filter[3].at<uchar>(y,x)>=125)
    + (filter[4].at<uchar>(y,x)>=125))>=NivelFiltro){
280.                             posfilter.at<uchar>(y,x) = 255;
281.                             //Grava distância do objeto em variavel para monitoração

282.                             if (x<=WIDTH*1/5 and saida[0]<3)
283.                                 saida[0]=3;
284.                             else if (x>WIDTH*1/5 and x<=WIDTH*2/5 and saida[1]<3)
285.                                 saida[1]=3;
286.                             else if (x>WIDTH*2/5 and x<=WIDTH*3/5 and saida[2]<3)
287.                                 saida[2]=3;
288.                             else if (x>WIDTH*3/5 and x<=WIDTH*4/5 and saida[3]<3)
289.                                 saida[3]=3;
290.                             else if (x>WIDTH*4/5 and x<=WIDTH*5/5 and saida[4]<3)
291.                                 saida[4]=3;
292.
293.                             auxpixel = (filter[0].at<uchar>(y,x) + filter[1].at<ucha
    r>(y,x) + filter[2].at<uchar>(y,x) + filter[3].at<uchar>(y,x) + filter[4].at<uchar>(
    y,x))/5;
294.                             //Grava cor do pixel referente à distância do objeto em
    variavel que será enviada ao controlador fuzzy
295.                             if (x<=WIDTH*1/5 and pixel[0]<auxpixel)
296.                                 pixel[0]=auxpixel;
297.                             else if (x>WIDTH*1/5 and x<=WIDTH*2/5 and pixel[1]<auxpi
    xel)
298.                                 pixel[1]=auxpixel;
299.                             else if (x>WIDTH*2/5 and x<=WIDTH*3/5 and pixel[2]<auxpi
    xel)
300.                                 pixel[2]=auxpixel;
301.                             else if (x>WIDTH*3/5 and x<=WIDTH*4/5 and pixel[3]<auxpi
    xel)
302.                                 pixel[3]=auxpixel;
303.                             else if (x>WIDTH*4/5 and x<=WIDTH*5/5 and pixel[4]<auxpi
    xel)
304.                                 pixel[4]=auxpixel;
305.                         }
306.                         else{
307.                             posfilter.at<uchar>(y,x) = 0;
308.                         }
309.                         //Insere linhas para demarcar a imagem, facilitando a monito
    ração do processo
310.                         if (x==WIDTH*1/5 or x==WIDTH*2/5 or x==WIDTH*3/5 or x==WIDTH
    *4/5){
311.                             posfilter.at<uchar>(y,x) = 255;
312.                         }
313.                     }
314.                 }
315.
316.                 imshow("PosFiltro", posfilter);
317.                 //Imprime informações referentes ao conjunto de pertinência e proxim
    idade dos pixels que serão enviadas ao controlador fuzzy
318.                 printf("%d   %d   %d   %d   %d\n%d %d %d %d %d\n\n", saida[0], saida
    [1], saida[2], saida[3], saida[4], pixel[0], pixel[1], pixel[2], pixel[3], pixel[4])
    ;
319.                 //Envia dados ao controlador fuzzy
320.                 sendto(socket_supervisorio,&pixel[0],sizeof(unsigned char[5]),0,(str
    uct sockaddr*)&endereco_supervisorio,socketsize);
```

```
            }
        }
    }

    destroyAllWindows();

    //Encerra câmera
    Dense3DClose(dense3d);
    CloseDUOCamera();
    //Encerra sockets
    close(socket_supervisorio);

    return 0;
}
```

APÊNDICE B – ALGORITMO PARA TROCA DE DADOS ENTRE COMPUTADOR EXTERNO E RASPBERRY

O algoritmo expresso abaixo é o responsável pelo recebimento e tratamento dos vetores provindos do computador externo, por meio de *sockets* UDP. Cada vez que recebe um novo dado, o mesmo é transmitido à endereços de memória compartilhada para que as demais rotinas processadas no Raspberry PI possam acessá-lo.

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <unistd.h>
4.  #include <arpa/inet.h>
5.  #include <string.h>
6.  #include <sys/ipc.h>
7.  #include <sys/shm.h>
8.
9.  //Endereço para comunicação através de UDP
10. #define PORTNOTEBOOK 2302
11.
12. //Endereços de memória compartilhada
13. #define KEYFinalizar 9912
14. #define SEND 9920
15.
16. typedef struct sample{
17.         unsigned char sendProcesso[5];
18. }sample;
19.
20. struct dados {
21.     float y;
22.     int finalizar;
23. }__attribute__((__packed__));
24.
25. int main() {
26.     //Variaveis referentes aos sockets
27.     socklen_t socketsize; //Recebe o tamanho da estrutura sockaddr_in
28.     int socket_notebook;
29.     struct sockaddr_in endereco_processo;
30.     struct sockaddr_in endereco_notebook;
31.
32.     unsigned char saida[5];
33.
34.     //Sockets
35.     socket_notebook= socket(AF_INET, SOCK_DGRAM,0);
36.     socketsize = sizeof(struct sockaddr_in); //Guarda o tamanho da estrutura sockadd
    r_in
37.     endereco_processo.sin_family = AF_INET;
38.     endereco_processo.sin_addr.s_addr = INADDR_ANY; //Utiliza qualquer interface de
    rede disponivel (Ex: Ethernet, Wireless)
39.     endereco_processo.sin_port = htons(PORTNOTEBOOK); //Define a porta de escuta da
    conexão
40.     memset(&(endereco_processo.sin_zero),'\0',sizeof(endereco_processo.sin_zero)); /
    /Zera o resto da estrutura
41.     bind(socket_notebook,(struct sockaddr *)&endereco_processo, sizeof(struct sockad
    dr)); //Liga o endereçamento do servidor ao socket
42.
43.     //Estabelece comunicação com computador externo
```

```
44.     endereco_notebook.sin_family = AF_INET;
45.     endereco_notebook.sin_addr.s_addr = inet_addr("192.168.1.39"); //IP do computado
    r do qual receberá pacotes
46.     endereco_notebook.sin_port = htons(PORTNOTEBOOK); //Define a porta de escuta da
    conexao
47.     memset(&(endereco_notebook.sin_zero),'\0',sizeof(endereco_notebook.sin_zero)); /
    /Zera o resto da estrutura
48.
49.     saida[0]=0;
50.     saida[1]=0;
51.     saida[2]=0;
52.     saida[3]=0;
53.     saida[4]=0;
54.
55.     //Configurações de memória compartilhada
56.     int shmidX, shmidFinalizar;
57.     char *path="/home/pi/Desktop/Software";
58.     int *finalizar;
59.     sample *structDados;
60.
61.     //Inicializa memória compartilhada
62.     if((shmidFinalizar=shmget(ftok(path,(key_t)KEYFinalizar),sizeof(int*),IPC_CREAT|
    SHM_R|SHM_W))==-1){
63.         printf("\n\nErro na criação do segmento de memoria...\n");
64.         exit(-1);
65.     }
66.
67.     finalizar=shmat(shmidFinalizar,0,0);
68.
69.     if((shmidX=shmget(ftok(path,(key_t)SEND),sizeof(sample),IPC_CREAT|SHM_R|SHM_W))=
    =-1){
70.         printf("\n\nErro na criação do segmento de memoria...\n");
71.         exit(-1);
72.     }
73.
74.     structDados=(sample *) shmat(shmidX,0,0);
75.
76.     //Aguarda comando através de endereço de memória compartilhada para iniciar
77.     while ((*finalizar)!=1){
78.         usleep(100000);
79.         finalizar=shmat(shmidFinalizar,0,0);
80.     }
81.
82.     while((*finalizar)==1) //Interrompe a execução quando solicitado por meio de mem
    ória compartilhada
83.     {
84.         recvfrom(socket_notebook,&saida[0],sizeof(unsigned char[5]),0,(struct sockad
    dr*)&endereco_notebook,&socketsize);
85.         structDados->sendProcesso[0] = saida[0];
86.         structDados->sendProcesso[1] = saida[1];
87.         structDados->sendProcesso[2] = saida[2];
88.         structDados->sendProcesso[3] = saida[3];
89.         structDados->sendProcesso[4] = saida[4];
90.         printf("Enviou: %d %d %d %d %d\n", structDados-
    >sendProcesso[0], structDados->sendProcesso[1], structDados-
    >sendProcesso[2], structDados->sendProcesso[3], structDados->sendProcesso[4]);
91.     }
92.
93.     //Encerra comunicação com o computador externo
94.     close(socket_notebook);
95.     //Encerra memória compartilhada
96.     shmdt(structDados);
97.     shmdt(finalizar);
98.
99.     return EXIT_SUCCESS;
100.}
```

APÊNDICE C – ALGORITMO PARA COMUNICAÇÃO COM A BÚSSOLA ELETRÔNICA HMC5883L

O seguinte *script* realiza a comunicação com a bússola eletrônica. Em seguida, armazena os dados referentes ao posicionamento do barco em endereços de memória compartilhada.

```
1.  #define _GNU_SOURCE
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #define _USE_MATH_DEFINES
5.  #include <math.h>
6.  #include <fcntl.h>
7.  #include <unistd.h>
8.  #include <string.h>
9.  #include <sys/ioctl.h>
10. #include <sys/types.h>
11. #include <sys/stat.h>
12. #include <sys/time.h>
13. #include <linux/i2c-dev.h>
14. #include <sys/ipc.h>
15. #include <sys/shm.h>
16.
17. //Inicialização e configuração do magnetometro
18. const int HMC5883L_I2C_ADDR = 0x1E;
19.
20. #define CONFIG_A 0x00
21. #define CONFIG_B 0x01
22. #define MODE 0x02
23. #define DATA 0x03 //Lê 6 bytes: x msb, x lsb, z msb, z lsb, y msb, y lsb
24. #define STATUS 0x09
25. #define ID_A 0x0A
26. #define ID_B 0x0B
27. #define ID_C 0x0C
28. #define ID_STRING "H43"
29. #define GAIN 1370 //000 setting
30.
31. //Endereços de memória compartilhada
32. #define POSX 9900
33. #define KEYFinalizar 9912
34.
35.  void selectDevice(int fd, int addr, char * name)
36.  {
37.      if (ioctl(fd, I2C_SLAVE, addr) < 0)
38.      {
39.          fprintf(stderr, "%s not present\n", name);
40.      }
41.  }
42.
43.  void writeToDevice(int fd, int reg, int val)
44.  {
45.     char buf[2];
46.     buf[0]=reg;
47.     buf[1]=val;
48.     if (write(fd, buf, 2) != 2){
49.         fprintf(stderr, "Can't write to ADXL345\n");
50.     }
51. }
52.
```

```
53. int main(int argc, char **argv)
54. {
55.     //Variáveis referentesà leitura de dados do magnetômetro
56.     int fd;
57.     unsigned char buf[16];
58.     struct timeval tv;
59.     struct timezone tz;
60.     struct timeval data_timestamp;
61.     int resolution = 100000;
62.     long next_timestamp;
63.
64.     //Configurações de memória compartilhada
65.     int shmidX, shmidFinalizar;
66.     char *path="/home/pi/Desktop/Software";
67.     int *finalizar;
68.     int *sendx;
69.     int k;
70.
71.     if ((fd = open("/dev/i2c-1", O_RDWR)) < 0)
72.     {
73.         // Open port for reading and writing
74.         fprintf(stderr, "Failed to open i2c bus\n");
75.
76.         return 1;
77.     }
78.
79.     //Inicializa módulo HMC5883L
80.     selectDevice(fd, HMC5883L_I2C_ADDR, "HMC5883L");
81.
82.     //first read the 3 ID bytes
83.     buf[0] = ID_A;
84.     if ((write(fd, buf, 1)) != 1){
85.         // Send the register to read from
86.         fprintf(stderr, "Error writing to i2c slave\n");
87.     }
88.     if (read(fd, buf, 3) != 3) {
89.             fprintf(stderr, "Unable to read from HMC5883L\n");
90.     }
91.     buf[3]=0;
92.     printf("Identification: '%s' ",buf);
93.     if (strncmp(buf, ID_STRING, 3) == 0)
94.         printf("HMC5883L sensor detected\n");
95.     else {
96.         printf("unknown sensor. Exiting.\n");
97.         exit(1);
98.     }
99.
100.    //Envia configurações ao matnetometro
101.    //writeToDevice(fd, 0x01, 0);
102.    writeToDevice(fd, CONFIG_A, 0b01101000); //8 sample averaging
103.    writeToDevice(fd, CONFIG_B, 0b00000000); //max gain
104.    writeToDevice(fd, MODE,     0b00000011); //idle mode
105.
106.    gettimeofday(&data_timestamp,&tz);
107.    data_timestamp.tv_sec += 1; //start loggin at start of next second
108.    data_timestamp.tv_usec = 0;
109.
110.    //Inicializa memória compartilhada
111.    if((shmidFinalizar=shmget(ftok(path,(key_t)KEYFinalizar),sizeof(int*),IPC_CREAT|
    SHM_R|SHM_W))==-1){
112.        printf("\n\nErro na criação do segmento de memoria...\n");
113.        exit(-1);
114.    }
115.
116.    finalizar=shmat(shmidFinalizar,0,0);
117.
```

```
118.     if((shmidX=shmget(ftok(path,(key_t)POSX),sizeof(int*),IPC_CREAT|SHM_R|SHM_W))==-
    1){
119.         printf("\n\nErro na criação do segmento de memoria...\n");
120.         exit(-1);
121.     }
122.
123.     sendx=shmat(shmidX,0,0);
124.
125.     //Aguarda comando através de endereço de memória compartilhada para iniciar
126.     while ((*finalizar)!=1){
127.         usleep(100000);
128.         finalizar=shmat(shmidFinalizar,0,0);
129.     }
130.
131.     while ((*finalizar)==1){ //Interrompe a execução quando solicitado por meio de m
    emória compartilhada
132.         usleep(200000);//0,2 segundo para iniciar nova leitura
133.         while (1) {
134.             //get time
135.             gettimeofday(&tv,&tz);
136.
137.             //if seconds >= next_secs && usecs >= next_usecs
138.             if (tv.tv_sec >= data_timestamp.tv_sec && tv.tv_usec >= data_timestamp.t
    v_usec)
139.                 break;
140.
141.             usleep(1000);
142.         }
143.
144.         //initiate single conversion
145.         writeToDevice(fd, MODE,     0b00000001); //single measurement
146.         usleep(7000);
147.         buf[0] = DATA;
148.
149.         //Confirma validade da leitura de dados
150.         if ((write(fd, buf, 1)) != 1){
151.             fprintf(stderr, "Error writing to i2c slave\n");
152.         }
153.
154.         if (read(fd, buf, 6) != 6){
155.             fprintf(stderr, "Unable to read from HMC5883L\n");
156.         }
157.         else {
158.             short x = (buf[0] << 8) | buf[1];
159.             short y = (buf[4] << 8) | buf[5];
160.             short z = (buf[2] << 8) | buf[3];
161.             float angle = atan2(y, x) * 180 / M_PI;
162.             float mag = sqrt(x*x+y*y+z*z);
163.
164.             //Por motivos desconhecidos, o magnetometro passava a decrementar o valo
    r de X quando esse
165.             //fosse maior do que 270, enquanto Y presentava valores positivos ao pas
    sar por esse ponto.
166.             //Dessa forma, corrige-se o posicionamento de x para evitar o problema
167.             if (y>0)
168.                 x=(270-x)+270;
169.
170.             //Imprime dados lidos do magnetômetro
171.             printf("X:%d,   Y:%d,   Z:%d,   %0.1f nT\n",x,y,z,1e5*mag/GAIN);
172.             *sendx=x;
173.             fflush(stdout);
174.         }
175.
176.         //advance data timestamp to next required time
177.         data_timestamp.tv_usec += resolution;
178.         if (data_timestamp.tv_usec >= 1e6) {
```

```
179.             data_timestamp.tv_sec += 1;
180.             data_timestamp.tv_usec -= 1e6;
181.         }
182.
183.     }
184.
185.     //Encerra memória compartilhada
186.     shmdt(sendx);
187.     shmdt(finalizar);
188.
189.     return 0;
190. }
```

APÊNDICE D – ALGORITMO PARA CONTROLE DO SISTEMA BASEADO EM LÓGICA *FUZZY*

O seguinte algoritmo realiza a leitura de todos os dados do conjunto, por meio de endereços de memória compartilhada, e os aplica em um controlador *fuzzy*. Assim, desempenha processos de fuzzificação, aplicação de regras nebulosas e defuzzificação dos valores obtidos. Na sequência, utiliza esses resultados para comandar a navegação do veículo, controlando o acionamento dos seus dois motores por meio de saídas digitais.

```
1.  #include <stdio.h>
2.  #include <math.h>
3.  #include <stdlib.h>
4.  #include <sys/ipc.h>
5.  #include <sys/shm.h>
6.  #include <unistd.h>
7.  #include <sys/time.h>
8.  #include <wiringPi.h>
9.  #include <arpa/inet.h>
10. #include <string.h>
11.
12. //Endereços de memória compartilhada
13. #define POSMagnet 9900
14. #define KEYFinalizar 9912
15. #define KEYSAIDA 9920
16.
17. //Estruturas para recebimento de dados de endereços de memória compartilhada
18. typedef struct sample{
19.     unsigned char entrada[5];
20. }sample;
21.
22. struct dados {
23.     float y;
24.     int finalizar;
25. }__attribute__((__packed__));
26.
27. //Estruturas referentes aos conjuntos utilizados no processo de fuzzificação
28. typedef struct StructDistancia {
29.     float Livre, Distante, DistanciaMedia, Proximo;
30. }StructDistancia;
31.
32. typedef struct StructPosicaoAngular {
33.     float Esquerda, EsquerdaLeve, Centro, DireitaLeve, Direita;
34. }StructPosicaoAngular;
35.
36. int main()
37. {
38.     //Variaveis referentes à memória compartilhada
39.     int shmidMagnet;
40.     int shmidFinalizar;
41.     int shmidSaida;
42.     char *path="/home/pi/Desktop/Software";
43.     int *finalizar;
44.     int *magnet;
45.     sample *saida;
46.
```

```
47.     //Declaração das demais variáveis
48.     int VariacaoAngular;
49.     int PosicaoAngularDesejada = 240; //Define a posição angular desejada
50.     int AuxMaiorValor;
51.     float Esquerda, EsquerdaLeve, Frente, DireitaLeve, Direita, Re;
52.     float AuxEsquerda1, AuxEsquerda2, AuxEsquerda3, AuxEsquerda4, AuxEsquerdaLeve1,
    AuxEsquerdaLeve2, AuxEsquerdaLeve3;
53.     float AuxDireitaLeve1, AuxDireitaLeve2, AuxDireitaLeve3, AuxDireita1, AuxDireita
    2, AuxDireita3, AuxDireita4;
54.     float SaidaMOM;
55.
56.     //Definição das estruturas para variáveis fuzzy
57.     StructDistancia u0; //Obstáculos à esquerda
58.     StructDistancia u1; //Obstáculos ao centro-esquerda
59.     StructDistancia u2; //Obstáculos ao centro
60.     StructDistancia u3; //Obstáculos ao centro-direita
61.     StructDistancia u4; //Obstáculos à direita
62.
63.     StructPosicaoAngular PosicaoAngular; //Posição angular da inclinação do barco
64.
65.     wiringPiSetup();        //Inicia a biblioteca WiringPi, responsável pelas entrad
    as e saídas do Raspberry
66.     pinMode(1, PWM_OUTPUT); //Configura o pino físico 12 como saída PWM dos motores
    -- NÃO UTILIZADO
67.     pinMode(0, OUTPUT); //Configura o pino físico 11 como saída IN1
68.     pinMode(2, OUTPUT); //Configura o pino físico 13 como saída IN2
69.     pinMode(3, OUTPUT); //Configura o pino físico 15 como saída IN3
70.     pinMode(4, OUTPUT); //Configura o pino físico 16 como saída IN4
71.
72.     int estado=0;
73.
74.     //Inicializa memória compartilhada
75.     if((shmidMagnet=shmget(ftok(path,(key_t)POSMagnet),sizeof(int*),IPC_CREAT|SHM_R|
    SHM_W))==-1){
76.         printf("\n\nErro na criação do segmento de memoria...\n");
77.         exit(-1);
78.     }
79.
80.     if((shmidSaida=shmget(ftok(path,(key_t)KEYSAIDA),sizeof(sample),IPC_CREAT|SHM_R|
    SHM_W))==-1){
81.         printf("\n\nErro na criação do segmento de memoria...\n");
82.         exit(-1);
83.     }
84.
85.     if((shmidFinalizar=shmget(ftok(path,(key_t)KEYFinalizar),sizeof(int*),IPC_CREAT|
    SHM_R|SHM_W))==-1){
86.         printf("\n\nErro na criação do segmento de memoria...\n");
87.         exit(-1);
88.     }
89.
90.     magnet = shmat(shmidMagnet, 0, 0);
91.     finalizar = shmat(shmidFinalizar, 0, 0);
92.     saida = (sample *) shmat(shmidSaida,0,0);
93.
94.     //Aguarda comando através de endereço de memória compartilhada para iniciar
95.     while ((*finalizar)!=1){
96.         usleep(100000);
97.         finalizar=shmat(shmidFinalizar,0,0);
98.     }
99.
100.    while((*finalizar)==1) //Interrompe a execução quando solicitado por meio de mem
    ória compartilhada
101.    {
102.        //Fuzzificação das variáveis de entrada
103.
104.        //u0
```

```
105.        if (saida->entrada[0]<=40)
106.            u0.Livre = 1;
107.        else if (saida->entrada[0]<=60)
108.            u0.Livre = (60.0-saida->entrada[0])/20.0;
109.        else
110.            u0.Livre = 0;
111.
112.        if (saida->entrada[0]<=40)
113.            u0.Distante = 0;
114.        else if (saida->entrada[0]<=60)
115.            u0.Distante = (saida->entrada[0]-40.0)/20.0;
116.        else if (saida->entrada[0]<=65)
117.            u0.Distante = 1;
118.        else if (saida->entrada[0]<=85)
119.            u0.Distante = (85.0-saida->entrada[0])/20.0;
120.        else
121.            u0.Distante = 0;
122.
123.        if (saida->entrada[0]<=65)
124.            u0.DistanciaMedia = 0;
125.        else if (saida->entrada[0]<=85)
126.            u0.DistanciaMedia = (saida->entrada[0]-65.0)/20.0;
127.        else if (saida->entrada[0]<=115)
128.            u0.DistanciaMedia = 1;
129.        else if (saida->entrada[0]<=135)
130.            u0.DistanciaMedia = (135.0-saida->entrada[0])/20.0;
131.        else
132.            u0.DistanciaMedia = 0;
133.
134.        if (saida->entrada[0]<=115)
135.            u0.Proximo = 0;
136.        else if (saida->entrada[0]<=135)
137.            u0.Proximo = (saida->entrada[0]-115.0)/20.0;
138.        else
139.            u0.Proximo = 1;
140.
141.        //u1
142.        if (saida->entrada[1]<=40)
143.            u1.Livre = 1;
144.        else if (saida->entrada[1]<=60)
145.            u1.Livre = (60.0-saida->entrada[1])/20.0;
146.        else
147.            u1.Livre = 0;
148.
149.        if (saida->entrada[1]<=40)
150.            u1.Distante = 0;
151.        else if (saida->entrada[1]<=60)
152.            u1.Distante = (saida->entrada[1]-40.0)/20.0;
153.        else if (saida->entrada[1]<=65)
154.            u1.Distante = 1;
155.        else if (saida->entrada[1]<=85)
156.            u1.Distante = (85.0-saida->entrada[1])/20.0;
157.        else
158.            u1.Distante = 0;
159.
160.        if (saida->entrada[1]<=65)
161.            u1.DistanciaMedia = 0;
162.        else if (saida->entrada[1]<=85)
163.            u1.DistanciaMedia = (saida->entrada[1]-65.0)/20.0;
164.        else if (saida->entrada[1]<=115)
165.            u1.DistanciaMedia = 1;
166.        else if (saida->entrada[1]<=135)
167.            u1.DistanciaMedia = (135.0-saida->entrada[1])/20.0;
168.        else
169.            u1.DistanciaMedia = 0;
170.
```

```
171.        if (saida->entrada[1]<=115)
172.            u1.Proximo = 0;
173.        else if (saida->entrada[1]<=135)
174.            u1.Proximo = (saida->entrada[1]-115.0)/20.0;
175.        else
176.            u1.Proximo = 1;
177.
178.        //u2
179.        if (saida->entrada[2]<=40)
180.            u2.Livre = 1;
181.        else if (saida->entrada[2]<=60)
182.            u2.Livre = (60.0-saida->entrada[2])/20.0;
183.        else
184.            u2.Livre = 0;
185.
186.        if (saida->entrada[2]<=40)
187.            u2.Distante = 0;
188.        else if (saida->entrada[2]<=60)
189.            u2.Distante = (saida->entrada[2]-40.0)/20.0;
190.        else if (saida->entrada[2]<=65)
191.            u2.Distante = 1;
192.        else if (saida->entrada[2]<=85)
193.            u2.Distante = (85.0-saida->entrada[2])/20.0;
194.        else
195.            u2.Distante = 0;
196.
197.        if (saida->entrada[2]<=65)
198.            u2.DistanciaMedia = 0;
199.        else if (saida->entrada[2]<=85)
200.            u2.DistanciaMedia = (saida->entrada[2]-65.0)/20.0;
201.        else if (saida->entrada[2]<=115)
202.            u2.DistanciaMedia = 1;
203.        else if (saida->entrada[2]<=135)
204.            u2.DistanciaMedia = (135.0-saida->entrada[2])/20.0;
205.        else
206.            u2.DistanciaMedia = 0;
207.
208.        if (saida->entrada[2]<=115)
209.            u2.Proximo = 0;
210.        else if (saida->entrada[2]<=135)
211.            u2.Proximo = (saida->entrada[2]-115.0)/20.0;
212.        else
213.            u2.Proximo = 1;
214.
215.        //u3
216.        if (saida->entrada[3]<=40)
217.            u3.Livre = 1;
218.        else if (saida->entrada[3]<=60)
219.            u3.Livre = (60.0-saida->entrada[3])/20.0;
220.        else
221.            u3.Livre = 0;
222.
223.        if (saida->entrada[3]<=40)
224.            u3.Distante = 0;
225.        else if (saida->entrada[3]<=60)
226.            u3.Distante = (saida->entrada[3]-40.0)/20.0;
227.        else if (saida->entrada[3]<=65)
228.            u3.Distante = 1;
229.        else if (saida->entrada[3]<=85)
230.            u3.Distante = (85.0-saida->entrada[3])/20.0;
231.        else
232.            u3.Distante = 0;
233.
234.        if (saida->entrada[3]<=65)
235.            u3.DistanciaMedia = 0;
236.        else if (saida->entrada[3]<=85)
```

```
237.            u3.DistanciaMedia = (saida->entrada[3]-65.0)/20.0;
238.        else if (saida->entrada[3]<=115)
239.            u3.DistanciaMedia = 1;
240.        else if (saida->entrada[3]<=135)
241.            u3.DistanciaMedia = (135.0-saida->entrada[3])/20.0;
242.        else
243.            u3.DistanciaMedia = 0;
244.
245.        if (saida->entrada[3]<=115)
246.            u3.Proximo = 0;
247.        else if (saida->entrada[3]<=135)
248.            u3.Proximo = (saida->entrada[3]-115.0)/20.0;
249.        else
250.            u3.Proximo = 1;
251.
252.        //u4
253.        if (saida->entrada[4]<=40)
254.            u4.Livre = 1;
255.        else if (saida->entrada[4]<=60)
256.            u4.Livre = (60.0-saida->entrada[4])/20.0;
257.        else
258.            u4.Livre = 0;
259.
260.        if (saida->entrada[4]<=40)
261.            u4.Distante = 0;
262.        else if (saida->entrada[4]<=60)
263.            u4.Distante = (saida->entrada[4]-40.0)/20.0;
264.        else if (saida->entrada[4]<=65)
265.            u4.Distante = 1;
266.        else if (saida->entrada[4]<=85)
267.            u4.Distante = (85.0-saida->entrada[4])/20.0;
268.        else
269.            u4.Distante = 0;
270.
271.        if (saida->entrada[4]<=65)
272.            u4.DistanciaMedia = 0;
273.        else if (saida->entrada[4]<=85)
274.            u4.DistanciaMedia = (saida->entrada[4]-65.0)/20.0;
275.        else if (saida->entrada[4]<=115)
276.            u4.DistanciaMedia = 1;
277.        else if (saida->entrada[4]<=135)
278.            u4.DistanciaMedia = (135.0-saida->entrada[4])/20.0;
279.        else
280.            u4.DistanciaMedia = 0;
281.
282.        if (saida->entrada[4]<=115)
283.            u4.Proximo = 0;
284.        else if (saida->entrada[4]<=135)
285.            u4.Proximo = (saida->entrada[4]-115.0)/20.0;
286.        else
287.            u4.Proximo = 1;
288.
289.        //Variação da posição angular
290.        VariacaoAngular = -*magnet + PosicaoAngularDesejada;
291.
292.        if (VariacaoAngular<=-60)
293.            PosicaoAngular.Esquerda = 1;
294.        else if (VariacaoAngular<=-30)
295.           PosicaoAngular.Esquerda = (-VariacaoAngular-30.0)/30.0;
296.        else
297.            PosicaoAngular.Esquerda = 0;
298.
299.        if (VariacaoAngular<=-60)
300.            PosicaoAngular.EsquerdaLeve = 0;
301.        else if (VariacaoAngular<=-30)
302.            PosicaoAngular.EsquerdaLeve = (60.0+VariacaoAngular)/30.0;
```

```
303.        else if (VariacaoAngular<=-10)
304.            PosicaoAngular.EsquerdaLeve = 1;
305.        else if (VariacaoAngular<=0)
306.            PosicaoAngular.EsquerdaLeve = (-VariacaoAngular)/10.0;
307.        else
308.           PosicaoAngular.EsquerdaLeve = 0;
309.
310.        if (VariacaoAngular<=-10)
311.            PosicaoAngular.Centro = 0;
312.        else if (VariacaoAngular<=0)
313.            PosicaoAngular.Centro = (10.0+VariacaoAngular)/10.0;
314.        else if (VariacaoAngular<=10)
315.            PosicaoAngular.Centro = (10.0-VariacaoAngular)/10.0;
316.        else
317.           PosicaoAngular.Centro = 0;
318.
319.        if (VariacaoAngular<=0)
320.            PosicaoAngular.DireitaLeve = 0;
321.        else if (VariacaoAngular<=10)
322.            PosicaoAngular.DireitaLeve = (VariacaoAngular)/10.0;
323.        else if (VariacaoAngular<=30)
324.            PosicaoAngular.DireitaLeve = 1;
325.        else if (VariacaoAngular<=60)
326.            PosicaoAngular.DireitaLeve = (60.0-VariacaoAngular)/30.0;
327.        else
328.           PosicaoAngular.DireitaLeve = 0;
329.
330.        if (VariacaoAngular<=30)
331.            PosicaoAngular.Direita = 0;
332.        else if (VariacaoAngular<=60)
333.           PosicaoAngular.Direita = (VariacaoAngular-30.0)/30.0;
334.        else
335.            PosicaoAngular.Direita = 1;
336.
337.        //Regras fuzzy
338.
339.        //nº1 - SE (max(u3, u4)=DISTANTE E max(u3, u4)=LIVRE) ENTAO DIREITA=(VPA-
    >ESQUERDA)
340.        if (((u3.Distante>=0.5) || (u3.Livre>=0.5)) && ((u4.Distante>=0.5) || (u4.Li
    vre>=0.5)))
341.            AuxDireita1 = PosicaoAngular.Esquerda;
342.        else
343.            AuxDireita1 = 0;
344.
345.        //nº2 - SE (max(u3, u4)=DISTANTE E max(u3, u4)=LIVRE) ENTAO DIREITA LEVE=(VP
    A->ESQUERDA LEVE)
346.        if (((u3.Distante>=0.5) || (u3.Livre>=0.5)) && ((u4.Distante>=0.5) || (u4.Li
    vre>=0.5)))
347.            AuxDireitaLeve1 = PosicaoAngular.EsquerdaLeve;
348.        else
349.            AuxDireitaLeve1 = 0;
350.
351.        //nº3 - SE (max(u0, u1)=DISTANTE E max(u0, u1)=LIVRE) ENTAO ESQUERDA=(VPA-
    >DIREITA)
352.        if (((u0.Distante>=0.5) || (u0.Livre>=0.5)) && ((u1.Distante>=0.5) || (u1.Li
    vre>=0.5)))
353.            AuxEsquerda1 = PosicaoAngular.Direita;
354.        else
355.            AuxEsquerda1 = 0;
356.
357.        //nº4 - SE (max(u0, u1)=DISTANTE E max(u0, u1)=LIVRE) ENTAO ESQUERDA LEVE=(V
    PA->DIREITA LEVE)
358.        if (((u0.Distante>=0.5) || (u0.Livre>=0.5)) && ((u1.Distante>=0.5) || (u1.Li
    vre>=0.5)))
359.            AuxEsquerdaLeve1 = PosicaoAngular.DireitaLeve;
360.        else
```

```
361.            AuxEsquerdaLeve1 = 0;
362.
363.        //nº5 - SE (VPA=CENTRO) E (u2=LIVRE) E u1=(DISTANTE OU LIVRE) E u3=(DISTANTE
     OU LIVRE) ENTAO FRENTE=(u2->LIVRE)
364.        if ((PosicaoAngular.Centro>=0.5) && (u2.Livre>=0.5) && ((u1.Distante>=0.5) |
    | (u1.Livre>=0.5)) && ((u3.Distante>=0.5) || (u3.Livre>=0.5)))
365.            Frente = u2.Livre;
366.        else
367.            Frente = 0;
368.
369.        //nº6 - SE max(u0,u1)<=max(u3,u4) E (max(u3,u4)=DISTANTE) ENTAO ESQUERDA LEV
    E=(u3->DISTANTE OU u4->DISTANTE)
370.        if (saida->entrada[3]>=saida->entrada[4])
371.            AuxMaiorValor = saida->entrada[3];
372.        else
373.            AuxMaiorValor = saida->entrada[4];
374.
375.        if ((saida->entrada[0]<=AuxMaiorValor) && (saida-
    >entrada[1]<=AuxMaiorValor) && ((u3.Distante>=0.5) || (u4.Distante>=0.5)))
376.            if (u3.Distante>u4.Distante)
377.                AuxEsquerdaLeve2 = u3.Distante;
378.            else
379.                AuxEsquerdaLeve2 = u4.Distante;
380.        else
381.            AuxEsquerdaLeve2 = 0;
382.
383.        //nº7 - SE max(u0,u1)<=max(u3,u4) E (max(u3,u4)=DISTANCIA MEDIA) ENTAO ESQUE
    RDA=(u3->DISTANCIA MEDIA OU u4->DISTANCIA MEDIA)
384.        if ((saida->entrada[0]<=AuxMaiorValor) && (saida-
    >entrada[1]<=AuxMaiorValor) && ((u3.DistanciaMedia>=0.5) || (u4.DistanciaMedia>=0.5)
    ))
385.            if (u3.DistanciaMedia>u4.DistanciaMedia)
386.                AuxEsquerda2 = u3.DistanciaMedia;
387.            else
388.                AuxEsquerda2 = u4.DistanciaMedia;
389.        else
390.            AuxEsquerda2 = 0;
391.
392.        //nº8 - SE max(u0,u1)<=max(u3,u4) E (max(u3,u4)=PROXIMO) ENTAO ESQUERDA=1
393.        if ((saida->entrada[0]<=AuxMaiorValor) && (saida-
    >entrada[1]<=AuxMaiorValor) && ((u3.Proximo>=0.5) || (u4.Proximo>=0.5)))
394.            AuxEsquerda3 = 1;
395.        else
396.            AuxEsquerda3 = 0;
397.
398.        //nº9 - SE max(u0,u1)<=max(u3,u4) E (u2=DISTANTE) ENTAO ESQUERDA LEVE=(u2-
    >DISTANTE)
399.        if ((saida->entrada[0]<=AuxMaiorValor) && (saida-
    >entrada[1]<=AuxMaiorValor) && (u2.Distante>=0.5))
400.            AuxEsquerdaLeve3 = u2.Distante;
401.        else
402.            AuxEsquerdaLeve3 = 0;
403.
404.        //nº10 - SE max(u0,u1)<=max(u3,u4) E (u2=DISTANCIA MEDIA) ENTAO ESQUERDA=(u2
    ->DISTANCIA MEDIA)
405.        if ((saida->entrada[0]<=AuxMaiorValor) && (saida-
    >entrada[1]<=AuxMaiorValor) && (u2.DistanciaMedia>=0.5))
406.            AuxEsquerda4 = u2.DistanciaMedia;
407.        else
408.            AuxEsquerda4 = 0;
409.
410.        //nº11 - SE max(u3,u4)<=max(u0,u1) E (max(u0,u1)=DISTANTE) ENTAO DIREITA LEV
    E=(u0->DISTANTE OU u1->DISTANTE)
411.        if (saida->entrada[0]>=saida->entrada[1])
412.            AuxMaiorValor = saida->entrada[0];
413.        else
```

```
414.            AuxMaiorValor = saida->entrada[1];
415.
416.        if ((saida->entrada[3]<=AuxMaiorValor) && (saida-
    >entrada[4]<=AuxMaiorValor) && ((u0.Distante>=0.5) || (u1.Distante>=0.5)))
417.            if (u0.Distante>u1.Distante)
418.                AuxDireitaLeve2 = u0.Distante;
419.            else
420.                AuxDireitaLeve2 = u1.Distante;
421.        else
422.            AuxDireitaLeve2 = 0;
423.
424.        //nº12 - SE max(u3,u4)<=max(u0,u1) E (max(u0,u1)=DISTANCIA MEDIA) ENTAO DIRE
    ITA=(u0->DISTANCIA MEDIA OU u1->DISTANCIA MEDIA)
425.        if ((saida->entrada[3]<=AuxMaiorValor) && (saida-
    >entrada[4]<=AuxMaiorValor) && ((u0.DistanciaMedia>=0.5) || (u1.DistanciaMedia>=0.5)
    ))
426.            if (u0.DistanciaMedia>u1.DistanciaMedia)
427.                AuxDireita2 = u0.DistanciaMedia;
428.            else
429.                AuxDireita2 = u1.DistanciaMedia;
430.        else
431.            AuxDireita2 = 0;
432.
433.        //nº13 - SE max(u3,u4)<=max(u0,u1) E (max(u0,u1)=PROXIMO) ENTAO DIREITA=1
434.        if ((saida->entrada[3]<=AuxMaiorValor) && (saida-
    >entrada[4]<=AuxMaiorValor) && ((u0.Proximo>=0.5) || (u1.Proximo>=0.5)))
435.            AuxDireita3 = 1;
436.        else
437.            AuxDireita3 = 0;
438.
439.        //nº14 - SE max(u3,u4)<=max(u0,u1) E (u2=DISTANTE) ENTAO DIREITA LEVE=(u2-
    >DISTANTE)
440.        if ((saida->entrada[3]<=AuxMaiorValor) && (saida-
    >entrada[4]<=AuxMaiorValor) && (u2.Distante>=0.5))
441.            AuxDireitaLeve3 = u2.Distante;
442.        else
443.            AuxDireitaLeve3 = 0;
444.
445.        //nº15 - SE max(u3,u4)<=max(u0,u1) E (u2=DISTANCIA MEDIA) ENTAO DIREITA=(u2-
    >DISTANCIA MEDIA)
446.        if ((saida->entrada[3]<=AuxMaiorValor) && (saida-
    >entrada[4]<=AuxMaiorValor) && (u2.DistanciaMedia>=0.5))
447.            AuxDireita4 = u2.DistanciaMedia;
448.        else
449.            AuxDireita4 = 0;
450.
451.        //nº16 - SE (u2=PROXIMO) OU (max(u0,u1)=PROXIMO E max(u3,u4)=PROXIMO) ENTÃO
    RÉ=1
452.        if ((u2.Proximo>=0.5) || (((u0.Proximo>=0.5) || (u1.Proximo>=0.5)) && ((u3.P
    roximo>=0.5) || (u4.Proximo>=0.5))))
453.            Re = 1;
454.        else
455.            Re = 0;
456.
457.        //Considera o maior valor de cada conjunto para utilizá-
    lo no processo de defuzzificação
458.        if ((AuxEsquerdaLeve1 >= AuxEsquerdaLeve2) && (AuxEsquerdaLeve1 >= AuxEsquer
    daLeve3))
459.            EsquerdaLeve = AuxEsquerdaLeve1;
460.        else if (AuxEsquerda2 >= AuxEsquerdaLeve3)
461.            EsquerdaLeve = AuxEsquerdaLeve2;
462.        else
463.            EsquerdaLeve = AuxEsquerdaLeve3;
464.
465.        if ((AuxEsquerda1 >= AuxEsquerda2) && (AuxEsquerda1 >= AuxEsquerda3) && (Aux
    Esquerda1 >= AuxEsquerda4))
```

```
466.            Esquerda = AuxEsquerda1;
467.        else if ((AuxEsquerda2 >= AuxEsquerda3) && (AuxEsquerda2 >= AuxEsquerda4))
468.            Esquerda = AuxEsquerda2;
469.        else if (AuxEsquerda3 >= AuxEsquerda4)
470.            Esquerda = AuxEsquerda3;
471.        else
472.            Esquerda = AuxEsquerda4;
473.
474.        if ((AuxDireitaLeve1 >= AuxDireitaLeve2) && (AuxDireitaLeve1 >= AuxDireitaLe
    ve3))
475.            DireitaLeve = AuxDireitaLeve1;
476.        else if (AuxDireitaLeve2 >= AuxDireitaLeve3)
477.            DireitaLeve = AuxDireitaLeve2;
478.        else
479.            DireitaLeve = AuxDireitaLeve3;
480.
481.        if ((AuxDireita1 >= AuxDireita2) && (AuxDireita1 >= AuxDireita3) && (AuxDire
    ita1 >= AuxDireita4))
482.            Direita = AuxDireita1;
483.        else if ((AuxDireita2 >= AuxDireita3) && (AuxDireita2 >= AuxDireita4))
484.            Direita = AuxDireita2;
485.        else if (AuxDireita3 >= AuxDireita4)
486.            Direita = AuxDireita3;
487.        else
488.            Direita = AuxDireita4;
489.
490.        //Defuzificação
491.
492.        //Cálculo da saída através da média do máximo MOM
493.        SaidaMOM = (5*Esquerda+25*EsquerdaLeve+45*Frente+65*DireitaLeve+85*Direita)/
    (Esquerda+EsquerdaLeve+Frente+DireitaLeve+Direita);
494.
495.        //Seleciona a saída com base no resultado obtido
496.        if (Re==1) //Ré
497.            estado = 6;
498.        else if (SaidaMOM<=15) //Esquerda
499.            estado = 1;
500.        else if (SaidaMOM<=35) //Esquerda leve
501.            estado = 2;
502.        else if (SaidaMOM<=55) //Frente
503.            estado = 3;
504.        else if (SaidaMOM<=75) //Direita leve
505.            estado = 4;
506.        else //Direita
507.            estado = 5;
508.
509.        //Estados referentes à saída
510.
511.        switch (estado)
512.        {
513.            case 0: //Parado
514.            printf("Parado\n");
515.                pwmWrite(1, 0);
516.                digitalWrite(0, LOW);
517.                digitalWrite(2, LOW);
518.                digitalWrite(3, LOW);
519.                digitalWrite(4, LOW);
520.            break;
521.
522.            case 1: //Esquerda
523.            printf("Esquerda\n");
524.                pwmWrite(1, 1024);
525.                digitalWrite(0, LOW);
526.                digitalWrite(2, HIGH);
527.                digitalWrite(3, HIGH);
528.                digitalWrite(4, LOW);
```

```
529.             break;
530.
531.             case 2: //Esquerda leve
532.             printf("Esquerda leve\n");
533.                 pwmWrite(1, 1024);
534.                 digitalWrite(0, LOW);
535.                 digitalWrite(2, HIGH);
536.                 digitalWrite(3, LOW);
537.                 digitalWrite(4, LOW);
538.             break;
539.
540.             case 3: //Frente
541.             printf("Frente\n");
542.                 pwmWrite(1, 1024);
543.                 digitalWrite(0, LOW);
544.                 digitalWrite(2, HIGH);
545.                 digitalWrite(3, LOW);
546.                 digitalWrite(4, HIGH);
547.             break;
548.
549.             case 4: //Direita leve
550.             printf("Direita leve\n");
551.                 pwmWrite(1, 1024);
552.                 digitalWrite(0, LOW);
553.                 digitalWrite(2, LOW);
554.                 digitalWrite(3, LOW);
555.                 digitalWrite(4, HIGH);
556.             break;
557.
558.             case 5: //Direita
559.             printf("Direita\n");
560.                 pwmWrite(1, 1024);
561.                 digitalWrite(0, HIGH);
562.                 digitalWrite(2, LOW);
563.                 digitalWrite(3, LOW);
564.                 digitalWrite(4, HIGH);
565.             break;
566.
567.             case 6: //Ré
568.             printf("Re\n");
569.                 pwmWrite(1, 1024);
570.                 digitalWrite(0, HIGH);
571.                 digitalWrite(2, LOW);
572.                 digitalWrite(3, HIGH);
573.                 digitalWrite(4, LOW);
574.             usleep(2000000); //Ré dura ao menos 2s, para evitar possíveis loops de m
    ovimento
575.             break;
576.
577.             case 7: //Monitora dados recebidos
578.             printf("x = %d\n",*magnet);
579.             //recvfrom(socket_supervisorio,&saida[0],sizeof(char[5]),0,(struct socka
    ddr*)&endereco_supervisorio,&socketsize);
580.             printf("Recebeu: %d %d %d %d %d\n", saida->entrada[0], saida-
    >entrada[1], saida->entrada[2], saida->entrada[3], saida->entrada[4]);
581.             //printf("Recebeu: %d\n", *saida);
582.             break;
583.
584.         }
585.     }
586.
587.     //Reset das saídas digitais ao encerrar programa
588.     pwmWrite(1, 0);
589.     digitalWrite(0, LOW);
590.     digitalWrite(2, LOW);
591.     digitalWrite(3, LOW);
```

```
592.    digitalWrite(4, LOW);
593.
594.    //Encerra memória compartilhada e comunicação wireless
595.    shmdt(magnet);
596.    shmdt(saida);
597.    shmdt(finalizar);
598.    if((shmctl(shmidMagnet,IPC_RMID,NULL))==-1){
599.        printf("Erro ao destruir o segmento.\n");
600.        exit(-1);
601.    }
602.    if((shmctl(shmidSaida,IPC_RMID,NULL))==-1){
603.        printf("Erro ao destruir o segmento.\n");
604.        exit(-1);
605.    }
606.
607.    return EXIT_SUCCESS;
608.}
```

APÊNDICE E – ALGORITMO PARA INICIAR E FINALIZAR PROCESSO

Este *script* realiza apenas o controle de início e parada dos demais processos, considerando comandos numéricos de um teclado como entrada.

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <unistd.h>
4.  #include <arpa/inet.h>
5.  #include <string.h>
6.  #include <sys/ipc.h>
7.  #include <sys/shm.h>
8.
9.  //Endereços de memória compartilhada
10. #define KEYFinalizar 9912
11.
12. int main() {
13.     //Configurações de memória compartilhada
14.     int shmidFinalizar;
15.     char *path="/home/pi/Desktop/Software";
16.     int *finalizar;
17.     int auxEntrada=0;
18.
19.     //Inicializa memória compartilhada
20.     if((shmidFinalizar=shmget(ftok(path,(key_t)KEYFinalizar),sizeof(int*),IPC_CREAT|
    SHM_R|SHM_W))==-1){
21.         printf("\n\nErro na criação do segmento de memoria...\n");
22.         exit(-1);
23.     }
24.
25.     finalizar=shmat(shmidFinalizar,0,0);
26.
27.     //Aguarda comando de teclado para inicializar execução
28.     while (auxEntrada!=1){
29.         usleep(100000);
30.         printf("Digite 1 para iniciar:");
31.         scanf("%i", &auxEntrada);
32.     }
33.
34.     //Seta endereço de memória compartilhada para iniciar execução de todos os progr
    amas
35.     *finalizar=1;
36.
37.     //Aguarda comando de teclado para finalizar programa
38.     while(auxEntrada!=2){
39.         usleep(100000);
40.         printf("Digite 2 para sair:");
41.         scanf("%i", &auxEntrada);
42.     }
43.
44.     //Encerra endereço de memória compartilhada interrompendo a execução de todos os
     programas
45.     *finalizar=0;
46.     shmdt(finalizar);
47.     if((shmctl(shmidFinalizar,IPC_RMID,NULL))==-1){
48.         printf("Erro ao destruir o segmento.\n");
49.         exit(-1);
50.     }
51.
52.     return EXIT_SUCCESS;
53. }
```