isec
Engenharia

MESTRADO EM ENGENHARIA ELETROTÉCNICA

Planeamento de Trajetórias para uma Máquina Florestal

Autor

Tiago da Costa Gameiro

Orientadores

Professor Doutor Nuno Miguel Fonseca Ferreira

Professor Doutor João Paulo Morais Ferreira

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

Coimbra, Outubro de 2023

Instituto Superior de Engenharia
Politécnico de Coimbra

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA

Planeamento de Trajetórias para uma Máquina Florestal

Relatório de Trabalho de Projeto para a obtenção do grau de Mestre em Engenharia Eletrotécnica

Especialização em Automação e Comunicações em Sistemas Industriais

Autor

Tiago da Costa Gameiro

Orientador

Professor Doutor Nuno Miguel Fonseca Ferreira

Co-Orientador

Professor Doutor João Paulo Morais Ferreira

Supervisor na empresa Universidade de Coimbra (UC)

Professor Doutor Carlos Xavier Pais Viegas

Coimbra, Outubro de 2023

INSTITUTO POLITÉCNICO DE COIMBRA

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

AGRADECIMENTOS

Quero expressar a minha gratidão a todos aqueles que desempenharam papéis significativos no meu percurso académico. Em primeiro lugar, desejo agradecer ao nosso respeitado Professor Nuno Ferreira, a sua presença constante, dedicação e orientação foram essenciais para guiar o desenvolvimento do meu projeto, um grande obrigado por estar sempre disponível para oferecer ajuda e conselhos. Gostaria de prestar uma homenagem especial ao Tiago Pereira onde o seu apoio e trabalho em equipa ao longo dos últimos anos foram inestimáveis. A sua dedicação e compromisso não passaram despercebidos e desempenharam um papel crucial no sucesso deste projeto. Expresso também o meu agradecimento ao Professor Carlos Viegas pelo seu apoio e por me ter proporcionado a oportunidade de explorar esta área de investigação. Quero também agradecer ao Engenheiro João Alves pela sua constante assistência e ensino, e ao Professor João Ferreira pelo seu precioso tempo e apoio. Quero estender o meu agradecimento à equipa que sofreu alterações ao longo do tempo, incluindo Diogo Valério, Ricardo Baila, Miguel Gaspar, Marta Nobre, Hugo Santos, Francesco Di Giorgo, Hamid e Babak, pelas discussões produtivas que levaram a soluções inovadoras. O vosso apoio foi a base para atingir resultados excelentes neste projeto. Além disso, os momentos partilhados em equipa, cheios de vontade de aprender, descobrir e enfrentar os desafios deste empolgante projeto, tornam estes últimos anos verdadeiramente memoráveis. Não posso deixar de reconhecer o apoio constante da minha família, desde o meu pai e mãe até ás minhas irmãs. Durante estes tempos desafiantes, eles nunca hesitaram em apoiar-me e dar-me a força necessária para perseguir os meus objetivos, sonhos e futuro. Por último, mas não menos importante, quero agradecer à minha incrível namorada. Ela não só me apoiou ao longo desta jornada, mas também proporcionou uma perspetiva única ao longo do projeto, alargando os meus horizontes para futuras oportunidades profissionais. Obrigado por ser a minha rocha e por se juntar a mim nesta jornada.

RESUMO

Neste trabalho fez-se uma análise de diferentes métodos de navegação de sistemas robóticos autónomos para a limpeza florestal, integrou-se numa máquina florestal um módulo sensorial universal que fornece informações necessárias para o desenvolvimento de um sistema de navegação de modo a ser utilizado em diferentes tipos de máquinas florestais. Apresentamos o protótipo desenvolvido aplicado ao trator LV-600 da MDB de modo a testar o módulo sensorial universal, bem como avaliação dos algoritmos desenvolvidos. O sistema de controlo utiliza um modelo sensorial composto por diversos sensores tais como, um "*GPS*" (*Duro Inertial da SwiftNavigation*), um "*LiDAR*" (Velodyne VLP-16), e diversas câmaras infravermelhas (FLIR ADK) e "*RGBD*" (Realsense d435i), assim como um computador industrial para processar toda a informação adquirida pelo módulo genérico. Para realizar uma navegação em áreas florestais de forma segura e confiável é necessária uma combinação de múltiplos sensores, que permitam aquisição de dados do ambiente em que a máquina florestal se encontra. A navegação neste tipo de ambientes só é possível com deteção e a identificação de obstáculos, bem como saber a posição e a orientação do sistema robótico. O sistema foi desenvolvido na plataforma *ROS (Robot Operating System)* que é uma *framework* bastante útil para o desenvolvimento de aplicações robóticas, este tipo de sistemas funciona suportado em sistemas operacionais abertos, tais como o Linux, neste caso específico foi utilizado o Ubuntu 20.04, além disso fornece ferramentas que permitem reutilizar código desenvolvido por outros investigadores e permite a comunicação de todos os sensores utilizados para este fim de forma distribuída.

A interação dos dados obtidos pelos sensores, bem como o planeamento das rotas a efetuar, são elementos importantes para a análise ao sistema. Apresenta-se uma comparação entre os diferentes métodos de navegação apresentados com o objetivo de descobrir quais são os mais eficazes para tarefas específicas da limpeza florestal.

As primeiras experiências realizadas foram efetuadas em ambiente de simulação e recorreu-se ao modelo do equipamento através da ferramenta de simulação Gazebo e as experiências seguintes efetuaram-se em ambiente real, desenvolvidas no trator florestal, que foi adaptado de forma a poder ser controlado pelo sistema desenvolvido.

Palavras-Chave: Sistemas de Navegação. Robótica Móvel, ROS, Simulação, Sensores.

ABSTRACT

In this work, an analysis of different methods for the navigation of autonomous robotic systems for forest environment was carried out. A universal sensor module was integrated into a forest machine providing the necessary information for the development of a navigation system to be used in various types of forest machinery. We present the developed prototype applied to the MDB LV-600 tractor to test the universal sensor module, as well as the evaluation of the developed algorithms. The control system utilizes a sensor model composed of various sensors such as a SwiftNavigation Duro Inertial GPS, a Velodyne VLP-16 LiDAR, and various infrared cameras (FLIR ADK) and RGBD cameras (Realsense d435i), as well as an industrial computer to process all the information acquired by the generic module. To safely and reliably navigate in forested areas, a combination of multiple sensors is necessary to acquire data from the environment in which the forest machine operates. Navigation in such environments is only possible with obstacle detection and identification, as well as knowing the position and orientation of the robotic system. The system was developed on the ROS (Robot Operating System) platform, which is a highly useful framework for robotic applications. Such systems are supported by open-source operating systems, such as Linux. In this specific case, Ubuntu 20.04 was used. Additionally, ROS provides tools for code reuse developed by other researchers and enables the communication of all sensors used for this purpose in a distributed manner.

The interaction of data obtained by sensors, as well as route planning, are important elements for system analysis. A comparison between the different navigation methods is presented to discover the most effective ones for specific forest cleaning tasks.

The initial experiments were conducted in a simulation environment using the equipment model through the Gazebo simulation tool. Subsequent experiments were carried out in a real environment, developed on the forest tractor, which was adapted to be controlled by the developed system.

Keywords: Navigation Systems, Mobile Robotics, ROS, Simulation, Sensors.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

SIMBOLOGIA E ABREVIATURAS

ADAI	*Associação para o Desenvolvimento da Aerodinâmica Industrial*
AI	*Artificial Intelligence*
APF	*Artificial Potential Field*
API	*Application Programming Interface*
CD	*Cell Decomposition*
GPS	*Global Positioning System*
IAE	*Integral Absolute Error*
ICR	*Instantaneous Center of Rotation*
IMU	*Inertial Measurement Unit*
ISE	*Integral Square Error*
ITAE	*Integral of the Time weighted Absolute Error*
LiDAR	*Light detection and ranging*
MIMO	*Multiple Inputs Multiple Outputs*
MPC	*Model Predictive Control*
PID	*Proportional-Integral-Derivative*
PRM	*Probabilistic RoadMap*
P2P	*Peer to Peer*
RA	*Roadmap Approach*
RGBD	*Red, Green, Blue, Depth*
ROS	*Robot Operating System*
RTK	*Real Time Kinematic*
RVIZ	*ROS Visualization*
SDF	*Simulation Description File*
SEMFIRE	*Safety, Exploration and Maintenance of Forests with Ecological Robotics*
SLAM	*Simultaneous Localization and Mapping*
STL	*Standard Triangle Language*
URDF	*Unified Robot Description Format*
VD	*Voronoi Diagram*
VFH	*Vector Field Histogram*

VG	*Visibility Graph*
XML	*eXtensible Markup Language*
YOLO	*You Only Look Once*

CAPÍTULO 1: INTRODUÇÃO

Os incêndios florestais infelizmente têm vindo a aumentar, tanto em frequência, como em tamanho e intensidade no decorrer dos últimos anos. Estes incêndios causaram perdas na ordem dos milhões de euros, bem como danos em propriedades públicas e privadas, assim como, destruíram bastantes recursos naturais [1]. Em Portugal as mudanças sociais e económicas nas últimas décadas são enormes, desde a diminuição da população nas áreas rurais, como resultado as terras agrícolas foram praticamente abandonadas, assim como o número de rebanhos reduziu de forma significativa, aumentando os combustíveis florestais, quer por o recurso à redução do pasto de gado, mas também através da redução da recolha de lenha, causando o aumento da quantidade de combustível florestal. Em relação aos incêndios florestais, foram feitos diversos estudos sobre as causas, estes revelaram que os incêndios caracterizados por negligência (não limpeza de terrenos, fogueiras, cigarros) foram os que apresentaram o maior número de ocorrências registadas, representando praticamente 49% de todos os casos de incêndio [2]. As remoções das vegetações, através das limpezas, são práticas e tem como principal objetivo a diminuição da probabilidade de ocorrência de incêndios florestais, assim como reduzir a sua rápida propagação, desta forma, possuem o propósito da redução dos danos causados eventualmente pelos incêndios e/ou a melhoria das condições para o seu combate eficaz [3]. Uma possibilidade de solução será a integração de sistemas robóticos, para efetuar a limpeza das florestas, permite que este sistema possa auxiliar, prevenir, bem como reduzir potenciais propagações e, em última análise, até evitar incêndios. A quantidade de material inflamável disponível nas florestas diminui quando o combustível florestal é removido, isso significa que consegue-se diminuir a probabilidade da ocorrência de um incêndio, assim como da probabilidade de este se espalhar com facilidade e de se transformar num incêndio em grande escala difícil de controlar.

1.1 Motivação

A ocorrência de incêndios florestais suscita em mim um profundo sentimento de tristeza e preocupação. A vasta área consumida pelo fogo e a degradação dos ecossistemas naturais provocam uma sensação de desgosto e inquietação. Uma questão que frequentemente me recorre é: "Como posso contribuir para diminuir ou, idealmente, terminar com estes incêndios?" Neste contexto, o desenvolvimento de um robô destinado à prevenção de incêndios florestais representa uma forma de contribuir para a preservação do meio ambiente e o bem-estar das gerações futuras. A aplicação da tecnologia com o propósito de proteger o nosso planeta é uma demonstração do nosso compromisso em utilizar recursos avançados em prol da conservação ambiental. O esforço coletivo e dedicado de equipas de investigação de todo o mundo que trabalham, passo a passo, dia a dia, na evolução da tecnologia em direção a soluções benéficas para o meio ambiente, é digno de admiração. Como Baden Powell dizia: "Deixar o mundo um pouco melhor do que o encontramos." Este lema ecoa na

missão de utilizar a ciência e a inovação para proteger e preservar o nosso mundo, promovendo a harmonia entre a tecnologia e o meio ambiente.

1.2 Enquadramento

Com a colaboração da ADAI (Associação para o Desenvolvimento da Aerodinâmica Industrial) e da UC (Universidade de Coimbra), têm vindo a ser desenvolvidos projetos para o combate e prevenção de incêndios florestais. Este relatório está presente num destes projetos, que tem como objetivo o desenvolvimento de um sistema autónomo para este feito. Desta forma, a equipa dentro deste projeto têm tido muitas alterações de membros ao longo desta jornada, uns vão outros vêm, possibilitando diferentes ideias e formas de realizar e idealizar estes projetos. Com isto menciono a possibilidade de existir informação neste relatório que poderá ir ao encontro de outros documentos, estes realizados pela equipa e pelos membros que por ela passaram. Este projeto resume-se à conversão de um trator que pode ser controlado por rádio frequência para um robô móvel autónomo. Desta forma, é desenvolvido um sistema sensorial, um cérebro para o trator, este que têm uma variedade de sensores e um sistema computacional que permita o controlo adequado para o trator executar estes trabalhos na perfeição.

1.3 Objetivos

Este trabalho tem como principal objetivo o desenvolvimento de um módulo sensorial capaz de controlar um sistema MIMO (*Multiple Input Multiple Output*) como o trator em questão, o LV600 da MDB, de forma autónoma. Múltiplos controladores e algoritmos de desvio de obstáculos serão implementados, onde neste trabalho irá ser realizado estudos e comparações entre eles de forma a ser validado o bom desempenho deste módulo sensorial designado de “Sentry”.

1.4 Estrutura do relatório

Este relatório está dividido e cinco capítulos, onde cada um deles refere-se a uma etapa do trabalho desenvolvido no decorrer deste projeto. O capítulo dois é referente a um estudo prévio realizado para obter uma noção do que já foi implementado. Nestes serão dados alguns exemplos assim como breves explicações dos funcionamentos algorítmicos e controladores. Também será mencionado alguns exemplos onde foram executados projetos robóticos interessantes na área florestal, de combate aos incêndios e da agricultura. O capítulo três refere-se à metodologia, ou seja, uma explicação mais detalhada dos algoritmos escolhidos assim como os controladores. O capítulo quatro mostra os resultados obtidos nos testes realizados em simulação e num ambiente real, nestes são usados os conteúdos explicados no capítulo três. Finaliza-se com o capítulo cinco onde será feita uma conclusão pessoal e científica do trabalho desenvolvido, mencionando alguns comentários para desenvolvimentos futuros.

CAPÍTULO 2: ESTADO DA ARTE

Apesar da inclusão de sistemas robóticos para a limpeza florestal ser uma ideia inovadora para a prevenção de fogos, ainda é algo que não está implementado no terreno, até porque o ambiente florestal apresenta um desafio bastante complexo, tais como a forma de obter uma boa navegação do sistema robótico [4]. Além disso, há um amplo uso desse tipo de sistemas robóticos em diversas áreas, incluindo a agricultura. Um exemplo é a plataforma ALICE da empresa da Solinftec que se destaca com a tecnologia AI. Esta empresa utiliza robôs que monitorizam os insetos e ervas daninhas, realizam análises do solo e avaliam a nutrição das plantas [5]. A empresa *Fieldwork Robotics* dedica-se à colheita de frutas, esta empresa desenvolveu robôs especialmente projetados para a colheita de framboesas, a fim de suprir a escassez de mão de obra [6]. A empresa Ecorobotix desenvolveu o robô ARA, destinado à pulverização de precisão, sistema baseado em inteligência artificial. Esta tecnologia incorpora diversos sensores tais como GPS, uma camera na parte frontal, para permitir a identificação das plantas daninhas para que seja possível a aplicação precisa do herbicida diretamente nestas plantas, utilizando apenas a quantidade exata necessária para o efeito [7].

No entanto em ambiente agrícola já existem diversos tipos de sistemas robóticos para inúmeras aplicações distintas, mas para ambientes florestais o número de aplicações continua a ser reduzido e pode ser compreendida pela complexidade geral do ambiente florestal em comparação ao ambiente agrícola. Esta complexidade manifesta-se nos terrenos íngremes de difícil acesso e não estruturados, o que torna bastante complexo o desenvolvimento de qualquer tipo de sistemas robóticos robustos. Além disso, é necessário haver uma infraestrutura de comunicação pré-instalada para que se possa operar este tipo de sistemas com segurança, o que é um desafio para este tipo de ambientes não estruturados, já que a comunicação remota é fortemente afetada pelo elevado número de árvores que funcionam como obstáculos entre o robô e o agente supervisor [8].

Devido ao ambiente não estruturado das florestas, é necessário um conjunto de sensores de forma a possibilitar a navegação, a informação de perceção 3D é exigida, e é possível obtê-la através de diversos sensores tais como câmaras stereo, de LiDARs 3D, ou através de uma fusão sensorial de diferentes sensores tais como do GPS, do IMU, entre outros [9].

A informação obtida pelos sensores, permite a locomoção do sistema robótico, mas também avaliar o estado do terreno e efetuar o corte necessário da vegetação que se pretende remover. Além disso, apresenta uma dificuldade acrescida, nomeadamente quando as tarefas se realizam em terrenos agressivos e com uma vegetação muito densa [4]. Com o objetivo de desenvolver um robô móvel autónomo flexível, confiável e de manutenção simples, é fundamental estabelecer uma arquitetura robusta, simples e modular, abrangendo os sensores, atuadores e os computadores responsáveis pela execução dos algoritmos [10].

Um outro problema destaca-se no facto de alguns sensores de posição, tais como o GPS/RTK, não apresentarem dados muito precisos, o que dificulta a obtenção correta da posição do sistema e geralmente são estes os locais onde se realizam as tarefas de limpeza [11]. Outra dificuldade no desenvolvimento deste tipo de sistemas é o tipo de solo, que muitas vezes apresenta uma elevada quantidade de elementos soltos tais como as pedras e troncos de árvores, bem como cavidades e buracos. Essas condições adversas podem fazer com que o sistema robótico escorregue ou até fique preso. Além disso, ao contrário dos robôs agrícolas durante tempestades são colocados em abrigos específicos, os robôs florestais operam após eventos climáticos adversos. Isso significa que precisam de sistemas de comunicação e locomoção específicos para estes tipos de ambientes, e também devem ser capazes de permanecer em pleno funcionamento mesmo após a ocorrência de uma tempestade ou até um incêndio [8].

Atualmente estas tarefas são realizadas de forma manual utilizando tratores, ou até tratores guiados de forma remota, estes últimos começam a ser bastante utilizados, porque foram desenvolvidos com o objetivo de proteger o utilizador durante as tarefas de limpeza. Este sistema permitem que o utilizador esteja a uma distância de segurança e consiga controlar a máquina através com um comando remoto, que permite controlar todas os comportamentos da máquina florestal [4].

O trator LV-600 da MDB é um sistema diferencial, ou seja, tem dois motores, no entanto, existem muitas outras empresas que possuem tratores semelhantes, tais como a Prinoth, o grupo FAE, a Energreen e a McConnel conforme se observam nas Figuras 1 à Figura 4 e na Tabela 1 é possível observar as características mais relevantes de cada um.

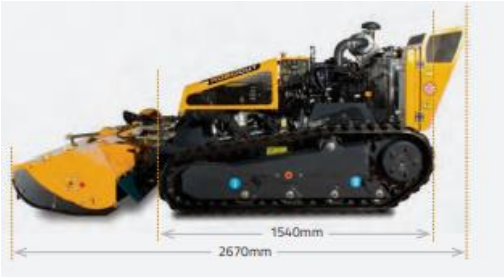


Figura 1 - McConnel Trator.

Figura 2 - Prinoth Trator.

Figura 3 - Energreen Trator.

Figura 4 - FAE Group Trator.

Tabela 1 - Características dos tratores.

	Largura	Comprimento	Altura	Peso
McConnel	1310 mm	2670 mm	1120 mm	1390 kg
Prinoth	1650 mm	2905 mm	1455 mm	2590 kg
Energreen	2032 mm	3140,2 mm	1270 mm	2949,7 kg
FAE Group	2463,8 mm	6477 mm	2717,8 mm	10840,85 kg

O desenvolvimento de um módulo sensorial universal centra-se na possibilidade de ser utilizado por qualquer tipo de máquina florestal que seja controlada da mesma forma que o trator em questão, ou seja, caso este módulo sensorial seja aplicado num outro trator, este terá o mesmo tipo de funcionamento. Ou seja, robotizar um trator.

2.1 Robótica

A área da robótica tem tido um desenvolvimento impressionante onde a expressão "robô" teve o seu primeiro uso registado na peça de 1920, "R.U.R." (Robôs Universais de Rossum), escrita pelo dramaturgo checo Karl Kapek. A palavra "robota", em tcheco, que significa trabalhador, foi utilizada pela primeira vez nesse contexto[12]. Foi em 1920 que esta palavra, hoje tão comum no nosso quotidiano, foi pronunciada pela primeira vez, marcando o início de um progresso tecnológico considerável desde então.

2.1.1 Classificação Robótica

Classificar robôs é uma tarefa complexa devido à diversidade em tamanho, forma e habilidades. Embora cada robô seja único, muitos compartilham características comuns que permitem agrupamentos. Muitas vertentes da robótica têm vindo a aparecer conforme a evolução aumenta, nomeadamente [13]:

- Robótica Espacial e aérea;
- Robótica Aquática;
- Robótica de Serviços;
- Robótica Autónoma;
- Robótica Médica;
- Robótica Educacional;
- Robótica Industrial;
- Robótica Florestal;
- Outras.

Como foi referido, é complexo classificar as áreas da robótica até porque a sua utilização apresenta-se em qualquer área. Mas, como já foi referido, a que iremos abordar serão apenas a área da robótica florestal. Em todas estas áreas da robótica existe a robótica móvel, onde os robôs podem ser capazes de se mover autonomamente e tomar decisões com base na perceção do ambiente [12]. Para tal

os robôs necessitam de entrada de dados, processamento e ações para responder às mudanças no mundo. Tendências atuais na robótica incluem inteligência artificial, condução autónoma, comunicação em rede, cooperação, nano robótica, interfaces mais amigáveis, interação segura com humanos e expressão de emoções. Essas tendências estão sendo aplicadas em diversas áreas, como medicina, indústria e serviços. O campo da robótica móveis continuará a evoluir nos próximos anos [14]. A robótica móvel abrange a locomoção, perceção, cognição e navegação. A locomoção envolve mecânica e controlo, a perceção abrange análise de sinais e visão computacional, a cognição analisa dados e toma ações, e a navegação requer planeamento e inteligência artificial [14]. Podemos distribuir a locomoção por duas áreas, robôs móveis de locomoção a pernas e a rodas [15]. Salientando que veículos com rodas funcionam bem em superfícies pavimentadas, mas enfrentam limitações em terrenos irregulares. Assim sendo, mais de metade da superfície terrestre é inacessível para este tipo de veículos, como opções a este tipo de robôs, existem os veículos com lagartas, que oferecem maior mobilidade, embora ainda apresentem desafios e elevado consumo de energia. No entanto, os veículos com pernas possuem vantagens em terrenos naturais, uma vez que podem usar apoios para cada pé, permitindo a movimentação em terrenos irregulares [16]. Os veículos com rodas deixam sulcos no solo, o que pode ser uma desvantagem na área que iremos estudar. Os veículos com pernas também podem economizar energia em superfícies macias e conseguem alterar direções sem derrapar devido às articulações existentes nas pernas. Em resumo, as inclusão de pernas proporcionam uma melhor mobilidade e eficiência em terrenos diversos [16]. No entanto, iremos abordar os robôs para o combate e a prevenção de fogos, robôs que utilizam lagartas para a sua locomoção.

2.1.2 Exemplos de Robótica Florestal e de Combate a Incêndios

Apesar de existirem poucos trabalhos de investigação nesta área, existem alguns exemplos realizados que podemos mencionar, nomeadamente o projeto SEMFIRE como se apresenta na Figura 5 que utiliza diferentes tecnologias robóticas, onde uma delas é o Ranger, que é um robô com lagartas de grande escala que têm como objetivo desbastar vegetação, além deste projeto foi implementado outro designado de AgRob V18 [4] onde um robô recolhe biomassa florestal.

Figura 5 - SEMFIRE Ranger Robot [17].

Uma empresa no sector militar, *Milrem Robotics*, desenvolveu duas versões de robôs que lutam nas operações de combate aos fogos, onde um deles centra-se em extinguir o fogo e o outro em assistir o transporte em ambientes hostis e de difícil acesso. Estes robôs são o *Multiscope Rescue with Hydra* e o *Multiscope Rescue Hose Cartridge* [8] como se mostram nas Figuras 6 e 7.

Figura 6 - Multiscope Rescue with Hydra [18].

Figura 7 - Multiscope Rescue Hose Cartridge [19].

A empresa *Shark Robotics*, desenvolveu um robô que em vez de substituir os bombeiros, colabora nas suas tarefas de combate a incêndios, o robô Colossus possui uma bomba de água para o combate de incêndios e consegue carregar até 500kg de material, pessoas feridas, etc..., é feito de alumínio soldado a aço aeronáutico, tornando-o assim leve e resistente a temperaturas até 900ºC [8] conforme se mostra na Figura 8.

Figura 8 – Collossus [20].

Foi este robô em concreto que auxiliou os bombeiros no combate ao fogo de Notre-Dame [20]. É evidente que o avanço tecnológico está a impulsionar a utilização de robôs em situações perigosas, como incêndios, trazendo consigo diversas vantagens. A tecnologia robótica, que faz parte da atual revolução industrial, desempenha um papel fundamental. Robôs podem operar em cenários perigosos, interagindo remotamente ou autonomamente com humanos. Isso não só reduz os riscos para os bombeiros, mas também agiliza as operações de combate a incêndios [21]. A robótica

traz consigo a capacidade de enfrentar tarefas complexas que podem ser perigosas ou desafiantes para os humanos, oferecendo uma abordagem mais eficaz e segura para lidar com situações críticas. O crescimento da população e o avanço tecnológico têm contribuído para o aumento de acidentes e riscos de incêndio [22]. A robótica surge como solução para proteger o ambiente e vidas humanas, através de robôs capazes de detetar incêndios remotamente. Num contexto de automação em que os veículos autónomos estão em ascensão, os bombeiros enfrentam riscos constantes. Para lidar com este problema, estão a ser desenvolvidos estes sistemas autónomos de robôs, que procuram e extinguem incêndios, reduzindo os riscos humanos envolvidos.

2.2 Controlo

Estes sistemas autónomos necessitam de uma componente computacional para que este tipo de máquinas possa realizar tarefas. O controlador de um robô desempenha o papel de processar informações e comandos para efetuar movimentos. Geralmente, trata-se de um microcontrolador ou de um computador que armazena dados sobre o robô e o seu meio ambiente, executando programas para operar o robô. O sistema de controlo coordena os dados adquiridos pelos sensores, efetua decisões com base nos objetivos do robô e envia comandos para os atuadores a fim de movimentar toda a estrutura. O sistema de controlo é essencial para o funcionamento do robô e a sua capacidade de se adaptar ao ambiente. Existem várias técnicas de controlo, incluindo estratégias de controlo em malha aberta e em malha fechada. No controlo em malha aberta, operadores humanos enviam instruções e o robô executa o que foi solicitado. Já numa estratégia em controlo em malha fechada, o robô utiliza informações dos sensores para ajustar o seu comportamento em tempo real. A escolha da estratégia de controlo depende da sua complexidade e das tarefas que deve executar. Em resumo, o sistema de controlo é responsável por traduzir informações em ações para permitir que o robô realize as suas tarefas de forma eficaz e precisa [23].

Podemos então referir três tipos de controladores muito utilizados nesta área, nomeadamente o controlo clássico designado por Proporcional, Integral e Derivativo (PID), o controlo inteligente e adaptativo (Fuzzy) e o controlo preditivo [24].

2.2.1 Controlador PID

O Controlador PID é um controlador vastamente utilizado em sistemas de controlo linear, fácil de implementar e específico para controlar processos e variáveis (tais como a temperatura, a pressão, a velocidade, etc.) para um valor desejado [25]. O controlo PID envolve três controlos separados cujas saídas se somam para formar um único sinal de saída. A leitura atual do parâmetro controlado é subtraída do comando para criar um sinal de erro. Esse sinal de erro é processado por três unidades distintas [26]. O PID é muito utilizado na indústria devido à sua eficácia e simplicidade de implementação conforme se mostra na Figura 9 [27].

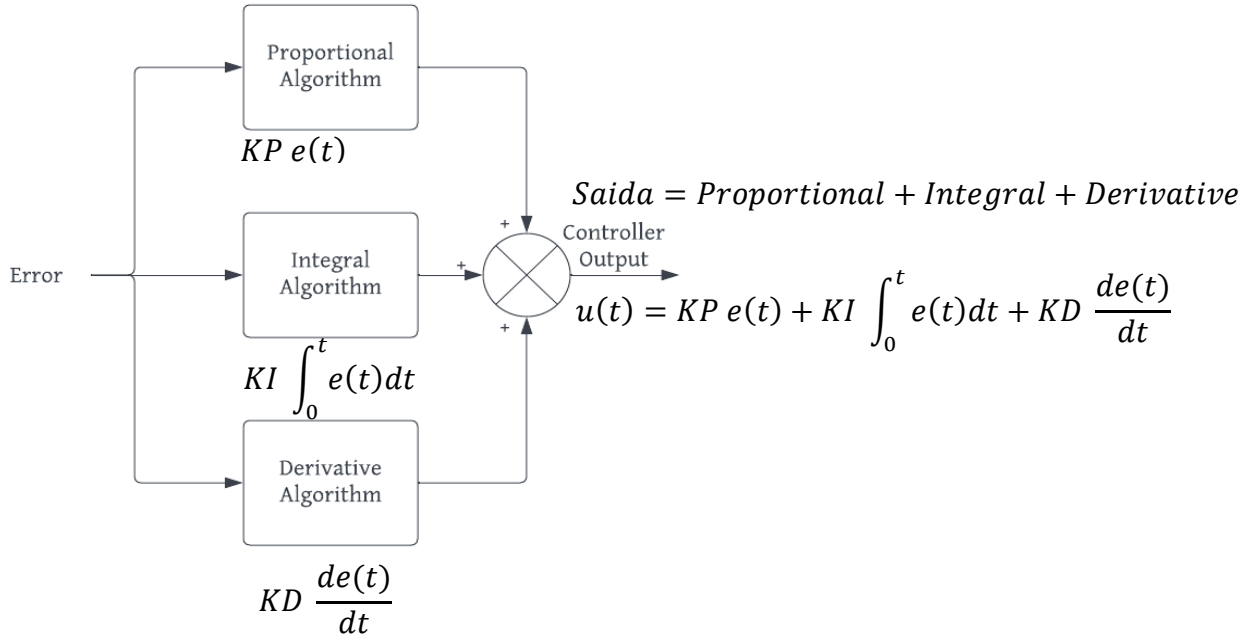


Figura 9 - Controlador PID.

A descrição do funcionamento das várias ações do controlador:

A ação proporcional (P) também designado por ganho proporcional (Kp) o erro entre o valor desejado e o valor real é multiplicando por este coeficiente, assim quanto maior o erro, maior a correção do controlador. O controlador proporcional, pode corrigir o erro, mas se o ganho for elevado este pode causar oscilações em torno do valor desejado [12], [14, 15].

$$Proportional = KP\ e(t) \quad (1)$$

A ação Integral (I) serve para eliminar o erro acumulado ao longo do tempo. Esta ação de controlo é designada por ganho integral (Ki), permite que o controlador corrija erros persistentes, mesmo que a ação proporcional, não seja suficiente para as eliminar. Além disso, o termo integral é útil para eliminar o erro de estado estacionário, que pode ocorrer quando existe uma perturbação constante no sistema[19][22].

$$Integral = KI \int_0^t e(t)dt \quad (2)$$

A ação Derivativa (D) prevê a variação da mudança do erro ao longo do tempo, esta ação de controlo é designada por ganho derivativo (Kd) e esta ação é útil para prevenir oscilações excessivas e serve também para reduzir a resposta do controlador a mudanças rápidas na variável erro, esta ação atua como um "amortecedor" que suaviza a resposta e melhora a estabilidade geral do sistema [23, 24].

$$Derivative = KD \frac{de(t)}{dt} \quad (3)$$

A combinação destes termos permite que o controlador reaja rapidamente a perturbações, corrija erros persistentes e mantenha a variável controlada próxima do

valor desejado. A sintonia dos coeficientes é essencial para otimizar o desempenho do sistema, minimizando oscilações e melhorando o tempo de resposta. O controlador PID é a contribuição das diferentes ações de controlo descritas. Mais de 95% dos problemas de controlo podem ser resolvidos por este tipo de sistemas de controlo, embora alguns possam ter apenas duas ações de controlo do tipo PI sem a ação derivativa ou PD sem a ação integral [30].

2.2.1.1 Cálculos dos parâmetros do PID

Para calcular os ganhos respetivos do controlador existem vários métodos, estes que podem ser distribuídos por duas categorias, os métodos em malha fechada e em malha aberta. Existem inúmeros métodos que podem ser implementados para fazer o *tuning* do controlador PID, como o método Ziegler-Nichols (Z-N), Tyreus-Luyben (T_L), C-H-R, Cohen and Coon (C-C), Fertik, Ciancone-Marline (C-M), etc [31]. Um sistema de controlo em malha fechada é um sistema que utiliza o controlo com retroação, começa-se por colocar os ganhos integral e derivativo a zero e, depois, aumenta-se gradualmente o ganho proporcional até que a saída do sistema oscile de forma sustentada e simétrica em torno do valor de referência, indicando uma estabilidade limite [32]. Num sistema em malha aberta, a saída não é comparada com a entrada [33] e precisa apenas da resposta ao degrau do sistema em malha aberta, o mais indicado é que esta resposta tenha o formato de um S[32]. Podemos observar na Figura 10 os exemplos de um sistema em malha aberta e fechada.

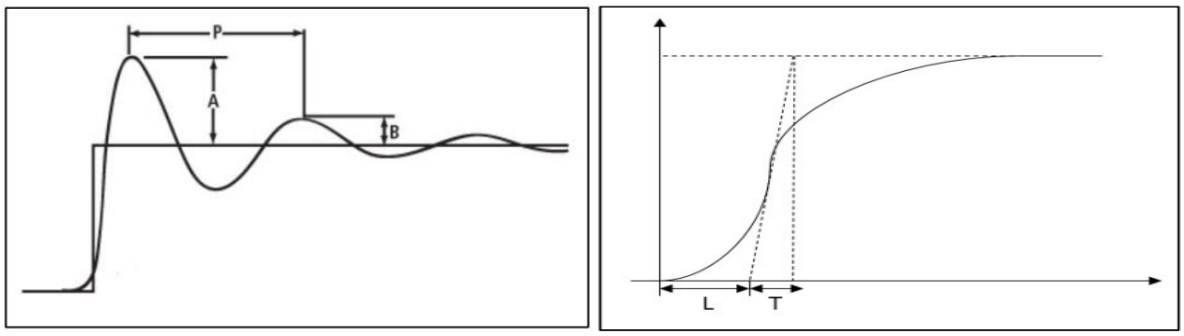


Figura 10 - Malha Fechada e Malha Aberta [32].

No sistema de malha fechada, o ***P*** representa o período de amortecimento e ***A*** e ***B*** são os primeiros valores de pico do sistema. No sistema em malha aberta o ***T*** representa a constante de tempo e o tempo ***L*** é o atraso na resposta [32].

Os controladores Z-N e C-C são controladores simples e fáceis de implementar. O método Cohen-Coon (CC) para o controlador PID é uma extensão do método Ziegler-Nichols. Esta técnica melhora a resposta em regime permanente, especialmente quando há um grande atraso no processo em relação à constante de tempo em malha aberta [34]. Para atingir este objetivo, são obtidos os parâmetros PID a partir de experiências em malha aberta, ver Figura 11.

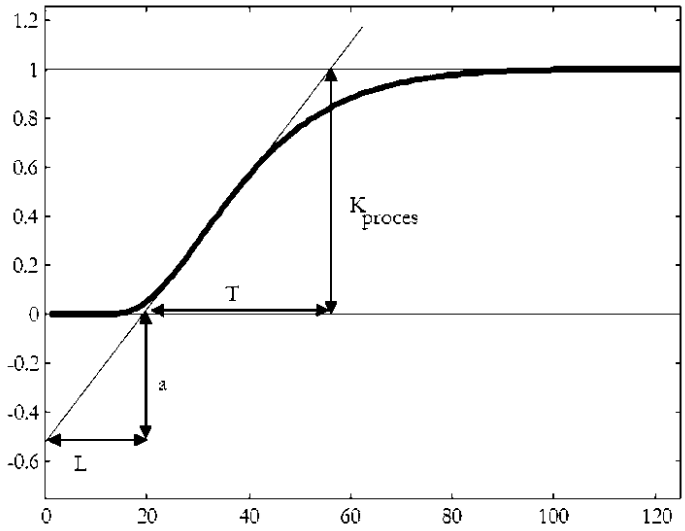


Figura 11 - Malha Aberta Cohen Coon [35].

A partir dos dados obtidos é possível calcular os ganhos associados conforme podemos observar na tabela 2.

Tabela 2 - Fórmulas Calculo dos Ganhos [34].

Cohen Coon	K_C	T_I	T_D
PID	$\frac{1,35}{a}\left(1+\frac{0,18\tau}{1-\tau}\right)$	$\frac{2,5-2,0\tau}{1-0,39\tau}*L$	$\frac{0,37-0,37\tau}{1-0,81\tau}*L$
$\tau=\frac{L}{L+T}$		$a=\frac{K_{process}*L}{T}$	

2.2.1.2 Erros dos Controladores PID

Na metodologia de conceção de um controlador PID, um dos critérios de desempenho mais importantes é a diferença (erro) entre a saída do sistema e o valor de referência. Geralmente, as funções de otimização são formuladas com base em equações de erro, sendo as três mais comuns: ITAE, IAE e ISE [36].

$\mathrm{IAE} = \int_0^t |e(t)|dt$ (Integral da Magnitude Absoluta do Erro)

$\mathrm{ISE} = \int_0^t e(t)^2 dt$ (Integral do Erro Quadrático)

$\mathrm{ITAE} = \int_0^t t|e(t)|dt$ (Integral do Tempo multiplicado pela Magnitude Absoluta do Erro)

2.2.2 Controlador com Lógica *Fuzzy*

A lógica *fuzzy* trata-se de uma maneira de modular a razão lógica quando a verdade de um estado não é binária, ela surgiu na década de 1960 como uma abordagem para lidar com a imprecisão e a incerteza presentes nos sistemas de perceção. Desde então, tem sido aplicada em diversas aplicações de engenharia. Esta abordagem é considerada uma solução mais simples para resolver muitos problemas de controlo não-linear, incluindo aqueles relacionados com navegação e controlo em robótica. A lógica *fuzzy* apresenta vantagens em relação às soluções tradicionais, uma vez que permite que os computadores se comportem de forma mais próxima aos humanos, respondendo eficazmente a entradas complexas e lidando com conceitos linguísticos como "muito quente", "muito frio" ou "ideal" [23].

Exemplo aplicado a um robô de lagartas que segue um destino onde o controlador *fuzzy* tem duas variáveis de entrada, o erro da distância a que o robô se encontra da trajetória e o erro relativamente ao seu ângulo de orientação. A variável de saída do robô é a velocidade angular do trator conforme se mostra na Figura 12.

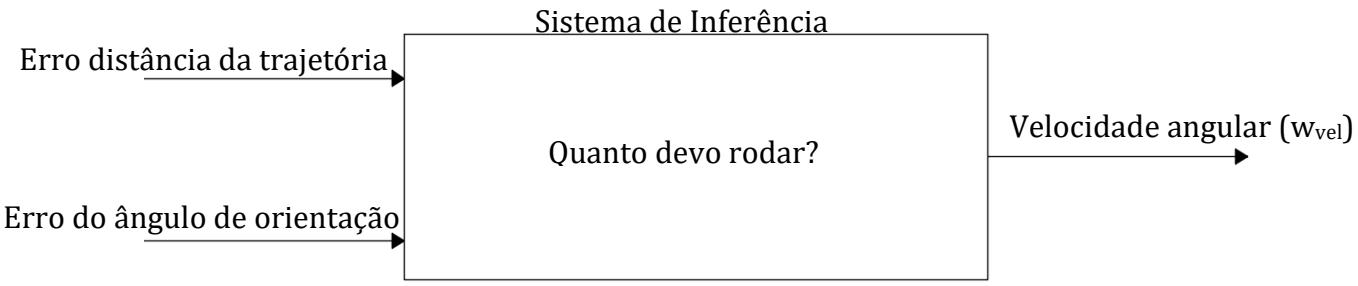


Figura 12 - Sistema de Inferência *Fuzzy*.

Podemos observar o sistema de inferência *fuzzy*, onde entram duas variáveis de entrada e sai uma variável de saída. O sistema completo é composto por várias partes, a fuzificação, onde converte o valor de entrada (valor crespo) na sua representação difusa [37]. Posteriormente esta variável difusa passa pela inferência, esta que é composta pela base de dados e pela base de regras onde são definidas estratégias de controlo que expressam a informação de todo o processo de controlo. Nesta mesma etapa, a variável difusa vai ser transformada numa outra variável difusa onde finaliza no processo de desfuzificação, esta que converte a variável de saída numa variável de valor crespo [38]. Podemos observar este processo na Figura 13, esta que foi adaptada do artigo [39].

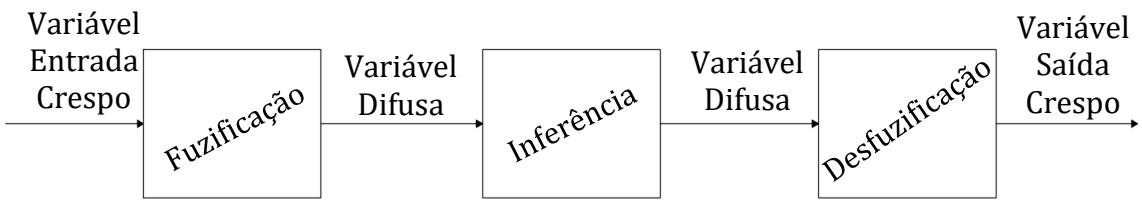


Figura 13 - Etapas da Lógica *Fuzzy*.

Na lógica *fuzzy* existem três operadores lógicos que podemos utilizar, eles são os conhecidos operadores AND, OR e NOT [40]. E estes comportam-se da seguinte forma:

Tabela 3 - Representação dos operadores lógicos e respetivas funções.

A	B	OR	Max(A,B)
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

A	B	AND	Min(A,B)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

A	NOT	1 – A
0	1	1
1	0	0

Como foi referenciado anteriormente existem duas variáveis crespas à entrada do controlador. Então vamos começar pelo processo de fuzificação. Imaginemos que neste momento os valores que recebemos das variaveis de erro são: 4,5 metros da trajetória e um erro de 2,5 graus da sua orientação, como podemos observar na Figura 14.

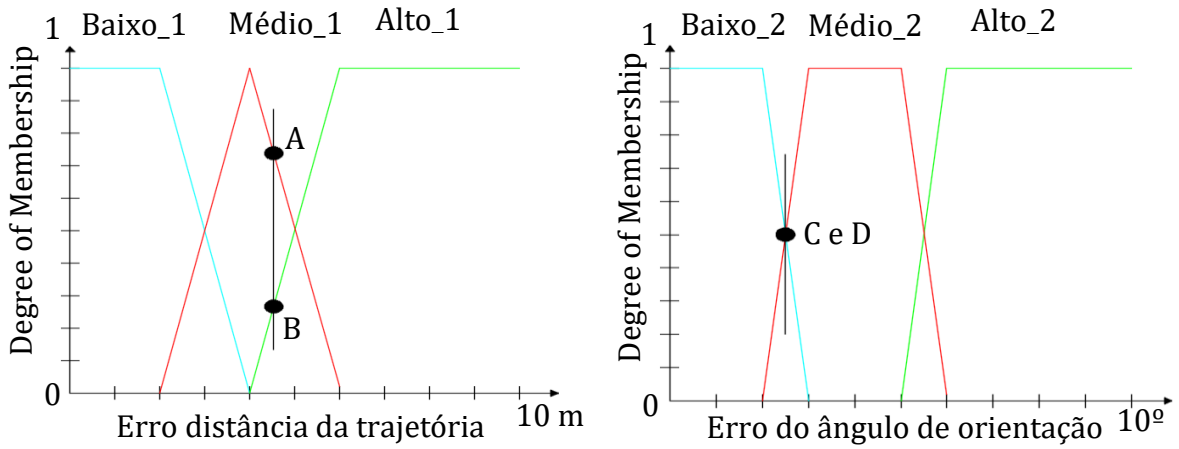


Figura 14 - Fuzificação das duas variáveis de entrada.

De acordo com estes valores podemos ver na Tabela 4, estão descritos os valores associados a cada *Degree of Membership*.

Tabela 4 - Valores referentes aos gráficos.

A	70 % de Médio$_1$ do erro da distância a que está da trajetória
B	30 % de Alto$_1$ do erro da distância a que está da trajetória
C	50 % de Baixo$_2$ do erro do seu angulo de orientação
D	50 % de Médio$_2$ do erro do seu angulo de orientação

Erro da distância a que está da trajetória tem um valor difuso, Dist = [0; 0,70; 0,30] respetivos aos valores [Baixo$_1$; Médio$_1$; Alto$_1$] e o erro do ângulo de orientação um valor difuso, Ang = [0,50; 0,50; 0]. Tendo convertido a variável para valores difusos, passamos para a parte da inferência. Onde para exemplificação foram descritas três regras para explicar o funcionamento desta lógica.

Tabela 5 - Base de Regras.

Nº Regra	Condição
1	**IF** dist = Baixo_1 **AND** angle = Baixo_2 **THEN** w_{vel} = Baixo_3
2	**IF** dist = Médio_1 **OR** angle = Médio_2 **THEN** w_{vel} = Médio_3
3	**IF** dist = Alto_1 **OR** angle = Alto_2 **THEN** w_{vel} = Alto_3

Estando estas regras implementadas, agora é possível converter a variável difusa para a próxima etapa.

Tabela 6 - Valores das Condições.

Nº Regra	Condição
1	AND -> w_{vel} = min (L_{Dist}, L_{Ang}) = min(0; 0,50) = 0
2	OR -> w_{vel} = max (M_{Dist}, M_{Ang}) = max(0,70; 0,50) = 0.,70
3	AND -> w_{vel} = max (A_{Dist}, A_{Ang}) = max(0,30; 0) = 0,30

Desta forma obteve-se a variável difusa convertida, onde w_{vel} = [0; 0,70; 0,30] e procedeu-se à desfuzificação onde a partir desta variável preencheu-se os gráficos com 70% do valor médio e com 30% do valor alto conforme a variavel difusa.

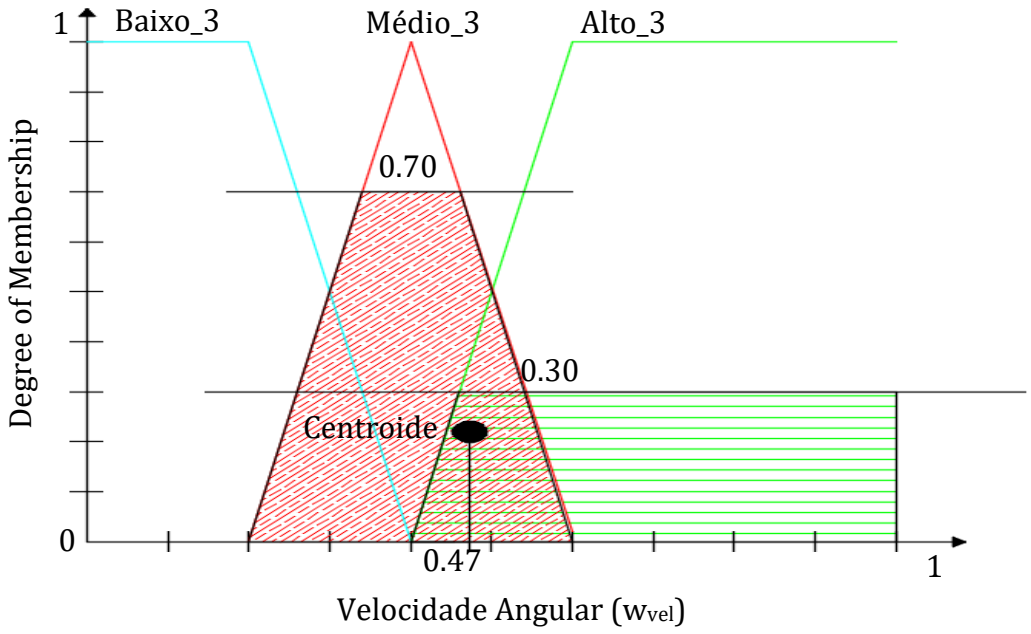


Figura 15 – Desfuzificação.

Após este preenchimento o valor de w_{vel} consiste no valor de x da coordenada do centroide da junção deste preenchimento. A Figura 15 é um mero exemplo demonstrativo e não calculado, cujo valor de x do centroide deste exemplo é de 0,47 rad/s. O que significa que através da lógica difusa, de das variáveis de entrada, a variável crespa de saída seria do valor de 0,47 rad/s.

A lógica *Fuzzy*, mesmo não sendo a única forma de lidar com conceitos ambíguos, destaca-se na engenharia de controlo devido à sua capacidade de aproximar funções complexas [41]. As suas vantagens incluem um design intuitivo da base de conhecimento, uma interface simples, uma simplificação de cálculos, uma aprendizagem adaptativa, uma validação consistente, um tratamento de ambiguidade e a integração de algoritmos de regulação. Esta abordagem oferece uma maior flexibilidade e eficácia na tomada de decisões em sistemas de controlo complexos. A lógica *Fuzzy* desempenha um papel crucial ao proporcionar uma ferramenta versátil e adaptável para enfrentar os desafios inerentes aos sistemas dinâmicos e incertos, permitindo uma abordagem mais natural e abrangente na resolução de problemas complexos de engenharia [39].

2.2.3 Controlador Preditivo

O conceito fundamental do controlo preditivo é bastante simples, são realizadas previsões comportamentais para diferentes ações de controlo. Através da otimização de um critério que considera essas ações, é possível determinar a melhor ação a aplicar ao processo. Uma analogia que ilustra bem este tipo de controlo dos anteriores é a condução de um automóvel em que o veículo é controlado através do retrovisor corrigindo a trajetória quando se detetada um erro. A abordagem preditiva é de antecipação e é mais eficaz, até porque envolve o modelo do veículo, e observa a trajetória mais à frente (com base em valores futuros de referência), considera possíveis perturbações (como um veículo à frente, um peão atravessando a estrada ou um buraco na estrada) e otimiza um critério tal como o (tempo, segurança, etc.). Dessa forma, por meio da previsão e antecipação, define-se uma sequência de ações de controlo para alcançar os melhores resultados [42]. O MPC *(Model Predictive Control*) é uma abordagem que se baseia em modelos disponíveis em várias disciplinas, isso permite que o MPC aproveita os conhecimentos e informações já existentes em várias áreas para melhorar o processo de controlo, evitando a necessidade de formulações complexas de leis de controlo explícitas. Em vez disso, utiliza a otimização baseada em modelo para determinar automaticamente a lei de controlo. Isso proporciona flexibilidade e utiliza o conhecimento acumulado ao longo do tempo, tornando o MPC uma opção vantajosa [43].

Na Figura 16 está representado o diagrama de blocos do Controlo Preditivo, este que se assemelha ao PID pois também é um controlo de malha fechada, no entanto usa o modelo que “prevê” eventos futuros [43].

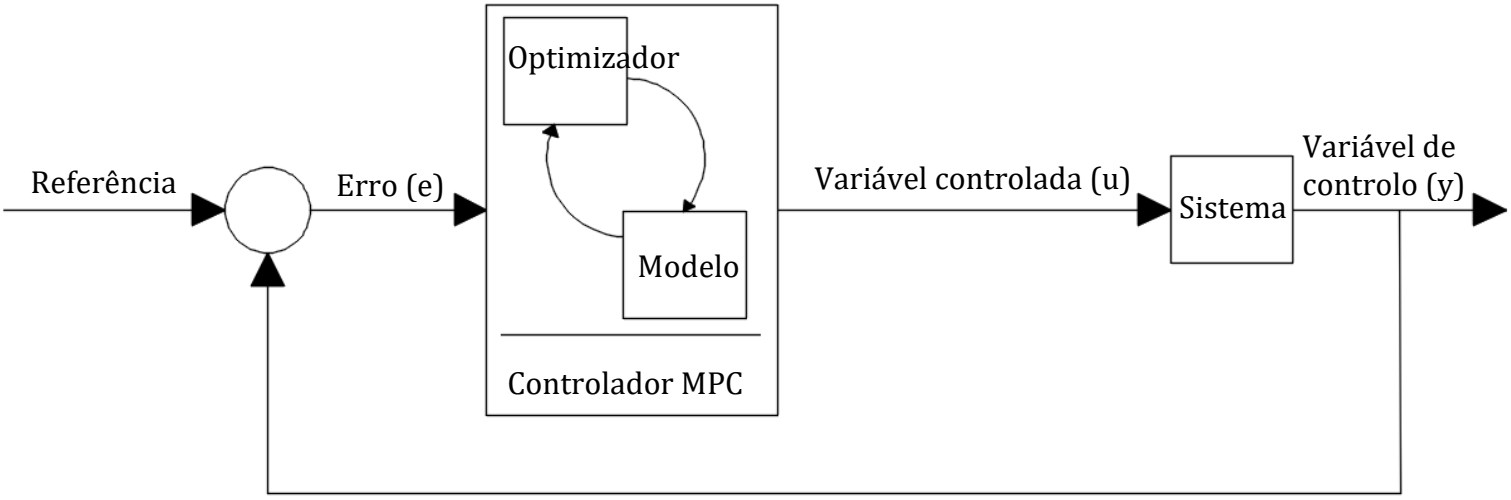


Figura 16 - Diagrama de Blocos de um Modelo de Controlo Preditivo.

Na Figura 17 está representado um trator que quer seguir uma trajetória, a linha azul é a nossa referência. As restantes linhas (Rosa, Laranja, Verde) são trajetos futuros possíveis que o robô pode realizar para se aproximar da referência. Este trajeto futuro está presente no horizonte de previsão, este que deve ser extenso o bastante para refletir o impacto de uma alteração na variável controlada "**u**" sobre a variável de controlo "**y**" [43].

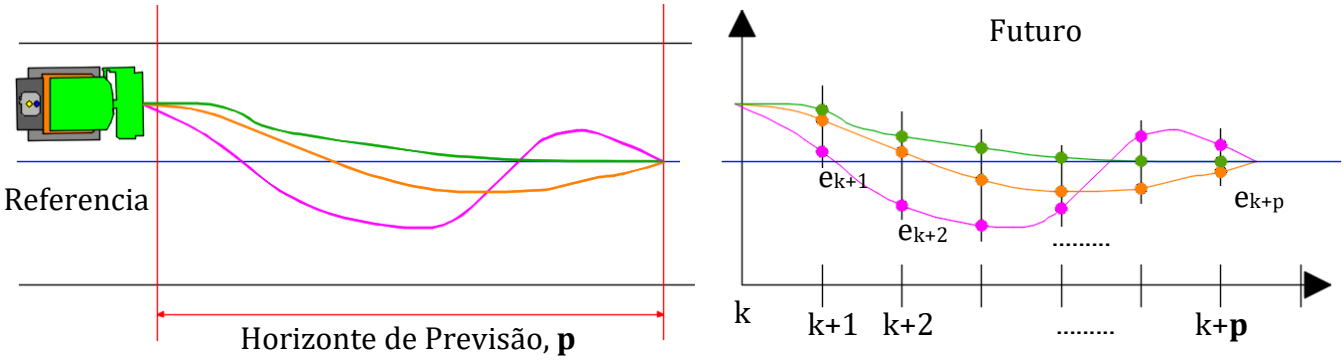


Figura 17 - Exemplo prático do funcionamento do MPC.

Podemos observar os pontos nos instantes temporais relativos aos respetivos caminhos previamente calculados. No entanto o controlador tem que escolher qual o melhor caminho que o robô tem que seguir. Considerando que a nossa variável controlada "**u**" representa a velocidade angular do trator (w_{vel}) e "**y**" a distância a que se encontra da trajetória. O MPC precisa de escolher qual é o melhor caminho preditivo que é mais próximo à referência, por isso simula múltiplos cenários idênticos, no entanto ele realiza de forma sistemática com o auxílio do Optimizador. Desta forma, o MPC tenta minimizar ao máximo o erro da distância a que está da referência (e_{k+p}) e a mudança nos valores da velocidade angular (Δu), como podemos observar na Figura 18.

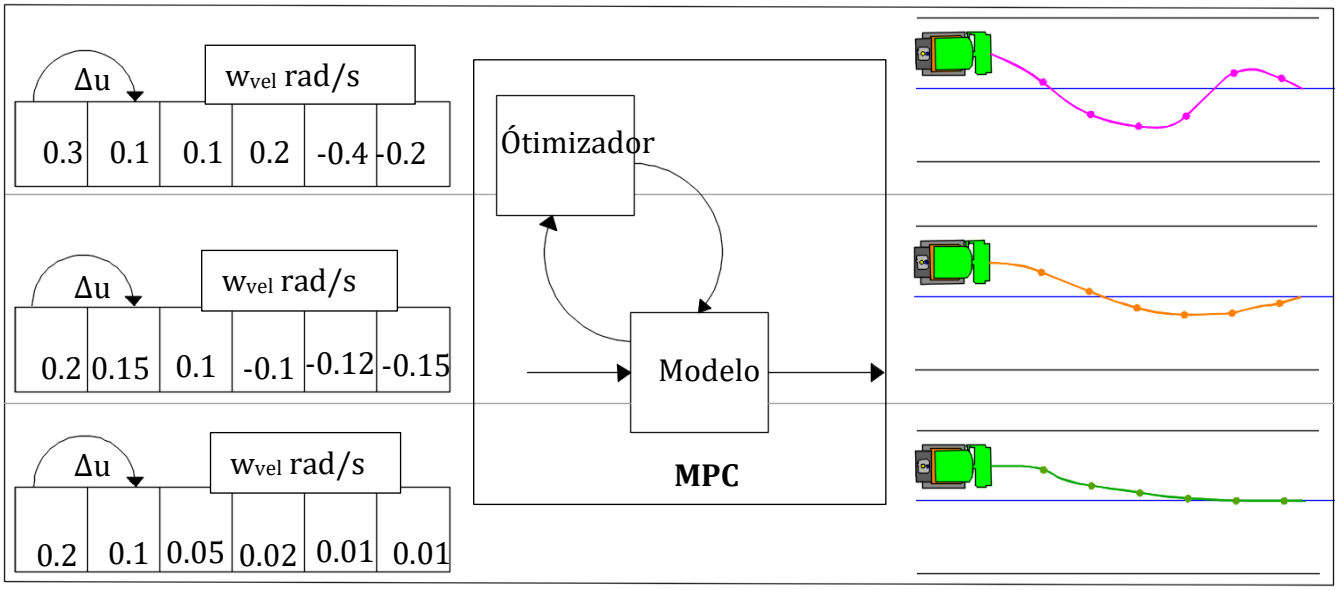


Figura 18 - Funcionamento do MPC.

O MPC realiza a sua escolha referente ao trajeto com menor valor proveniente da função de custo. Este valor é calculado da seguinte forma, obtido do artigo [44]:

$$J = \sum_{i=1}^{p} {e_{k+i}}^2 + \sum_{i=0}^{p-1} {\Delta u_{k+i}}^2 \qquad (4)$$

A cada iteração (k+i), todo este processo é repetido, pois como o horizonte de previsão mudou, todo o seu controlo tem que ser outra vez realizado porque a cada momento que passa em todo o nosso meio ambiente pode haver alterações tais como deslizes no terreno ou influências do vento que mudam a posição do trator. No artigo [45], foi recomendada a utilização de um controlador PID para garantir uma navegação estável entre pontos de referência num robô de tração diferencial. Os resultados mostraram que o planeador recomendado é eficiente em termos de comprimento do percurso e também considera a praticidade do trajeto proposto.

2.3 Tipos de Navegação

A navegação de robôs móveis é um tema crucial no campo da robótica. Estes robôs são conhecidos pelas suas capacidades inteligentes e têm uma ampla gama de aplicações, desde o transporte até à indústria e robótica de salvamento. O planeamento de trajetos é uma das partes mais importantes e essenciais na navegação autónoma de robôs móveis [46]. Vários investigadores desenvolveram algoritmos para evitar obstáculos, dividindo-se principalmente em duas áreas: planeamento de trajetória global e planeamento de trajetória local. No primeiro, o robô navega com base em informações prévias sobre o ambiente e os obstáculos. No segundo, o robô adapta-se dinamicamente com base em obstáculos detetados

localmente. Os métodos de planeamento de trajetória local são mais práticos para robôs móveis, uma vez que lidam com ambientes complexos e em constante mudança [47].

2.3.1 Navegação Global

Existem alguns métodos de navegação global como por exemplo:

2.3.1.1 Dijkstra

Este algoritmo baseia-se na da teoria dos grafos, este que calcula o caminho mais curto entre um nó inicial e todos os outros nós num grafo. Ele tem a abordagem escolher as distâncias mais curtas à medida que percorre o grafo. À exceção do nó inicial, que tem uma distância de 0, o algoritmo começa com distâncias infinitas entre todos os nós. À medida que avança, atualiza a distância de cada nó, sempre escolhendo o nó não visitado com a menor distância. O objetivo é identificar a rota mais curta para cada nó em relação ao ponto de partida [48].

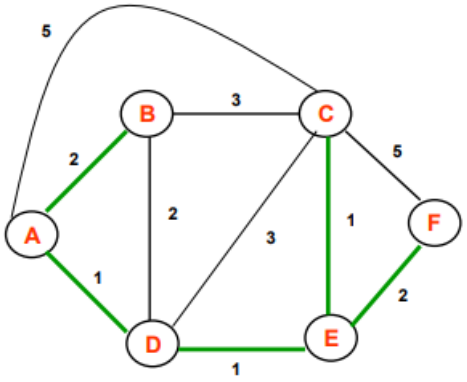


Figura 19 - Dijkstra Grafos.

A Figura 19 apresenta um conjunto de caminhos como se pode observar no trabalho [48] e demonstra todos os caminhos mais curtos entre o nó A e os restantes.

2.3.1.2 Cell Decomposition (CD)

Neste método, a área é dividida em grades não sobrepostas e utiliza-se grafos interligados para permitir a transição de uma célula para outra, alcançando o objetivo desejado. Durante essa transição, dá-se prioridade às células puras, ou seja, aquelas sem obstáculos, para o planeamento do percurso desde o ponto inicial até o ponto de destino. Caso surjam células corrompidas, que contenham obstáculos no trajeto, estas são subdivididas em duas novas células para obter uma célula pura, que é incorporada à sequência ao determinar o caminho ideal entre as posições iniciais e de destino. A sequência de células puras que liga essas posições define o percurso necessário. O método CD é classificado como adaptativo, aproximado e exato [49].

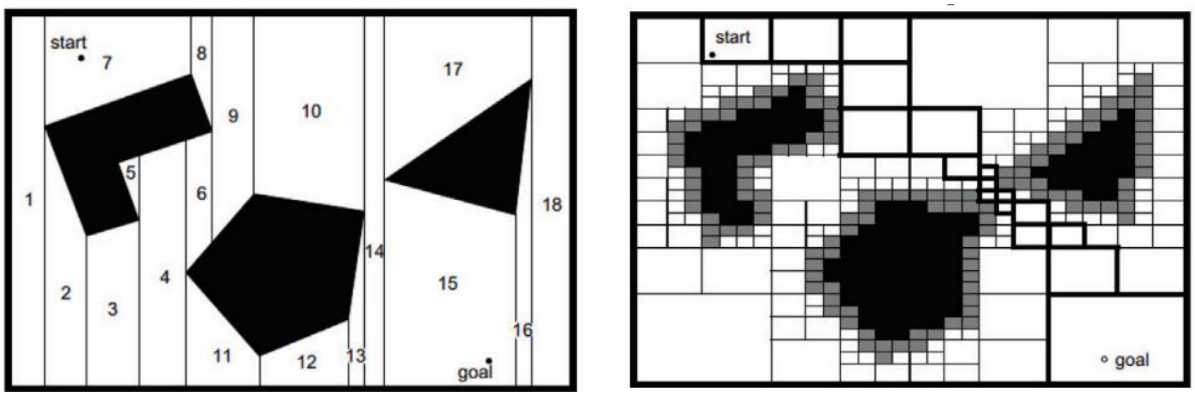


Figura 20 – Exemplo - *Cell decomposition*.

Na Figura 20, podemos observar um exemplo do trabalho [50], onde o passo inicial deste método envolve a divisão do espaço livre, que é delimitado por obstáculos poligonais tanto externa como internamente, em células trapezoidais e triangulares através da criação de segmentos de linha paralelos a partir de cada vértice de cada polígono interior no espaço de configuração, estendendo-se até ao limite exterior. Depois, cada célula, representada como um nó, é numerada no grafo de conectividade. Nós que são adjacentes no espaço de configuração são interligados no grafo. Ao seguir simplesmente as células livres adjacentes desde a localização inicial até ao ponto de objetivo, é possível encontrar um percurso contínuo a partir deste grafo de interligação [50].

2.3.1.3 A*

O A* é um algoritmo de pesquisa que também pode ser usado para encontrar percursos. Este algoritmo procura de forma contínua por locais não explorados num grafo. Todos os locais são percorridos no grafo e, quando o local alvo é alcançado, o algoritmo para. E caso o alvo não seja atingido, então todos os vizinhos são selecionados para exploração, de forma a procurar o percurso mais curto [46]. Basicamente consiste na junção do algoritmo Dijkstra com uma função heurística [51], a Figura 21 é uma representação deste algoritmo.

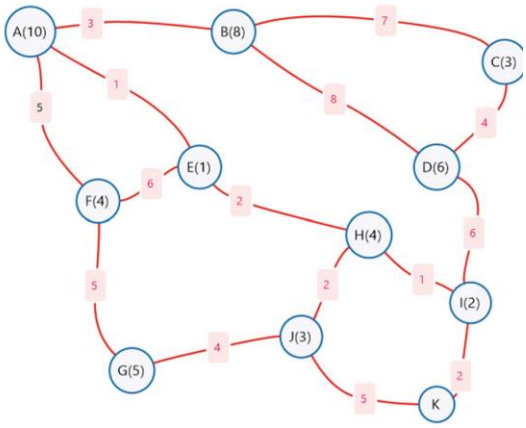


Figura 21 - Representação do Algoritmo A* [23].

2.3.1.4 Roadmap Aproach (RA)

O RA é um método usado para encontrar percursos em espaços complexos, cria uma rede de nós dentro desse espaço e, caso não haja obstáculos entre eles, estes são ligados por linhas. Existem três técnicas principais: o grafo de visibilidade (VG), o diagrama de Voronoi (VD) e o roadmap probabilístico (PRM) [50]. Nos métodos VG e VD, os obstáculos são tratados como polígonos, e os seus vértices são considerados nós. O VG, por exemplo, liga esses nós através de arestas que representam trajetos possíveis [52]. No entanto, à medida que o número de obstáculos aumenta, esta abordagem pode tornar-se mais lenta. Para acelerar o processo, é sugerido construir o VG e otimizar o percurso de forma simultânea. Isto permite que o grafo seja construído enquanto se procura um percurso ideal, reduzindo o tempo necessário [53].

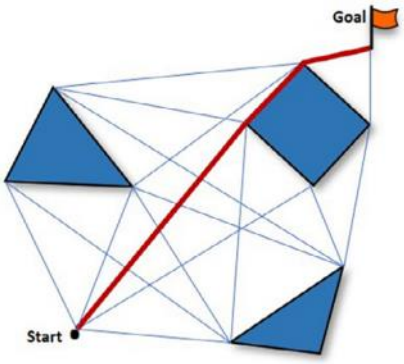


Figura 22 - Grafo de Visibilidade.

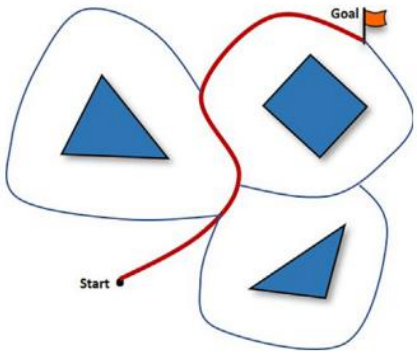


Figura 23 - Diagrama de Voronoi.

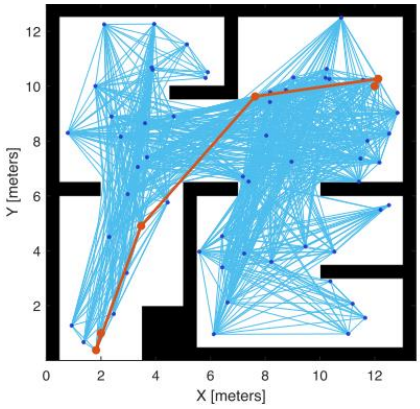


Figura 24 - Roadmap Probabilistico.

As Figuras 22 à 24 foram retiradas do artigo [50] como ilustrações que podem exemplificar o que foi referido.

2.3.2 Navegação Local

Além de navegação a nível global, também existem métodos de navegação a nível local, onde ao contrário dos planeadores de trajetórias globais, não necessitam de um mapa conhecido do ambiente para ser fornecido ao robô. Em vez disso, esses métodos baseiam-se em informações atuais e locais dos sensores para conferir ao robô móvel a capacidade de navegação em tempo real [54]. Como por exemplo:

2.3.2.1 Artificial Potencial Field (APF)

As abordagens do Campo Potencial Artificial (APF) são inspiradas na natureza. O objetivo principal do APF envolve simular o ambiente em redor do robô, onde os obstáculos são afastados por uma força repulsiva e o robô é atraído em direção ao alvo por uma força atrativa. Este campo de potencial é influenciado por duas forças: a força de atração e a força de repulsão. O alvo exerce uma força atrativa em direção a ele próprio, enquanto os obstáculos geram uma força repulsiva que varia inversamente com a distância entre o alvo e os obstáculos, afastando o alvo deles.

No contexto do APF, o robô desloca-se de zonas de maior potencial para zonas de menor potencial [46].

2.3.2.2 Vector Field Histogram (VFH)

O método do histograma de campo vetorial utiliza uma representação do ambiente por meio de uma grade dividida em células. Cada célula da grade contém um valor numérico entre 0 e 15, que indica se a área que ela representa está ocupada ou livre. O valor 0 representa certeza absoluta de ausência de obstáculos, enquanto o valor 15 representa certeza absoluta de presença de obstáculos. Esse método passa por um processo de redução de dados em duas etapas, de forma recursiva, para calcular o movimento desejado do robô em cada instante [54].

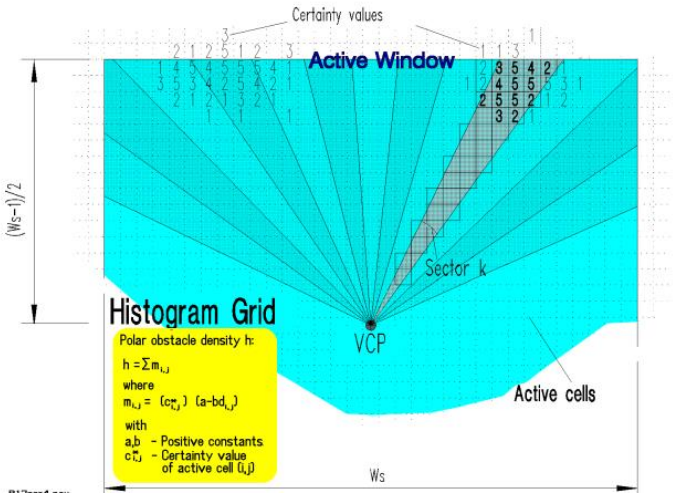


Figura 25 - Histogram Grid [55].

O método VFH utiliza uma técnica de redução de dados com três níveis de representação. O nível mais alto guarda a descrição detalhada do ambiente do robô. Nesse nível, a grelha de histograma bidimensional cartesiano é atualizada em tempo real com dados de alcance obtidos pelos sensores de alcance incorporados, como mostra a Figura 25. No nível intermédio, é construído um histograma polar unidimensional em torno da localização momentânea do robô, como podemos observar na Figura 26. Uma transformação mapeia a região ativa, resultando em cada setor k a conter um valor que representa a densidade polar de obstáculos na direção k. O nível mais baixo de representação de dados é a saída do algoritmo VFH, que consiste nos valores de referência para os controladores de direção e orientação do veículo [55].

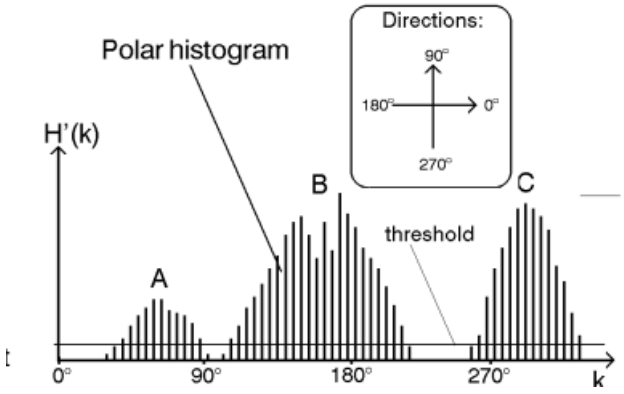


Figura 26 - Polar Histogram [55].

2.3.2.3 Simultaneous Localization and Mapping (SLAM)

O SLAM é um desafio na robótica móvel que envolve permitir que um robô navegue autonomamente por um ambiente enquanto cria um mapa desse ambiente e acompanha a sua própria localização dentro dele. Isso é crucial para a autonomia e exploração de robôs. A localização é essencial para que o robô entenda qual é a posição a que está no mapa, que é composto por elementos como paredes e obstáculos [56]. SLAM é um processo pelo qual um robô móvel pode construir um mapa de um ambiente e, ao mesmo tempo, usar esse mapa para deduzir a sua localização. No SLAM, tanto a trajetória da plataforma quanto a localização de todos os pontos de referência são estimadas em tempo real, sem a necessidade de qualquer conhecimento prévio de localização [57].

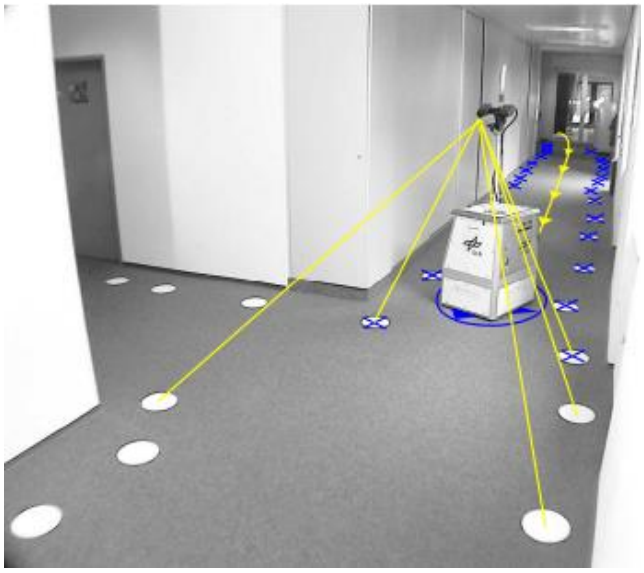


Figura 27 - Robô com SLAM [58].

Na Figura 27 o robô observa o ambiente em relação à sua própria posição desconhecida. Além disso, o movimento relativo do robô é medido onde a partir destas informações, um algoritmo SLAM calcula estimativas da pose do robô e da geometria do ambiente. No exemplo ilustrado aqui, uma câmara num robô mede a posição

relativa de características artificiais no chão (círculos brancos), enquanto o movimento do sensor é fornecido pela odometria do robô (setas amarelas). O resultado é a pose do robô (seta azul circular debaixo do robô) e a posição global de cada característica (cruzes azuis) [58].

2.4 Robot Operating System (ROS)

Para controlar eficientemente diversas configurações robóticas, são necessárias ferramentas de programação apropriadas. Embora os robôs possam ser programados utilizando linguagens como C/C++ ou Python, não é prático aprenderem várias linguagens para diferentes partes de um projeto. O ROS oferece uma solução ao fornecer um ambiente unificado que combina componentes de software e hardware. Isto é vantajoso para os alunos, que só precisam aprender a linguagem de programação e a infraestrutura do ROS para gerirem várias tarefas. O ROS, um projeto global de código aberto, oferece uma extensa base de dados de modelos de robôs, sensores, linguagens suportadas e ambientes de simulação [59].

2.4.1 Arquitetura ROS

No ROS, são usados termos específicos para descrever os elementos centrais. O conceito de "nó" é fundamental, representando uma entidade que executa operações de processamento. A comunicação entre os nós ocorre através de "tópicos" e "serviços". Os "tópicos" seguem um modelo de publicação e subscrição, permitindo a troca de dados entre vários nós num espaço partilhado. Por sua vez, os "serviços" disponibilizam recursos que podem ser solicitados por outros nós. A forma de comunicação adotada baseia-se no modelo "*peer-to-peer*" (P2P), descentralizado, à exceção do serviço de resolução de nomes que é gerido por um "nó master". Esse "nó master" mantém registos dos serviços e tópicos disponíveis, possibilitando que os nós interajam através de tópicos e serviços [60] conforme a Figura 28.

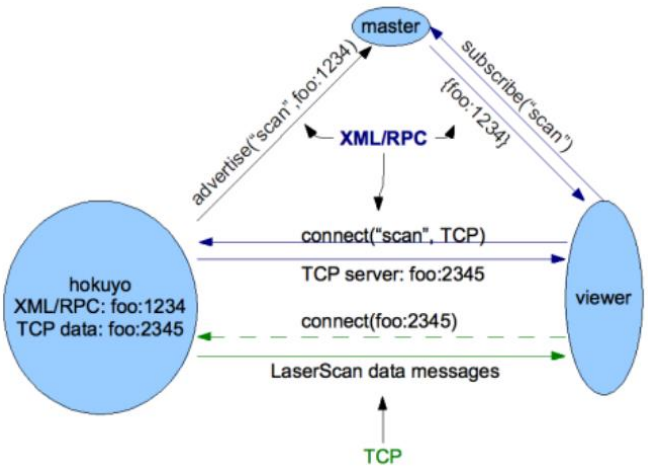


Figura 28 - Arquitetura Básica de Comunicação do ROS [60].

2.4.2 Gráfo

Uma das "questões desafiadoras" que impulsionou o desenvolvimento do ROS era conhecida como o problema de "buscar um objeto". Nesse cenário, um robô com recursos a câmaras, scanners, um braço manipulador e uma base móvel deve navegar num ambiente, encontrar um objeto específico e entregá-lo ao local desejado. Esta tarefa levou a importantes conclusões que moldaram o design do ROS. A tarefa pode ser dividida em subsistemas independentes, como navegação e visão computacional. Com abstração adequada de hardware e geometria, o software pode ser reutilizado em diferentes robôs. Esses objetivos são refletidos na arquitetura do ROS, onde programas distintos se comunicam por mensagens. Isso é representado visualmente por um grafo, Figura 29, em que os programas são nós interligados por arestas.

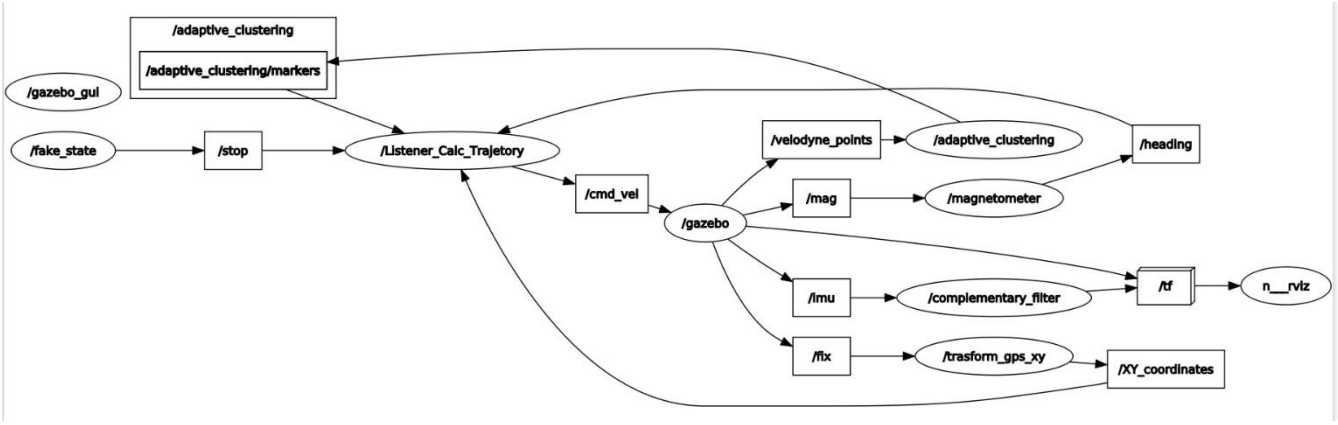


Figura 29 - Exemplo de um grafo retirado de uma simulação do ROS.

2.4.3 Simulação

O Gazebo é um reconhecido simulador 3D multi-robô que possibilita a criação de ambientes tanto internos como externos, com a capacidade de modelar diversos tipos de robôs, incluindo humanoides, móveis, submarinos, entre outros [61]. No contexto do Gazebo, a cena 3D é denominada de "mundo" e é composta por objetos estáticos e dinâmicos chamados de "modelos". O utilizador define essa paisagem através de um formato de Ficheiro de Descrição de Simulação (SDF). Além disso, é permitida uma personalização minuciosa dos detalhes de cada modelo, incluindo propriedades de inércia, aspeto visual e características de colisão. A capacidade de criar simulações realistas no Gazebo é devida às suas três bibliotecas principais: física, renderização e comunicação. O Gazebo é amplamente adotado na comunidade robótica, em grande parte devido à sua perfeita integração com o ROS [62]. Essa integração possibilita aos utilizadores desfrutar das visualizações 3D do Gazebo, juntamente com o suporte providenciado pelo ROS. Na Figura 30, podemos observar um exemplo de uma simulação 3D no Gazebo, onde o robô em questão está a utilizar um LiDAR (feixo a azul), esta imagem foi retirada da fonte [63].

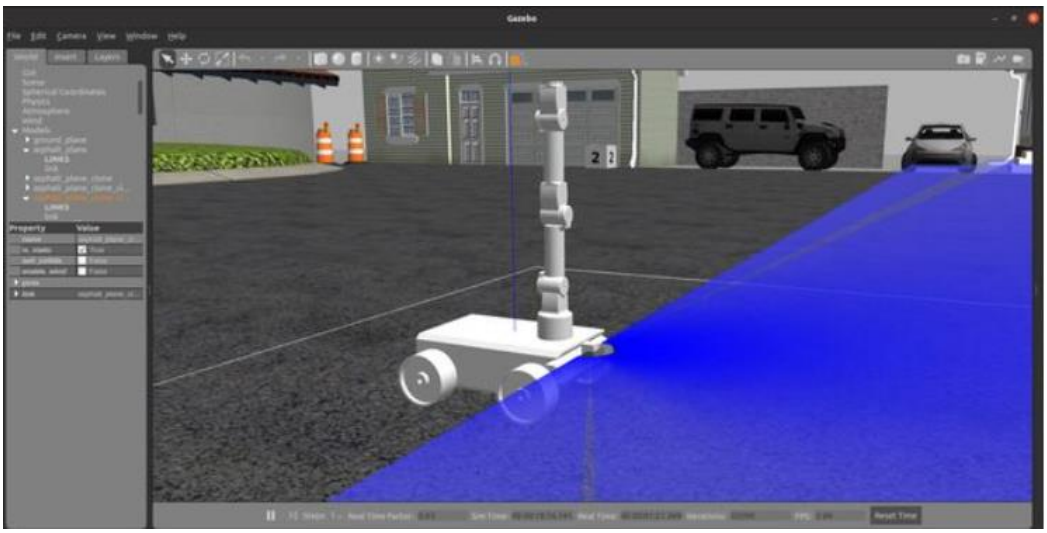


Figura 30 - Exemplo Simlação no Gazebo.

CAPÍTULO 3: METODOLOGIA

3.1 Arquitetura do Sistema Robótico

A arquitetura do sistema robótico bem como o funcionamento de todo o módulo sensorial pode ser observada na Figura 31 onde se apresenta de forma sucinta todo o processo e demonstra o gráfico obtido através do ROS, para ilustrar o conceito geral deste tipo de sistema. Esta representação pode ser aplicada a qualquer sistema independentemente do tamanho. De facto, essa forma de representação é tão útil para o desenvolvimento de software que os especialistas se referem aos programas ROS como nós, o que ajuda a lembrar que cada programa é apenas uma parte do sistema [64].

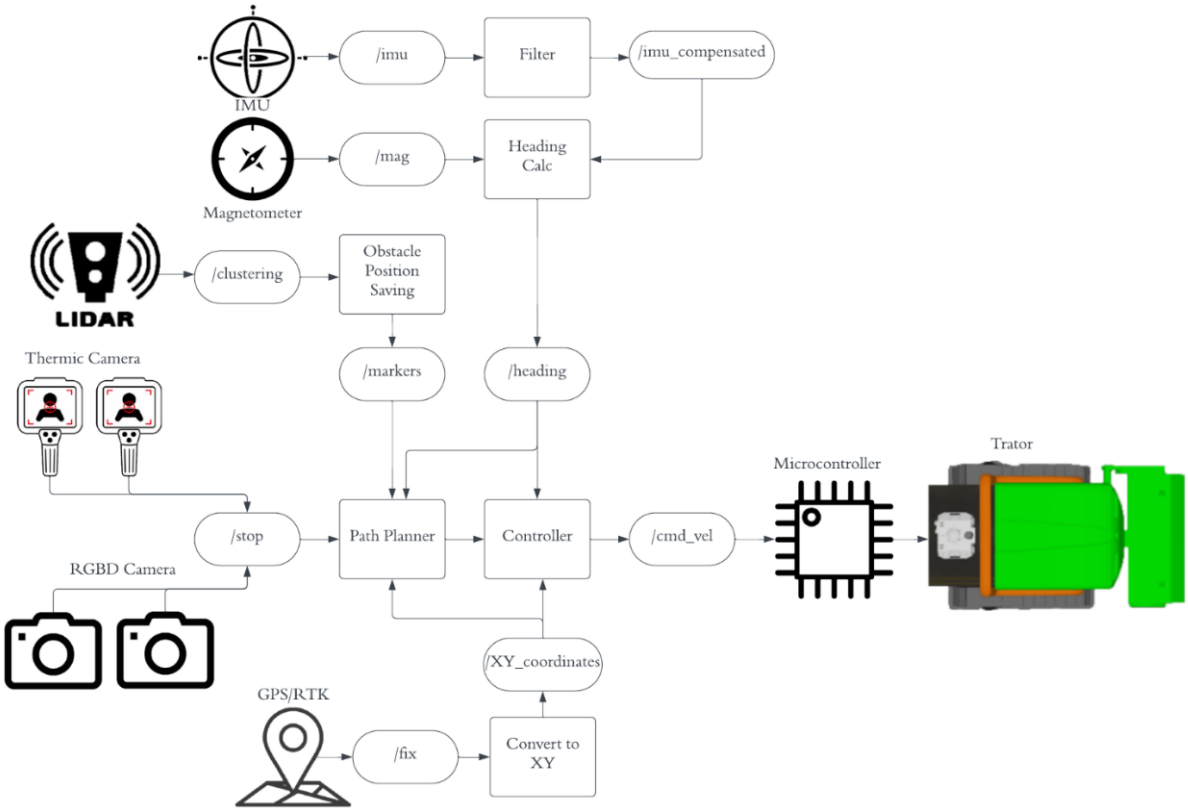


Figura 31 - Arquitetura do Sistema Robótico.

Os blocos presentes na Figura 31 são representados como “nós” e as elipses são designados “tópicos” provenientes dos sensores ou até dos “nós”. Na Figura 31 podemos observar o sensor IMU (*Inertial Measurement Unit*) que possui 6DOF (*Degrees Of Freedom*), com um acelerómetro e um giroscópio utilizados para obter informações do comportamento do trator. Este tipo de sensor fornece informações que foram adquiridas diretamente, ou seja, estes dados não foram modificados, formatados ou processados [65]. Antes de ser utilizados para análises ou outros propósitos, os dados geralmente precisam de ser organizados, limpos e processados.

Este tipo de processamento pode incluir eliminação de ruídos, correções de erros, conversões de formatos de dados e transformações de dados em estruturas que sejam mais adequadas para análises futuras [66], ou seja, na aquisição de dados neste tipo de sistemas existe sempre ruído nos valores adquiridos e por esta razão, todos os dados adquiridos são subscritos por um "nó" que têm como objetivo efetuar uma filtragem da informação, de seguida efetua uma publicação num outro "tópico" que se designa por *"/imu_compensated"* onde se observam os dados adquiridos através do filtro que elimina o ruído. O magnetómetro, pública os dados do valor do campo magnético que está a receber, estes dados adquiridos possuem ruído e também necessitam de ser tratados. O "nó" *"Heading_Calc"* converte o campo magnético da Terra em orientação, e caso exista uma componente angular, esta será corrigida de acordo com os dados adquiridos pelo IMU, de modo que o *Tilt* seja compensado. A orientação do trator é o *"/heading"* e vai ser subscrito pelo algoritmo que efetua o planeamento da trajetória a executar, designada *"Path_Planner"* e de seguida controlada pelo sistema de controlo designado "Controller" atuando sempre que exista um erro entre a trajetória real com a planeada.

O LiDAR *(Light Detection and Ranging)* 3D fornece informação tridimensional através de uma *PointCloud*, onde a partir desta informação é realizado um *clustering*, ou seja, sempre que se encontra um aglomerado de pontos, é considerado um obstáculo onde será definida uma caixa cujas dimensões envolvem esse aglomerado de pontos, conforme se observa na Figura 32. Os dados obtidos desse *"/clustering"* são subscritos pelo "nó" designado de *"Obstacle_Position_Saving"* que guarda a informação das caixas num "tópico" designado por *"/markers".* Estas localizações dos objetos são obtidas relativamente à posição do LiDAR do robô no mapa, informação esta que é subscrita pelo sistema de planeamento da trajetória designado por *"Path_Planner".*

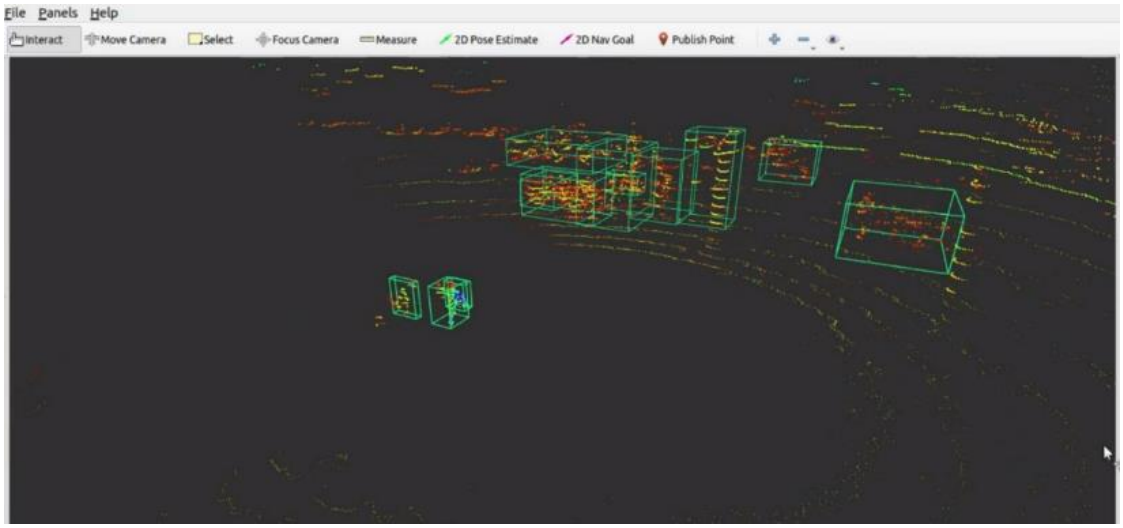


Figura 32 - PointCloud e Clustering.

As câmaras utilizadas, tanto as RGBD como as térmicas correm no mesmo algoritmo, no nosso caso, utilizamos o algoritmo YoloV5, e desta forma, além de conseguirmos obter os obstáculos é também possível detetar a presença de pessoas. No entanto, o

cálculo da distância só foi implementado nas câmaras RGBD e no LiDAR. Desta forma, sempre que se deteta uma pessoa a menos de 7 metros, efetua-se uma publicação num "tópico" designado de *"/stop",* este tópico é subscrito pelo sistema de planeamento de trajetórias como entrada de emergência para parar o sistema robótico.

O GPS, fornece dados relativos à posição da máquina, este é o tópico *"/fix"* onde se apresenta os dados da latitude e da longitude. Este tópico é subscrito por *"Convert_to_XY"* que pretende guardar a posição inicial da máquina, assumindo o ponto de origem (0, 0), a partir do ponto inicial converte a posição (latitude, longitude) em coordenadas (x, y) referentes à posição do robô, que são publicadas como *"/XY_coordinates"* e também subscritas pelo *"Path_Planner"*. Posteriormente, o "*Controller",* no decorrer da navegação o tópico, *"cmd_vel"* referente às velocidades angular e linear do sistema robótico vai converter estes dados em valores de tensão (V) para cada uma das rodas do trator. Na Figura 33 podemos observar o posicionamento dos sensores referidos no módulo sensorial designado de "Sentry".

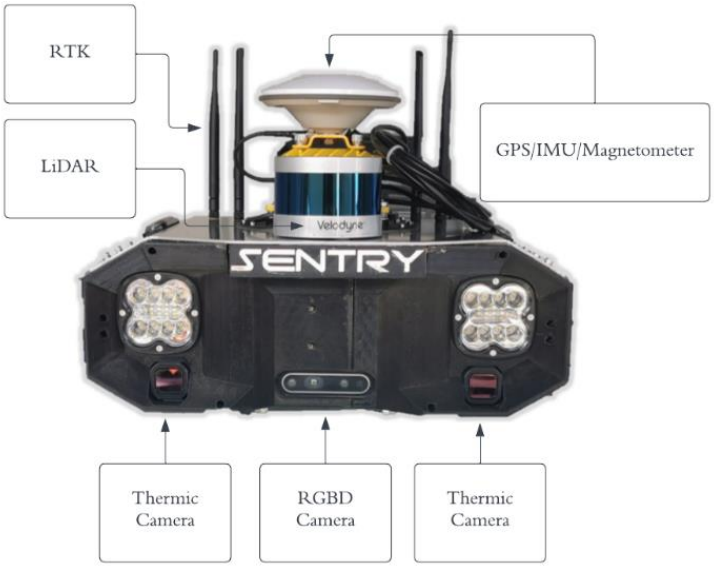


Figura 33 - Sistema sensorial Sentry.

3.2 Deteção por Câmara

Para a deteção e classificação visual procedeu-se à utilização de duas câmaras RGBD da Realsense e duas térmicas da D435i da FLIR ADK, como se apresenta na Figura 34, onde se integrou o algoritmo YOLO (*You Only Look Once*) para a deteção e identificação dos obstáculos.

Figura 34 - Realsense e FLIR.

O algoritmo YOLO pode identificar objetos em imagens, este algoritmo funciona de forma rápida e apresenta uma forma expedita e pode ser integrado sem a necessidade de várias etapas complexas. Este algoritmo utiliza uma imagem completa como entrada para a rede neural em vez de dividir a tarefa de deteção em partes. Em seguida, o algoritmo determina diretamente onde as caixas delimitadoras dos objetos estão na imagem e classifica cada objeto detetado e retorna a posição e a classe exata de cada objeto que encontrou [67] apresentado na Figura 35. As informações da imagem são usadas para projetar cada caixa delimitadora, cada caixa contém cinco valores: as coordenadas, a largura, a altura da caixa e uma confiança associada a cada deteção [68] conforme se mostra na Figura 35. Esses valores estão associados a uma grelha de células que cobrem a imagem. Além disso, cada célula localizada no centro de uma caixa delimitadora contém informações sobre como detetar um determinado objeto[68].

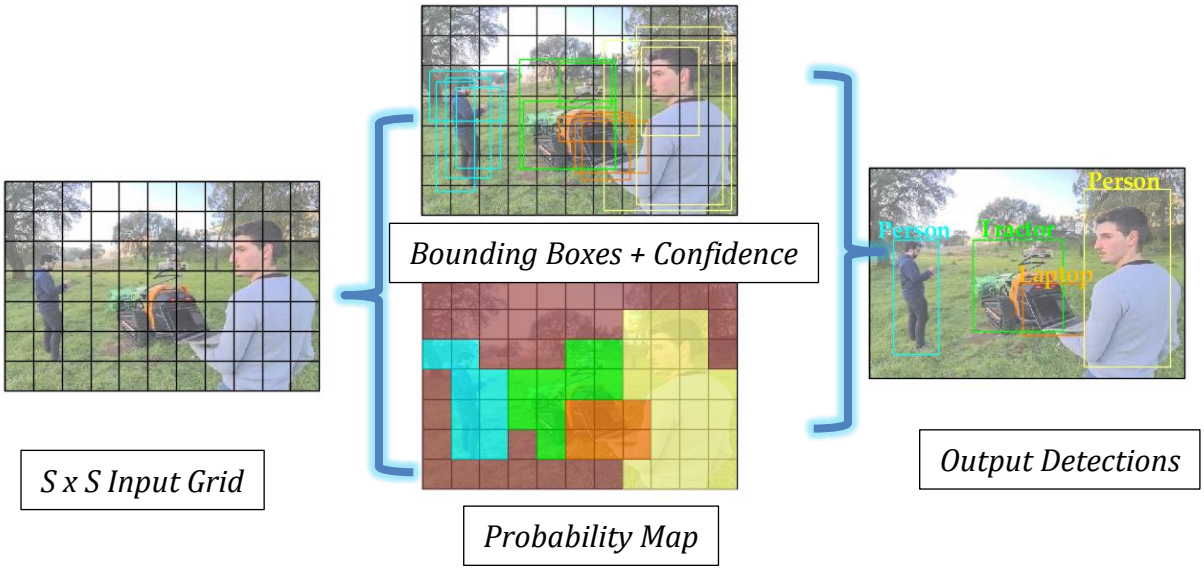


Figura 35 - Algoritmo do YOLO.

O YOLO usa informações de toda a imagem para encontrar objetos nas imagens de uma só vez. Em comparação com outras abordagens que usam métodos mais complexos, isso acelera o YOLO[67].

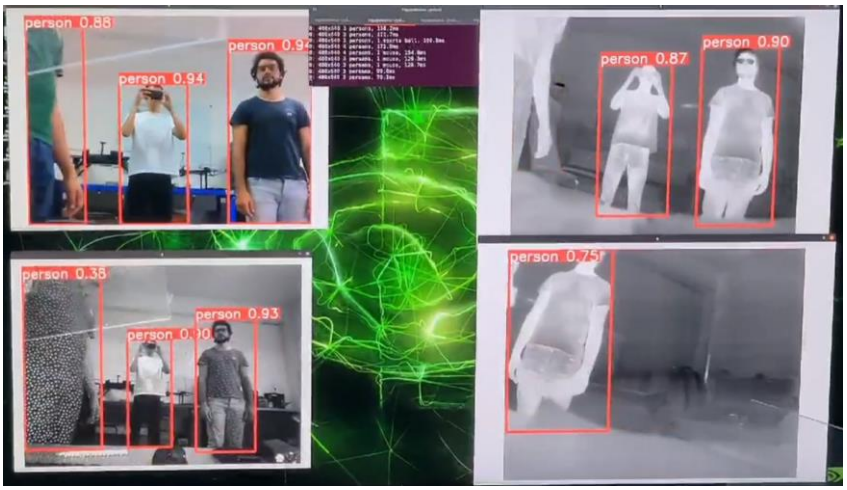


Figura 36 - Realsense e FLIR com algoritmo do YOLO.

É importante realçar que, como em qualquer modelo de aprendizagem de máquina, o desempenho do YOLO depende do conjunto de dados efetuados no treino bem como a qualidade dos dados disponíveis. Para a existência de mais redundância de dados utilizou-se as duas tecnologias de câmaras. Os sensores térmicos formam imagens do ambiente ou de objetos com base na quantidade de energia térmica emitida pelo objeto. Os sensores térmicos são imunes às condições de iluminação, sendo robustos a diversas variações de luz e condições climáticas [69]. As câmaras térmicas são muito úteis em aplicações de segurança em condições climáticas adversas, tais como a chuva, a neblina, nas quais as RGB comuns apresentam alguma dificuldade em produzir resultados precisos. Além disso, as câmaras térmicas também são mais valiosas e em situações de completa escuridão, quando as convencionais não são eficazes[70]. Em contrapartida, as câmaras infravermelhas são altamente sensíveis às variações de temperatura do ambiente ao redor, porém oferecem menos detalhes em comparação com as câmaras de luz visível. Isso acontece porque as cores capturadas no espectro visível fornecem muito mais informações e são mais fáceis de interpretar do que as imagens térmicas capturadas pelas câmaras infravermelhas[70].

Devido a uns sensores serem mais propícios a uns ambientes que outros, optou-se por utilizar ambas as tecnologias. Na Figura 37 podemos observar uma experiência entre a câmara RGB-D e a térmica.

Figura 37 - Testes de comparação entre realsense e FLIR.

Na Figura 38 podemos observar um caso na floresta onde a câmara térmica e a RGBD deteta uma pessoa no meio da florestação. Ter este tipo de informação é muito útil para a deteção do que desejamos.

Figura 38 - Teste de Comparação entre realsense e FLIR.

Foi realizado um processo de treino neural de forma ao algoritmo detetar também árvores, uma vez que estas representam obstáculos que o robô não deve cortar. Para isso, foram utilizados conjuntos de dados específicos para essa identificação, contendo milhares de imagens com anotações dos troncos [71]. Após a conclusão do treino neural, o algoritmo foi submetido a testes novamente, e o resultado obtido está representado na Figura 39.

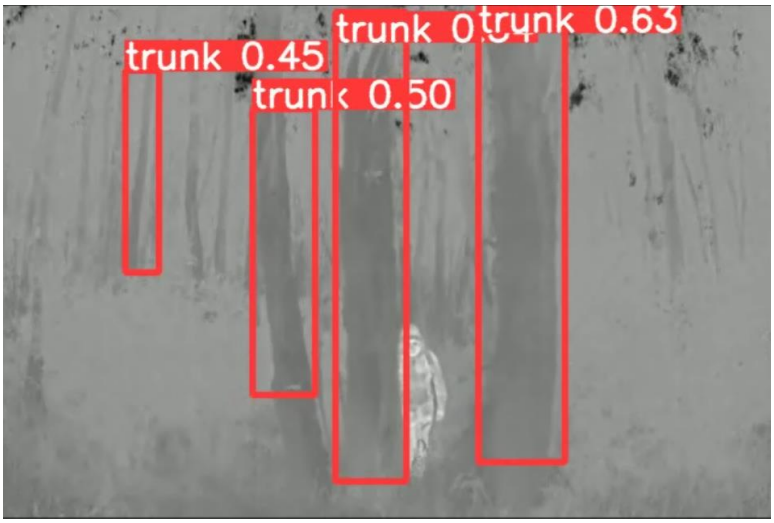


Figura 39 - Deteção de Árvores.

3.3 Funcionamento

O robô comunica com uma plataforma online, controlada por um utilizador, normalmente o dono do trator. Esta plataforma online está em fase de desenvolvimento, no entanto já é possível entender como se realiza o processo de utilização. A partir da *web*, o utilizador tem acesso a toda a informação do trator, desde a posição até o que é visível pelas câmaras. Ou seja, todas as informações que o robô está a receber e a visualizar é possível ser publicado nesta plataforma. O utilizador

pode delinear uma área no mapa, no *website* e a partir desta um algoritmo converte em percursos navegáveis, onde cada vértice é convertido em coordenada que é enviada para o robô, criando assim um caminho para a zona de limpeza. Também é possível que o utilizador mencione no mapa a presença de obstáculos, esses serão convertidos também em coordenadas das quais o robô sabe que não pode transgredir.

Devido às limitações de manobrabilidade dos veículos florestais, é necessário reservar uma área do terreno conhecida como "áreas de manobra" para permitir as viragens do veículo [72]. A abordagem mais simples é atribuir uma zona de largura constante ao redor do terreno. No entanto, essa estratégia alocaria uma grande quantidade de espaço a uma área de utilidade reduzida. Dependendo da disposição das faixas, algumas áreas de manobra podem ser paralelas às faixas, tornando-as desnecessárias para as viragens. Para otimizar esta situação, é possível construir áreas de manobra apenas ao longo das margens do terreno, onde as manobras efetivamente ocorrem, reduzindo assim a área necessária para este propósito [73]. As faixas de trabalho são posteriormente criadas na parte interna do terreno, que é a região que sobra após subtrair as áreas de manobra. Em terrenos bidimensionais, uma linha de referência pode ser usada como guia para a criação das faixas, onde cada linha paralela define uma faixa [74].

Figura 40 - Escolha da área.

Figura 41 - Criação do caminho.

Figura 42 - Áreas proibidas.

Figura 43 - Novo caminho.

As Figuras 40 até à 43, são representações, ou seja, apenas mostram a ideia do website e a criação das trajetórias planeadas. A Figura 40 representa a área que foi delineada pelo utilizador, onde, caso este não mencione áreas restritas, o algoritmo irá criar automaticamente uma trajetória para o robô navegar, conforme a Figura 41. Caso sejam adicionadas áreas proibidas, como mostra a Figura 42, o algoritmo irá realizar um novo caminho onde não transgrida essas áreas proibidas, conforme se mostra na Figura 43.

No decorrer deste projeto, ao observar o trabalho de um profissional de desbastamento florestal que utiliza este trator diariamente, o processo de desbastamento apresenta algumas regras para que a vegetação seja desbastada de forma correta. A Figura 44 mostra a máquina de estados do funcionamento da máquina quando está no processo de desbaste.

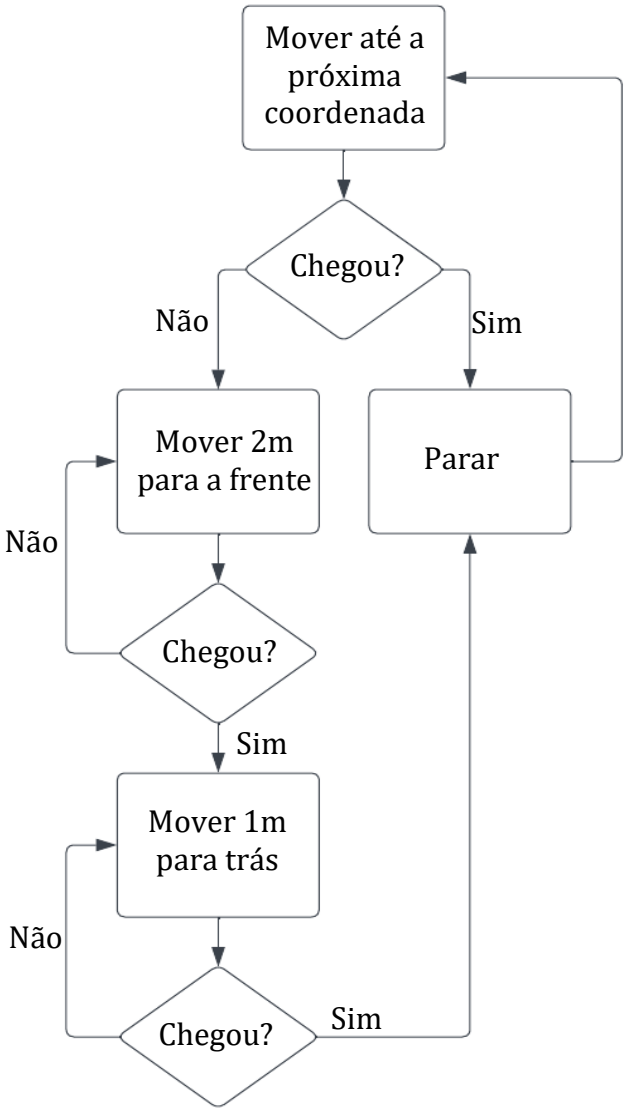


Figura 44 – Máquina de estados para desbastamento da vegetação.

3.4 Cinemática

Este robô possui duas rodas fixas num eixo comum onde cada roda pode de forma independente rodar para a frente ou para trás [75] conforme se observa na Figura 45. No entanto estas rodas são lagartas e não são circulares, o que significa que a cinemática é diferente do robô comum diferencial, como demonstra a Figura 46.

Figura 45 - Trator com lagartas.

Figura 46 - Trator com rodas.

Este robô tem duas variáveis de controlo de entrada, que são as velocidades das suas lagartas (as lagartas da esquerda e da direita) designadas por (V_l, V_r) respetivamente. Desta forma a cinemática direta no plano pode ser representada da seguinte forma [76].

$$(v_x, v_y, \omega_z) = f_d(V_l, V_r) \tag{5}$$

A velocidade de translação é designada por $\mathbf{v = (v_x, v_y)}$ e a velocidade angular por (ω_z) referentes à referência do robô. Para encontrarmos as ações de controlo necessárias para a locomoção desejada, podemos expressar a cinemática inversa da seguinte forma:

$$(V_l, V_r) = f_i(v_x, v_y, \omega_z) \tag{6}$$

O modelo de simulação é construído com base no modelo cinemático ICR (*Instantaneous Center of Rotation*), o qual tem a capacidade de simular o movimento do veículo em relação aos seus centros de rotação. Em idealidade, esses centros deveriam situar-se no centro de cada via de rodagem (lagarta), mas podem mover-se ao atravessar terrenos diversos [77]. A Figura 47 representa um esquema relacionado com a cinemática do robô em questão:

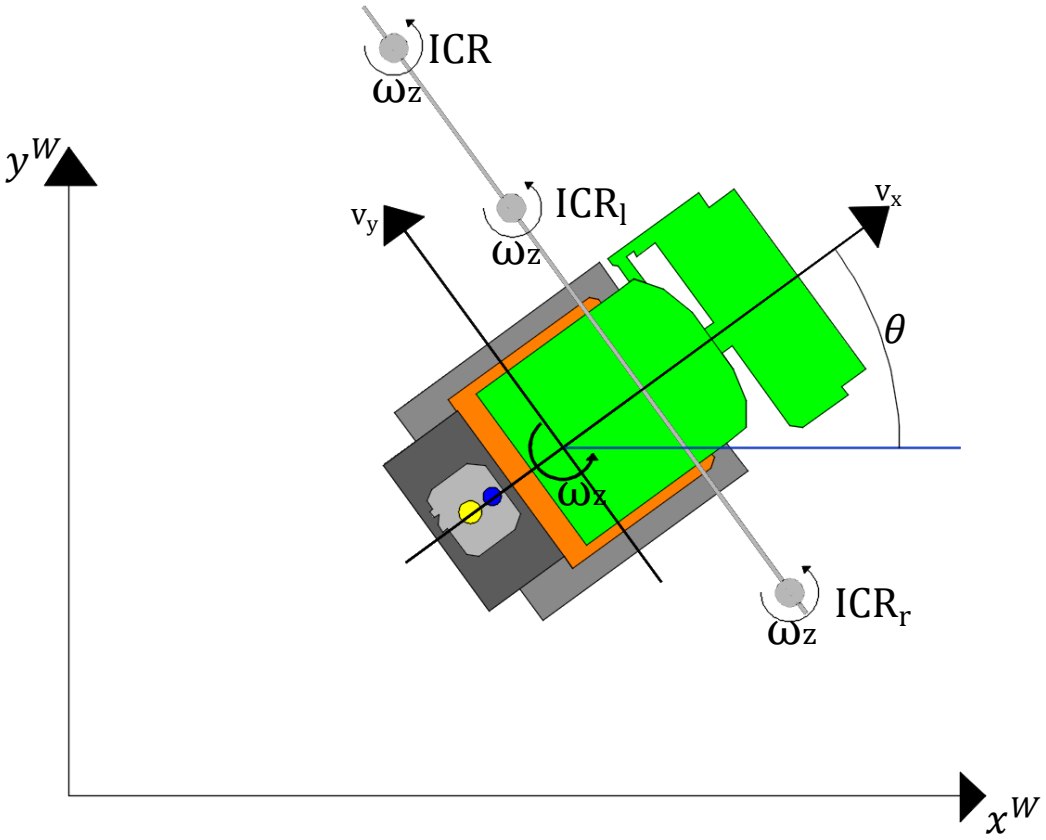


Figura 47 - Cinemática do trator.

No movimento de um veículo com lagartas, existe um ponto especial no plano horizontal chamado Centro Instantâneo de Rotação (CIR). Nesse ponto, o movimento do veículo parece ser apenas uma rotação, sem qualquer movimento lateral [76]. Na movimentação de um veículo de lagartas, é importante ter em conta o movimento individual das lagartas em contato com o solo e não o comportamento do veículo como um todo. Neste tipo de sistemas cada lagarta é um corpo rígido adicional, possuindo uma velocidade distinta, significa que o movimento de um ponto específico na superfície da lagarta é resultado da combinação do movimento do veículo e o movimento de cada lagarta. Devido a essa combinação, o CIR para cada lagarta (esquerda e direita) no plano do solo é diferente do CIR do veículo inteiro [78] conforme se observa na Figura 47. Assim, podemos definir os ICRs para as lagartas esquerda e direita como:

$$ICR_l = (x_{ICRl}, y_{ICRl}) \; e \; ICR_r = (x_{ICRr}, y_{ICRr}) \tag{7}$$

É importante destacar que essa definição está relacionada ao ponto no solo onde as lagartas estão em contato com a superfície e não ao eixo de rolagem das próprias lagartas. As coordenadas de referência local para o CIR do veículo e para os CIRs de lagartas podem ser encontradas geometricamente através das seguintes funções[76]:

$$x_{ICR} = \frac{-v_y}{\omega_z} \qquad x_{ICRl} = \frac{V_l - v_y}{\omega_z} \qquad x_{ICRr} = \frac{V_r - v_y}{\omega_z} \tag{8}$$

$$y_{ICR} = y_{ICRl} = y_{ICRr} = \frac{v_x}{\omega_z} \tag{9}$$

No cálculo das funções inversas, é possível obter as velocidades instantâneas de translação e rotação com relação ao sistema de referência, estas representam a cinemática direta do robô caso os CIRS das lagartas serem obtidas por:

$$v_x = \frac{V_r - V_l}{x_{ICRr} - x_{ICRl}} * y_{ICR} \tag{10}$$

$$v_y = \frac{V_r + V_l}{2} - \frac{V_r - V_l}{x_{ICRr} - x_{ICRl}} * \frac{x_{ICRr} + x_{ICRl}}{2} \tag{11}$$

$$\omega_z = \frac{V_r - V_l}{x_{ICRr} - x_{ICRl}} \tag{12}$$

Desta forma gera o seu movimento globalmente da seguinte forma:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{13}$$

Por outro lado, de forma a representar a cinemática inversa do robô podemos expressá-las da seguinte forma:

$$V_L = v_y + x_{ICRl} * \omega_z \qquad V_R = v_y + x_{ICRr} * \omega_z \tag{14}$$

No nosso caso, nem as lagartas nem os motores de combustão apresentam qualquer sensor e apesar do conhecimento da cinemática do robô, torna-se mais preciso para a navegação usar a posição do robô invés da velocidade de cada lagarta então, para efetuar o controlo da velocidade de cada lagarta, é aplicada nas entradas diferentes níveis de tensão consoante a posição do robô. A Figura 48 demonstra o funcionamento do robô para diferentes valores de tensão V_L e V_R no robô em questão.

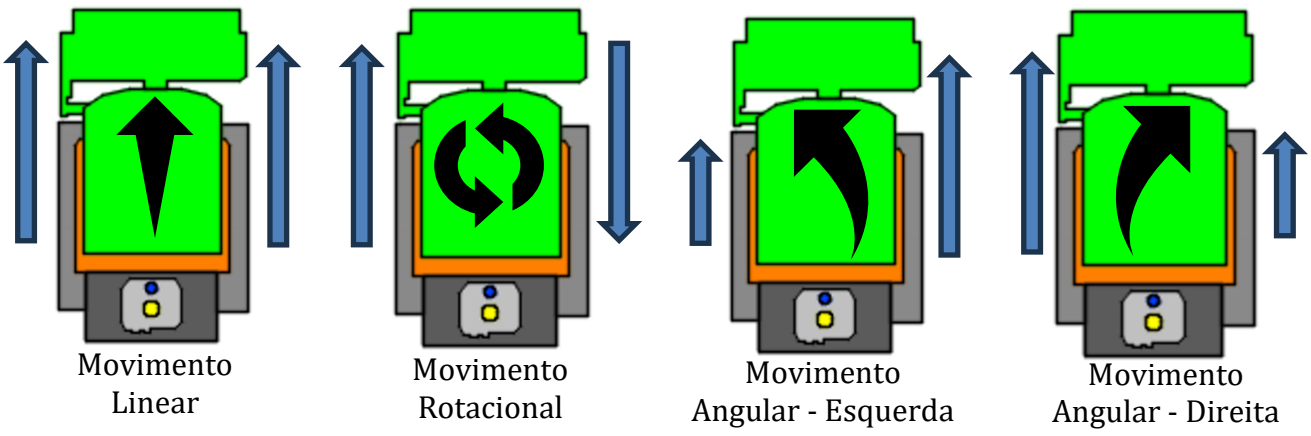


Figura 48 - Movimentos das lagartas.

3.5 Módulo Simulação

Antes de realizar qualquer teste no terreno foi desenvolvido um robô similar num ambiente de simulação designado Gazebo, onde o ROS integra-se através da *package* gazebo_ros. Essa *package* fornece um módulo de *plugin* (é um componente de software que adiciona funcionalidades específicas a um programa ou sistema maior, sem a necessidade de modificar o código fonte principal desse programa ou sistema) para o Gazebo que permite uma comunicação bidirecional entre o Gazebo e o ROS. Dados simulados de sensores e informações físicas podem ser transmitidos do Gazebo para o ROS, e os comandos dos atuadores podem ser enviados do ROS de volta para o Gazebo. Ao utilizar nomes e tipos de dados coerentes para esses fluxos, o Gazebo pode coincidir precisamente com a API do ROS de um robô. Isso permite que o software do robô seja executado de forma semelhante tanto no ambiente real como no simulador, após o ajuste dos parâmetros [64].

O sistema desenvolvido tem as mesmas dimensões, foi construído em SolidWorks e exportado cada parte para um ficheiro STL. A partir da aquisição de todas as peças em separado para a simulação, passou-se ao desenvolvimento da junção de todas elas num ficheiro URDF (*Unified Robot Description Format*) onde a estrutura, a cinemática e outros detalhes podem ser representados em ambientes de robótica [79]. O URDF é baseado no XML (*eXtensible Markup Language*), e contém informações sobre vários componentes do robô, tais como os elos, as juntas, os sensores entre outros [64]. Desta forma, apresenta-se a geometria dos elos, incluindo a forma e tamanho, bem como as suas propriedades físicas, tais como a massa e a inércia. Além disso, este simulador permite incluir sensores associados ao sistema robótico e específica as relações hierárquicas entre eles bem como as suas posições e limitações [79].

É possível visualizar e simular qualquer sistema robótico usando a sua representação através de um ficheiro URDF e também é possível efetuar diferentes tarefas, tais como, os movimentos do robô, a possibilidade de detetar colisões e também fazer uma análise da cinemática e da dinâmica do sistema que se pretende simular. Como resultado, o URDF é uma ferramenta muito importante para o desenvolvimento deste tipo de aplicação.

Para tornar a simulação o mais real possível, começou-se por calcular a inercia para cada componente do respetivo robô, as fórmulas seguintes representam as fórmulas inerciais usadas na simulação. No caso do cálculo da inércia do sistema de locomoção do robô, de modo a simplificar o seu cálculo, adotamos para a inércia as equações 15 a 17 de acordo com a Figura 49 que representa um cilindro.

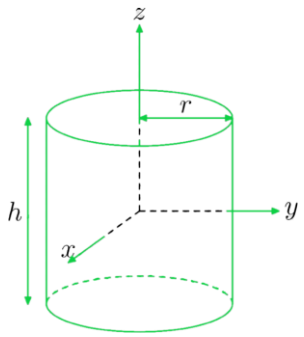


Figura 49 - Eixos e variáveis referentes ao cilindro.

$$I_x = \frac{1}{12} m(3r^2 + h^2) \tag{15}$$

$$I_y = \frac{1}{12} m(3r^2 + h^2) \tag{16}$$

$$I_z = \frac{1}{2} mr^2 \tag{17}$$

Do mesmo modo, para determinar a inércia do trator adotamos as equações 18 a 20 de acordo com a Figura 50 que representa o volume de um prisma retangular.

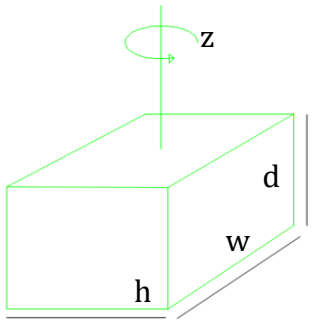


Figura 50 - Eixos e variáveis do prisma.

$$I_h = \frac{1}{12} m(w^2 + d^2) \tag{18}$$

$$I_w = \frac{1}{12} m(d^2 + h^2) \tag{19}$$

$$I_d = \frac{1}{12} m(w^2 + h^2) \tag{20}$$

O trator contém as seguintes características apresentadas na Tabela 5, e as mencionadas são as suficientes para a realização do URDF.

Tabela 7 - Parâmetros do trator.

Propriedades	Parâmetros
Largura	1360 mm (300 mm por lagarta)
Comprimento com ferramenta	2160 mm + 750 mm
Altura	1200 mm (400 mm por lagarta)
Peso com ferramenta	1292 Kg (200 Kg por lagarta) + 227Kg

A Tabela 8 demonstra os resultados inerciais do corpo e da ferramenta do robô, assim como a representação de como foi calculado no ficheiro URDF do robô.

Tabela 8 - Cálculo e demonstração no URDF da Inércia da ferramenta e do corpo do robô.

Corpo e Ferramenta do Robô	$I = \begin{bmatrix} \frac{1}{12}m(d^2+h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+h^2) \end{bmatrix}$
Corpo do Robô	$I = \begin{bmatrix} 394.36 & 0 & 0 \\ 0 & 389.73 & 0 \\ 0 & 0 & 90.50 \end{bmatrix}$
Ferramenta do Robô	$I = \begin{bmatrix} 1.09 & 0 & 0 \\ 0 & 39.33 & 0 \\ 0 & 0 & 28.93 \end{bmatrix}$
Cálculo da Inércia do corpo e da ferramenta no ficheiro URDF	<inertia ixx="${0.0833333 * mass * (y*y + z*z)}" ixy="0.0" ixz="0.0" ixy=”0.0” iyy="${0.0833333 * mass * (x*x + y*y)}" iyz="0.0" izx=”0.0” izy=”0.0” izz="${0.0833333 * mass * (x*x + z*z)}" /> </inertial>

A Tabela 9 demonstra os resultados inerciais das rodas do robô, assim como a representação de como foi calculado no ficheiro URDF do robô.

Tabela 9 - Cálculo e demonstração no URDF da Inércia das rodas do robô.

Roda Esquerda e Direita do Robô	$I = \begin{bmatrix} \frac{1}{12}m(3r^2 + h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2 + h^2) & 0 \\ 0 & 0 & \frac{1}{2}mr^2 \end{bmatrix}$
Rodas do Robô	$I = \begin{bmatrix} 9.5 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 16 \end{bmatrix}$
Cálculo da Inércia das rodas no ficheiro URDF	<inertia ixx="${0.0833333 * mass * (3 * r*r + h*h)}" ixy="0.0" ixz="0.0" iyx="0.0" iyy="${0.0833333 * mass * (3 * r*r + h*h)}" iyz="0.0" izx="0.0" izy="0.0" izz="${0.5 * mass * r*r}" /> </inertial>

Na simulação foram criadas formas geométricas que representassem as características do robô real. Ou seja, foram criados dois prismas, um a representar o corpo do robô e outro a representar a sua ferramenta, o mesmo se considerou para a construção das rodas/lagartas, estas foram representadas por cilindros como se podem observar na Figura 51.

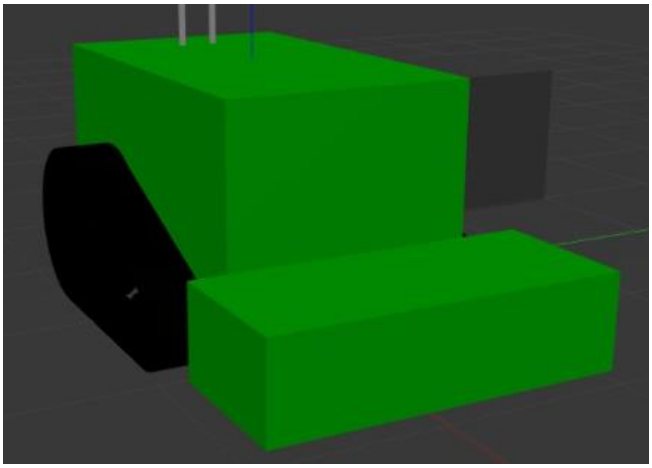


Figura 51 – Geometria do robô na simulação.

Após este procedimento procedeu-se a um detalhe de forma a tornar o trator o mais parecido com a realidade conforme se pode observar na Figura 52.

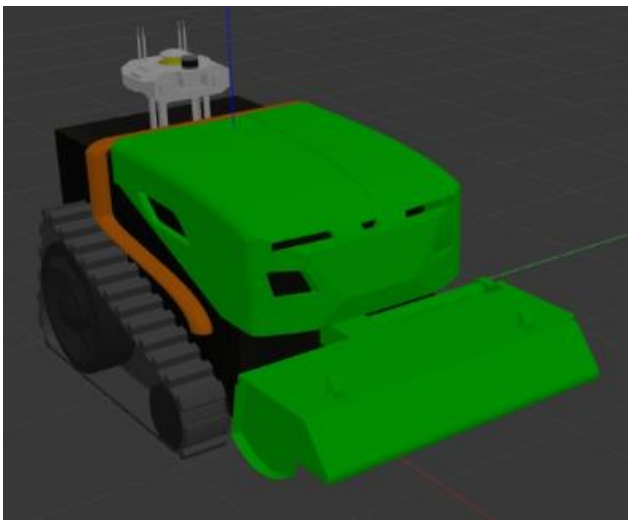


Figura 52 - Aperfeiçoamento visual do trator.

Além do cálculo das inercias, também foi necessário calcular os atritos, estes dados permitem que quando o robô esteja em contacto com a superfície mundo, na simulação, consiga ter o atrito suficiente para se poder deslocar. Estes parâmetros são inseridos no ficheiro URDF do trator num parâmetro designado de *"friction".*

Por último, foram adicionados os planos de colisão para simular o contacto do robô com qualquer objeto presente na simulação, após todos os dados estarem devidamente parametrizados de acordo com a realidade, assim como a ligação de todas as partes do robô. Na Figura 53 podemos observar os *"links"* e as suas "TFs" de todas as componentes do trator presentes no RVIZ, esta ferramenta de visualização tridimensional é amplamente utilizada, e é uma abreviação do ROS *Visualization*. Permite que os utilizadores observem e interajam em tempo real com várias partes do funcionamento do sistema robótico e com seu ambiente. O RVIZ mostra uma representação fácil de entender, mostra os estados e os dados sensoriais do sistema robótico, tornando-o especialmente útil durante o desenvolvimento, na deteção de erros e nos testes realizados, e permite desta forma entender melhor o comportamento do robô com uma imagem mais detalhada [80] conforme se mostra na Figura 53.

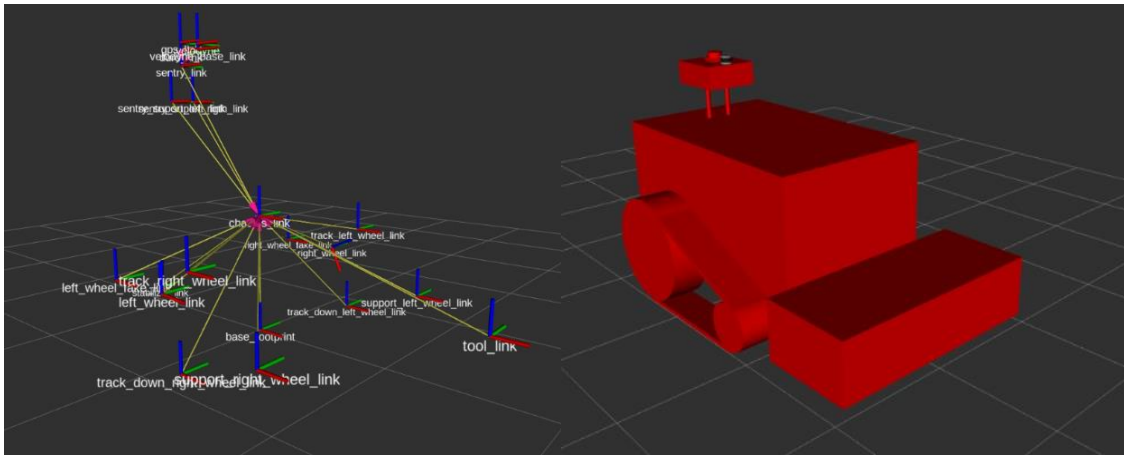


Figura 53 - Links e RVIS.

Podemos ver como resultado de uma comparação do trator real e da simulação, onde existe uma apenas uma pequena diferença, apresentada na Figura 54, onde se observa que no robô real não está presente o módulo sensorial, “Sentry”.

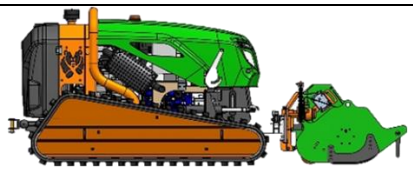
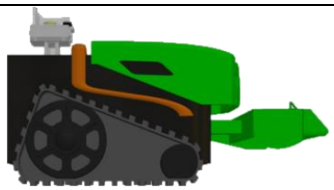
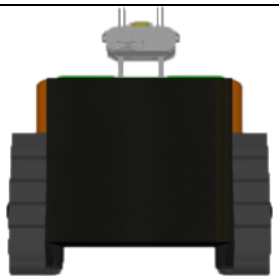
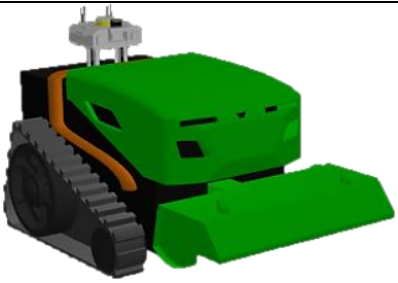
Vista	Real	Simulação
Lateral		
Traseira		
Perspetiva		

Figura 54 - Vistas referentes do robô real e do robô simulado.

3.6 Controladores

O sistema de controlo do robô é um sistema do tipo MIMO (*Multiple Inputs and Multiples Outputs*) este está equipado com vários controladores. Este trator é especialmente útil para atender às necessidades específicas da operação florestal devido à natureza imprevisível e variável desses ambientes. Ao trabalhar em florestas, o terreno frequentemente é irregular e acidentado, com obstáculos naturais que podem dificultar a locomoção e a manobra [8]. Podemos considerar que o controlador se resume em duas componentes, uma componente que se refere aos tipos de controlo, esta é realizada no “Controller” onde é referido e apresentado na Figura 31 e a outra componente de controlo incide-se no “Microcontrolador”, também demonstrado e apresentado na Figura 31. Devido à inexistência de saber quais as velocidades de cada lagarta, o seu controlo é realizado obtendo a posição e a orientação do trator. Através da plataforma ROS, é possível adquirir através dos diversos “tópicos” publicados quer a posição quer a orientação do robô assim como através do tópico “cmd_vel”, obter a velocidade linear e a velocidade angular do robô. Não sendo possível controlar a velocidade do trator, assumimos a velocidade como

variáveis de controlo mas como valores que permitem entender como vai ser o movimento do trator.

Sabendo que estamos perante uma estrutura diferencial, esta apenas possui dois eixos em x e z, sendo o eixo x referente à velocidade linear e no eixo z à velocidade angular. De forma a poder deixar o sistema em segurança, foi incluída uma proteção em software onde a soma destas duas componentes não pode ser superior a 1. Ou seja, 1 é considerada a velocidade máxima do sistema e 0 a velocidade mínima. Na Tabela 10 é possível observar diversos exemplos de como é possível entender como o robô teria de se movimentar de acordo com os valores obtidos do tópico “cmd_vel”.

Tabela 10 - Comportamentais referentes aos valores do "cmd_vel".

i	**Velocidade Linear**	**Velocidade Angular**	**Comportamento**
1	x = 1	z = 0	Mover para Frente
2	x = -1	z = 0	Mover para trás
3	x = 0	z = 1	Rodar *Anti-ClockWise*
4	x = 0	z = -1	Rodar *ClockWise*
5	x = 0.5	z = 0.5	Movimento circular no Sentido *Anti-ClockWise*
6	x = 0.5	z = -0.5	Movimento circular no sentido *ClockWise*

Este tópico é publicado pelo nosso *“Controller”,* no entanto quem vai injetar a tensão aplicada ao trator é um driver que é controlado pelo “Microcontrolador”. Ou seja, existem valores específicos que fazem com que o “Microcontrolador” reaja de formas diferentes. O código presente no microcontrolador é basicamente uma máquina de estados que dependendo dos valores presentes no tópico “cmd_vel” os valores de tensão que vão ser enviados para o driver de cada lagarta vão se alterando. Esta máquina de estados é representada como mostra a Figura 55.

No início, realizaram-se diversos testes com o propósito de determinar quais os valores máximos e os valores mínimos de tensão que podem ser aplicados aos terminais do trator. Estes testes tiveram como objetivo estabelecer um intervalo apropriado de valores que viabilizasse o funcionamento otimo das eletroválvulas nas lagartas do trator. Nesta fase de testes efetuou-se a aplicação gradual de diferentes níveis de tensão nos terminais das eletroválvulas. Para cada valor de tensão aplicado, monitorizaram-se os efeitos nas lagartas do trator, tais como a velocidade de deslocação, a resposta da direção, a estabilidade e outros parâmetros pertinentes. Ao analisar os resultados destes testes, foi possível determinar os limites máximos e mínimos de tensão nos quais as eletroválvulas e as lagartas do trator operavam de forma fiável e eficiente. Estes limites definiram a amplitude de tensão nas quais foram posteriormente estabelecidas nas funções. Baseado nos valores máximos e mínimos de tensão, foram reslizadas as equações/funções destinadas a modelar o

comportamento das eletroválvulas e das lagartas do trator em resposta à variação da tensão aplicada, conforme a Figura 55. Isto permitiu a criação de um conjunto de regras e/ou padrões que poderiam ser utilizados para regular o funcionamento das eletroválvulas e otimizar o desempenho do trator.

IF $x = 0$ **AND** $z = 0$

$V_R = V_L = 0$

IF $x = 0$ **AND** $z \,!= 0$

$V_R = V_L = abs(\frac{Max_{Rot} - Min_{Rot}}{0.9} * z + Max_{Rot} - \frac{Max_{Rot} - Min_{Rot}}{0.9})$

IF $z <= -0.01$

TURN CLOCKWISE $(V_R = -V_R \; AND \; V_L = V_L)$

IF $z >= 0.01$

TURN ANTI-CLOCKWISE $(V_R = V_R \; AND \; V_L = -V_L)$

IF $x \,!= 0$ **AND** $z = 0$

$V_R = V_L = abs(\frac{Max_{Linear} - Min_{Linear}}{0.9} * x + Max_{Linear} - \frac{Max_{Linear} - Min_{Linear}}{0.9})$

IF $x <= -0.01$

GO BACK $(V_R = -V_R \; AND \; V_L = -V_L)$

IF $x >= 0.01$

GO FRONT $(V_R = V_R \; AND \; V_L = V_L)$

ELSE

(A diferença caso z seja negativo os valores V_R e V_L são inversos ou seja $V_R = V_L$ e $V_L = V_R$)

IF $x > 0$ **AND** $z > 0$ **AND** $z \leq 0.1$

$x = 1.1 * (Max_{Linear} - Min_{Linear}) * x + Min_{Linear}$

$V_R = x$

$V_L = x * (1 + 1.2 * z)$

IF $x > 0$ **AND** $z > 0.1$ **AND** $z \leq 0.03$

$x = 1.1 * (Max_{Linear} - Min_{Linear}) * x + Min_{Linear}$

$V_R = x - \frac{4z}{0.3} - 60z$

$V_L = x - \frac{7z}{0.3} + 60z$

IF $x > 0$ **AND** $z > 0.3$

$V_R = 1.1 * (110 + \frac{10}{0.7 * abs(z - 0.3)})$

$V_L = 1.1 * (70 + \frac{15}{0.7 * abs(z - 0.3)})$

IF $z > 1$

$V_R = Max_{Linear}$

$V_L = Min_{Linear}$

Figura 55 – Equações do controlo da tensão aplicada às lagartas pelo Arduino.

As variáveis de entrada designadas referências R1 e R2 são valores calculados desejados, conforme se observa na Figura 56, estas referências são obtidas através do sistema de planeamento da trajetória do trator e servem como referências de entrada de cada um dos controladores. As variáveis dos erros e1 e e2 são obtidas

pela diferença entre os valores desejados menos o valor real, e ambos os controladores enviam os seus respetivos valores para o Microcontrolador este que posteriormente aciona diretamente o sistema de locomoção do trator nas suas lagartas, esquerda e direita, provocando assim o seu movimento e corrigindo o movimento face ao erro detetado em cada controlador.

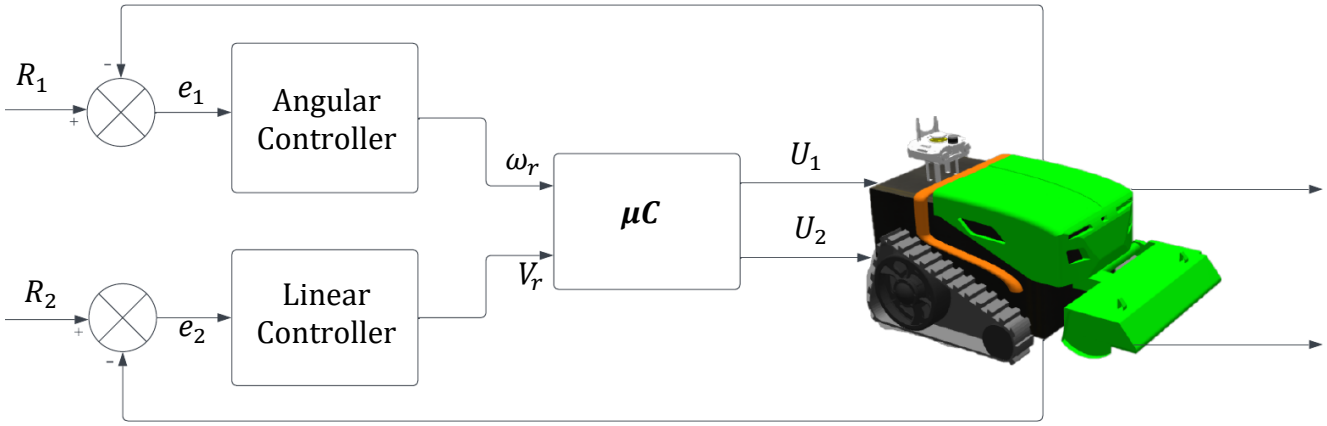


Figura 56 - Sistema de controlo do trator.

Os vários controladores do trator disponíveis podem ser escolhidos de acordo com as circunstâncias de operação. Por exemplo, um controlador pode ser ativado para maximizar a tração e a estabilidade do trator em terrenos inclinados ou escorregadios, melhorando assim a segurança e a sua eficiência. Além disso, ter a capacidade de substituir de forma dinâmica o controlador é possível a reduzir os efeitos no meio ambiente, por exemplo optar por um controlador mais suave em áreas sensíveis, onde a preservação do solo e da vegetação é mais sensível e evitar danos inevitáveis ao ecossistema circundante. O facto de possuir múltiplos controladores em operações florestais é uma inovação significativa neste contexto. Esta metodologia proporciona vantagens técnicas e aumenta a sustentabilidade das operações em ambientes desafiadores, demonstrando a capacidade da tecnologia de lidar com desafios diversos e ao mesmo tempo promover práticas mais éticas e eficazes na navegação destes equipamentos.

Neste tópico são abordados diferentes tipos de controladores desenvolvidos para controlo do robô tais como o controlador ON-OFF e diferentes tipos de controladores PIDs para controlar diversas variáveis.

3.6.1 ON-OFF

O controlador ON-OFF também conhecido como *Bang-Bang* (BB) está representado na Figura 57 e baseia-se na comutação entre dois valores, ou seja, neste tipo de controlo existe apenas uma mudança instantânea entre dois estados [81].

Neste contexto de estudo e análise, diversas variáveis desempenham papéis cruciais na compreensão e no controlo do sistema presente na Figura 57. A variável *Target* (T) define o local desejado no mapa, a variável *Actual Position* (AP) é a posição precisa em relação a um ponto de referência. A variável *On-Off Error* (eON-OFF) é o erro da posição onde este seja maior que um certo *threshold* uma nova trajetória é calcuada ela é o erro entre a posição desejada (T) e a posição real (AP) do sistema, a variável *Angular Value* (ωr) expressa o valor angular associado ao movimento rotacional ou curvilíneo. O *Microcontroller* (µC) é o "cérebro" do sistema, responsável por processar os valores obtidos no tópico “cmd_vel” para o controlo do trator. As variáveis Voltage *Right Wheel* (VR) e *Voltage Left Wheel* (VL) indicam as tensões aplicadas às rodas direita e esquerda, respetivamente, do sistema de

locomoção de forma a influenciar o movimento e a direção.

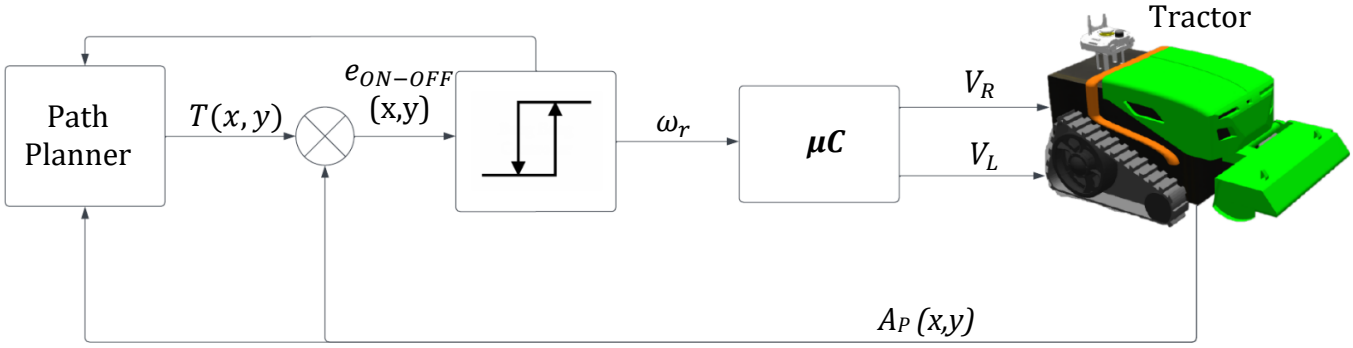


Figura 57 - Controlador ON-OFF.

Esta técnica apenas foi utilizada no início dos testes do sistema implementado, para verificar o movimento do robô numa direção constante entre duas coordenadas. A partir do conhecimento da trajetória pretendida, foram criadas duas trajetórias paralelas com uma distância de tolerância que consideramos de ±0,25 metros. No início o robô alinha-se com a trajetória pretendida. Caso a posição do robô coincida com qualquer ponto dessas trajetórias paralelas durante a navegação este algoritmo efetua um novo cálculo entre a posição atual e o destino. A ação é repetida até que a posição do robô esteja dentro de um raio de 0,25 metros da posição destino.

3.6.2 PID

Como foi referido no estado da arte, o controlador PID é um controlador utilizado em sistemas de controlo linear, fácil de implementar e específico para controlar processos e variáveis para um valor desejado [25] onde neste envolve três controlos separados cujas saídas se somam para formar um único sinal de saída. A leitura atual do parâmetro controlado é subtraída do comando para gerar um sinal de erro este que é processado por três unidades distintas [26].

3.6.2.1 PID Heading

O controlador PID *Heading* (PIDH) pretende controlar as trajetórias lineares e trajetórias angulares, desta forma são utilizados dois PID independentes conforme se representa na Figura 58, em que o foco inicial está localizado numa linha com as coordenadas do destino. Este alinhamento inicial é feito para ajustar o trator em relação ao ponto destino com uma precisão até quatro graus, sendo este o *treshold* angular admitido. Os valores de erro relacionados à orientação do robô e à posição em relação ao destino são atualizados durante o processo de navegação. O sistema permite assim uma limitação máxima de ±2 graus no âmbito do erro de orientação. A tensão aplicada nas lagartas são corrigidas quando o controlador deteta que foi ultrapassado este limite. Assim o trator permaneçe no caminho entre origem e o destino. Considerando que o erro angular é reduzido, este efeito contribui para uma trajetória suave e aplica um patamar de tensão às lagartas do robô correspondente ao erro calculado. O erro angular entre o robô e o ponto destino é inserido como referência no controlador PID à medida que o robô começa a sua trajetória com o intuito de alinhar o robô com o ponto destino da forma mais precisa possível.

Neste contexto de estudo e análise, diversas variáveis desempenham papéis cruciais na compreensão e no controlo do sistema presente na Figura 58. A *variável Desired Heading* (θ_{Des}) representa a orientação desejada, ou o ângulo a ser alcançado ou mantido pelo sistema, a variável *Target* (*T*) define o local desejado no mapa, a variável *Actual Position* (A_P) é a posição precisa em relação a um ponto de referência. A variável Heading Error (eHead) é o erro entre o ângulo desejado (θ_{Des}) e o ângulo real (θ) do sistema. A variável *Position Error* (e_{Pos}) é o erro entre a posição desejada (*T*) e a posição real (A_P) do sistema, a variável *Angular Value* (ωr) expressa o valor angular associado ao movimento rotacional ou curvilíneo. A variável *Linear Value* (V_R) expressa o movimento linear do sistema e a variável *Actual Heading* (θ) indica o ângulo real ou orientação atual do sistema. A variável *Microcontroller* (µC) é um "cérebro" do sistema, responsável por processar os valores obtidos no tópico “cmd_vel” para o controlo do trator. As variáveis *Voltage Right Wheel* (V_R) e *Voltage Left Wheel* (V_L) indicam as tensões aplicadas às rodas direita e esquerda, respetivamente, do sistema de locomoção de forma a influenciar o movimento e a direção. Este controlador está também associado ao algoritmo *Vector-Field*.

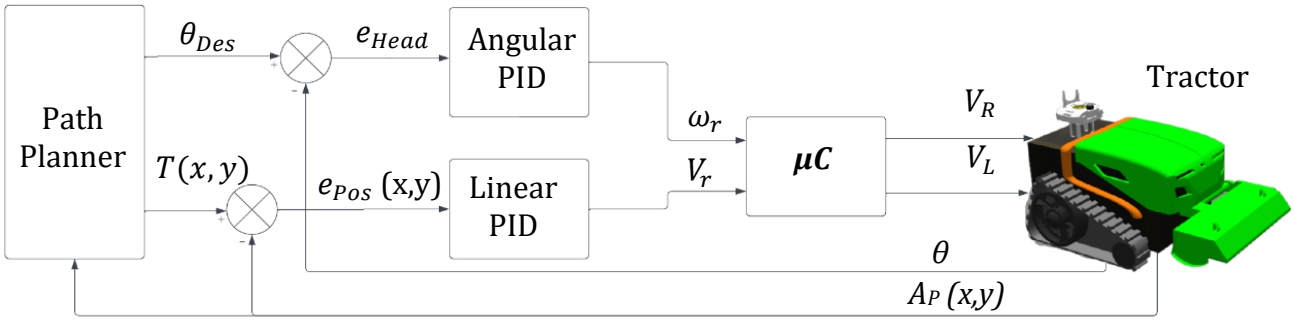


Figura 58 - Controlador com erro associado ao *Heading*.

3.6.2.2 PID Cross Track Error

O controlador PID CTE (*Cross Track Error*) controla trajetórias lineares e angulares de forma autónoma e efetua este tipo de controlo através de dois controladores PID, como representa na Figura 59. Os valores de erro são continuamente atualizados durante a operação de navegação e incluem a posição relativa do robô em relação à linha e ao seu destino final. Foi considerada tolerância máxima de ±10 centímetros no erro de posição em relação à trajetória desejada. É feita uma correção à tensão que alimenta as duas lagartas quando o controlador deteta que está próxima da tolerância admitida. Esta correção permite que o trator navegue com precisão, mantendo-o perto da trajetória planeada entre a origem e o destino. O erro de posição como referência, permite que o sistema também possa alterar a tensão aplicada às lagartas de acordo com a proximidade do ponto destino. Isso significa que a tensão pode aumentar ou diminuir dependendo da distância, o que permite que o trator se adapte às circunstâncias.

Neste contexto de estudo e análise, diversas variáveis desempenham papéis cruciais na compreensão e no controlo do sistema presente na Figura 59. Todas são idênticas às variáveis referentes anteriormente exceto à distância desejada da linha (D_{DFL}), a distância atual a que está da linha (A_{DFL}) e ao erro dessa distância (e_{CTE}).

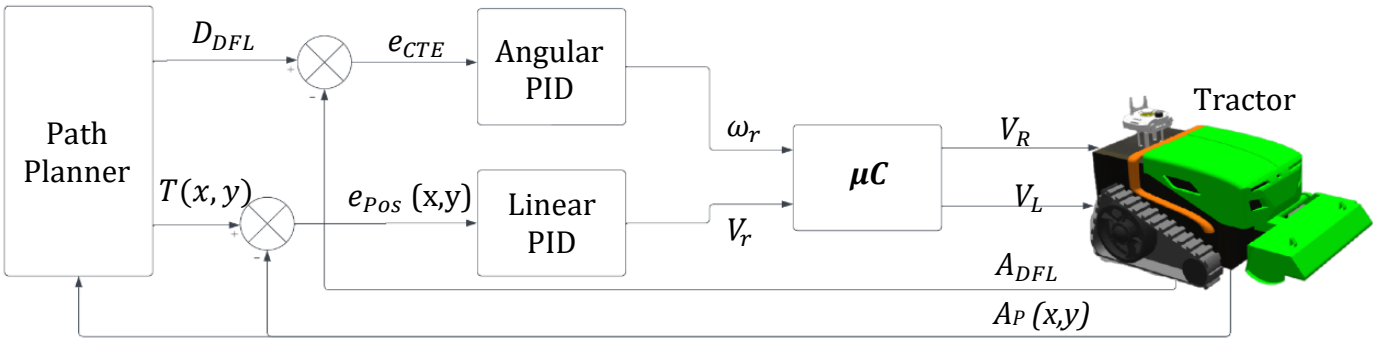


Figura 59 - Controlador com erro associado à distância da trajetória.

3.6.2.3 PID Non-Linear

O controlador PID NL (*Non-Linear*) realiza o controlo das trajetórias lineares e angulares do trator de forma autónoma utilizando um conjunto de dois controladores PID, como se representa na Figura 60 este controlador não-linear é constituído por duas partes distintas. Inicialmente, obtêm-se o ângulo de referência theta fornecido pelo sistema de planeamento de trajetórias o *"Path Planner"*. No diagrama de blocos podemos observar que o circuito externo faz a realimentação da posição do robô comparando assim a trajetória atual com a trajetória planeada disponibilizada pelo *"Path Planner"* [82]. E o valor obtido é convertido num ângulo de correção através de um controlador proporcional com um ganho Kct, originando assim um termo de correção φ. Esta correção é subtraída ao valor θ para obter o novo θ desejado, que será a nova referência de entrada no circuito interno do diagrama de blocos. Este circuito, por sua vez, recorre à orientação atual com a realimentação para calcular o efeito de atuação necessária.

Neste contexto de estudo e análise, diversas variáveis desempenham papéis cruciais na compreensão e no controlo do sistema presente na Figura 60. A variável "*Bearing of path"* (ϕ) representa a orientação que o sistema deve seguir em relação a um ponto de referência. A variável *"Current path segment"* (C_{PS}) indica em que parte do trajeto o sistema se encontra no momento, permitindo tomar ações apropriadas com base nessa informação. A *"Distance from point to line"* (D_{DFPL}) mede a distância perpendicular entre um ponto específico e a linha de referência. Essa medida ajuda a avaliar quão próximo o sistema está da rota desejada e a corrigir desvios. O *"Cross track error"* (e_{CTE}) mede o desvio lateral do sistema em relação à trajetória desejada. Ele informa o quão longe o sistema está da rota planejada e é usado para calcular correções. O *"Gain"* (K_{CT}) é um fator multiplicativo aplicado a certas variáveis ou ações do controlador. Ele regula a intensidade das correções ou respostas do sistema conforme a necessidade. A variável *"Corrective turn angle"* (φ) determina o ângulo pelo qual o sistema deve se ajustar para corrigir desvios da trajetória planeada, sendo fundamental para guiar o sistema de volta à rota correta. A variável *Desired Heading* (θ_{Des}) representa a orientação desejada, ou o ângulo a ser alcançado ou mantido pelo sistema, a variável *Target* (*T*) define o local desejado no mapa, a variável *Actual Position* (A_P) é a posição precisa em relação a um ponto de referência. A variável *Heading Error* (e_{Head}) é o erro entre o ângulo desejado (θ_{Des}) e o ângulo real (θ) do sistema. A variável *Position Error* (e_{Pos}) é o erro entre a posição desejada (T) e a posição real (A_P) do sistema, a variável *Angular Value* (ω_r) expressa o valor angular associado ao movimento rotacional ou curvilíneo. A variável *Linear Value* (V_r) expressa o movimento linear do sistema e a variável *Actual Heading* (θ) indica o ângulo real ou orientação atual do sistema. O *Microcontroller* (µC) é o "cérebro" do sistema, responsável por processar os valores obtidos no tópico “cmd_vel” para o controlo do trator. As variáveis *Voltage Right Wheel* (V_R) e *Voltage Left Wheel* (V_L) indicam as tensões aplicadas às rodas direita e esquerda, respetivamente, do sistema de locomoção de forma a influenciar o movimento e a direção.

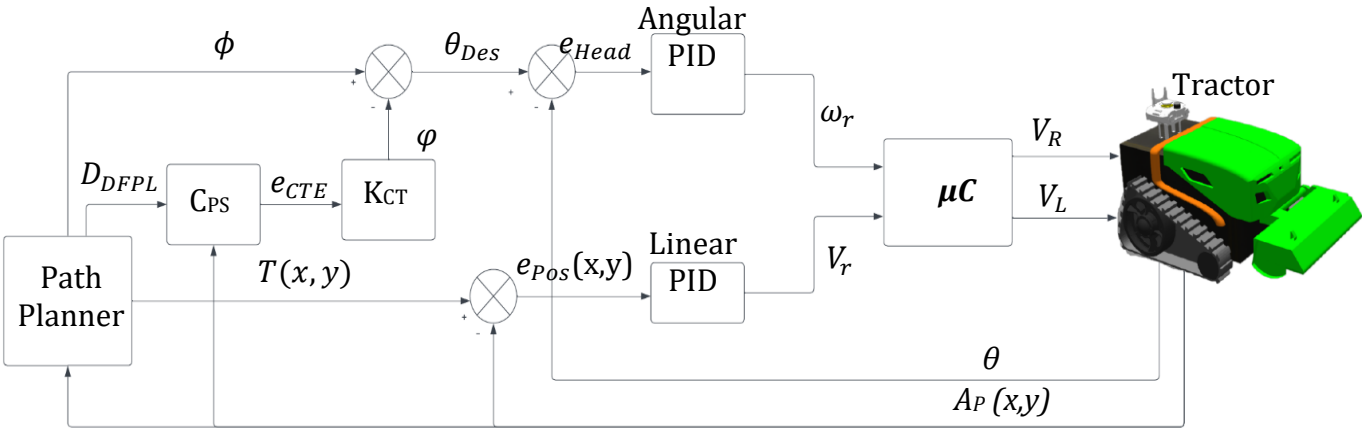


Figura 60 - Controlador Non-Linear.

3.7 Algoritmos de Navegação

Esta seção é destinada para descrever uma variedade de algoritmos de navegação adotados neste sistema. Cada um destes algoritmos fornece uma forma distinta de resolver os problemas de navegação autónoma e fornece informações importantes sobre como podem movimentar e se adaptar a diferentes ambientes.

3.7.1 Algoritmo ON-OFF

Na Figura 61 mostra-se o algoritmo ON-OFF considerado o algoritmo mais fácil de implementar, este tem como principal função estabelecer as fronteiras que limitam os movimentos do trator conforme se mostra na Figura 62, em que o trator segue um movimento dentro dos limites definidos.

```
Flag_Goal_Reached = FALSE
Flag_Rotation = TRUE

FOR coordinates in FILE:
    WHILE Flag_Goal_Reached = FALSE:
        IF detect_people = TRUE:
                STOP MACHINE
        ELSE:
                IF Goal_Reached:
                        STOP MACHINE
                        Flag_Goal_Reached = TRUE
                ELIF Flag_Rotation = TRUE:
                        IF Rotation_Finished:
                                STOP MACHINE
                                Flag_Rotation = FALSE
                        IF Rotation_Not_Finished:
                                ROTATING MACHINE
                                IF 10_Seconds_Passed_Without_Changing_Position:
                                        INCREASE ROTATION
                ELSE:
                        IF Not_Between_Lines:
                                HeadingRotation = ActualHeading
                                Flag_Rotation = TRUE
                                STOP MACHINE
                        ELSE:
                                GO IN FRONT
```

Figura 61 - Algoritmo ON-OFF.

O pseudocódigo delineado na Figura 61 traduz um processo de navegação bem como a tomada de decisões definidas para o movimento do trator. O sistema é projetado para responder dinamicamente às condições do ambiente e à proximidade do ponto destino. O algoritmo começa por estabelecer duas principais *flags*: *"Flag_Goal_Reached"* e *"Flag_Rotation"*, inicialmente configuradas como verdadeiro ou falso, respetivamente. O algoritmo percorre as coordenadas até que a *"Flag_Goal_Reached"* seja falsa. No algoritmo o primeiro teste de verificação é a deteção de pessoas, e no caso de serem detetadas, o trator irá parar o seu movimento para garantir a segurança das mesmas. Se não houver deteção, o algoritmo avalia a condição de chegada de destino, ou seja, se já alcançou ou não o ponto de destino. Se o objetivo for alcançado, a máquina irá parar e a *"Flag_Goal_Reached"* é definida como verdadeira. No caso de existir alguma rotação (ou seja, *"Flag_Rotation"* esta ser verdadeira), o algoritmo verifica se a rotação já foi ou não concluída. Se esta for concluída, a máquina será parada e a *"Flag_Rotation"* é desativada (definida como falsa). Se a rotação ainda não estiver completa, a máquina continua a rodar, e após passarem 10 segundos sem alteração de posição, a velocidade de rotação é aumentada. Caso não seja necessária efetuar uma rotação (*"Flag_Rotation* esta flag é falsa), e o algoritmo verifica se o trator está ou não está entre as linhas desejadas.

Se não estiver, a *"Flag_Rotation"* é novamente ativada, a máquina é parada e o trator realinha sua orientação. Se o trator estiver entre os limites admitidos, este continua a mover-se para frente em direção ao objetivo conforme se mostra na Figura 62.

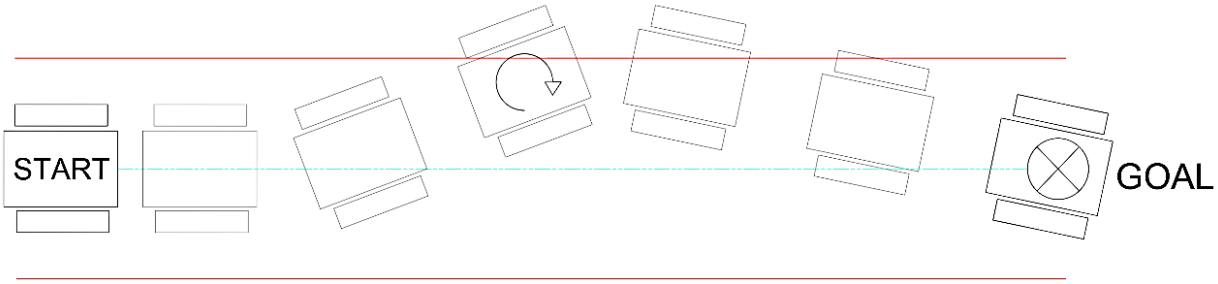


Figura 62 - Trajetória com o algoritmo bang-bang.

3.7.2 Algoritmo PIDH

O algoritmo apresenta-se na Figura 63 com intuito de se deslocar conforme a Figura 64. É um algoritmo um pouco mais complexo relativamente ao anterior, pois este já introduz um PID no seu controlo.

```
Flag_Goal_Reached = FALSE
Flag_Rotation = TRUE

FOR coordinates in FILE:
    WHILE Flag_Goal_Reached = FALSE:
        IF detect_people = TRUE:
                STOP MACHINE
        ELSE:
                IF Flag_Rotation = TRUE:
                        IF Rotation_Finished:
                                STOP MACHINE
                                Flag_Rotation = FALSE
                        ELIF 10_Seconds_Passed_Without_Changing_Position:
                                Flag_Rotation = FALSE
                        ELSE:
                                ROTATING MACHINE
                ELSE:
                        IF Goal_Reached = TRUE
                                STOP MACHINE
                                Flag_Goal_Reached = TRUE
                                Flag_Rotation = TRUE
                        ELSE:
                                Calculate Distance_From_Goal_Error – Linear PID
                                Calculate Heading_Error – Angular PID
```

Figura 63 - Algoritmo PID Heading.

O cenário delineado pelo pseudocódigo apresentado na Figura 63 configura um processo de navegação altamente sensível onde o trator se adapta de forma dinâmica a variadas circunstâncias. A base deste processo reside nestas duas *flags*, *"Flag_Goal_Reached"* e *"Flag_Rotation"*. A primeira arranca inativa, indicando que o ponto destino ainda não foi alcançado, enquanto a segunda começa como ativa, sinalizando a necessidade de existir uma rotação. A deteção de pessoas é uma prioridade, levando à interrupção imediata do trator caso exista a deteção. Na ausência de pessoas, o algoritmo direciona o trator de acordo o planeamento de trajetória efetuado. Quando é necessária efetuar uma rotação (*"Flag_Rotation"* é ativa), e o algoritmo verifica se esta foi concluída ou se não houve mudança de posição, após 10 segundos. Se a rotação terminou a máquina pára e a *"Flag_Rotation"* é desativada. No entanto, se após 10 segundos se não existir mudança de posição, a *"Flag_Rotation"* também é desativada. Caso contrário, a máquina é orientada para efetuar a rotação. Quando não é mais necessária efetuar qualquer rotação (*"Flag_Rotation"* esta é inativa), o algoritmo verifica se o ponto destino foi alcançado. Em caso afirmativo, a máquina pára, e a *"Flag_Goal_Reached"* é ativada e a *"Flag_Rotation"* é reativada, de modo a poder calcular a melhor orientação. Caso contrário, o algoritmo entra em ação cálculo dos erros de distância e orientação em relação ao ponto destino. Os controladores PID lineares e angulares asseguram que o trator mantenha a sua trajetória precisa e alinhada ao ponto destino conforme se mostra na Figura 64.

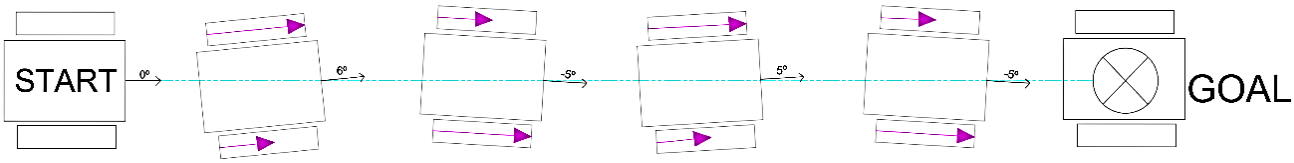


Figura 64 - Trajetória com o algoritmo PID Heading.

3.7.3 Algoritmo PID CTE

O algoritmo PID CTE conforme se mostra na Figura 65 tem como intuito de deslocar o trator conforme se mostra na Figura 66. É um algoritmo muito idêntico ao anterior, mas a variável erro usado como entrada dos PID é relativo à distância entre o robô e trajetória desejada.

```
Flag_Goal_Reached = FALSE
Flag_Rotation = TRUE

FOR coordinates in FILE:
    WHILE Flag_Goal_Reached = FALSE:
        IF detect_people = TRUE:
                STOP MACHINE
        ELSE:
                IF Flag_Rotation = TRUE
                        IF Not_Alligned AND
10_Seconds_Passed_Without_Changing_Position:
                                ROTATING MACHINE
                        ELSE:
                                Original_Position = Actual_Position
                                Flag_Rotation = FALSE
                ELSE:
                        IF Goal_Reached:
                                STOP MACHINE
                                Flag_Goal_Reached = TRUE
                        ELSE:
                                Calculate Distance_From_Goal_Error – Linear PID
                                Calculate Distance_From_Line_Error – Angular PID
```

Figura 65 - Algoritmo PID CTE.

Este pseudocódigo apresenta a abordagem da navegação do trator, promovendo a eficiência e a segurança ao conduzi-lo ao longo de um percurso predefinido. As *flags "Flag_Goal_Reached"* e *"Flag_Rotation"* são elementos centrais deste algoritmo, direcionando o comportamento do trator em resposta a diversos contextos. Enquanto percorre as coordenadas, o algoritmo apresenta duas prioridades, a deteção de pessoas e o ajuste à trajetória desejada. Quando há deteção de pessoas, ocorre uma paragem imediata do trator para garantir a segurança. Quando não há deteção de pessoas, a atenção volta-se para a precisão do movimento. Quando é necessária efetuar uma rotação (*"Flag_Rotation"* é ativada), o sistema avalia se a máquina não está alinhada e se nessa rotação após 10 segundos não existiu mudança de posição (significa que ficou preso). Caso os critérios sejam atendidos, o trator realiza a rotação. Caso contrário, a posição original é registada, e a *"Flag_Rotation"* é desativada. Quando não é mais necessário efetuar uma rotação (*"Flag_Rotation" = FALSE*), o algoritmo verifica se o ponto destino foi alcançado. Se o objetivo for atingido, a máquina é parada, a *"Flag_Goal_Reached"* é ativada e o trator cessa seu movimento. Se o objetivo ainda não foi atingido, os erros da distância em relação ao objetivo e à linha são calculados, permitindo ao trator ajustar sua trajetória de forma precisa.

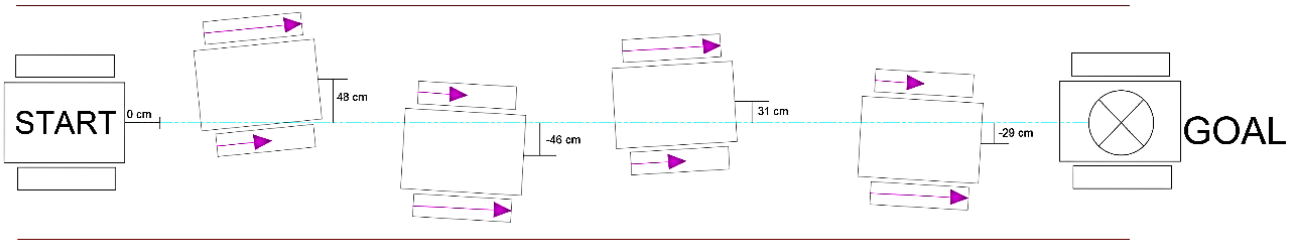


Figura 66 - Trajetória com o algoritmo PID CTE.

3.7.4 Algoritmo NL

O algoritmo NL mostra-se na Figura 67 e apresenta-se conforme a Figura 68. É um algoritmo mais complexo que os anteriores porque possuí as duas variáveis de erro dos algoritmos anteriores. Podemos caracterizar este algoritmo como integração dos algoritmos PIDH e PIDCTE.

```
IF No_Goals = TRUE

        RETURN FALSE

Flag_Goal_Reached = FALSE
Flag_Rotation = TRUE

WHILE Flag_Goal_Reached = FALSE:

        IF detect_people = TRUE:

                STOP MACHINE
        ELSE:
                IF Flag_Rotation = TRUE
                        IF Not_Alligned AND
10_Seconds_Passed_Without_Changing_Position:
                                ROTATING MACHINE
                        ELSE:
                                Original_Position = Actual_Position
                                Flag_Rotation = FALSE
                ELSE:
                        IF Goal_Reached:
                                STOP MACHINE
                                Flag_Goal_Reached = TRUE
                        ELSE:
                                Calculate Cross_Track_Error – Heading PID
                                Calculate Position_Error – Linear PID
                                Calculate Heading_Error – Angular PID
```

Figura 67 - Algoritmo Non-Linear.

O pseudocódigo delineia uma abordagem sistemática para a navegação do trator, incorporando múltiplos fatores para garantir a eficácia e a segurança. A análise começa com uma verificação de *"No_Goals"*, retornando *FALSE* se nenhum objetivo estiver definido, evitando a execução do algoritmo. Posteriormente, duas *flags* *"Flag_Goal_Reached"* e *"Flag_Rotation"* em que a primeira é inicializada como FALSE, indicando que o objetivo ainda não foi alcançado, enquanto a segunda começa como TRUE, forçando a necessidade de rotação. No cerne do algoritmo está em ciclo enquanto o objetivo não for atingido (*"Flag_Goal_Reached" = FALSE*). Durante este ciclo, a deteção de pessoas assume prioridade, parando de imediato o trator caso alguém seja detetado. Em cenários onde a rotação é necessária (*"Flag_Rotation" = TRUE*), o algoritmo avalia a alinhamento da trajetória e o tempo decorrido sem mudança de posição. Caso haja a necessidade de ajustar a trajetória, o trator é orientado a realizar a rotação. Caso contrário, a posição original é atualizada, e a *"Flag_Rotation"* é desativada. Quando não é mais necessário efetuar uma rotação (*"Flag_Rotation" = FALSE*), o algoritmo verifica se o ponto destino foi alcançado. Se o objetivo for cumprido, a máquina é parada, e a *"Flag_Goal_Reached"* é ativada. Se o objetivo não foi alcançado, uma série de cálculos de erros é realizada, envolvendo o erro de *"Cross_Track_Error"* (erro de trajetória transversal) para controlo de orientação, o erro de posição para controlo linear e o erro de orientação para controlo angular.

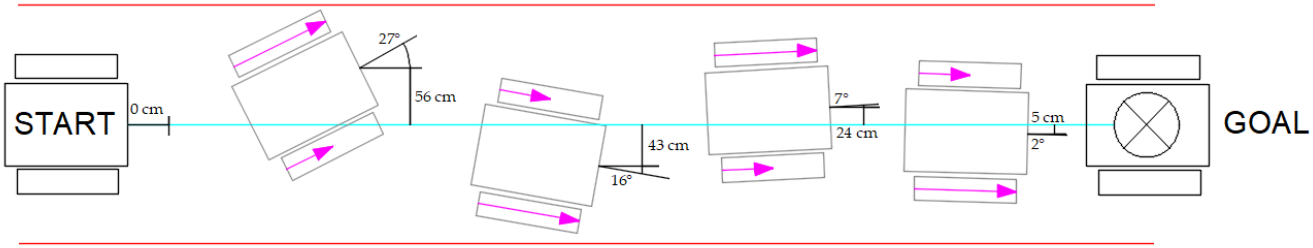


Figura 68 - Trajetória com o algoritmo Non-Linear.

3.7.5 Algoritmo VF

O algoritmo mostra-se na Figura 69 e apresenta-se na Figura 70. É um algoritmo um pouco diferente dos outros, apesar de também usar um controlo PID, o seu erro é proveniente da soma vetorial das forças virtuais existentes no decorrer da navegação.

```
IF No_Goals = TRUE

    RETURN FALSE

Flag_Goal_Reached = FALSE
Flag_Rotation = TRUE

WHILE Flag_Goal_Reached = FALSE:

    IF detect_people = TRUE:

        STOP MACHINE
    ELSE:
        IF Flag_Rotation = TRUE
            IF Not_Alligned AND 10_Seconds_Passed_Without_Changing_Position:
                ROTATING MACHINE
            ELSE:
                Original_Position = Actual_Position
                Flag_Rotation = FALSE
        ELSE:
            IF Goal_Reached:
                STOP MACHINE
                Flag_Goal_Reached = TRUE
            ELSE:
                Calculate S_Star – Linear PID
                Calculate Tau_Error – Angular PID
```

Figura 69 - Algoritmo Vector Field.

O pseudocódigo apresenta uma estratégia dinâmica para a navegação do trator, incorporando múltiplas verificações e cálculos para assegurar uma trajetória precisa e segura. O processo começa com a verificação de *"No_Goals"*, que retorna FALSE caso não haja objetivos definidos, evitando a continuidade do algoritmo sem um propósito. A seguir, duas *flags* a *"Flag_Goal_Reached"* e a *"Flag_Rotation"*. A primeira começa como *FALSE*, indicando que o objetivo ainda não foi alcançado, enquanto a segunda é inicializada como *TRUE*, sinalizando a necessidade de rotação. O coração do algoritmo é um loop que prossegue enquanto o objetivo não for atingido (*"Flag_Goal_Reached" = FALSE*). Durante cada iteração, há uma verificação prioritária de deteção de pessoas. Caso uma deteção positiva ocorra, o trator é imediatamente interrompido, priorizando a segurança. Nos casos em que a rotação é necessária (*"Flag_Rotation" = TRUE*), o algoritmo avalia se o trator não está alinhado e se 10 segundos passaram sem mudança de posição. Se ambos os critérios forem atendidos, o trator é orientado para realizar a rotação. Caso contrário, a posição original é atualizada, e a *"Flag_Rotation"* é desativada. Quando a rotação não é mais necessária (*"Flag_Rotation" = FALSE*), o algoritmo verifica se o objetivo foi alcançado. Se o objetivo for atingido, a máquina é parada, e a *"Flag_Goal_Reached"* é ativada. Se o objetivo não foi alcançado, o algoritmo entra em uma série de cálculos

envolvendo o erro de *"S_Star"* (erro de trajetória linear) para controlo de posição, e o erro de *"Tau_Error"* (erro de orientação angular) para controlo da orientação.

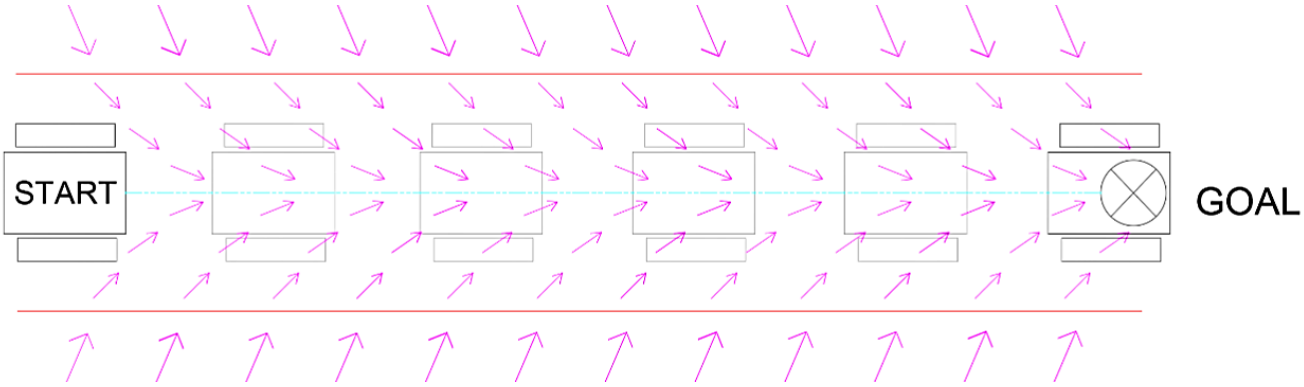


Figura 70 - Trajetória com o algoritmo Vector Field.

3.8 Algoritmos de Desvio de Obstáculos

Este tópico não apenas discute a variedade de métodos algorítmicos usados na navegação, mas também discute testes empíricos realizados em Python para garantir a eficácia dessas técnicas, particularmente os algoritmos de desvio de obstáculos. Duas estratégias notórias são o foco de nossa análise:

O algoritmo *A*,* que é conhecido por sua capacidade de encontrar rotas eficientes em meios de obstáculos e o algoritmo *Vector Field*, também conhecido como campo vetorial, que utiliza forças direcionais para contornar obstáculos e alcançar destinos.

Os testes realizados em Python fornecem uma visão concreta e verificada. Ambos mostram como os algoritmos de desvio de obstáculos ajudam o robô a evitar obstáculos e alcançar os seus objetivos com segurança, demonstrando a aplicação dessas abordagens. Ao examinarmos os detalhes destas experiências, ficará claro que os algoritmos *A** e *Vector Field* são soluções bastante robustas e confiáveis para lidar com a complexidade da navegação autónoma. Como resultado, podemos apreciar a importância desses algoritmos na capacidade dos dispositivos de se movimentarem de maneira autónoma e inteligente ao combinar conceitos teóricos com evidências da vida real.

3.8.1 Algoritmo A*

O *A** é um algoritmo de pesquisa frequentemente utilizado em encontrar caminhos. Este método calcula um percurso eficiente onde a navegação é realizada entre pontos, esses que são denominados de nós [83]. O algoritmo foi documentado por Peter Hart, Nils Nilsson e Bertram Raphael em 1966 [84], e pode ser encarado como uma melhoria do algoritmo de 1959 de Dijkstra.

3.8.1.1 Algoritmo Dijkstra

O algoritmo de Dijkstra aborda a questão de encontrar o caminho mais curto a partir de um ponto até um destino conforme se observa na Figura 71 com os dados apresentados na Tabela 11.

Tabela 11 - Distâncias entre o início e qualquer outro nó.

Nó	Distancia mais próxima desde o nó A	Nó Anterior
A	0	
B	4	C
C	3	A
D	6	C
E	8	B

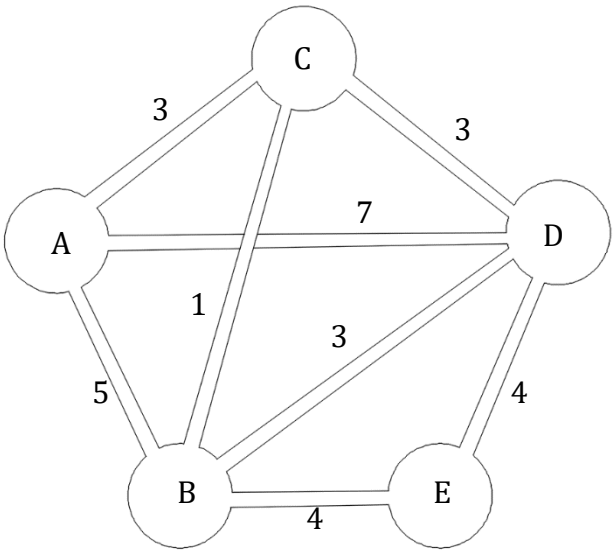


Figura 71 - Exemplo configurativo dos nós.

Tornou-se evidente que é possível encontrar os caminhos mais curtos de uma fonte específica para todos os pontos presentes num gráfico no mesmo intervalo de tempo. Por isso, esse problema é por vezes denominado de problema dos caminhos mais curtos de origem única [83], este problema é corrigido no Algoritmo *A**. Da Figura 72 até à Figura 75 estão presentes os caminhos mais curtos entre o nó A e os respetivos.

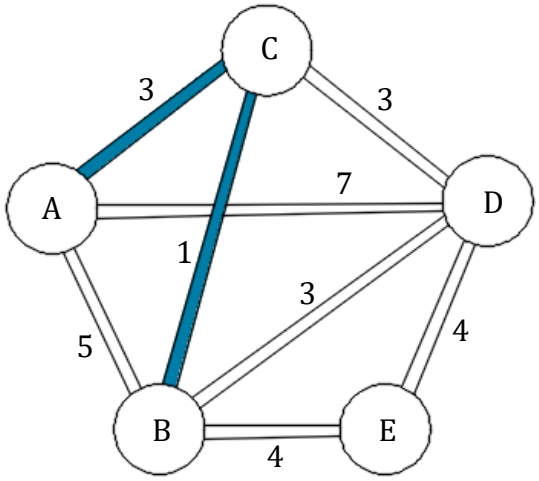


Figura 72 - Caminho mais curto do nó A ao nó B.

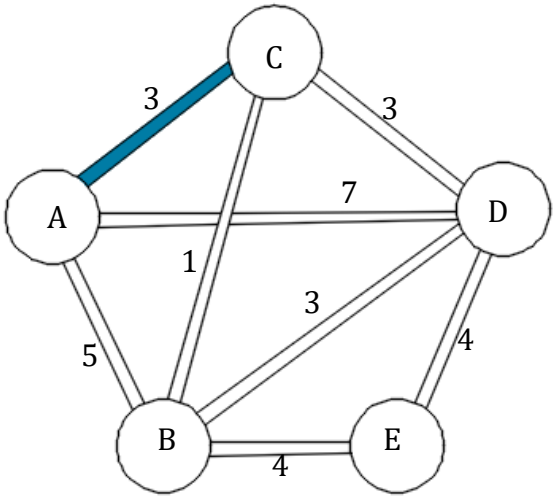


Figura 73 - Caminho mais curto do nó A para o nó C.

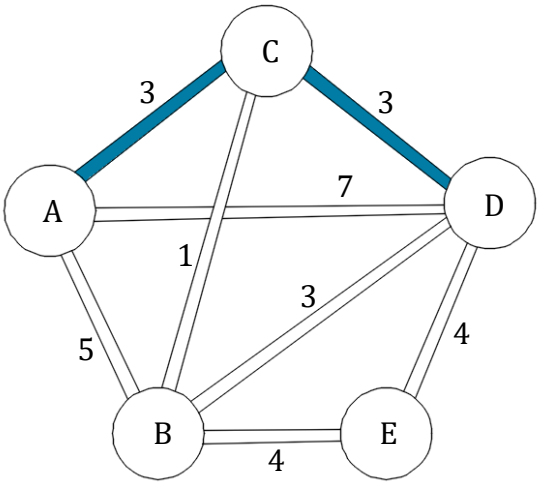


Figura 74 - Caminho mais curto do nó A para o nó D.

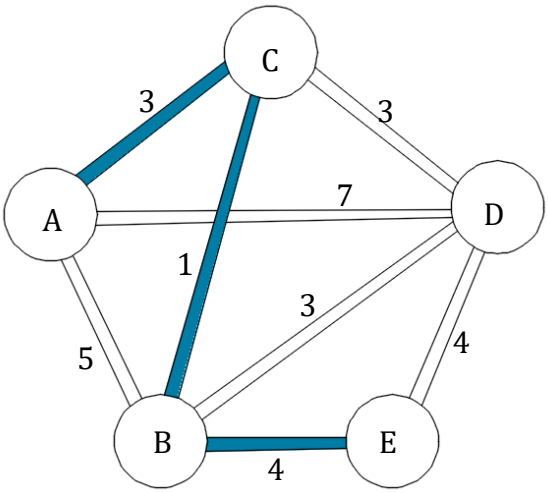


Figura 75 - Caminho mais curto do nó A para o nó E.

3.8.1.2 Algoritmo A* (Dijkstra + Função Heuristica)

Como foi referido anteriormente, o algoritmo A* é uma evolução do algoritmo de Dijkstra. A diferença está presente ao adicionarmos uma abordagem heurística de forma a encontrar os caminhos mais curtos, mas a grande diferença centra-se na maneira dos caminhos também serem mais eficientes. Ou seja, além de considerar o custo real da deslocação entre dois nós (como faz o algoritmo de Dijkstra), o A* também leva em conta uma estimativa heurística do custo de "esforço" para alcançar o destino a partir de cada nó. A grande vantagem do *A** é que ele evita a exploração excessiva de caminhos improváveis, isso resulta numa busca mais eficiente, muitas vezes alcançando soluções mais rapidamente do que o algoritmo de Dijkstra, especialmente nos gráficos com muitos nós. O algoritmo *A** foi criado para seguir o trajeto que implica o menor custo [51]. Para atingir esse objetivo, ele mantém uma fila

de prioridade organizada com diferentes percursos, o que se torna útil quando o robô precisa mudar de direção. Quando se depara com um obstáculo ou algo que bloqueie a primeira direção escolhida, o algoritmo procura encontrar uma nova direção viável e calcula qual é o novo caminho que minimiza os custos. Portanto, se a qualquer momento for identificado um trajeto com custo menor, os trajetos com custo mais alto são descartados e o robô segue em direção ao trajeto de custo menor [83]. Este processo prolonga-se até que o objetivo seja alcançado. O *A** é construído com base na pesquisa do melhor caminho em primeiro lugar e descobre um percurso de custo mínimo desde um nó inicial dado até um nó final. Na base do raciocínio subjacente ao algoritmo, encontra-se uma função $f(x)$ que resulta na soma das funções $g(x)$ e $h(x)$.

$$f(x) = g(x) + h(x) \tag{21}$$

A função $f(x)$ é também conhecida como "função distância-mais-custo-heurístico", onde x é o próximo nó no caminho, $g(x)$ é o custo do percurso do nó de partida até x e $h(x)$ é uma função heurística que estima o menor custo do caminho de x até o objetivo. No entanto, como $h(x)$ é uma estimativa heurística da distância até ao destino, este pode ser calculado de várias formas, mas é comum usar a distância em linha reta até ao destino devido à sua simplicidade. A função heurística é chamada de "consistente" quando satisfaz:

$$h(x) \leq d(x, y) + h(y) \tag{22}$$

"d" é o comprimento da aresta. Isso faz com que o *A** seja mais rápido e eficaz. Não precisa processar um nó mais do que uma vez e é equivalente ao algoritmo de Dijkstra com um custo reduzido:

$$d'^{(x,y)} \coloneqq d(x, y) - h(x) + h(y) \tag{23}$$

A complexidade temporal do *A** varia consoante a função heurística escolhida. No pior dos casos, a expansão dos nós pode ser exponencial em relação ao comprimento da solução. No cenário mais favorável onde $h*$ é a heurística ótima, cumpre a seguinte fórmula:

$$|h(x) - h^*(x)| = O(\log(h^*(x))) \tag{24}$$

Também surgem desafios na definição da função heurística do algoritmo *A** e devido a isso muitas funções heurísticas são desenvolvidas de forma a obter o valor ótimo. Uma das funções heurísticas comuns em busca de trajetos é a distância heurística. Dentre várias opções, temos distâncias heurísticas como a de Manhattan, a de exclusão e a diagonal [85]. Para este caso, uma função heurística foi desenvolvida de forma a obter o melhor caminho no menor espaço de tempo, como podemos observar:

$$h(x) = W_{Eucl} * d_{Eucl}(x) + W_{Obj} * d_{Obj}(x) + W_{Line} * d_{Line}(x) \quad (25)$$

Esta função possui pesos, estes que podem ser mudados de forma a serem obtidos os melhores resultados para uma melhor navegação. O d_{Eucl} representa a distância Euclidiana a que o robô está do objetivo (distância mais curta), o d_{Obj} representa a distância inversa a que o robô está do obstáculo, pois como o valor desta distância aumenta á medida que se aproxima do obstáculo ele vai dar um maior peso a esta proximidade ao calcular a função e o d_{Line} representa a distância a que o robô se encontra da trajetória. Os Ws simbolizam o peso das respetivas funções.

Na Figura 76 está representado um exemplo que vai ser feito passo a passo de forma a mostrar como funciona o algoritmo A*.

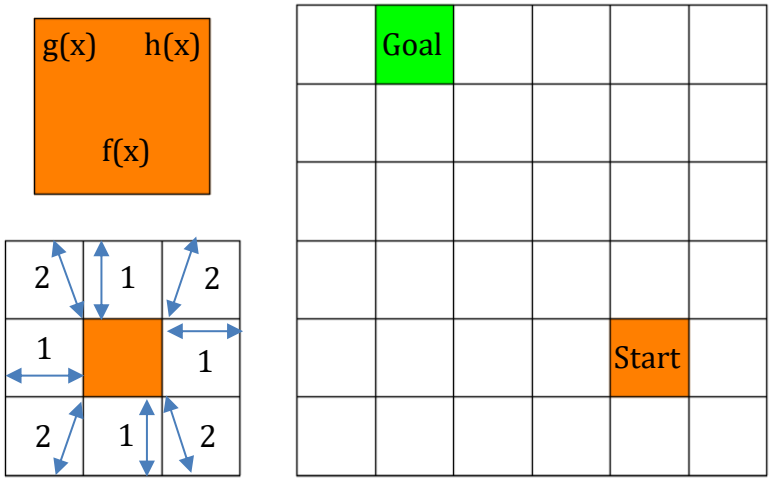


Figura 76 - Exemplo A* Algoritmo e Valores de Custo.

Foram criadas algumas representações visuais em formato de matrizes, como podemos observar da Figura 77 até à 80. Nas caixas ao longo da diagonal, atribuísse um custo de 2, enquanto nas caixas posicionadas horizontalmente e verticalmente, o custo é de 1 e temos uma caixa que representa o destino e outra que representa o ponto de partida. Cada caixa possui três valores: o valor no canto superior esquerdo que representa o resultado do algoritmo de Dijkstra $(g(x))$, o valor no canto superior direito que reflete o valor da função heurística $(h(x))$ e o valor no centro da caixa é o resultado da função distância-mais-custo-heurístico $(f(x))$. É importante sublinhar que estas imagens ilustram passo a passo o funcionamento do algoritmo *A**. Elas representam a aplicação prática do algoritmo para encontrar a rota mais eficiente.

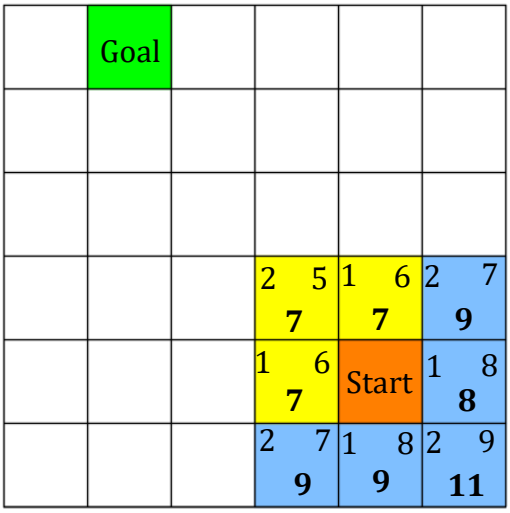


Figura 77 - 1ª Iteração do Algoritmo A*.

Figura 78 - 2ª Iteração do Algoritmo A*.

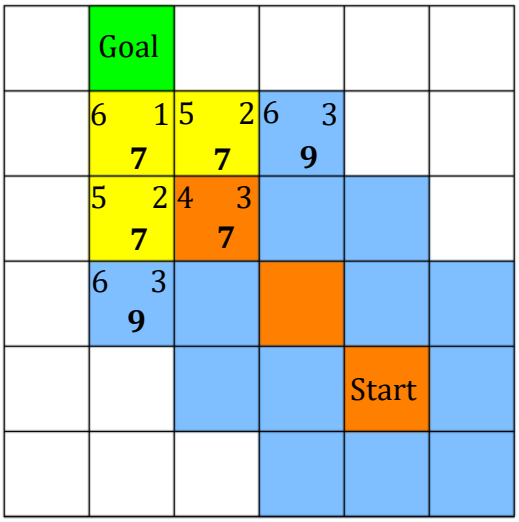


Figura 79 - 3ª Iteração do Algoritmo A*.

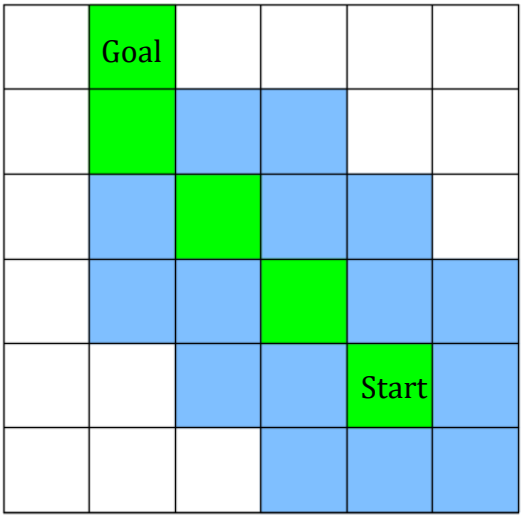


Figura 80 - Finalização do melhor trajeto.

Após esta explicação do cálculo do melhor trajeto, podemos prosseguir para o patamar onde se inclui obstáculos no seu percurso. Na Figura 81 está presente um exemplo que possui obstáculos e a partir deste irá ser feita uma demonstração de como o algoritmo *A** funciona na presença de obstáculos ao longo do cálculo do melhor trajeto possível.

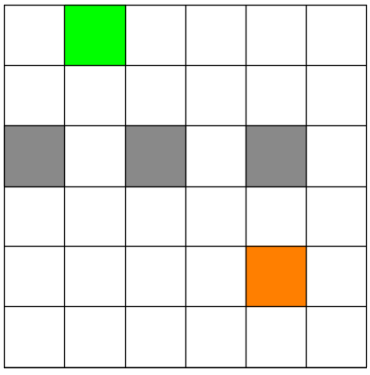


Figura 81 - Exemplo com obstáculos.

Assim como exemplificado anteriormente, os custos permanecem uniformes. Das Figuras 82 até à 85 ilustram passo a passo o comportamento do algoritmo *A**. Na Figura 85, é possível observar a existência de dois caminhos igualmente viáveis. Embora compartilhem os mesmos valores de custo, o caminho representado em tom azul claro já havia completado o trajeto. Assim, essa opção foi escolhida como a rota preferencial.

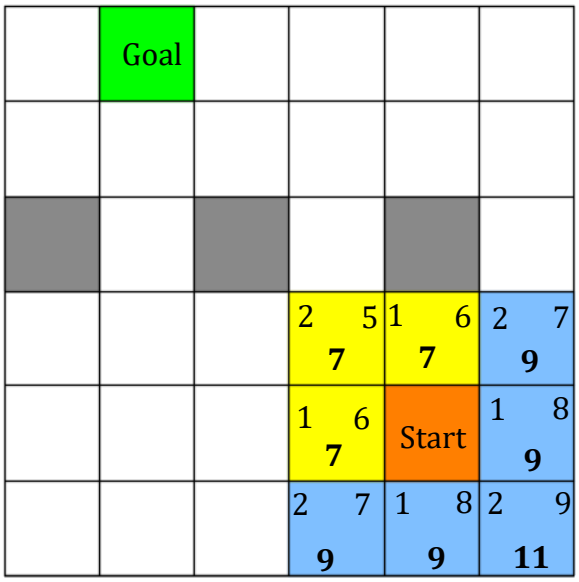


Figura 82 - 1ª Iteração do Algoritmo A* com obstáculos.

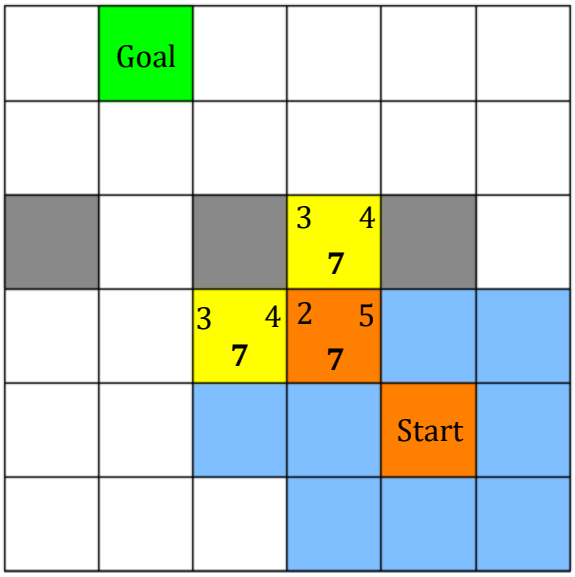


Figura 83 - 2ª Iteração do Algoritmo A* com obstáculos.

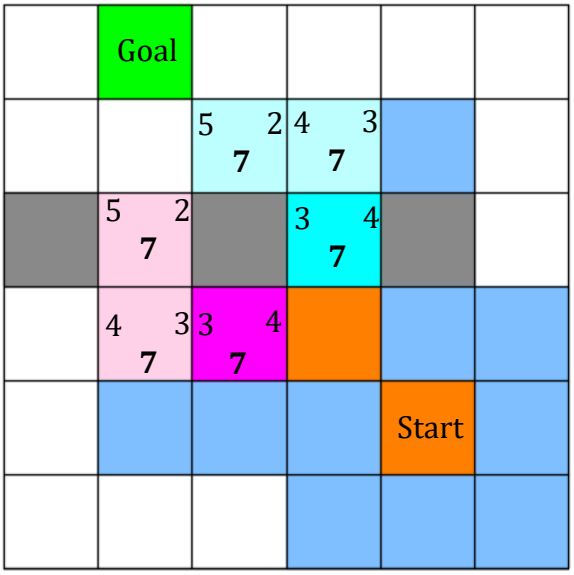


Figura 84 - 3ª Iteração do Algoritmo A* com obstáculos.

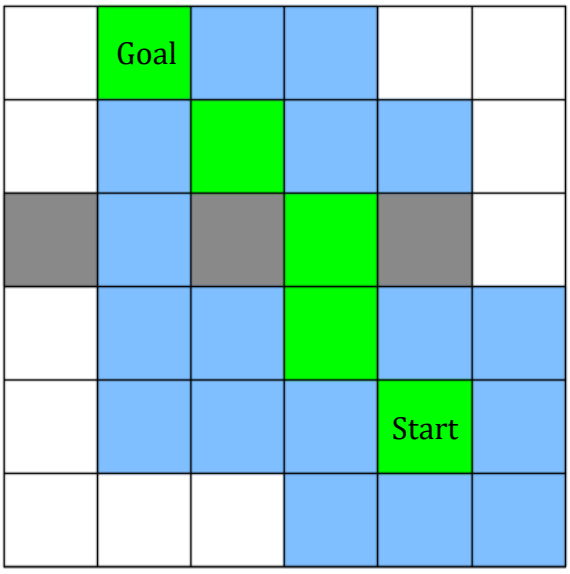


Figura 85 - Finalização do melhor trajeto.

3.8.2 Algoritmo VF

Este algoritmo parte do conceito de seguir uma trajetória através da criação de um campo vetorial em torno do percurso a ser seguido. Os vetores desse campo fornecem comandos de orientação para guiar o robô em direção à trajetória desejada. O objetivo não é acompanhar um ponto em movimento, mas sim posicionar-se na trajetória enquanto se desloca. Os vetores nesse campo estão direcionados para a trajetória que deve ser seguida, de acordo com a direção desejada da navegação [86]. Esses vetores atuam como comandos de orientação para o robô. Como podemos observar na Figura 86 que representa uma amostra de um campo vetorial entre a partida e o destino, desta forma direciona desde o seu início de navegação a linha entre pontos.

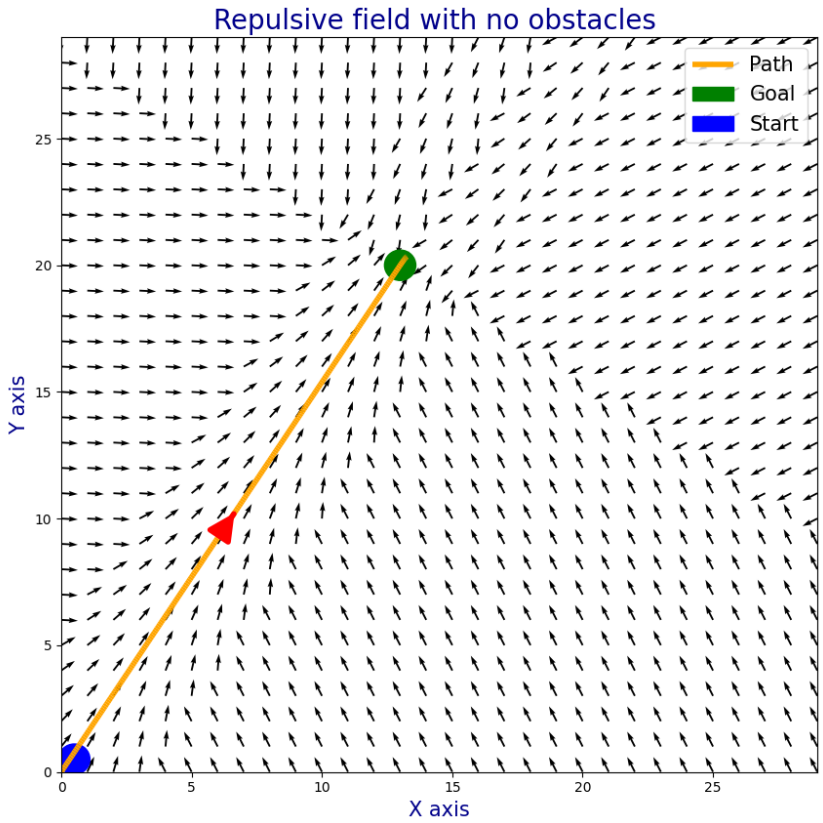


Figura 86 - Exemplo do campo magnético.

A ideia de campos vetoriais é comparável à dos campos potenciais, que têm sido amplamente utilizados como ferramenta para o planeamento de trajetórias. Os campos potenciais podem ser aplicados na navegação de robôs para situações de evasão de obstáculos e prevenção de colisões [55]. Os campos vetoriais distinguem-se dos campos potenciais no sentido de que nem sempre representam o gradiente de um potencial. Em vez disso, o campo vetorial simplesmente indica uma direção desejada de movimento.

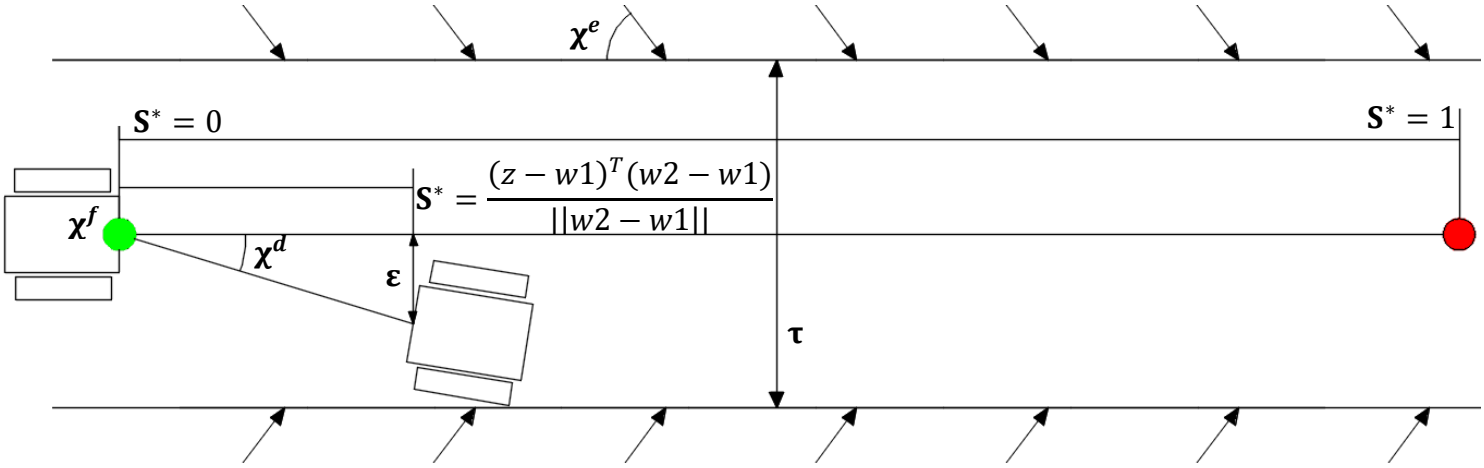


Figura 87 - Representação das variáveis do Algoritmo Vector Field.

Os cálculos e etapas mostrados a seguir representam passo a passo o funcionamento do algoritmo do *Vector Field,* a Figura 87 é uma representação das variáveis deste algoritmo.

Passos:

1º - Cálculo do Ângulo de Orientação (χf): Nesta etapa, é calculado o ângulo de orientação desejado (χf) para a robô ir do primeiro *waypoint* (w1) para o segundo *waypoint* (w2).

$$\chi f = atan2(w2_y - w1_y\,, w2_x - w1_x) \tag{26}$$

2º - Cálculo da Posição ao Longo da Trajetória (S^*): Aqui, é determinada a posição do robô ao longo da trajetória entre os *waypoints*. É projetado o vetor que une a posição atual (z) e o primeiro *waypoint* (w1) no vetor entre w1 e w2 fornecendo uma medida de onde o robô está posicionado ao longo da trajetória.

$$S^* = \frac{(z - w1)^T (w2 - w1)}{||w2 - w1||} \tag{27}$$

3º - Cálculo da Distância da Trajetória (ε): Essa etapa envolve o cálculo da distância entre a posição real do robô (z) e o ponto mais próximo na trajetória entre os *waypoints*. Isso ajuda a entender quão distante o robô está da trajetória desejada.

$$\varepsilon = ||z - (S^*(w2 - w1) + w1)|| \tag{28}$$

4º - Determinação do Lado da Trajetória (ρ): Aqui, é calculado um valor (ρ) que indica em qual lado da trajetória o robô se encontra. É realizado ao usar o produto vetorial entre o vetor que conecta os *waypoints* (w2 - w1) e o vetor que conecta a posição atual (z) ao primeiro *waypoint* (w1). O sinal desse valor indica se o robô está à esquerda ou à direita da trajetória.

$$\boldsymbol{\rho} = sign[(w2 - w1) * (z - w1)] \quad (29)$$

5º - Verificação de Posição em Relação ao *Waypoint* 2: Se o valor S^* for maior que 1, significa que o robô já passou do segundo *waypoint*. Nesse caso é mudado para o próximo *waypoint* pois o robô já alcançou o destino do *waypoint* atual.

6º - Mudança de *Waypoint*: Aqui, é feita a transição para o próximo *waypoint* na trajetória.

7º - Continuação do Processo: Caso o valor S^* seja menor ou igual a 1, isso indica que o robô ainda está entre os *waypoints* e devido a isso a sua navegação continua.

8º - Definição de ε como ρε: Aqui, o valor de **ε** é atualizado para ρ**ε**, o que representa a distância do robô à trajetória, considerando o lado em que o robô está.

9º - Verificação da Distância da Trajetória: Se o valor absoluto de **ε** for maior que um limite pré-definido τ, isso indica que o robô está a uma distância significativa da trajetória desejada.

10º - Cálculo do Ângulo Desejado (χ^d): Nesse caso, o χ^d é calculado como a diferença entre o ângulo de referência χ^f e o produto de $\rho * \chi^e$, onde χ^e é um fator de correção.

$$\chi^d = \chi^f - \rho * \chi^e \quad (30)$$

11º - Caso Contrário (Distância da Trajetória dentro do Limite): Se o valor absoluto de **ε** for menor ou igual a τ, isso significa que o robô está próximo o suficiente da trajetória.

12º - Cálculo do Ângulo Desejado com Correção (χ^d): Nesse cenário, o χ^d é calculado como a diferença entre o ângulo de referência χ^f e o produto de χ^e, τ e um fator k.

$$\chi^d = \chi^f - \chi^e \left(\frac{\boldsymbol{\varepsilon}}{\tau}\right)^k \quad (31)$$

13º - Finalização do Processo: O processo é encerrado.

Com base na compreensão do funcionamento deste algoritmo, é possível ampliar a sua aplicação para lidar com a presença de obstáculos. Nas Figuras 88 e 89, é apresentado o comportamento vetorial tanto em relação ao destino quanto em relação a um obstáculo [87]. A interação entre os vetores que conduzem ao destino e aqueles que evitam o obstáculo resulta num padrão de trajetória que contorna eficazmente as áreas de risco. Este passo adicional ilustra como o algoritmo é adaptativo, permitindo ao robô navegar de forma inteligente mesmo em ambientes desafiantes com obstáculos.

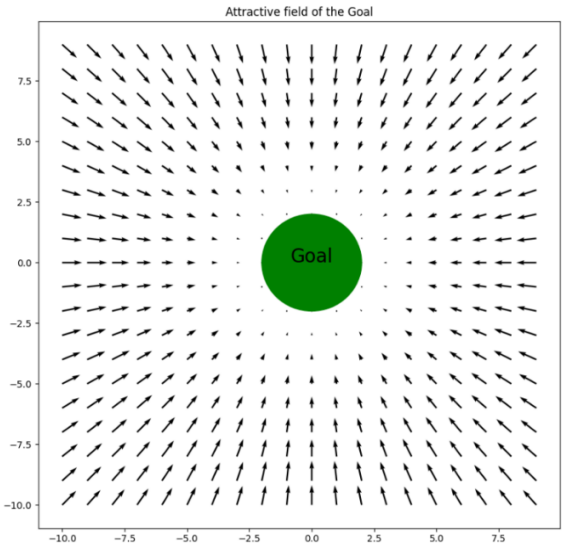


Figura 88 - Campo vetorial de um destino.

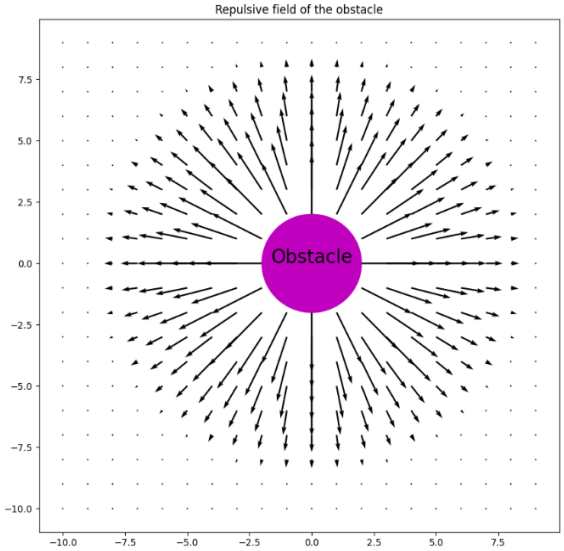


Figura 89 - Campo vetorial de um obstáculo.

Deste modo, o algoritmo ganha novas nuances, uma vez que passamos a considerar outros campos vetoriais. Isso implica que ao longo da trajetória podem surgir campos vetoriais associados a obstáculos, que precisam ser combinados de forma a permitir desvios conforme necessário. Na Figura 90, é ilustrado um exemplo de campos vetoriais entre dois pontos, com um obstáculo no meio. É visível que as setas laranjas representam as forças vetoriais atrativas em direção ao destino, enquanto as setas azuis representam as forças vetoriais repulsivas do obstáculo. Essa situação ilustrativa demonstra como a interação entre diversos campos vetoriais contribui para uma navegação inteligente em torno de obstáculos, enriquecendo a capacidade adaptativa do algoritmo.

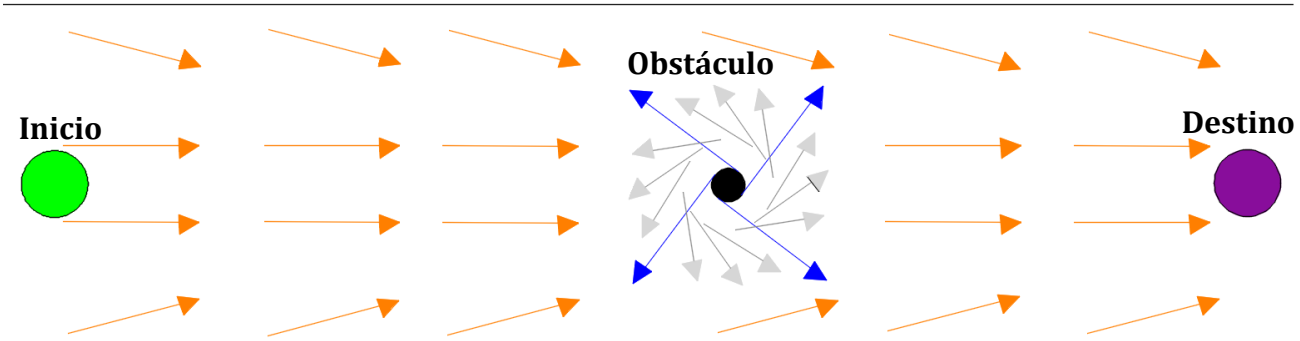


Figura 90 - Exemplo de campos vetoriais (Inicio - Destino).

A presença de diversas forças vetoriais nesse cenário implica que elas se combinem para determinar o caminho mais otimizado entre o ponto de início e o ponto de destino. Da Figura 91 até à 94 é exemplificada a combinação das componentes vetoriais, evidenciando como a soma dessas influências vetoriais resulta em uma trajetória coerente e adaptada às circunstâncias.

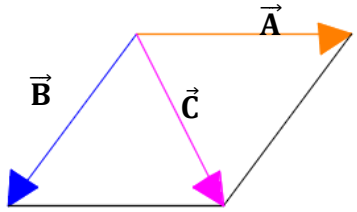


Figura 91 – Soma vetorial do ponto 1 da demonstração gráfica.

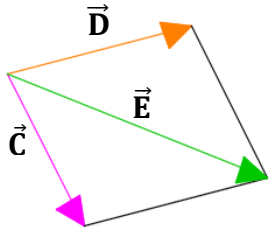


Figura 92 - Soma vetorial do ponto 2 da demonstração gráfica.

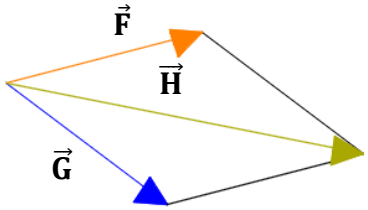


Figura 93 - Soma vetorial do ponto 3 da demonstração gráfica.

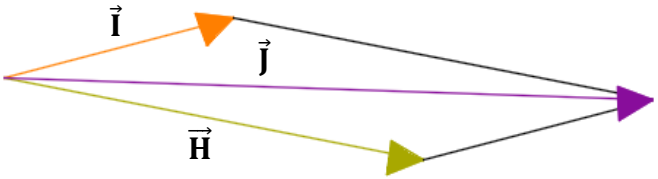


Figura 94 - Soma vetorial do ponto 4 da demonstração gráfica.

Na Figura 95, encontram-se representados os vetores calculados previamente, sendo cada um identificado pelo seu respetivo nome. Ao analisarmos mais detalhadamente, constatamos que ao somar esses vetores, eles formam um padrão de movimento que contorna o obstáculo presente na trajetória.

Esta abordagem através da vectorização permite uma navegação adaptativa, orientando o percurso de modo a contornar obstáculos de forma eficaz. Isto assegura uma navegação mais segura e eficiente, evitando colisões indesejadas de forma a proporcionar uma resposta inteligente à presença de obstáculos.

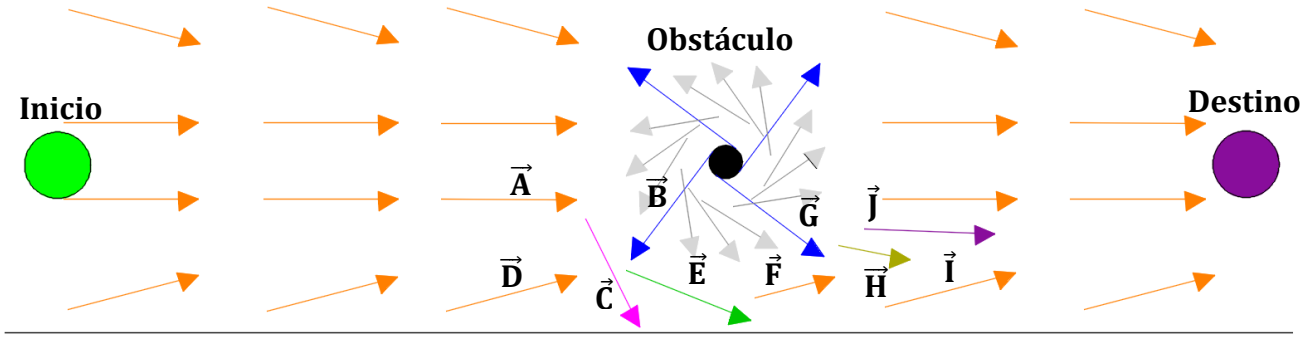


Figura 95 - Representação das somas vetoriais.

Após a agregação dos vetores através de somas, é delineado um percurso otimizado em vermelho na Figura 96, estendendo-se entre o ponto de início e o ponto de destino. Esse caminho é resultado da combinação dos vetores previamente calculados, que orientam o sistema de modo a contornar obstáculos e seguir a trajetória desejada.

A representação visual do trajeto mais eficiente evidencia a eficácia desse método em superar desafios impostos pelo ambiente, oferecendo uma rota segura e direcionada

para o robô prosseguir. Essa bordagem de cálculo e visualização de trajetórias ilustra a aplicação prática dos conceitos de vectorização e navegação inteligente.

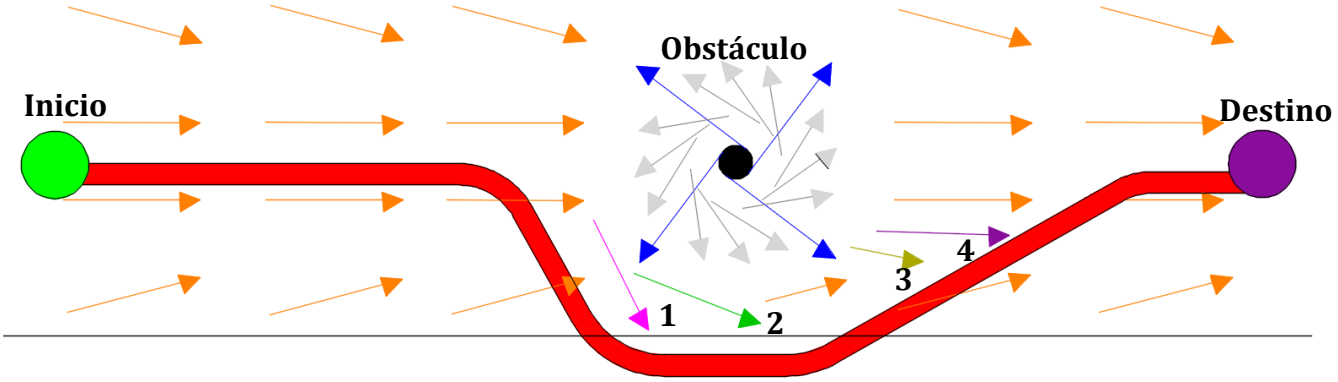


Figura 96 - Caminho realizado com os cálculos vetoriais.

CAPÍTULO 4: TESTES E RESULTADOS

Os vários algoritmos de navegação propostos foram analisados e avaliados, todos foram testados com sucesso quer no simulador quer no ambiente real. É possível determinar quais algoritmos que apresentam maior eficiência e precisão na orientação em ambientes desconhecidos. Assim como a implementação dos algoritmos para efetuar o desvio de obstáculos, sendo esta última experiência efetuada apenas para os algoritmos que apresentaram melhores resultados na primeira tarefa. Esta etapa visa melhorar a navegação, o que permite, que os sistemas reajam e evitem os obstáculos de forma inteligente, tornando-os mais confiáveis e seguros em ambientes difíceis. Este tópico examinará os algoritmos de desvio de obstáculos e de navegação destacando suas características.

4.1 Gridmap

Este processo de utilização dos algoritmos de desvio de obstáculos representa um avanço significativo na capacidade de navegação autónoma dos robôs. Ao utilizar um LiDAR, pois este sensor é capaz de emitir pulsos de laser e medir o tempo que demoram a regressar após refletirem em objetos, o robô pode criar uma representação detalhada do ambiente à sua volta. Isso inclui a identificação precisa da localização e forma dos obstáculos que podem obstruir o seu caminho. Com base nos dados obtidos pelo LiDAR, é gerado um mapa de grelha de ocupação, que divide o espaço em células e indica quais delas estão ocupadas por obstáculos e quais estão livres [88]. Essa informação é fundamental para que o robô possa tomar decisões informadas sobre a sua rota.

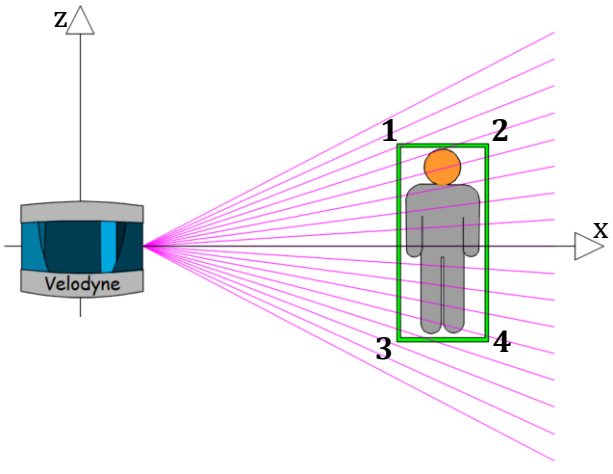


Figura 97 - LiDAR deteção eixo z e x.

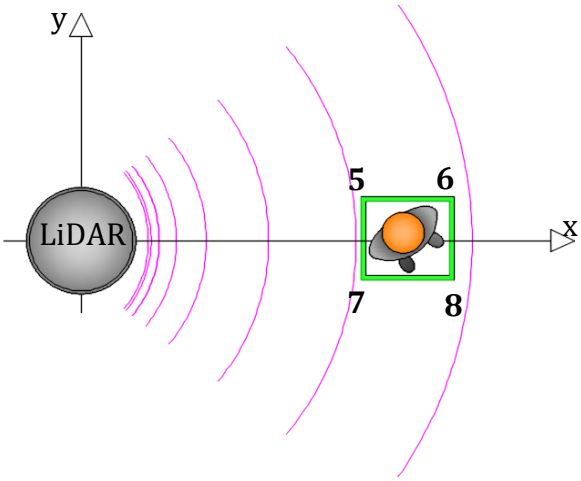


Figura 98 - LiDAR deteção eixo y e x.

Nas Figuras 97 e 98, podemos observar o agrupamento de pontos realizado pelo LiDAR em todas as direções. Isso nos permite identificar os pontos mais altos e mais baixos da caixa que envolve o objeto ou pessoa em análise. Ao obter esses pontos

extremos, é possível inserir no *gridMap* os pontos que correspondem à presença do obstáculo. Vale destacar que o *gridMap* é uma representação bidimensional, onde os valores inseridos refletem o cenário ilustrado na Figura 98. No entanto, os dados obtidos da Figura 97 também são de extrema importância, uma vez que a altura do obstáculo influência diretamente as ações que o robô deve executar.

O mapa de ocupação é uma representação que divide o espaço em uma grelha, onde cada célula está associada a uma probabilidade de ocupação específica (1 se estiver claramente ocupada por um obstáculo, 0 se a probabilidade de ocupação for nula) [32]. Ao conhecer os valores máximos e mínimos do agrupamento de pontos detetados pelo LiDAR, é possível delimitar a área que abrange esses quatro pontos. Todas as células que fazem parte dessa área serão associadas à probabilidade de ocupação 1 (representada a cinzento), enquanto as restantes células permanecerão com probabilidade de ocupação 0 (representada a branco). Para uma compreensão prática, as Figuras 99 e 100 apresentam um exemplo de como o *gridMap* é preenchido com base nos dados das Figuras 97 e 98.

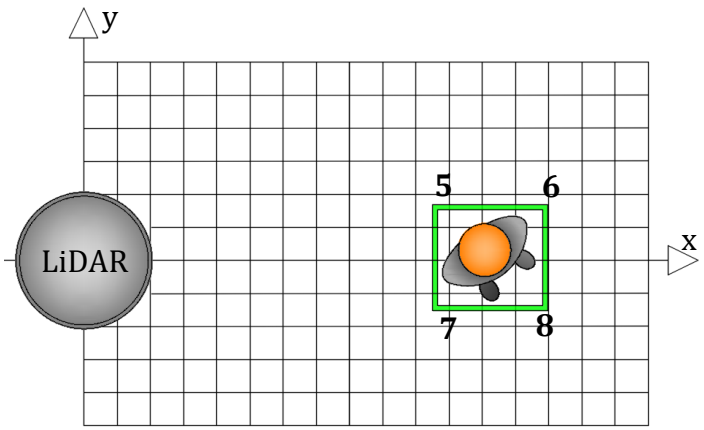


Figura 99 - Grid Map.

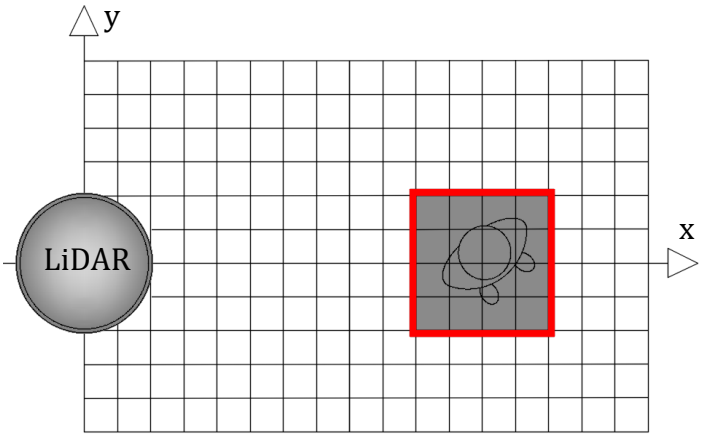


Figura 100 - Grid Map com obstáculo.

Este mapa está em constante atualização durante a navegação do robô, por isso o *"PathPlanner"* utiliza este mapa de grelha de ocupação como entrada para calcular as trajetórias. A atualização dos dados da grelha a uma frequência de 10 Hz permite que o *"PathPlanner"* reaja de forma ágil às mudanças no ambiente, garantindo que o robô esteja sempre a par das condições ao seu redor. A cada 100 ms, uma nova trajetória é gerada pela combinação dos algoritmos de desvio de obstáculos. Estes algoritmos são responsáveis por analisar o mapa de grelha de ocupação, identificar os obstáculos presentes e calcular rotas alternativas que evitem colisões [88]. Dessa forma, o robô pode manobrar seguindo uma trajetória que otimiza tanto a evasão de obstáculos como a eficiência no percurso. As variáveis de atuação resultantes dessa trajetória são essenciais para que o robô possa seguir a rota planeada. Elas determinam as ações a serem tomadas nos motores e nos sistemas de controlo, ajustando a velocidade, direção e outros parâmetros para que o robô siga o trajeto desviando-se de obstáculos.

4.2 Simulação

Neste estudo, realizamos uma investigação abrangente através da simulação no ambiente Gazebo, focada na avaliação e comparação dos diferentes controladores de navegação apresentados. Os resultados das simulações foram gravados permitindo uma análise detalhada sobre os dados de navegação recolhidos. Para fornecer uma visão mais clara e elucidativa sobre o desempenho dos diversos controladores apresentamos os resultados de forma visual de modo a realçar as diferenças entre cada abordagem de controlo. Num esforço para avaliar a eficácia dos controladores realizaram-se testes onde a trajetória executada pelo robô foi configurada como um quadrado de 8 por 8 unidades. Essa escolha desafiadora de trajetória permitiu examinar a capacidade do controlador em manter o alinhamento do robô com a rota desejada. Esses testes proporcionam informações cruciais sobre como o controlador responde a mudanças de direção e ajusta suas variáveis para alcançar navegação precisa.

A Figura 101 demonstra que todos os algoritmos funcionam bem na navegação realizada no Gazebo, como podemos observar, elas realizam um quadrado quase perfeito em qualquer um dos controladores.

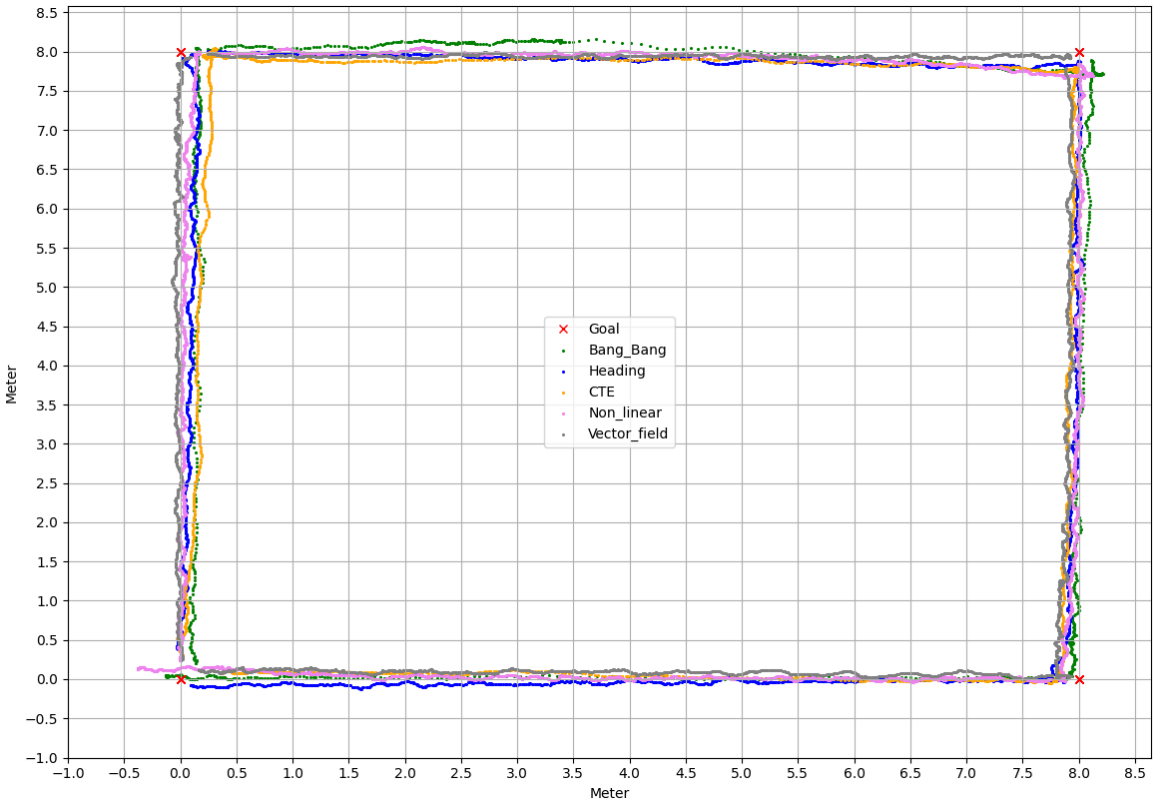


Figura 101 – Simulação de todos os algoritmos da navegação do quadrado.

As seguintes Figuras (102 até 106) representam exatamente o que representa a Figura 101, mas colocou-se para ser mais percetível cada controlador isoladamente.

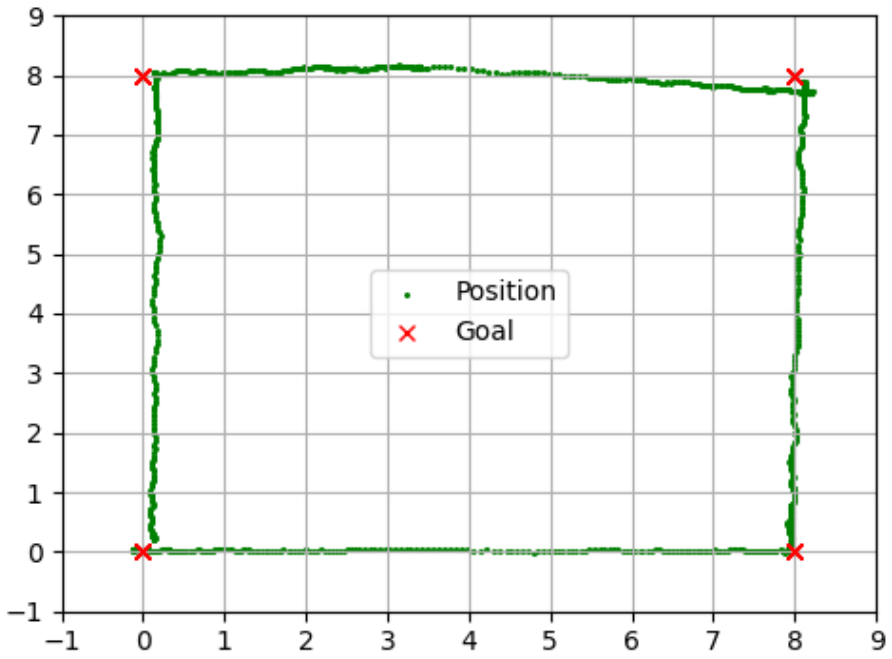


Figura 102 - Realização do quadrado com o controlador Bang-Bang em ambiente de simulação.

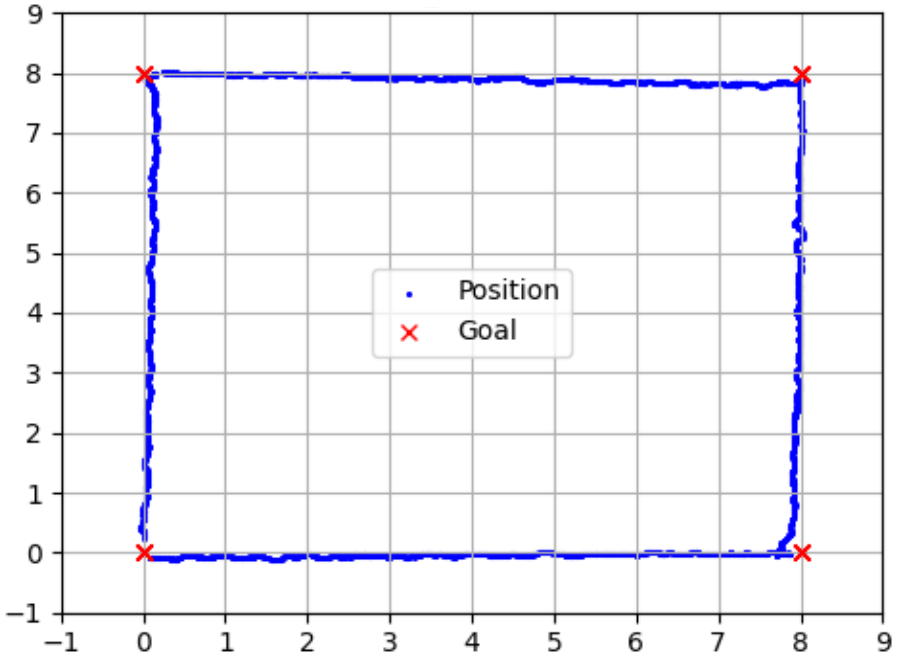


Figura 103 - Realização do quadrado com o controlador PIDH em ambiente de simulação.

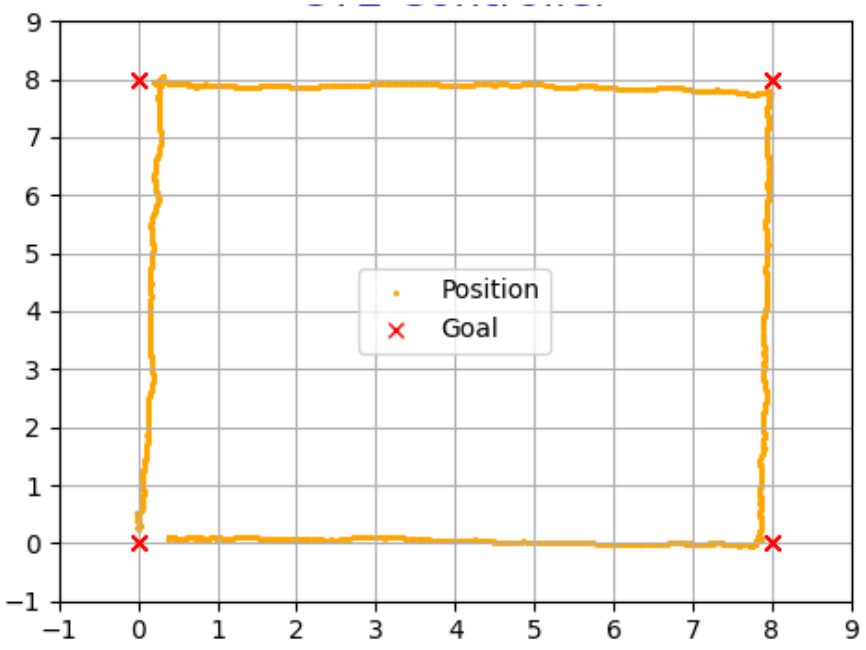


Figura 104 - Realização do quadrado com o controlador PID CTE em ambiente de simulação.

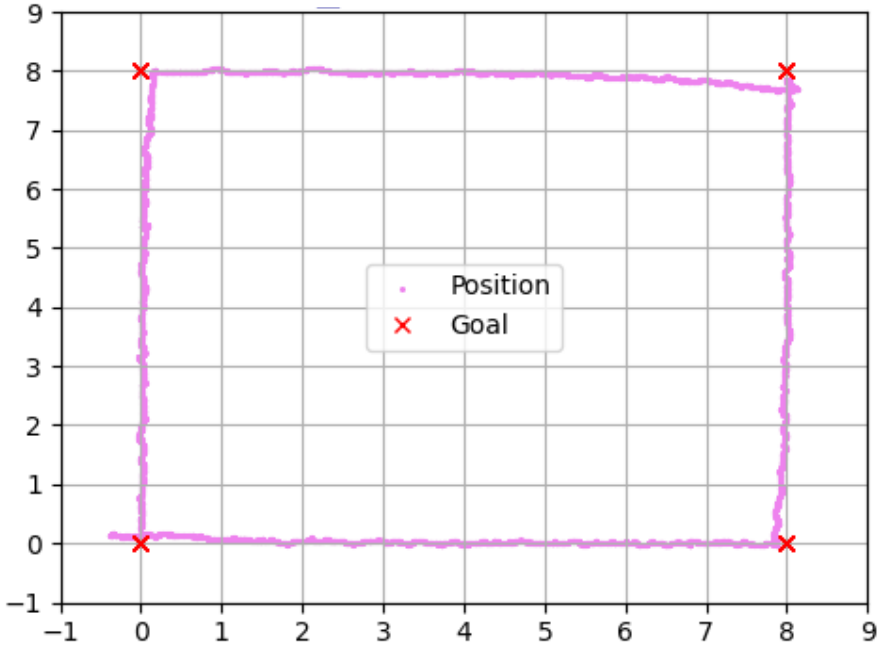


Figura 105 - Realização do quadrado com o controlador NL em ambiente de simulação.

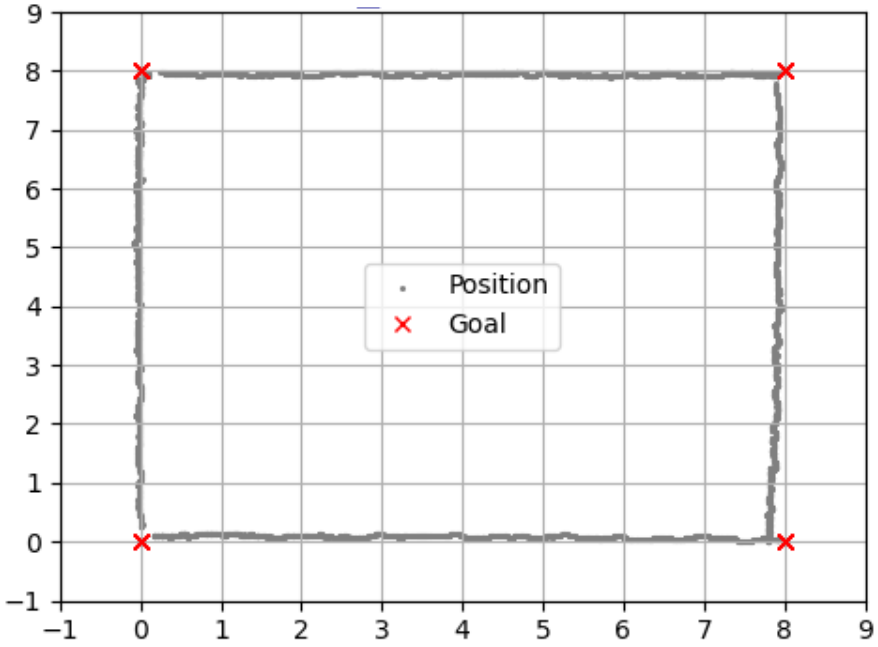


Figura 106 - Realização do quadrado com o controlador VF em ambiente de simulação.

Da Figura 102 até à Figura 106 está demonstrado os caminhos realizados pelo sistema robótico. Podemos observar que chegou aos destinos propostos com sucesso. Devido a este teste ser realizado em simulação, significa que o ambiente é um ambiente estruturado e perfeito, o que torna a navegação com valores muito ideais. Na Tabela 12 podemos observar associado a cada controlador, valores de erro interessantes, sendo eles quase 0. Através desta tabela, mesmo que os valores sejam quase ótimos. Como forma de seleção foram também calculados os erros IAE, ITAE e ISE para ver qual controlador PID com melhor desempenho. Os gráficos do ganho pode ser vistos em anexo.

Tabela 12 – Comparação dos diferentes algoritmos, erro e tempo de navegação em simulação.

Algoritmo	ITAE	IAE	ISE	Média [m]	Desvio padrão [m]	Máximo [m]	Tempo de navegação [s]
Bang-Bang	46,4922	3,2179	0,0357	0,08	0,06	0,22	115,80
Heading	67,7122	8,2875	0,1885	0,03	0,03	0,16	166,70
CTE	32,9667	5,5367	0,1202	0,04	0,03	0,15	92,05
Vector-Field	43,4623	8,6495	0,3299	0,05	0,04	0,14	188,60
Non-Linear	26,4528	4,5833	0,0816	0,05	0,04	0,18	226,56

Da Figura 107 até à 110 podemos observar passo a passo o robô a chegar ao destino pretendido.

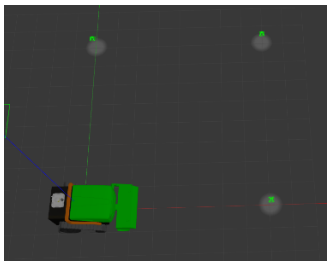


Figura 107 - Posição Inicial.

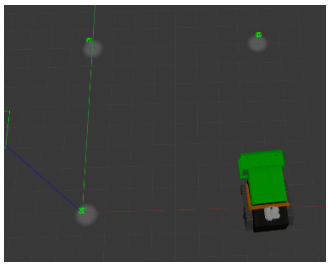


Figura 108 - 1ª Coordenada.

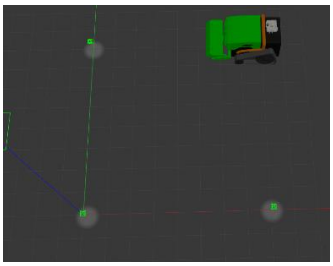


Figura 109 - 2ª Coordenada.

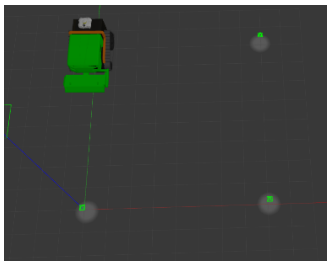


Figura 110 - 3ª Coordenada.

Na Figura 111, é possível observar o êxito do robô ao alcançar os destinos correspondentes. Essa eficácia foi observada em todos os controladores utilizados.

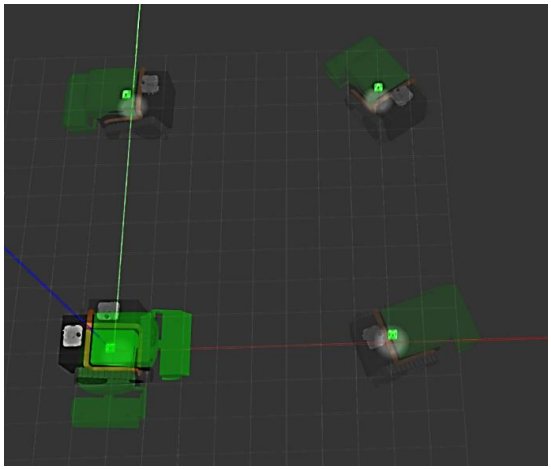


Figura 111 - Simulação em Gazebo.

Para a realização dos cálculos do ganho do PID procedeu-se ao método Cohen Coon malha-aberta. Ou seja, para cada controlador foram realizados dois testes, um para a componente angular e outra para a componente linear. Onde os valores cálculados foram os apresentados na Tabela 13.

Tabela 13 – Parâmetros dos diversos PIDs

Controlador	PID Componente Linear			PID Componente Angular			PID Orientação		
	Kp	Ki	Kd	Kp	Ki	Kd	Kp	Ki	Kd
PIDH	0,642	0,071	0,90	0,101	0,0054	0,0040			
PID CTE	1,882	0,607	0,83	0,084	0,0295	0,0376			
PID NL	0,298	0,021	0,69	0,297	0,411	0,0546	0,155	0,0787	0,0469
PIDH (VF)	Utiliza o S*			0,385	0,1026	0,0211			

Como forma de seleção foram calculados os erros IAE, ITAE e ISE para ver qual controlador PID com melhor desempenho. Os gráficos do ganho podem ser visualizados em anexo.

4.3 Real

Após as experiências de navegação autónoma em ambiente virtual passamos para um cenário real, colocamos o trator num terreno a fim de investigar as diferenças entre os controladores apresentados, nestas experiências práticas foram recolhidos dados da navegação em tempo real. O robô foi colocado num terreno semelhante ao simulado para que seja semelhante à simulação. Para visualizar os dados das experiências realizadas para cada sistema de controlo, efetuamos uma apresentação visual do mesmo para proporcionar uma perspetiva clara e comparativa do desempenho de cada controlador, para um ambiente real. Esta abordagem prática visa fornecer conhecimentos mais práticos e valiosos sobre o comportamento e a eficácia dos controladores de navegação no mundo real. Podemos observar todos os testes realizados na Figura 112, como podemos ver:

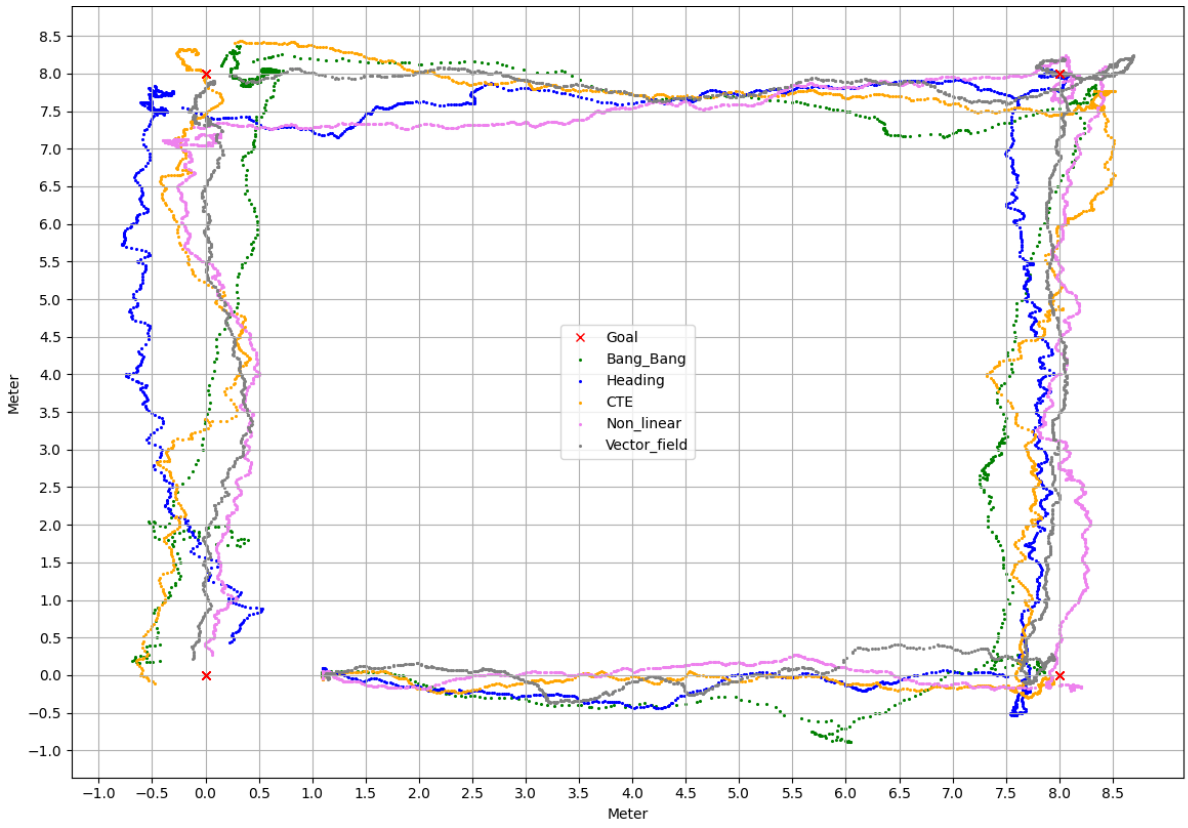


Figura 112 - Testes realizados em ambiente real juntos.

Após analisarmos minuciosamente os resultados obtidos nos testes realizados no terreno, optámos por realizar testes adicionais para identificar de forma mais precisa quais os métodos mais promissores para serem incorporados num algoritmo de desvio de obstáculos durante a navegação. Através desta análise cuidadosa, procuramos selecionar as abordagens que demonstrem consistência e eficácia nas mais diversas situações.

Tabela 14 – Parâmetros dos Ganhos dos PIDs.

Controlador	PID Componente Linear			PID Componente Angular			PID Orientação		
	Kp	Ki	Kd	Kp	Kp	Ki	Kd	Ki	Kp
PIDH	0,1	0,005	0,05	0,02	0,0	0,0			
PID CTE	0,25	0,0	0,0	0,25	0,0	0,0			
PID NL	0,08	0,0	0,0	0,30	0,0	0,0	0,00833	0,0	0,0
PIDH (VF)	Utiliza o S*			0,01	0,00005	0,0001			

A tabela 14, mostra os valores dos ganhos usados nestas experiências, estes que foram todos testados, *trial-error*, até obtermos os valores que achamos que melhor se adaptavam ao trator em questão, onde se utilizou os valores calculados na simulação como condição inicial ajustando os valores dos parâmetros ao ambiente real.

Da Figura 113 até à 117 apresentam uma representação visual dos diversos percursos realizados nos diferentes tipos de controlo. Esta visualização é de extrema importância para compreender qual a abordagem que melhor se adapta às condições específicas de navegação e para fundamentar a decisão de incorporar os resultados nos futuros algoritmos de desvio de obstáculos.

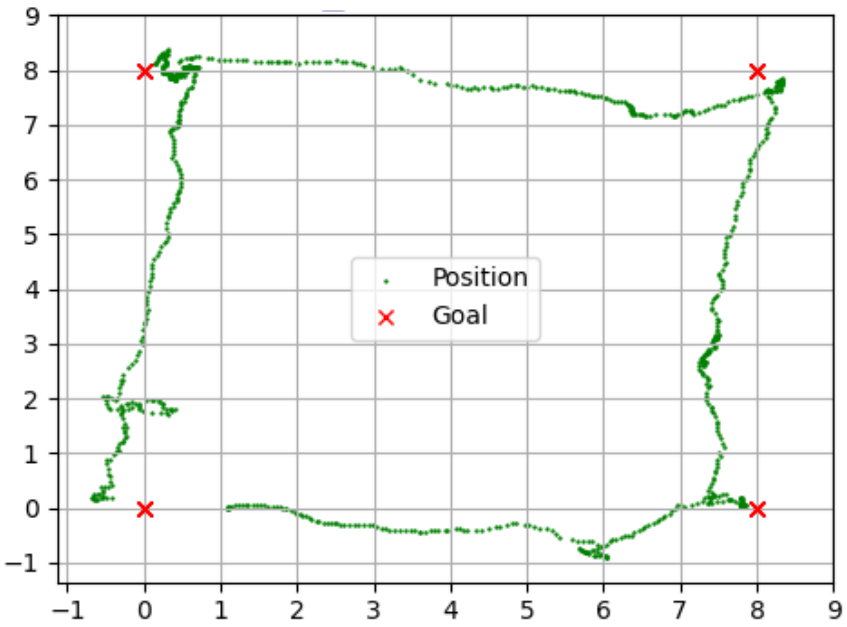


Figura 113 - Realização do quadrado com o controlador Bang-Bang em ambiente real.

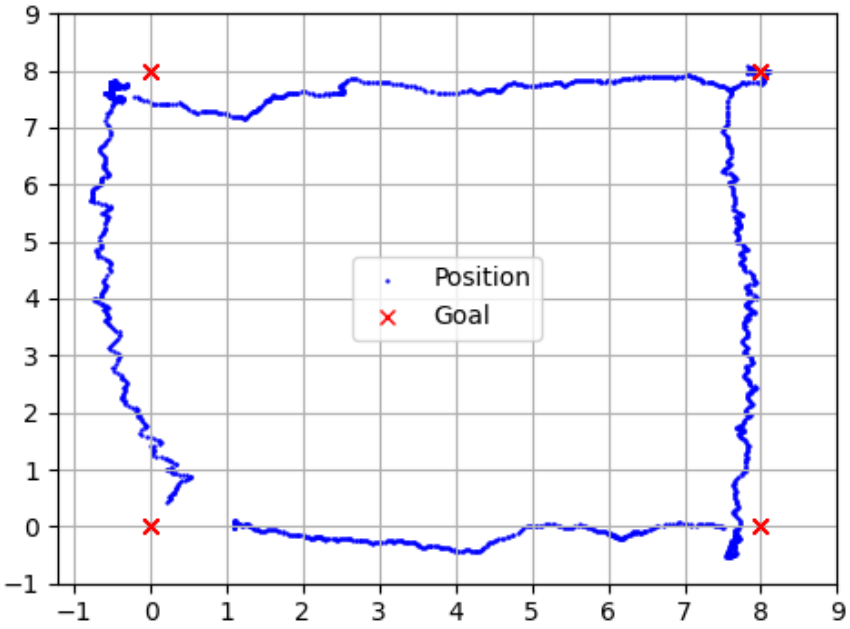


Figura 114 - Realização do quadrado com o controlador PIDH em ambiente real.

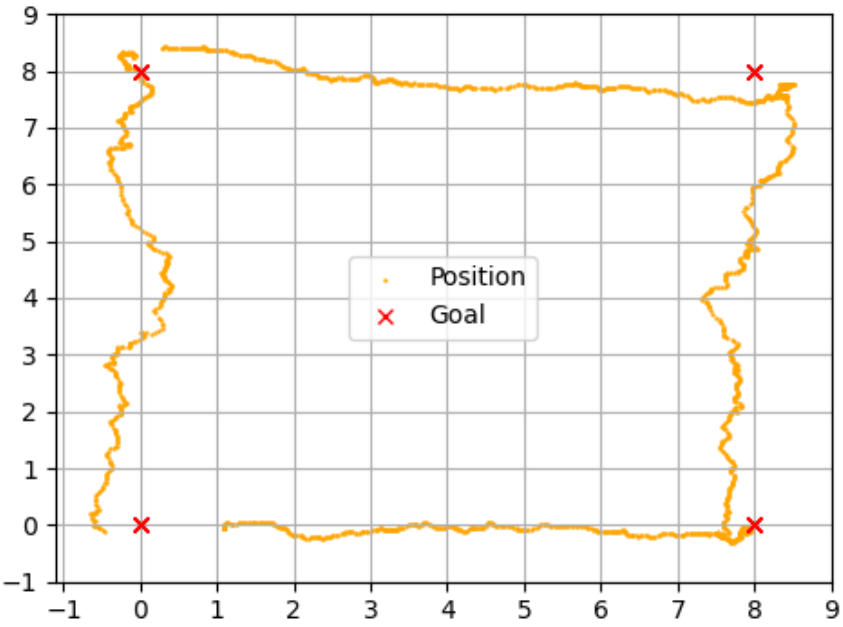


Figura 115 - Realização do quadrado com o controlador PID CTE em ambiente real.

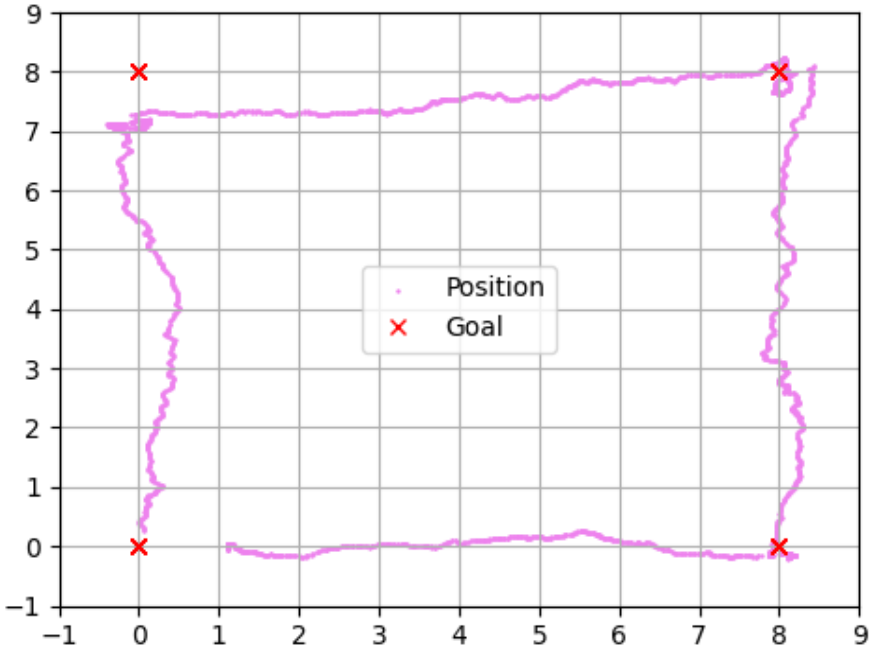


Figura 116 - Realização do quadrado com o controlador NL em ambiente real.

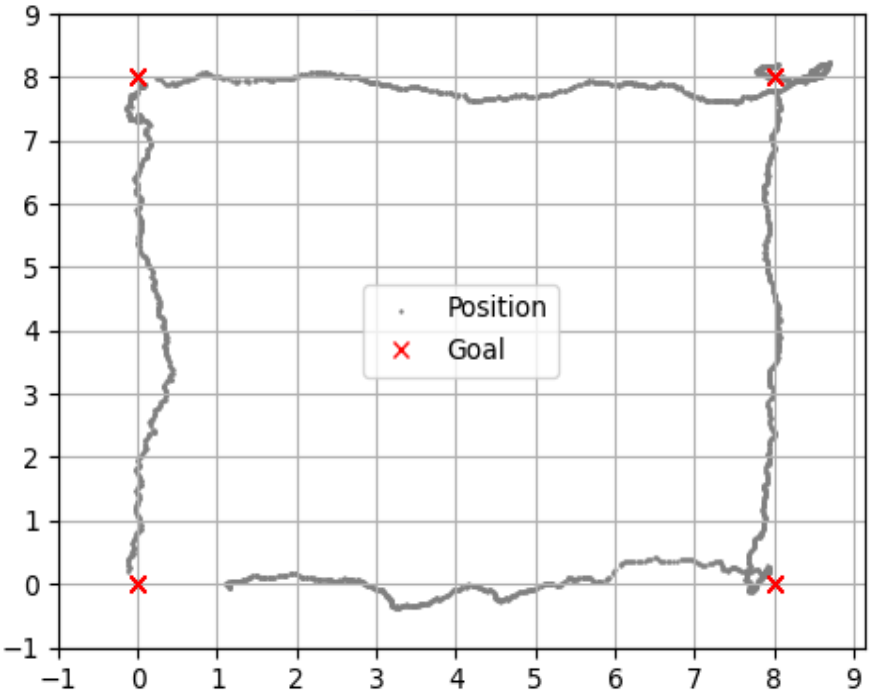


Figura 117 - Realização do quadrado com o controlador VF em ambiente real.

Podemos observar que chegou aos destinos propostos com sucesso, no entanto como este ambiente não ideal e possui perturbações é notório que o trajeto não se encontra tão perfeito como o de simulação.

Para efetuar uma comparação a todos os algoritmos foram repetidas as mesmas trajetórias, no nosso caso consideramos o movimento em quadrado, onde o robô passa pelas coordenadas [(8.0, 0.0) ;(8.0, 8.0) ;(0.0, 8.0) ;(0.0, 0.0)] respetivamente. Todos os testes começaram com o robô direcionado para ESTE previamente antes da navegação. Em todos os testes o robô está com a tecnologia GPS+RTK, dando lhe assim uma precisão até 2 cm. Para todos os testes de navegação foi considerado haver um *offset* de 0,2 metros até chegar ao destino. Para cada um destes testes foi feito uma análise do erro da posição do robô relativamente à trajetória que este deve de seguir, ou seja, foram calculados os erros médio, máximo e o desvio padrão para cada uma das situações. Nos gráficos, é percetível uma translação de posição de 1,1 metros. Isso significa que a localização do robô, em vez de estar no ponto "Sentry", onde está presente o GPS, passa a situar-se no centro de rotação do robô, afetando as trajetórias observadas, visto que a posição não começa em início do ponto (0,0), mas sim no ponto (1,0).

Como observado na Tabela 15, a maneira que estes dados foram calculados foram através de um método utilizado no “pandas”, esta que é uma biblioteca do Python própria para manipulação e análise de dados, ou seja, foram usadas três funções, a função `.mean()` é utilizada para calcular o valor médio dos dados, a função `.std()` para calcular o desvio padrão, medida de dispersão ou variabilidade de modo a quantificar o afastamento da média, ou seja um desvio padrão mais elevado indica uma maior variabilidade ou dispersão nos dados, enquanto um desvio padrão mais reduzido indica ma menor variabilidade. Por último, a função `.max()`para encontrar o máximo valor. Como forma de seleção foram também calculados os erros IAE, ITAE e ISE para ver qual controlador PID com melhor desempenho.

Tabela 15 - Comparação dos diferentes algoritmos, erros e tempo de navegação em ambiente real.

Algoritmo	ITAE	IAE	ISE	Média [m]	Desvio padrão [m]	Máximo [m]	Tempo de navegação [s]
Bang-Bang	2815,625	118,19	46,844	0,33	0,19	0,89	103,60
Heading	1695,307	138,11	44,1836	0,21	0,18	0,85	88,90
CTE	1541,619	131,13	46,9071	0,18	0,14	0,65	86,45
Non-Linear	1257,305	95,303	19,6773	0,21	0,19	0,74	93,60
Vector-Field	520,9629	66,989	11,8488	0,18	0,17	0,66	82,46

Podemos observar que os algoritmos que apresentam melhores resultados são o CTE, o NLC e o VFC. No entanto, no caso do CTE não é o mais indicado para o desvio de obstáculos uma vez que, num percurso não retilíneo (como no caso da presença de um obstáculo), a distância requer tempo para aumentar e torna-se completamente diferente da direção que aponta para a referência seguinte, como demonstra a Figura 118. Isso indica a necessidade de uma rotação forte. É possível reverter esta situação

ao aumentar os ganhos do PID, no entanto, para existir um controlo que realize este comportamento poderá tornar o controlador instável.

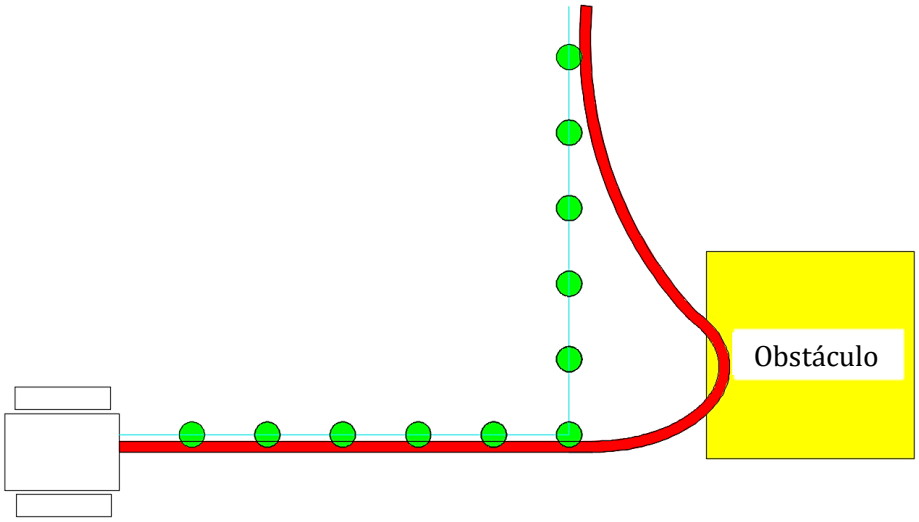


Figura 118 - Comportamento do controlador CTE no Desvio de Obstáculos.

Devido a esta característica do controlador CTE prosseguiu-se para a experiência com obstáculos com os algoritmos VFC e o NLC. Da Figura 119 à Figura 122 podemos observar o trator a realizar o seu percurso do quadrado onde cada imagem representa cada vértice do mesmo.

Figura 119 - Posição Inicial.

Figura 120 - 1ª Coordenada.

Figura 121 - 2ª Coordenada.

Figura 122 - 3ª Coordenada.

A Figura 123 é uma representação de todos os trajetos realizados no terreno na realização dos quadrados de 8 por 8 metros.

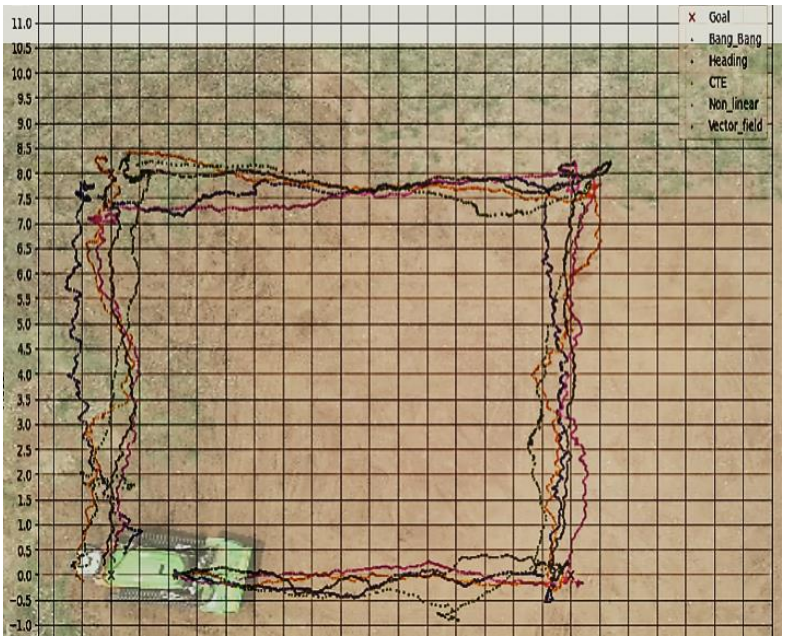


Figura 123 - Representação de todos os caminhos de todos os controladores.

4.4 Comparação dos controladores

Como foi referido anteriormente, os controladores que foram escolhidos são o controlador NL e o controlador VF. Neste tópico iremos realizar uma comparação entre estes de forma a podermos escolher um vencedor. Para isto, em vez de realizar-mos uma trajetória em formato de um quadrado de oito por oito, foi feita uma experiência de um caminho possível para um desbastamento real. Foram definidos os seguintes pontos por onde o robô os alcançou respetivamente [(0; 0), (10; 0), (10; 2), (0; 2), (0; 4), (10; 4), (10; 6), (0; 6) e (0; 0)]. A Figura 124 mostra a junção dos dados de cada controlo retirados nesta trajetória.

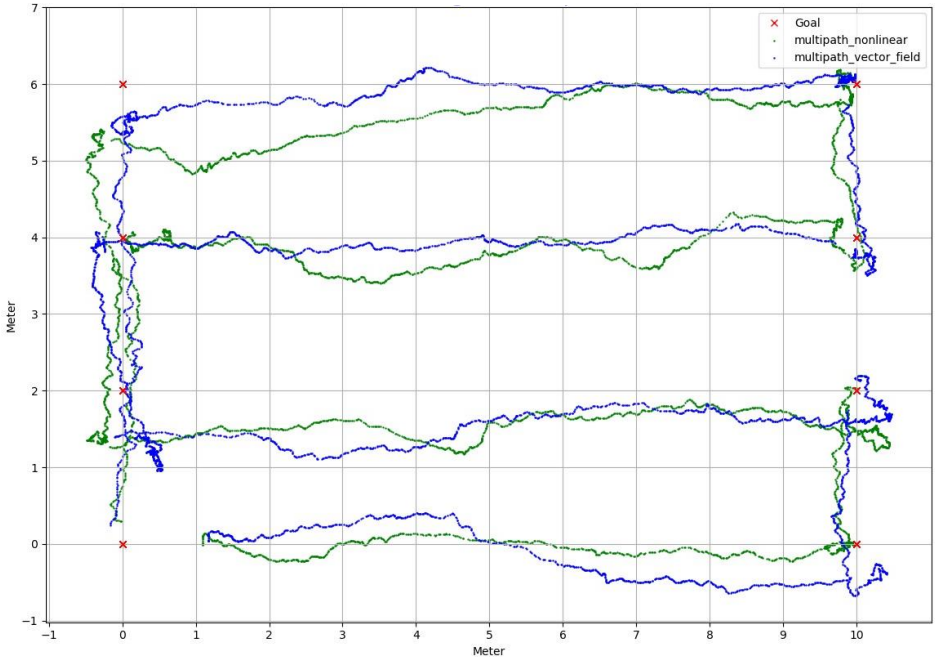


Figura 124 - Junção das trajetórias de ambos os controladores (NL e VF).

Como podemos observar, todos os pontos foram alcançados. Para uma melhor visualização foram colocados estes trajetos separadamente, como mostra a Figura 125.

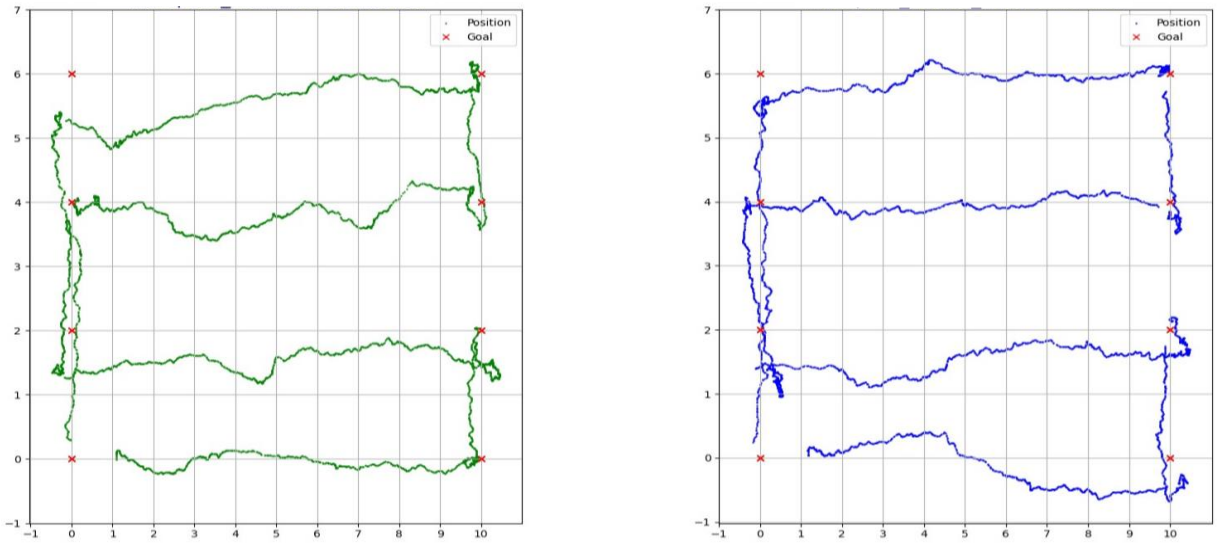


Figura 125 - Trajetórias de cada controlador (NL e VF) respetivamente.

Após termos os dados destas trajetórias, é possível calcular os erros pretendidos de forma a tirar a conclusão de qual o melhor entre os dois algoritmos como podemos observar na Tabela 16.

Tabela 16 - Erros e Tempo de navegação dos controladores NL e VF.

Algoritmo	Média [m]	Desvio padrão [m]	Máximo [m]	Tempo de navegação [s]
NL	0,28	0,25	1,16	175,03
VF	0,25	0,21	0,93	185,5

Com estes resultados, apresar do controlador VF ter um tempo de navegação de aproximadamente mais 10 segundos que o controlador NL, todos os erros envolventes na navegação são menores. Desta forma conclui-se que o controlador VF é o controlador que menos erro gera nesta navegação.

4.5 Validação dos algoritmos de desvios de obstáculos

Para validar os algoritmos foram desenvolvidos scripts em Python de forma a realizar diversos testes práticos para validar a eficácia dos algoritmos de desvio de obstáculos. Estes testes envolveram simulações em vários cenários, nos quais os algoritmos foram submetidos a diferentes situações de deteção e evasão de obstáculos.

Os resultados obtidos nestes testes foram analisados e comparados com as expectativas teóricas, proporcionando uma avaliação abrangente do desempenho dos algoritmos em situações reais. Estes testes práticos trouxeram contribuições valiosas para aperfeiçoar e otimizar os algoritmos.

4.5.1 Algoritmo A*

A Figura 126 é usada como representação visual onde se destaca uma grelha composta por quadrados pretos, cada um correspondendo a um obstáculo, além de apresentar um ponto de partida (*start*), um ponto de destino (*goal*), uma linha de referência (*reference line*) e a trajetória que o algoritmo *A** percorre, habilmente desviando-se dos obstáculos ao longo do caminho. O notável aspeto aqui é que o código do algoritmo implementa obstáculos de forma aleatória em cada execução, simulando uma variedade de cenários possíveis. Essa imagem ilustrativa, que confirma o funcionamento preciso do Algoritmo *A**, é um testemunho visual da sua confiabilidade. Ao observarmos como o algoritmo é capaz de mapear uma rota segura e eficiente, contornando obstáculos de maneira consistente, independentemente das variações introduzidas a cada execução, reforça-se a convicção de que o algoritmo opera de acordo com as expectativas.

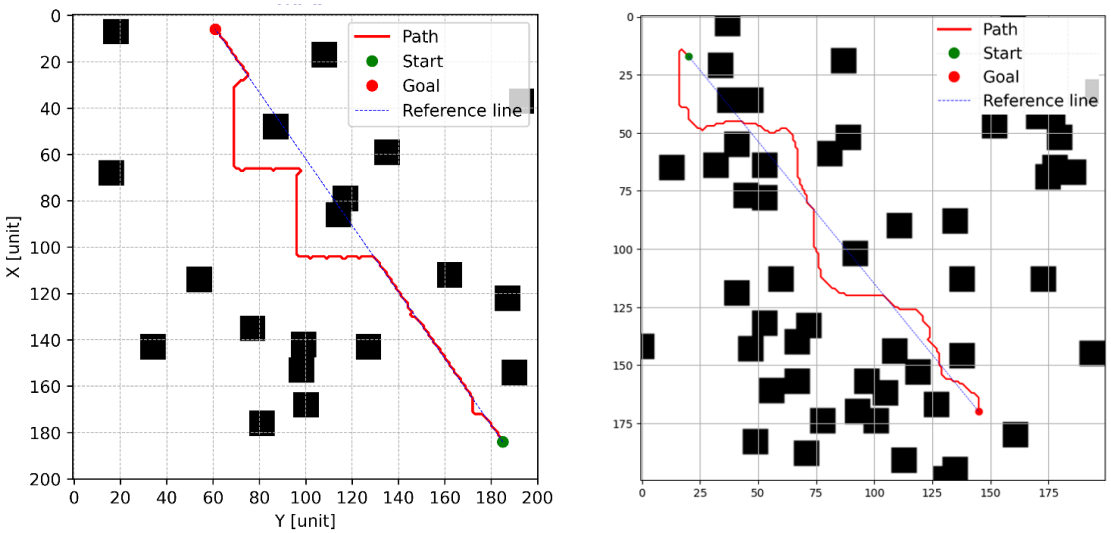


Figura 126 – Validação do algoritmo A*.

4.5.2 Algoritmo VF

Na Figura 127, é possível observar que cada gráfico apresenta elementos essenciais tais como um ponto de "partida" (start), um ponto de "destino" (*goal*), uma "trajetória" (*path*) e um obstáculo composto por quatro pontos, acompanhado de um campo vetorial repulsivo. A escolha de quatro pontos para representar o obstáculo baseia-se no facto de este não se limitar a uma única posição pontual. Para garantir uma abordagem abrangente, foram adicionados pontos ao redor do obstáculo, formando um campo repulsivo que abrange todas as áreas da sua extensão. Durante os testes

realizados, emergiu uma observação importante, pois a variação angular das forças repulsivas dos obstáculos tem um impacto significativo na eficácia da trajetória resultante. Na Figura 127, é percetível que um ângulo de inclinação excessivo resulta numa soma vetorial que induz um trajeto em *loop*. Especificamente, quando o ângulo atinge 90 graus, a trajetória entra num ciclo repetitivo, no qual a presença de forças vetoriais insuficientes para interromper o ciclo dificulta a orientação do robô em direção ao destino. Esta análise enfatiza a importância de ajustar cuidadosamente os ângulos das forças repulsivas dos obstáculos, a fim de garantir um comportamento de navegação coeso e eficiente, evitando armadilhas indesejadas como os *loops* mencionados.

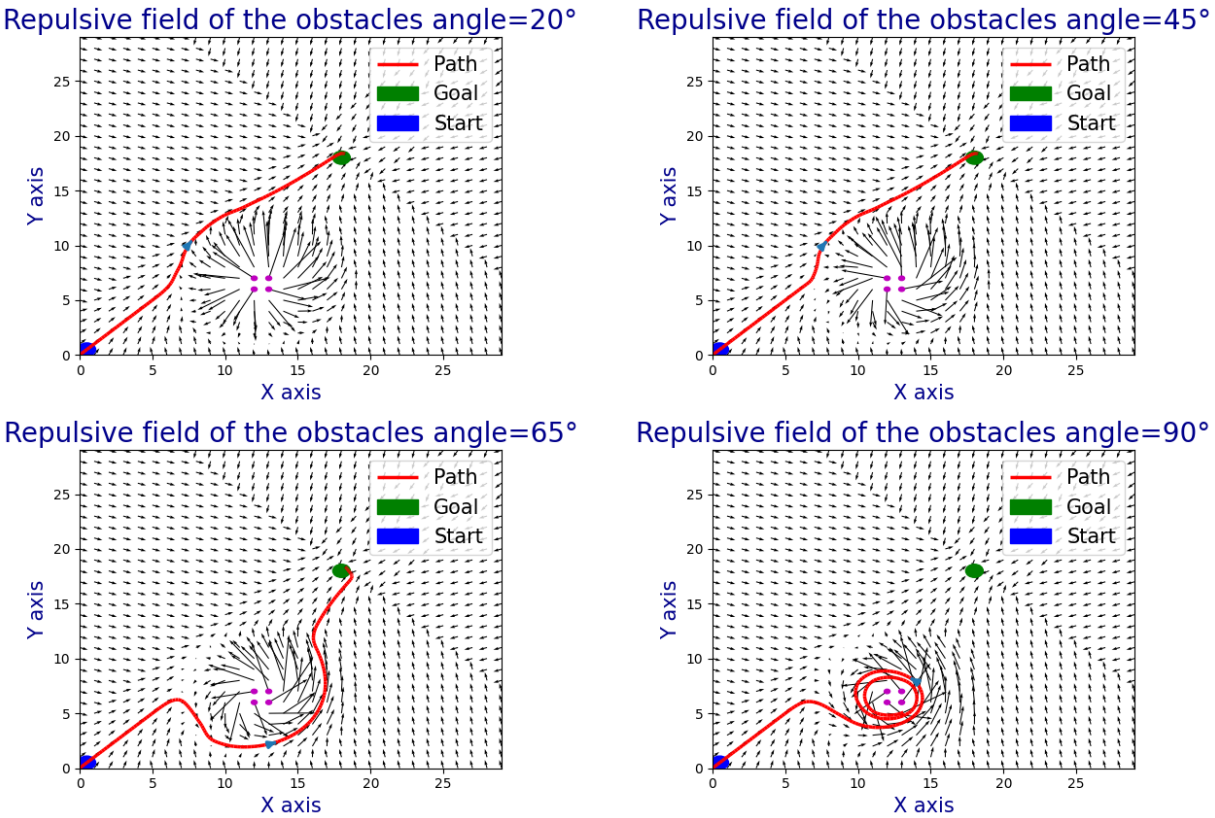


Figura 127 – Validação do algoritmo Vector Field com um obstáculo.

Na Figura 127, procedemos à validação do algoritmo na presença de um obstáculo, resultando numa observação de importância: quando o ângulo de inclinação das forças é excessivamente alto, o robô pode entrar num estado de "*loopTrap*" (armadilha em *loop*). Assim, ao considerar a existência de um obstáculo, identificamos que os ângulos ideais de inclinação das forças vetoriais são 20°, 45° e 65°. Esta constatação motivou a realização de um novo teste, no qual adicionou-se mais um obstáculo ao campo vetorial, Figura 128. Nesse cenário, constatámos que um ângulo de inclinação excessivamente pequeno resulta na anulação das forças, levando a que a trajetória termine antes de atingir o destino, uma situação designada como "*wallTrap*" (armadilha de parede). A introdução de novas forças vetoriais permitiu-nos constatar que os ângulos de inclinação de 20° e 45° não são suficientes, enquanto os de 65° e

90° revelaram-se ideais para este contexto específico. Dado que o ângulo de 65° demonstrou ser eficaz em ambos os cenários, foi selecionado como parâmetro padrão para todos os testes subsequentes. Essa adaptação visa otimizar o desempenho do algoritmo ao lidar com diversas situações, proporcionando uma abordagem coesa e confiável.

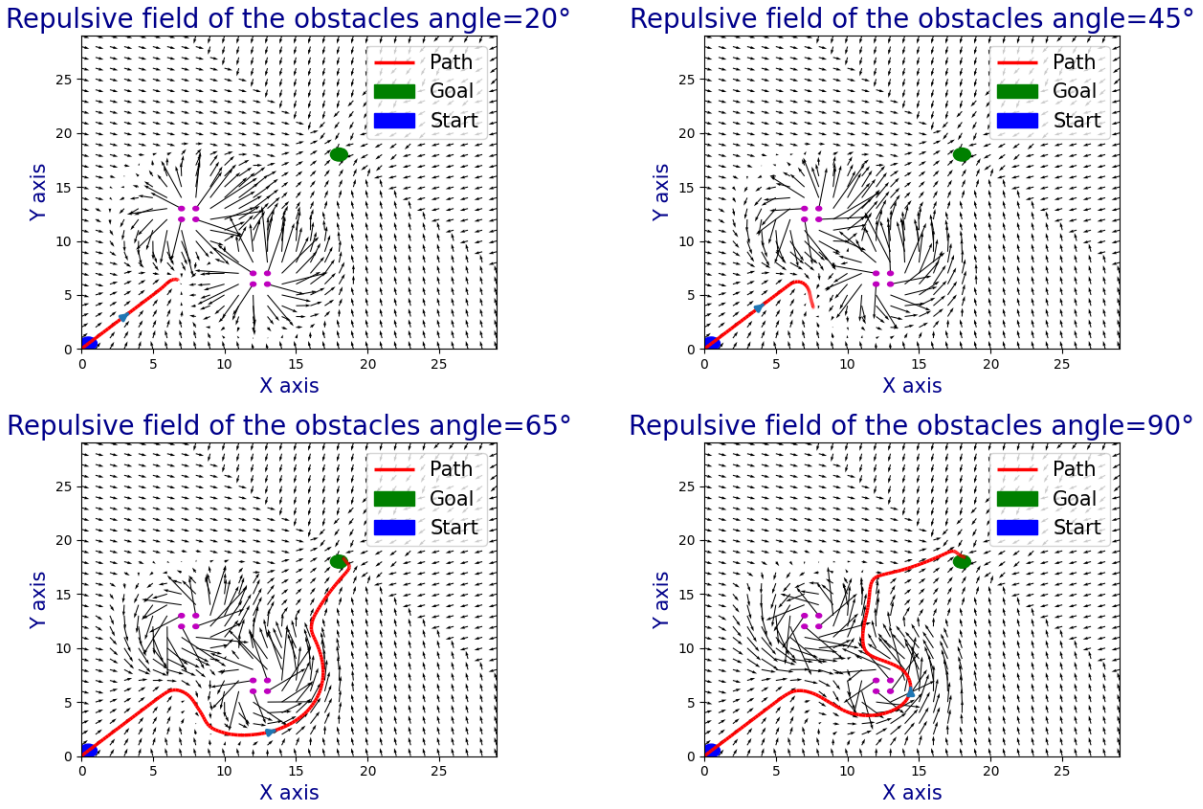


Figura 128 - Validação do algoritmo Vector Field com dois obstáculos.

Após a realização desses testes, estabeleceu-se a confiabilidade do algoritmo, o que levou à decisão de implementá-lo tanto em simulações como em situações reais.

4.6 Comparação dos algoritmos de desvio de obstáculos

4.6.1 Simulação

As Figuras 129 e 130 retratam os resultados derivados da simulação da posição do robô quando submetido à execução dos algoritmos de desvio de obstáculos. Na Figura 129, o algoritmo *A** primeiramente calculou a trajetória inicial do robô. Isso ocorreu devido à deteção de um obstáculo pelo LiDAR, cujas coordenadas máximas e mínimas foram registadas como [(8, 1), (8, -1), (10, 1), (10, -1)]. Consequentemente, o *GridMap* foi preenchido com áreas ocupadas, marcadas com o valor 1. Com essas regiões identificadas, o algoritmo iniciou sua operação, otimizando a trajetória do robô. Este processo foi continuamente recalculado durante a navegação.

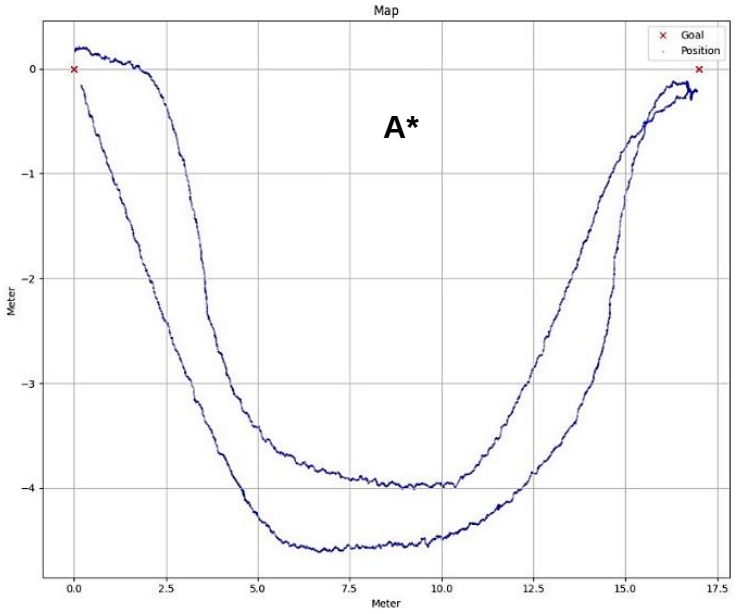


Figura 129 - Desvio de obstáculos com o algoritmo A*.

Já na Figura 130, deparamo-nos com a aplicação do algoritmo VF. Nas posições [(8, 1), (8, -1), (10, 1), (10, -1)], o LiDAR detetou os extremos do obstáculo. Essa deteção resultou na criação de vetores de força repulsiva associados ao obstáculo em questão. Inicialmente, as forças geradas pelo obstáculo não foram suficientes para provocar uma alteração na trajetória do robô. No entanto, à medida que o robô se aproximava do obstáculo, as forças repulsivas aumentavam significativamente, possibilitando uma soma vetorial que permitiu ao robô desviar-se e alcançar seu destino. Uma observação relevante é que essas forças repulsivas possuem uma natureza circular, com um ângulo de dispersão de 65º, conforme mencionado anteriormente. Isso indica que o robô realizou uma manobra de desvio por cima do obstáculo durante o contorno, conforme podemos observar na Figura 130.

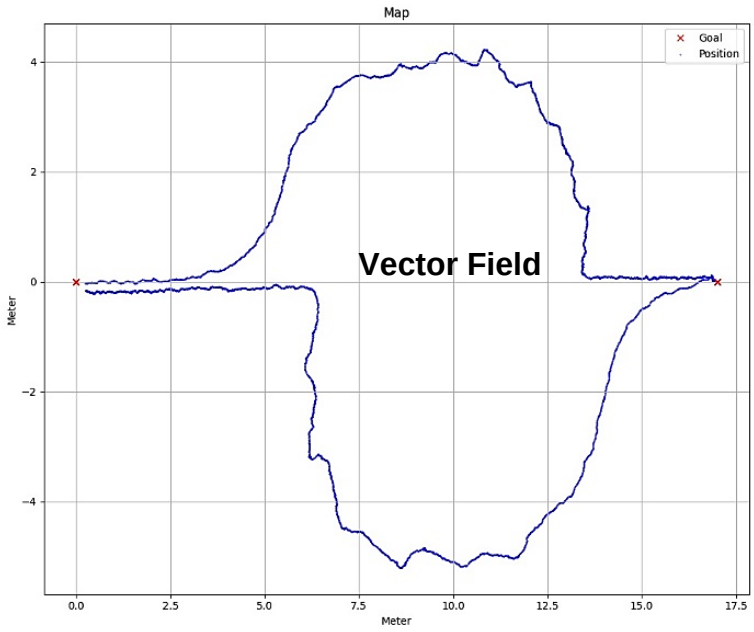


Figura 130 - Desvio de obstáculos com o algoritmo Vector Field.

Os resultados obtidos sugerem que as execuções desses algoritmos decorreram de maneira bem-sucedida. Tendo em vista essas conclusões promissoras, a decisão tomada foi de avançar para a próxima etapa, que envolve a implementação dos algoritmos em ambiente real. Na Figura 131 podemos observar que o *clustering* dos obstáculos está bem implementado, pois uma caixa e realizada á volta da *PointCloud* que o LiDAR detetou.

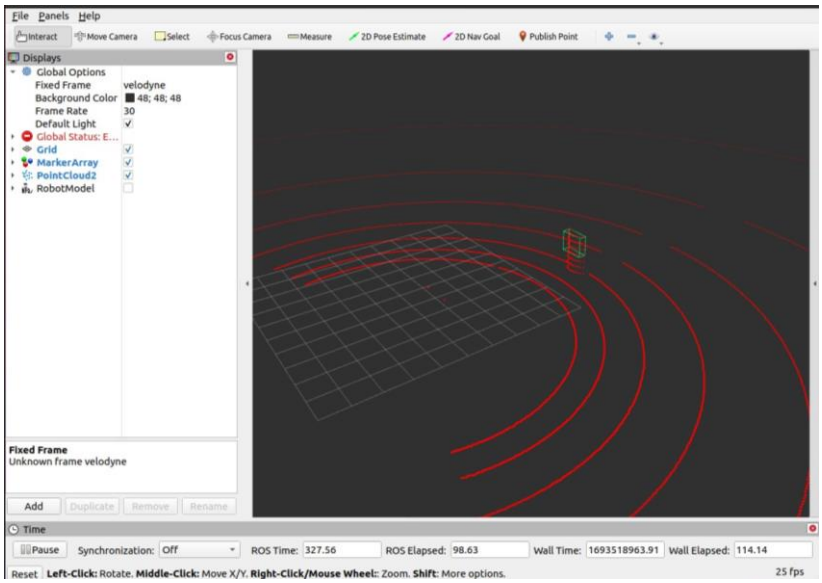


Figura 131 - Deteção do obstáculo com o clustering.

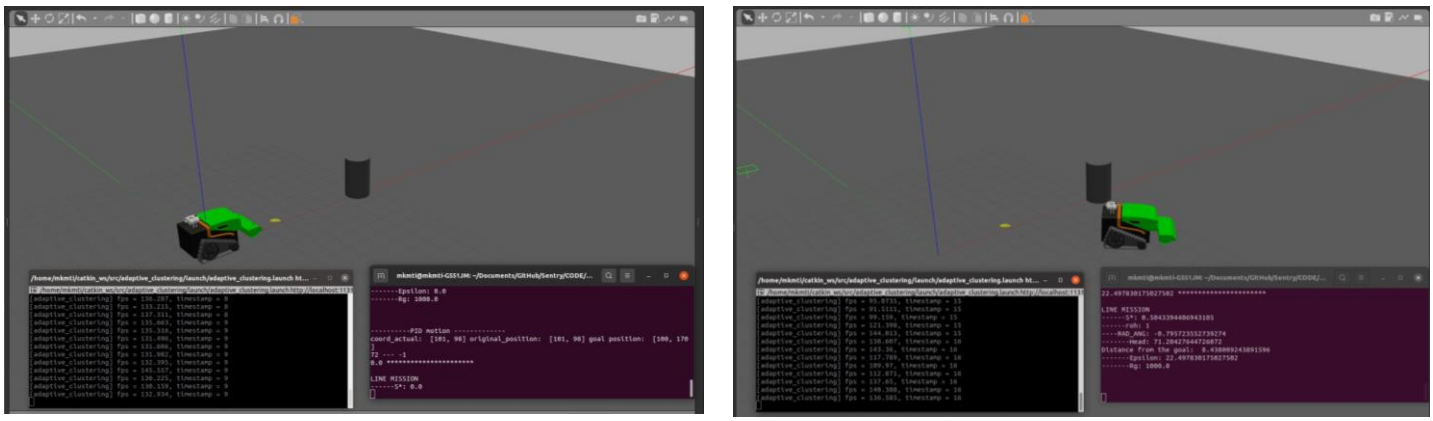


Figura 132 - Simulação do desvio do obstáculo em Gazebo.

Na Figura 132 podemos observar que como o robô já está a detetar o obstáculo, significa que já sabe a sua respetiva posição, desta forma é possível que ele utilize os algoritmos para contornar o obstáculo.

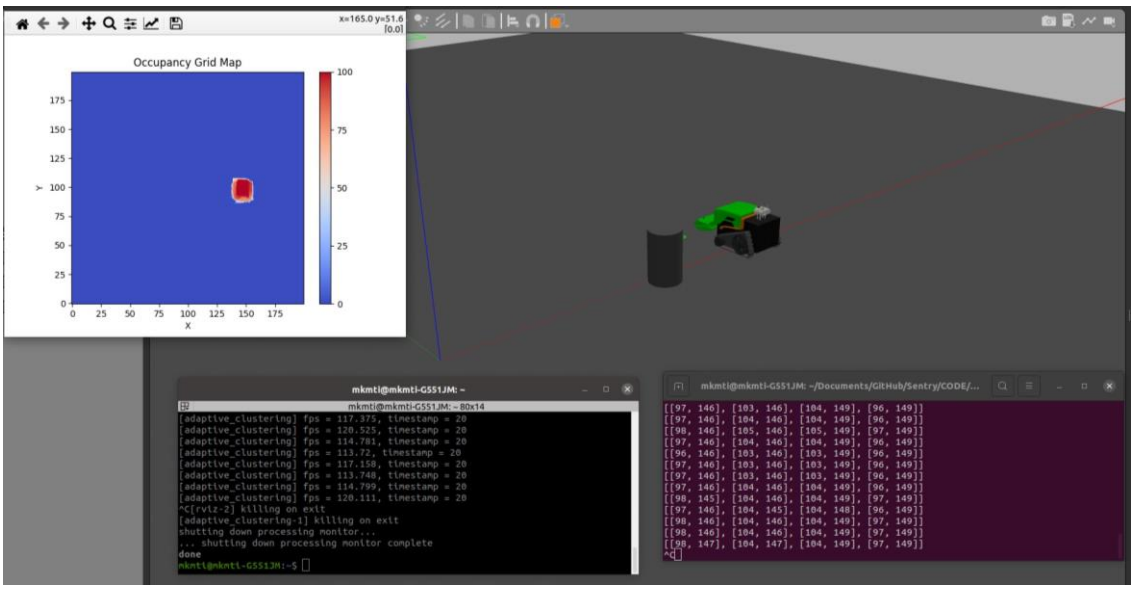


Figura 133 - Simulação Gazebo ao chegar ao objetivo.

A Figura 133, mostra que o robô chegou ao objetivo com sucesso desviando do obstáculo que estava presente no caminho. Finalizando este processo, é mostrado o *GridMap*, este que mostra a posição na sua grelha do objeto detetado.

4.6.2 Real

Na análise realizada no ambiente real, podemos observar nas Figuras 134 e 135 os resultados obtidos que proporcionam uma visão do que foi alcançado. É notório a conformidade entre os testes práticos e as simulações prévias, o que evidencia a eficácia dos algoritmos em situações reais de navegação mesmo com obstáculos. Relativamente à posição estratégica do obstáculo, esta foi selecionada considerando uma caixa com dimensões semelhantes ao objeto utilizado nas simulações. Antes do início da navegação, procedeu-se à medição da localização de forma a garantir a coerência e comparabilidade entre os resultados. Esta abordagem de confrontar os resultados práticos com os simulados contribui para uma compreensão da eficácia do algoritmo no desvio de obstáculos.

Algoritmo A*

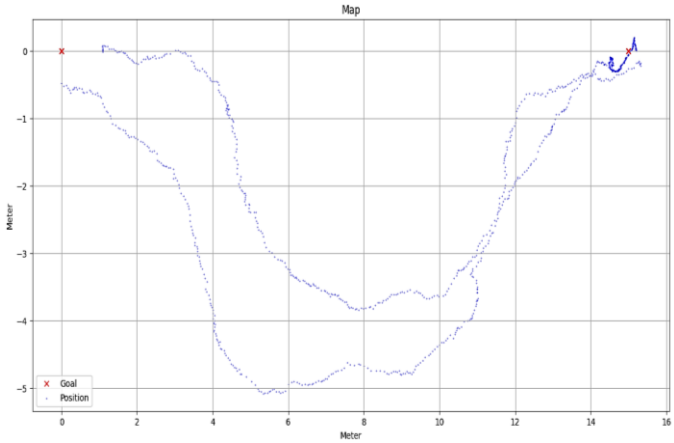


Figura 134 – Teste Real do desvio de obstáculos com o Algoritmo A*.

Algoritmo Vector Field

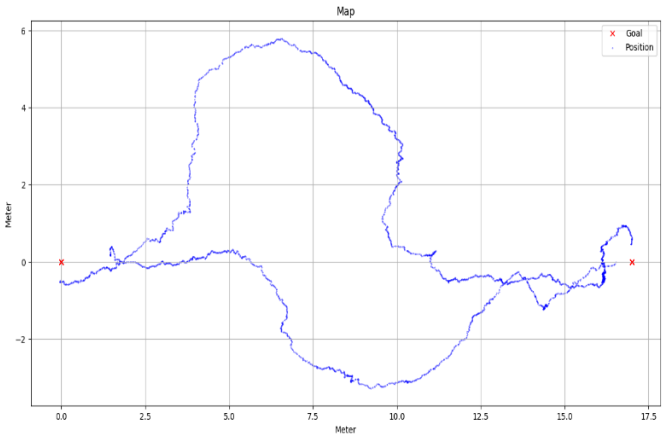


Figura 135 - Teste Real do desvio de obstáculos com o Algoritmo VF.

Na representação visual apresentada na Figura 136, é possível verificar com mais detalhe os testes realizados em ambiente real. As imagens destacam de forma tangível a aplicação do Algoritmo *A**, demonstrando como o robô consegue contornar obstáculos conforme previsto.

Figura 136 - Fotos do teste prático do controlador Non-Linear com o algoritmo A* para desvio de obstáculos.

Na representação gráfica da Figura 137, podemos observar uma combinação de imagens que proporcionam uma visualização do teste. Ao comparar as Figuras 136 e 137, é evidente que as imagens ilustram como o trator seguiu o caminho delineado pelos dados obtidos. Esta análise visual permite validar de forma notória a correspondência entre a teoria e a aplicação, fortalecendo a confiança na precisão e eficácia do algoritmo implementado.

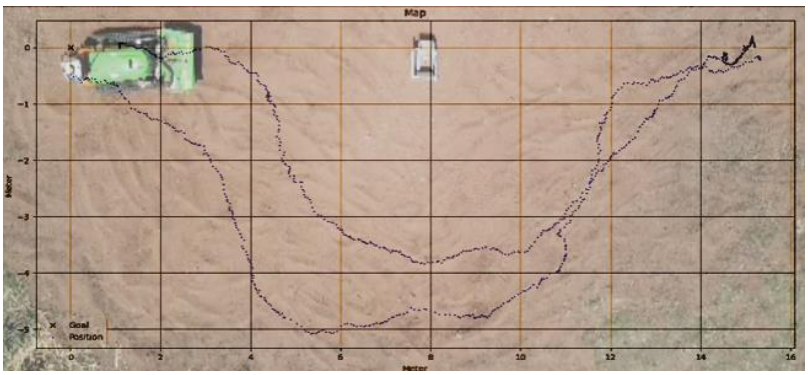


Figura 137 - Representação dos dados com as imagens reais.

Na ilustração visual na Figura 138, é viável analisar a progressão dos testes conduzidos no cenário real. As imagens conferem uma perceção do algoritmo VF, evidenciando a forma de como o robô consegue contornar obstáculos conforme o planeado.

Figura 138 – Experiências do controlador PID Heading (versão Vector Field) com o algoritmo Vector Field para desvio de obstáculos.

Na Figura 139, é possível observar uma junção de imagens que se relacionam com os valores recolhidos e a realização do teste prático. Ao efetuarmos uma breve comparação entre as Figuras 138 e 139, conseguimos constatar pelas figuras que o percurso efetuado pelo trator está em concordância com os dados obtidos validando a correspondência entre a trajetória planeada e executada, reforçando a fiabilidade do desempenho do algoritmo em questão.

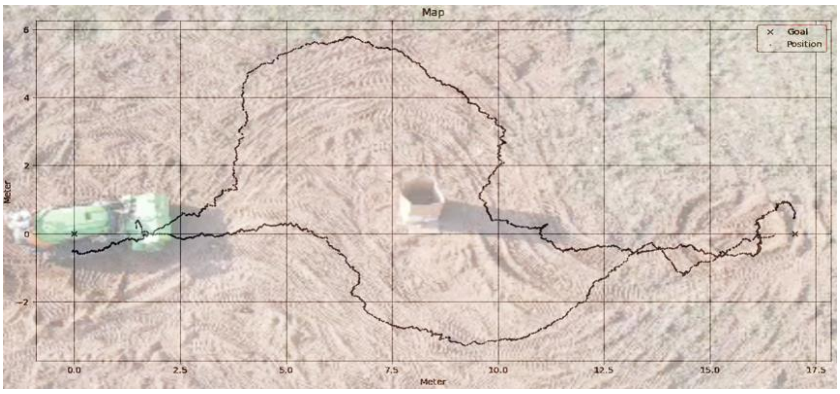


Figura 139 - Representação dos dados conjugado com as fotos reais.

Os resultados mostrados acima concluem que ambos os algoritmos fazem o que é suposto, o desvio de obstáculos. No entanto podemos referir que estes possam ser aprimorados, pois existem parâmetros que podem ser alterados de forma a influenciar a otimização da navegação. Dentro destes parâmetros, os pesos da função heurística presente no algoritmo *A** está presente, ao ajustarmos estes parâmetros torna-se possível reduzir o erro relativo á referência da trajetória mais rapidamente.

O Algoritmo *A** apresenta uma vantagem relativamente ao algoritmo VF, pois uma trajetória otimizada é calculada no inicio da navegação enquanto o Algoritmo VF apenas segue o campo vetorial a que está inserido e caso encontre alterações desvia-se ou aproxima-se de acordo com estas.

Um outro ponto a acrescentar relativamente ao algoritmo VF é que como o campo vetorial repulsivo têm a mesma orientação relativamente ao obstáculo, significa que o robô irá sempre se desviar no sentido desta orientação, mesmo que o percurso não seja o mais eficaz. Isto pode aumentar o tempo de navegação e até dificultar a própria navegação do robô. Como podemos observar na Figura 140 o VF percorre uma trajetória muito maior relativamente ao *A**.

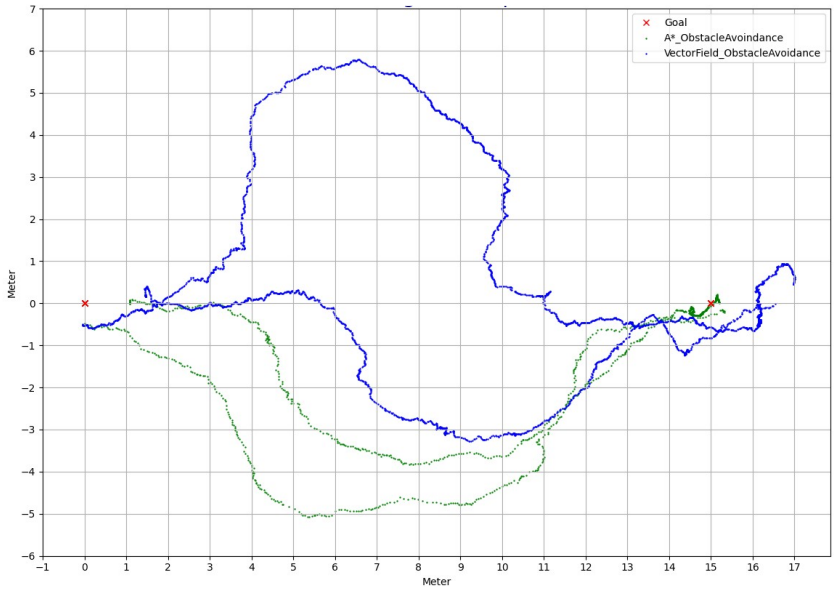


Figura 140 - Comparação de Algoritmos (A* e VF).

Na Tabela 17 estão representados os valores referentes aos erros de cada algoritmo. É normal estes erros serem muito maiores que os anteriormente vistos pois como existe desvio de obstáculos significa que o robô vai se desviar bastante da trajetória.

Tabela 17 - Valores comparativos entre algoritmos de desvio de obstáculos.

Algoritmo	Média [m]	Desvio padrão [m]	Máximo [m]	Tempo de navegação [s]
A*	1,70	1,59	4,96	32,67
VF	1,54	1,53	5,62	112,47

A Tabela 17 mostra os valores comparativos entre ambos os algoritmos, referenciando que o algoritmo *A** tem melhor desempenho que o algoritmo VF. O seu tempo de navegação torna-se quase quatro vezes inferior ao algoritmo VF, tem um erro máximo menor, os restantes estão bastante equiparados. Podemos concluir que o algoritmo *A** sai vencedor neste trabalho.

CAPÍTULO 5: CONCLUSÃO

No decurso deste projeto, exploraram-se e testaram-se vários métodos de controlo e algoritmos de navegação, revelando resultados de grande interesse. Contudo, a limitação do poder computacional apresentou-se como um desafio significativo, restringindo a viabilidade de aplicação de certos algoritmos devido à carga adicional que impunham ao já sobrecarregado sistema computacional do "Sentry". Essa limitação tornou-se particularmente evidente durante os testes realizados, quando o computador do "Sentry" deu sinais de esforço. Vários obstáculos surgiram ao longo do desenvolvimento, entre estes, a incerteza quanto à velocidade de cada lagarta do trator, o que levou à adoção da posição como referência. No entanto, esta abordagem resultou em saltos significativos na navegação do robô sempre que ocorria uma perda momentânea de sinal, desestabilizando a sua trajetória. Torna-se claro, portanto, que a melhoria da odometria do trator é um aspeto crucial a considerar para avanços futuros. A revisão do estado da arte e a pesquisa realizada permitiram conhecer algoritmos altamente promissores para a aplicação em questão, especialmente aqueles baseados no conceito de SLAM, que conferem ao robô uma referência espacial mais precisa e um entendimento mais profundo do ambiente circundante. Destacamos a utilização bem-sucedida de algoritmos notáveis, como o A* e o VF, que contribuíram de forma positiva para a capacidade de navegação do trator. Além disso, é importante mencionar que a investigação realizada ao longo deste projeto resultou numa reformulação das abordagens de controlo, desvio de obstáculos e navegação autónoma. A experiência adquirida ao longo deste percurso revelou-se extremamente enriquecedora, expandindo significativamente o conhecimento neste vasto campo da navegação autónoma. Em resumo, este projeto não só promoveu a exploração e experimentação de vários métodos e algoritmos, como também nos permitiu adquirir uma nova perspetiva sobre o desenvolvimento do robô, abrangendo o seu controlo, desvio de obstáculos e navegação em toda a sua extensão. O sentimento é de enriquecimento e ampliação da compreensão deste vasto mundo da navegação autónoma.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] L. M. Ferreira, A. P. Coimbra, e A. T. De Almeida, «Autonomous System for Wildfire and Forest Fire Early Detection and Control», *Inventions*, vol. 5, n.º 3, p. 41, ago. 2020, doi: 10.3390/inventions5030041.

[2] A. C. Meira Castro, A. Nunes, A. Sousa, e L. Lourenço, «Mapping the Causes of Forest Fires in Portugal by Clustering Analysis», *Geosciences*, vol. 10, n.º 2, p. 53, jan. 2020, doi: 10.3390/geosciences10020053.

[3] S. T. McKinney, I. Abrahamson, T. Jain, e N. Anderson, «A systematic review of empirical evidence for landscape-level fuel treatment effectiveness», *fire ecol*, vol. 18, n.º 1, p. 21, set. 2022, doi: 10.1186/s42408-022-00146-3.

[4] M. S. Couceiro, D. Portugal, J. F. Ferreira, e R. P. Rocha, «SEMFIRE: Towards a new generation of forestry maintenance multi-robot systems», em *2019 IEEE/SICE International Symposium on System Integration (SII)*, jan. 2019, pp. 270–276. doi: 10.1109/SII.2019.8700403.

[5] A. Oliveira, «Robótica na agricultura: o que é e como vai impactar o agro?», Blog da Aegro. Acedido: 20 de julho de 2023. [Em linha]. Disponível em: https://blog.aegro.com.br/robotica-na-agricultura/

[6] «Fieldwork Robotics raises £1.5M in funding for AI-supported harvesting robot», Tech.eu. Acedido: 5 de setembro de 2023. [Em linha]. Disponível em: https://tech.eu/2023/08/10/fieldwork-robotics-ai-supported-harvesting-robot/

[7] «ARA in the spotlight in “Schweizer Landtechnik”», Ecorobotix. Acedido: 5 de setembro de 2023. [Em linha]. Disponível em: https://ecorobotix.com/en/discover/news-novelties/ara-in-the-spotlight-in-schweizer-landtechnik/

[8] L. F. P. Oliveira, A. P. Moreira, e M. F. Silva, «Advances in Forest Robotics: A State-of-the-Art Survey», *Robotics*, vol. 10, n.º 2, p. 53, mar. 2021, doi: 10.3390/robotics10020053.

[9] B. Suger, B. Steder, e W. Burgard, «Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3D-lidar data», em *2015 IEEE International Conference on Robotics and Automation (ICRA)*, mai. 2015, pp. 3941–3946. doi: 10.1109/ICRA.2015.7139749.

[10] L. Emmi, M. Gonzalez-de-Soto, G. Pajares, e P. Gonzalez-de-Santos, «New Trends in Robotics for Agriculture: Integration and Assessment of a Real Fleet of Robots», *The Scientific World Journal*, vol. 2014, pp. 1–21, 2014, doi: 10.1155/2014/404059.

[11] K. M. Gayathri, N. Thangadurai, e M. P. Vasudha, «Positioning and signal strength analysis of IRNSS and GPS receiver in plain and vegetation area», em *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, mai. 2016, pp. 251–256. doi: 10.1109/ICACCCT.2016.7831641.

[12] R. Raj e A. Kos, «A Comprehensive Study of Mobile Robot: History, Developments, Applications, and Future Research Perspectives», *Applied Sciences*, vol. 12, n.º 14, p. 6951, jul. 2022, doi: 10.3390/app12146951.

[13] E. GuizzoUpdated, «Types of Robots», ROBOTS: Your Guide to the World of Robotics. Acedido: 28 de agosto de 2023. [Em linha]. Disponível em: https://robotsguide.com/learn/types-of-robots

[14] F. Rubio, F. Valero, e C. Llopis-Albert, «A review of mobile robots: Concepts, methods, theoretical framework, and applications», *International Journal of Advanced Robotic Systems*, vol. 16, n.º 2, p. 172988141983959, mar. 2019, doi: 10.1177/1729881419839596.

[15] S. G. Tzafestas, *Introduction to Mobile Robot Control*, 1st ed. 225 Wyman Street, Waltham, MA 02451, USA: Elsevier, 2013. [Em linha]. Disponível em: https://books.google.com/books/about/Introduction_to_Mobile_Robot_Control.html?hl=pt-PT&id=gmYALDVqlLUC

[16] Y. Gong *et al.*, «Legged robots for object manipulation: A review», *Front. Mech. Eng.*, vol. 9, p. 1142421, abr. 2023, doi: 10.3389/fmech.2023.1142421.

[17] «SEMFIRE - Segurança, Exploração e Manutenção de Florestas com Integração de Robótica Ecológica». Acedido: 26 de setembro de 2023. [Em linha]. Disponível em: https://semfire.ingeniarius.pt/

[18] «Multiscope Rescue with Hydra», Milrem. Acedido: 29 de agosto de 2023. [Em linha]. Disponível em: https://milremrobotics.com/product/multiscope-rescue-hydra/

[19] «Multiscope Rescue Hose Cartridge», Milrem. Acedido: 29 de agosto de 2023. [Em linha]. Disponível em: https://milremrobotics.com/product/firehouse-container/

[20] «A firefighting robot called Colossus helped battle the Notre-Dame blaze», TechSpot. Acedido: 29 de agosto de 2023. [Em linha]. Disponível em: https://www.techspot.com/news/79706-firefighting-robot-called-colossus-helped-battle-notre-dame.html

[21] «C. S. Conceição <Robôs para Deteção de Incêndios em Edifícios> , Universidade Coimbra, 2019», https://estudogeral.uc.pt/bitstream/10316/93579/1/Rob%C3%B4s%20para%20Dete%C3%A7%C3%A3o%20de%20Inc%C3%AAndios%20em%20Edif%C3%ADcios.pdf.

[22] N. A. Sathiabalan *et al.*, «Autonomous robotic fire detection and extinguishing system», *J. Phys.: Conf. Ser.*, vol. 2107, n.º 1, p. 012060, nov. 2021, doi: 10.1088/1742-6596/2107/1/012060.

[23] D. Herath e D. St-Onge, Eds., *Foundations of Robotics: A Multidisciplinary Approach with Python and ROS*. Singapore: Springer Nature Singapore, 2022. doi: 10.1007/978-981-19-1983-1.

[24] N. A. I. Ruslan, N. H. Amer, K. Hudha, Z. A. Kadir, S. A. F. M. Ishak, e S. M. F. S. Dardin, «Modelling and control strategies in path tracking control for autonomous tracked vehicles: A review of state of the art and challenges», *Journal of Terramechanics*, vol. 105, pp. 67–79, fev. 2023, doi: 10.1016/j.jterra.2022.10.003.

[25] R. P. Borase, D. K. Maghade, S. Y. Sondkar, e S. N. Pawar, «A review of PID control, tuning methods and applications», *Int. J. Dynam. Control*, vol. 9, n.º 2, pp. 818–827, jun. 2021, doi: 10.1007/s40435-020-00665-4.

[26] J. M. Holland, *Designing mobile autonomous robots*. Amsterdam ; Boston: Elsevier Newnes, 2004.

[27] C. B. Kadu e C. Y. Patil, «Design and Implementation of Stable PID Controller for Interacting Level Control System», *Procedia Computer Science*, vol. 79, pp. 737–746, 2016, doi: 10.1016/j.procs.2016.03.097.

[28] Kiam Heong Ang, G. Chong, e Yun Li, «PID control system analysis, design, and technology», *IEEE Trans. Contr. Syst. Technol.*, vol. 13, n.º 4, pp. 559–576, jul. 2005, doi: 10.1109/TCST.2005.847331.

[29] «Proportional Integral Derivative (PID)». Acedido: 20 de julho de 2023. [Em linha]. Disponível em: https://apmonitor.com/pdc/index.php/Main/ProportionalIntegralDerivative

[30] K. J. Åström e R. M. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton: Princeton University Press, 2008.

[31] M. Shahrokhi e A. Zomorrodi, «Comparison of PID Controller Tuning Methods», Sharif University of Technology,Department of Chemical & Petroleum Engineering, dec.2014, http://maulana.lecture.ub.ac.id/files/2014/12/Jurnal-PID_Tunning_Comparison.pdf.

[32] F. F. França, «Controlador PID para controle de temperatura de uma carga resistiva AC» nov. 2018, Dissertação UNESP, Guarantinguetá, https://repositorio.unesp.br/server/api/core/bitstreams/7329fa57-7dc5-4de6-9a16-e2bcd6a1da95/content.

[33] «9.3: PID Tuning via Classical Methods», Engineering LibreTexts. Acedido: 12 de outubro de 2023. [Em linha]. Disponível em: https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Chemical_Process_Dynamics_and_Controls_(Woolf)/09%3A_Proportional-Integral-Derivative_(PID)_Control/9.03%3A_PID_Tuning_via_Classical_Methods

[34] R. Sen, C. Pati, S. Dutta, e R. Sen, «Comparison Between Three Tuning Methods of PID Control for High Precision Positioning Stage», *MAPAN*, vol. 30, n.º 1, pp. 65–70, mar. 2015, doi: 10.1007/s12647-014-0123-z.

[35] «PID tuning methods», INCATools. Acedido: 13 de outubro de 2023. [Em linha]. Disponível em: https://www.incatools.com/pid-tuning/pid-tuning-methods/

[36] A. Almabrok, M. Psarakis, e A. Dounis, «Fast Tuning of the PID Controller in An HVAC System Using the Big Bang–Big Crunch Algorithm and FPGA Technology», *Algorithms*, vol. 11, n.º 10, p. 146, set. 2018, doi: 10.3390/a11100146.

[37] C. Moraga, «Introduction to fuzzy logic», *Facta Univ Electron Energ*, vol. 18, n.º 2, pp. 319–328, 2005, doi: 10.2298/FUEE0502319M.

[38] T. J. Ross, *Fuzzy Logic with Engineering Applications*, 1.ª ed. Wiley, 2010. doi: 10.1002/9781119994374.

[39] E. Universitat Politècnica De València, «Universitat Politècnica de València», *ing.agua*, vol. 18, n.º 1, p. ix, set. 2014, doi: 10.4995/ia.2014.3293.

[40] S. Karakaya e H. Ocak, «Fuzzy logic-based moving obstacle avoidance method», *GJCS*, vol. 9, n.º 1, pp. 1–9, abr. 2019, doi: 10.18844/gjcs.v9i1.3972.

[41] A. Sobrino, «Fuzzy Logic and Education: Teaching the Basics of Fuzzy Logic through an Example (by Way of Cycling)», *Education Sciences*, vol. 3, n.º 2, pp. 75–97, abr. 2013, doi: 10.3390/educsci3020075.

[42] D. Q. Mayne, «Model predictive control: Recent developments and future promise», *Automatica*, vol. 50, n.º 12, pp. 2967–2986, dez. 2014, doi: 10.1016/j.automatica.2014.10.128.

[43] M. Schwenzer, M. Ay, T. Bergs, e D. Abel, «Review on model predictive control: an engineering perspective», *Int J Adv Manuf Technol*, vol. 117, n.º 5–6, pp. 1327–1349, nov. 2021, doi: 10.1007/s00170-021-07682-3.

[44] E. F. Camacho e C. Bordons, *Model Predictive control*. em Advanced Textbooks in Control and Signal Processing. London: Springer London, 2007. doi: 10.1007/978-0-85729-398-5.

[45] M. H. A. Sidi, K. Hudha, Z. A. Kadir, e N. H. Amer, «Modeling and path tracking control of a tracked mobile robot», em *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, mar. 2018, pp. 72–76. doi: 10.1109/CSPA.2018.8368688.

[46] F. Gul, W. Rahiman, e S. S. Nazli Alhady, «A comprehensive study for robot navigation techniques», *Cogent Engineering*, vol. 6, n.º 1, p. 1632046, jan. 2019, doi: 10.1080/23311916.2019.1632046.

[47] R. Abiyev, D. Ibrahim, e B. Erin, «Navigation of mobile robots in the presence of obstacles», *Advances in Engineering Software*, vol. 41, n.º 10–11, pp. 1179–1186, out. 2010, doi: 10.1016/j.advengsoft.2010.08.001.

[48] M. A. Javaid, «Understanding Dijkstra Algorithm», *SSRN Journal*, 2013, doi: 10.2139/ssrn.2340905.

[49] B. K. Patle, G. Babu L, A. Pandey, D. R. K. Parhi, e A. Jagadeesh, «A review: On path planning strategies for navigation of mobile robot», *Defence Technology*, vol. 15, n.º 4, pp. 582–606, ago. 2019, doi: 10.1016/j.dt.2019.04.011.

[50] A. Loganathan e N. S. Ahmad, «A systematic review on recent advances in autonomous mobile robot navigation», *Engineering Science and Technology, an International Journal*, vol. 40, p. 101343, abr. 2023, doi: 10.1016/j.jestch.2023.101343.

[51] D. Rachmawati e L. Gustin, «Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem», *J. Phys.: Conf. Ser.*, vol. 1566, n.º 1, p. 012061, jun. 2020, doi: 10.1088/1742-6596/1566/1/012061.

[52] L. Lacasa, B. Luque, F. Ballesteros, J. Luque, e J. C. Nuño, «From time series to complex networks: The visibility graph», *Proc. Natl. Acad. Sci. U.S.A.*, vol. 105, n.º 13, pp. 4972–4975, abr. 2008, doi: 10.1073/pnas.0709247105.

[53] T. Lv, C. Zhao, e J. Bao, «A Global Path Planning Algorithm Based on Bidirectional SVGA», *Journal of Robotics*, vol. 2017, pp. 1–11, 2017, doi: 10.1155/2017/8796531.

[54] L. G. Hee e M. H. A. Jr, «MOBILE ROBOTS NAVIGATION, MAPPING & LOCALIZATION: PART I», *MOBILE ROBOTS NAVIGATION, jan.2018*,DOI: 10.4018/9781599048499.ch158.

[55] J. Borenstein e Y. Koren, «The vector field histogram-fast obstacle avoidance for mobile robots», *IEEE Trans. Robot. Automat.*, vol. 7, n.º 3, pp. 278–288, jun. 1991, doi: 10.1109/70.88137.

[56] H. Taheri e Z. C. Xia, «SLAM; definition and evolution», *Engineering Applications of Artificial Intelligence*, vol. 97, p. 104032, jan. 2021, doi: 10.1016/j.engappai.2020.104032.

[57] H. Durrant-Whyte e T. Bailey, «Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms», jun.2006, IEEE Robotics & Automation Magazine, doi:10.1109/MRA.2006.1638022.

[58] U. Frese, R. Wagner, e T. Röfer, «A SLAM Overview from a User's Perspective», *Künstl Intell*, vol. 24, n.º 3, pp. 191–198, set. 2010, doi: 10.1007/s13218-010-0040-4.

[59] K. Alisher, K. Alexander, e B. Alexandr, «Control of the Mobile Robots with ROS in Robotics Courses», *Procedia Engineering*, vol. 100, pp. 1475–1484, 2015, doi: 10.1016/j.proeng.2015.01.519.

[60] V. A. Hax e N. L. D. Filho, «ROS MMaster: adaptando o ROS para maior escalabilidade», jan. 2014, ResearchGate,NAVTEC 2014At: Rio Grande - RS - Brasil, https://www.researchgate.net/publication/276920583_ROS_MMaster_adaptando_o_ROS_para_maior_escalabilidade..

[61] N. Koenig e A. Howard, «Design and use paradigms for gazebo, an open-source multi-robot simulator», em *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, Sendai, Japan: IEEE, 2004, pp. 2149–2154. doi: 10.1109/IROS.2004.1389727.

[62] P. Kaur, Z. Liu, e W. Shi, «Simulators for Mobile Social Robots:State-of-the-Art and Challenges». arXiv, 7 de fevereiro de 2022. Acedido: 31 de agosto de 2023. [Em linha]. Disponível em: http://arxiv.org/abs/2202.03582

[63] «Setting Up the ROS Navigation Stack for a Simulated Robot – Automatic Addison». Acedido: 31 de agosto de 2023. [Em linha]. Disponível em: https://automaticaddison.com/setting-up-the-ros-navigation-stack-for-a-simulated-robot/

[64] M. Quigley, B. Gerkey, e W. D. Smart, «Programming Robots with ROS»,nov.2015, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 978-1-4493-2389-9, First Edition.

[65] H. Wang, L. Li, H. Chen, Y. Li, S. Qiu, e R. Gravina, «Motion Recognition for Smart Sports Based on Wearable Inertial Sensors», em *Body Area Networks: Smart IoT and Big Data for Intelligent Health Management*, vol. 297, L. Mucchi, M. Hämäläinen, S. Jayousi, e S. Morosi, Eds., em Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 297. , Cham: Springer International Publishing, 2019, pp. 114–124. doi: 10.1007/978-3-030-34833-5_10.

[66] J. Hislop, M. Isaksson, J. McCormick, e C. Hensman, «Validation of 3-Space Wireless Inertial Measurement Units Using an Industrial Robot», *Sensors*, vol. 21, n.º 20, p. 6858, out. 2021, doi: 10.3390/s21206858.

[67] M. Horvat e G. Gledec, «A comparative study of YOLOv5 models performance for image localization and classification», 2022.

[68] M. Karthi, V. Muthulakshmi, R. Priscilla, P. Praveen, e K. Vanisri, «Evolution of YOLO-V5 Algorithm for Object Detection: Automated Detection of Library Books and Performace validation of Dataset», em *2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*, Chennai, India: IEEE, set. 2021, pp. 1–6. doi: 10.1109/ICSES52305.2021.9633834.

[69] Z. Wu, N. Fuller, D. Theriault, e M. Betke, «A Thermal Infrared Video Benchmark for Visual Analysis», em *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Columbus, OH, USA: IEEE, jun. 2014, pp. 201–208. doi: 10.1109/CVPRW.2014.39.

[70] M. Ivašić-Kos, M. Krišto, e M. Pobar, «Human Detection in Thermal Imaging Using YOLO», em *Proceedings of the 2019 5th International Conference on Computer and Technology Applications*, Istanbul Turkey: ACM, abr. 2019, pp. 20–24. doi: 10.1145/3323933.3324076.

[71] J. P. P. Monteiro, «Deteção e classificação de obstáculos à navegação de veículos autónomos em ambientes florestais»feb.2023, University of Coimbra, Volume1, https://estudogeral.uc.pt/handle/10316/107945.

[72] G. Mier, J. Valente, e S. De Bruin, «Fields2Cover: An Open-Source Coverage Path Planning Library for Unmanned Agricultural Vehicles», *IEEE Robot. Autom. Lett.*, vol. 8, n.º 4, pp. 2166–2172, abr. 2023, doi: 10.1109/LRA.2023.3248439.

[73] J. Jin, «Optimal field coverage path planning on 2D and 3D surfaces»,2019, Iowa State University, https://core.ac.uk/download/pdf/38925159.pdf, Dissertation.

[74] M. Nørremark, R. S. Nilsson, e C. A. G. Sørensen, «In-Field Route Planning Optimisation and Performance Indicators of Grain Harvest Operations», *Agronomy*, vol. 12, n.º 5, p. 1151, mai. 2022, doi: 10.3390/agronomy12051151.

[75] «icckinematics.pdf». Acedido: 1 de agosto de 2023. [Em linha]. Disponível em: https://www.cs.columbia.edu/~allen/F19/NOTES/icckinematics.pdf

[76] J. L. Martínez, A. Mandow, J. Morales, S. Pedraza, e A. García-Cerezo, «Approximating Kinematics for Tracked Mobile Robots», *The International Journal of Robotics Research*, vol. 24, n.º 10, pp. 867–878, out. 2005, doi: 10.1177/0278364905058239.

[77] A. Tisland, M. Baksaas, e K. Mathiassen, «How extending the kinematic model affects path following in off-road terrain for differential drive UGVs», em *Unmanned Systems Technology XXIV*, P. L. Muench, H. G. Nguyen, e B. K. Skibba, Eds., Orlando, United States: SPIE, mai. 2022, p. 8. doi: 10.1117/12.2618731.

[78] H. Lu, G. Xiong, e K. Guo, «Motion Predicting of Autonomous Tracked Vehicles with Online Slip Model Identification», *Mathematical Problems in Engineering*, vol. 2016, pp. 1–13, 2016, doi: 10.1155/2016/6375652.

[79] D. Tola e P. Corke, «Understanding URDF: A Survey Based on User Experience». arXiv, 12 de março de 2023. Acedido: 20 de julho de 2023. [Em linha]. Disponível em: http://arxiv.org/abs/2302.13442

[80] «What is the Difference Between rviz and Gazebo? – Automatic Addison». Acedido: 20 de julho de 2023. [Em linha]. Disponível em: https://automaticaddison.com/what-is-the-difference-between-rviz-and-gazebo/

[81] «Bang-Bang Control with BangBangController», FIRST Robotics Competition Documentation. Acedido: 20 de julho de 2023. [Em linha]. Disponível em: https://docs.wpilib.org/en/stable/docs/software/advanced-controls/controllers/bang-bang.html

[82] C. Kitts *et al.*, «Field operation of a robotic small waterplane area twin hull boat for shallow-water bathymetric characterization: A Robotic SWATH Boat for Bathymetry», *J. Field Robotics*, vol. 29, n.º 6, pp. 924–938, nov. 2012, doi: 10.1002/rob.21427.

[83] M. Chiaberge e L. Galtarossa, «Obstacle Avoidance Algorithms for Autonomous Navigation system in Unstructured Indoor areas», POLITECNICO DI TORINO, Master Thesis, oct.2018, https://webthesis.biblio.polito.it/8991/

[84] «astar.pdf». Acedido: 18 de agosto de 2023. [Em linha]. Disponível em: https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf

[85] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah, e E. Gunawan, «A Systematic Literature Review of A* Pathfinding», *Procedia Computer Science*, vol. 179, pp. 507–514, 2021, doi: 10.1016/j.procs.2021.01.034.

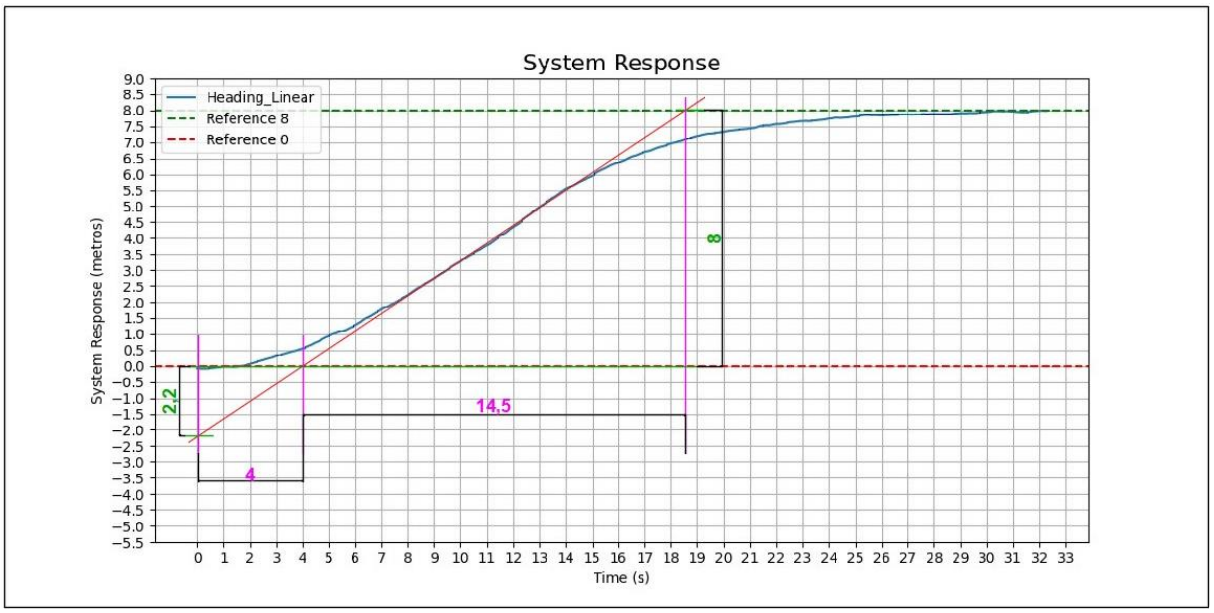
[86] D. R. Nelson, D. B. Barber, T. W. McLain, e R. W. Beard, «Vector field path following for small unmanned air vehicles», em *2006 American Control Conference*, Minneapolis, MN, USA: IEEE, 2006, p. 7 pp. doi: 10.1109/ACC.2006.1657648.

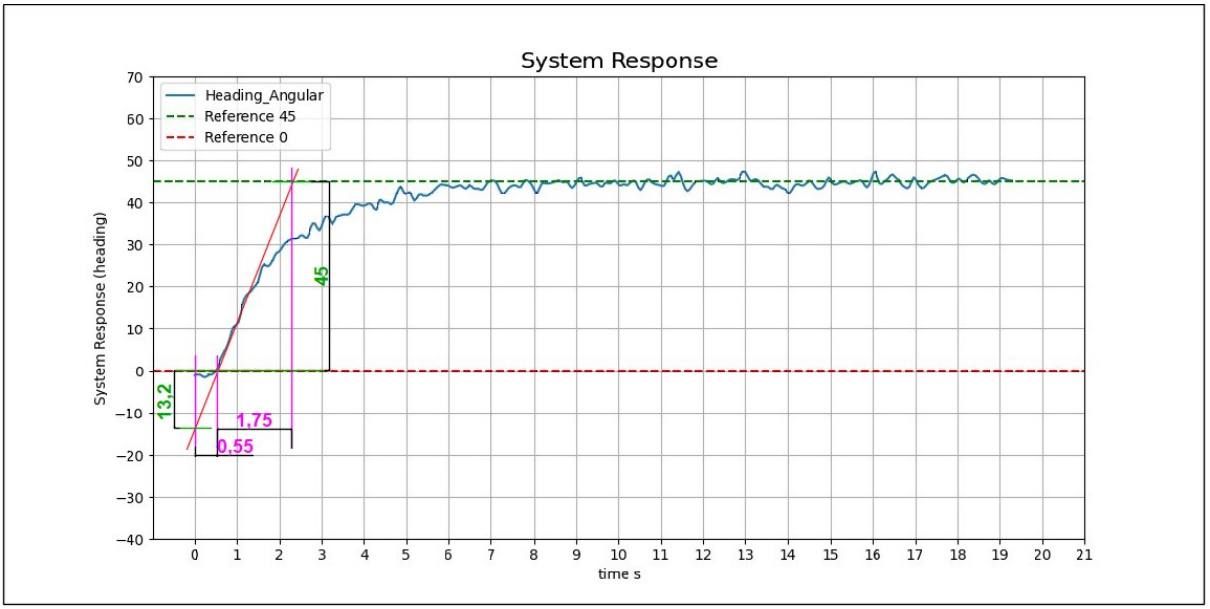
[87] Y. Koren e J. Borenstein, «Potential field methods and their inherent limitations for mobile robot navigation», em *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, Sacramento, CA, USA: IEEE Comput. Soc. Press, 1991, pp. 1398–1404. doi: 10.1109/ROBOT.1991.131810.

[88] R.Leo «Trajectory planning for a self-driving Electrical Vehicle.pdf» oct. 2021, POLITECNICO DI TORINO, Master Degree Thesis,https://webthesis.biblio.polito.it/20448/.

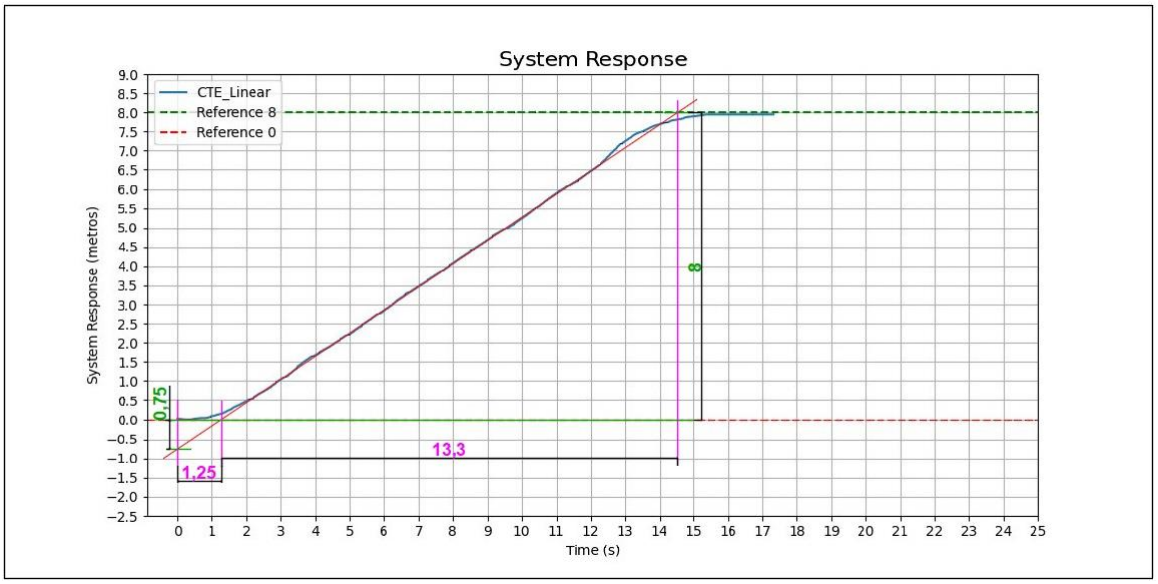
ANEXOS

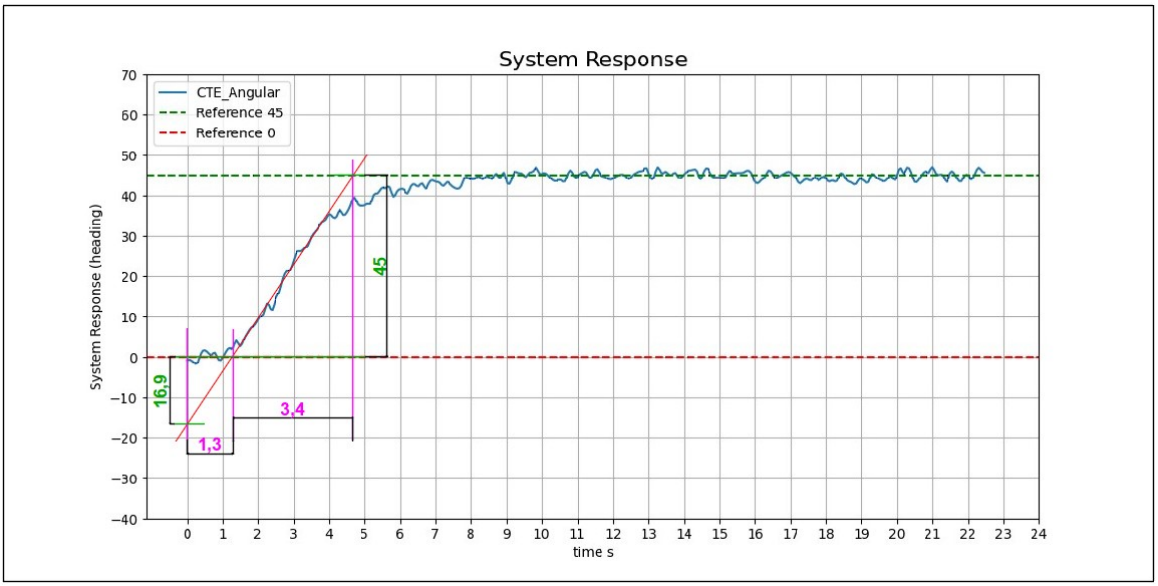
Anexo A – Experiência para o cálculo dos valores de ganho do PID linear e angular do algoritmo PIDH.



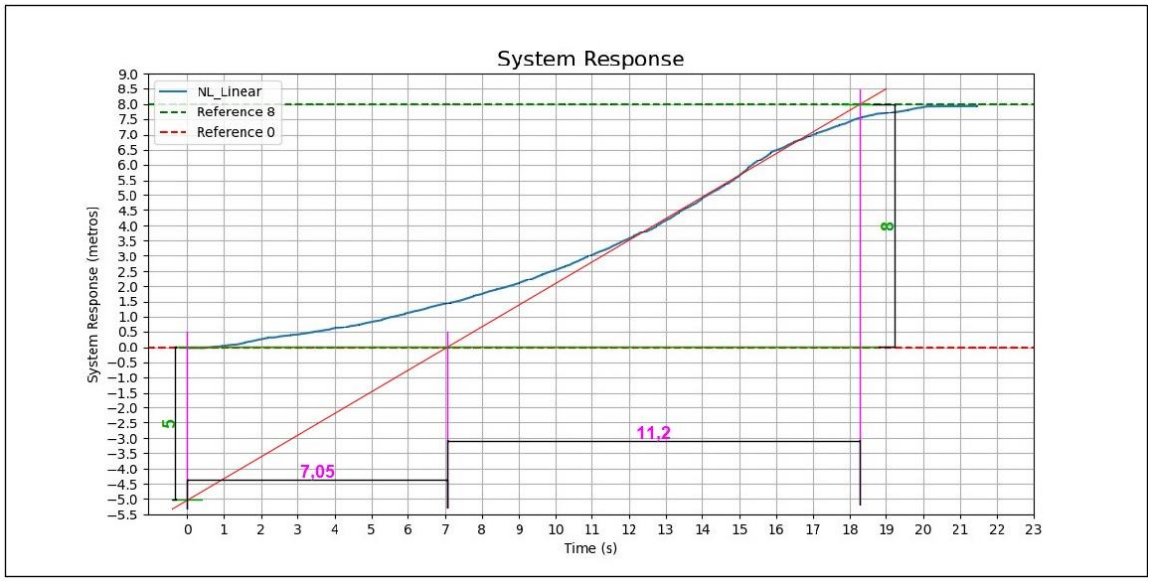


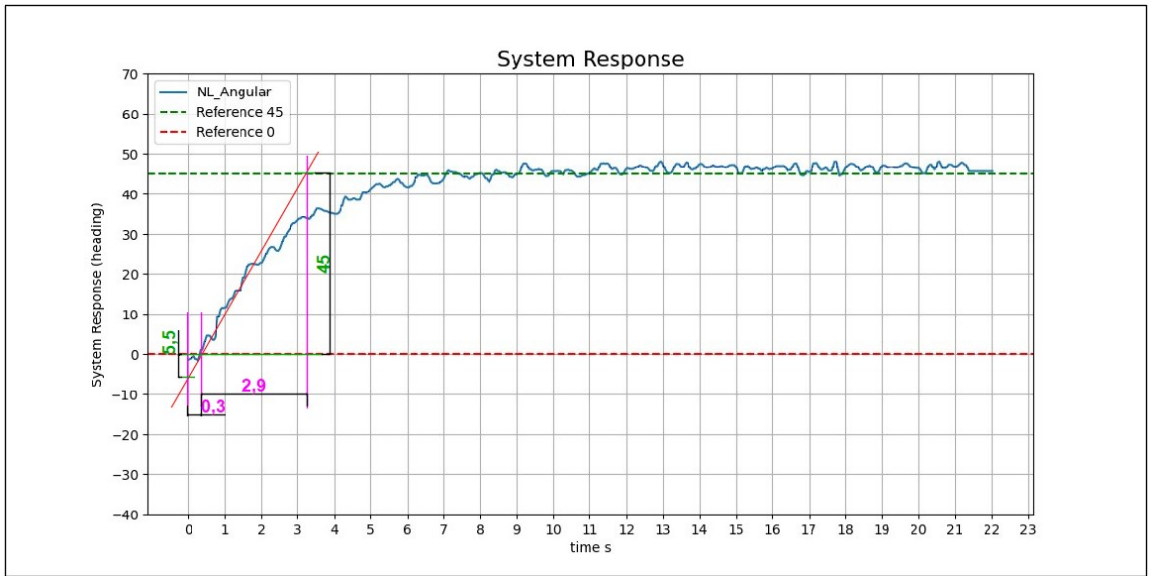
Anexo B – Experiência para o cálculo dos valores de ganho do PID linear e angular do algoritmo PID-CTE.

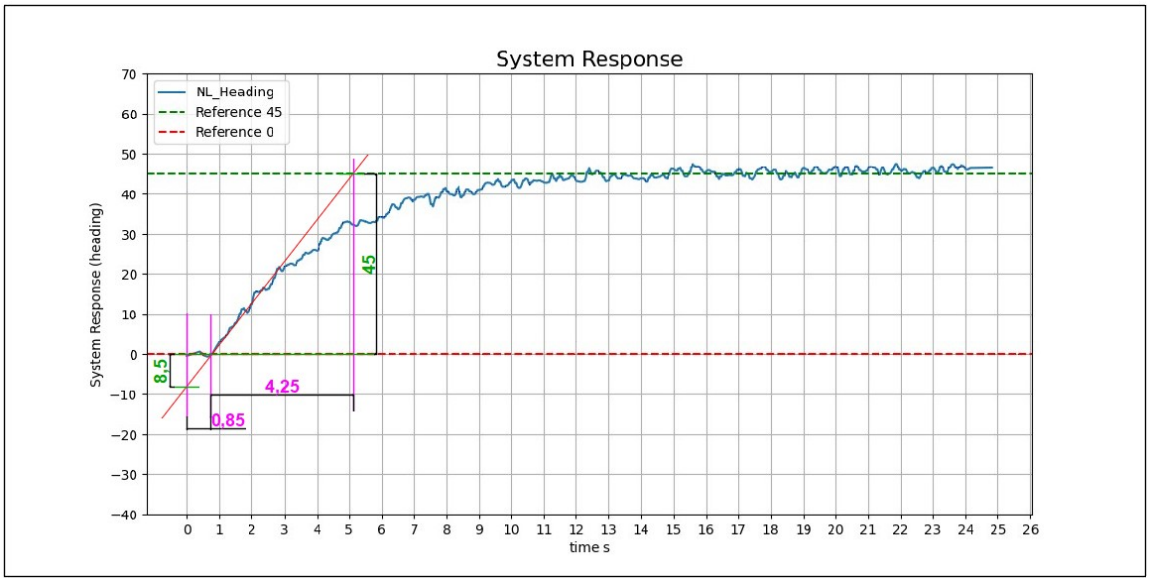




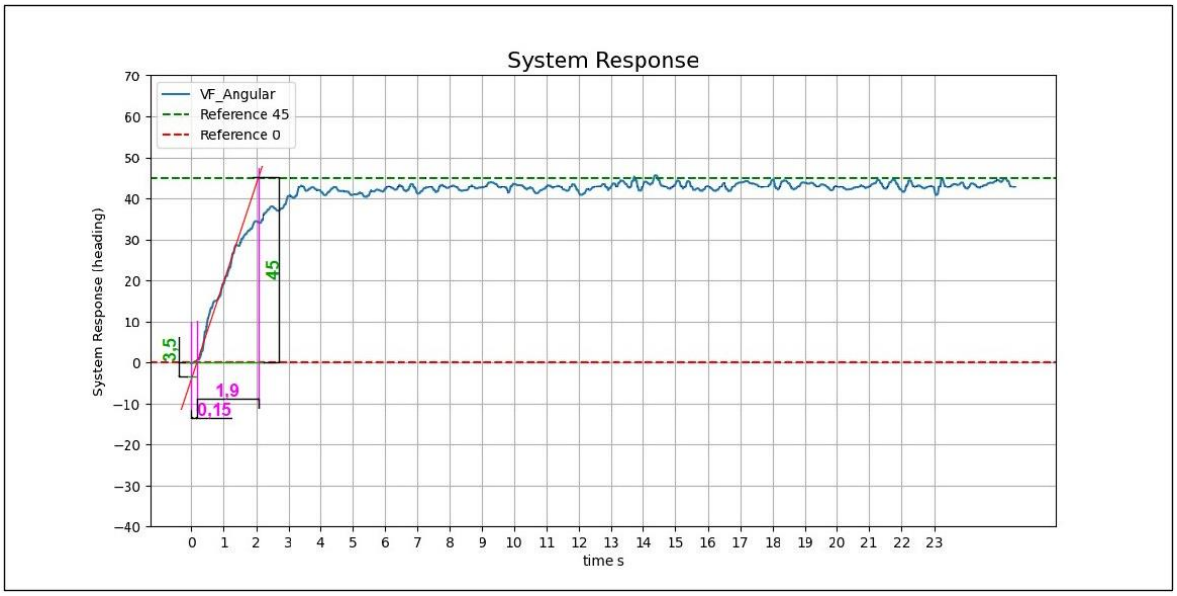
Anexo C – Experiência para o cálculo dos valores de ganho do PID linear, angular e orientação do algoritmo NL.







Anexo D – Experiência para o cálculo dos valores de ganho do PID angular do algoritmo VF.



Instituto Superior
de Engenharia
Politécnico de Coimbra