INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

# Controlo de um robô móvel seguidor de linha através de redes neuronais

**HUGO MESQUITA LEAL**
outubro de 2024

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

# Controlo de um robô móvel seguidor de linha através de redes neuronais

**Hugo Mesquita Leal**

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Automação e Sistemas


Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

Outubro, 2024

*Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores, Área de Especialização em Automação e Sistemas.*

**Candidato:** Hugo Mesquita Leal, Nº 1190660, 1190660@isep.ipp.pt

**Orientação Científica:** Ramiro De Sousa Barbosa, RSB@isep.ipp.pt


Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

Outubro, 2024

# Agradecimentos

Antes de mais, gostaria de expressar a minha profunda gratidão aos meus professores pelo seu inestimável apoio ao longo do meu percurso académico. Em particular, devo um agradecimento especial ao meu orientador, o Professor Ramiro de Sousa Barbosa, pela sua orientação nos aspetos técnicos deste projeto. A sua ajuda na resolução de problemas, na revisão do meu trabalho e a sua disponibilidade constante e respostas rápidas foram fundamentais para o sucesso deste relatório. Agradeço também a todos os meus outros professores que me ensinaram e inspiraram ao longo da minha licenciatura e mestrado. Para além disso, gostaria de agradecer à minha primeira professora, Raquel Seabra, por ter despertado a minha paixão pela aprendizagem e pela educação desde tenra idade.

Estou profundamente grato à minha família, cujo apoio inabalável tem sido uma fonte constante de força. Ao meu pai, Martinho Leal, à minha mãe, Paula Mesquita, à minha irmã, Diana Leal, e ao meu irmão, Bruno Leal, obrigada pelo vosso encorajamento, mesmo quando não sabiam exatamente como ajudar. Os vossos esforços para me manterem concentrado e no caminho certo, ao mesmo tempo que se certificavam de que eu não me distraía demasiado, desempenharam um papel crucial na conclusão desta dissertação.

Gostaria também de agradecer aos meus colegas e amigos universitários, que têm estado ao meu lado desde o início do meu percurso académico, há cinco anos. Trabalhar e conviver convosco foi fundamental para a minha formação, e não o teria conseguido sem vocês. Gostaria de mencionar especialmente Diogo Resende, Leonardo Silva, Álvaro Cardoso, Gonçalo Araújo, Katharina Mees, Diogo Carvalho, Francisco Encarnação e Pedro Costa. A vossa colaboração e camaradagem foram inestimáveis.

Por último, aos meus amigos ao longo dos anos, apesar de não terem contribuído direta ou indiretamente para esta dissertação, o vosso apoio moral e os bons momentos que partilhámos permitiram-me encontrar um equilíbrio entre trabalho e diversão. Aprecio profundamente o papel que desempenharam para me manterem de pé. Gostaria de mencionar nominalmente Carlos Silva, Eduardo Moreira, Jorge Ribeiro, Pedro Pinto, Sydney Riches, Lucas Jorge, Daniel Gonçalves e Tiago Gaspar.

Obrigado a todos por tornarem esta viagem possível.

# Resumo

Este trabalho teve como objetivo desenvolver e comparar o desempenho de um robô seguidor de linha utilizando redes neuronais e controladores tradicionais, como o PID. Após a análise de diferentes técnicas e robôs, foram desenvolvidas e testadas várias abordagens baseadas em *Deep Learning*.

Na primeira experiência utilizou-se os sensores infravermelhos do robô para controlador o robô para seguir uma linha desenhada no chão. Para este controlo, foi primeiramente implementado um controlador PID. Os dados obtidos com este método foram depois utilizados para treinar uma rede neuronal LSTM. Esta rede foi capaz de simular o comportamento do controlador PID e guiar-se pelo percurso eficazmente.

Na segunda experiência utilizou-se a câmera incluída no robô para realizar a mesma tarefa de controlar um robô seguidor de linha através de redes neuronais. Para isto, começou-se por tirar várias imagens do circuito com essa câmera. As imagens foram depois processadas e posteriormente categorizadas. Para a categorização foram testados dois métodos, onde o método de categorização manual em 8 diferentes classes mostrou-se ser o mais eficaz para este efeito. Com isto foi treinada uma rede CNN capaz de categorizar em tempo real novas imagens. Este modelo foi então utilizado para o controlo do robô.

Os resultados mostraram que as redes neuronais conseguem lidar com trajetos mais complexos, mas apresentam desafios como maior carga de processamento e necessidade de calibração. Em contrapartida, o controlador PID se mostrou mais simples e eficaz em circuitos menos exigentes. Concluiu-se que, as redes neuronais são promissoras para cenários mais complexos, mas podem ser também utilizadas eficientemente para um simples seguimento de linha.

**Palavras-Chave**: AGV, Robô, Raspbot, PID, LSTM, CNN, IA, *Deep Learning*, Seguidor de linha.

# Abstract

The aim of this work was to develop and compare the performance of a line-following robot using neural networks and traditional controllers such as the PID. After analyzing different techniques and robots, several approaches based on Deep Learning were developed and tested.

In the first experiment, the robot's infrared sensors were used to control the robot to follow a line drawn on the floor. For this control, a PID controller was first implemented. The data obtained with this method was then used to train an LSTM neural network. This network was able to simulate the behavior of the PID controller and guide itself along the route effectively.

In the second experiment, the camera included in the robot was used to perform the same task of controlling a line-following robot using neural networks. To do this, we began by taking several images of the circuit with the camera. The images were then processed and categorized. For the categorization, two methods were tested, where the manual categorization method into 8 different classes proved to be the most effective for this purpose. This trained a CNN network capable of categorizing new images in real time. This model was then used to control the robot.

The results showed that neural networks can handle more complex paths, but present challenges such as a higher processing load and the need for calibration. In contrast, the PID controller proved to be simpler and more effective for less demanding circuits. It was concluded that neural networks are promising for more complex scenarios, but can also be used efficiently for simple line following.

**Keywords**: AGV, Robot, Raspbot, PID, LSTM, CNN, AI, Deep Learning, Line follower.

# Índice

# Lista de Figuras

# Lista de Tabelas

# Listagens

# Lista de Acrónimos

| | |
|---|---|
| **4WD** | *Four-wheel Drive* |
| **AGV** | *Automated Guided Vehicle* |
| **ANN** | *Artificial Neural Networks* |
| **API** | *Application Programming Interface* |
| **CNN** | *Convolutional Neural Network* |
| **CSI** | *Camera Serial Interface* |
| **CSV** | *Comma-Separated Values* |
| **DL** | *Deep Learning* |
| **GPIO** | *General Purpose Input/Output* |
| **IA** | Inteligência Artificial |
| **IR** | *Infrared* |
| **LED** | *Light-Emitting Diode* |
| **LiDAR** | *Light Detection and Ranging* |
| **LSTM** | *Long Short-Term Memory* |
| **MES** | *Manufacturing Execution System* |
| **ML** | *Machine Learning* |
| **MLP** | *Multilayer Perceptron* |
| **NLP** | *Natural Language Processing* |
| **OS** | *Operating System* |
| **PID** | Proporcional, Integral e Derivativo |
| **PR** | *Pattern Recognition* |
| **RNN** | *Recurrent Neural Network* |

**SLAM** *Simultaneous Localization and Mapping*

**TMS** *Transport Management System*

**TTL** *Transistor-Transistor Logic*

## Capítulo 1

# Introdução

Nos últimos anos, técnicas de inteligência artificial, como o *Machine Learning* e o *Deep Learning*, têm revolucionado diversas áreas da ciência e engenharia, incluindo a robótica. Essas técnicas têm mostrado um potencial significativo para melhorar a autonomia e a inteligência de robôs, permitindo-lhes executar tarefas complexas com maior eficiência e precisão. Um dos campos promissores onde essas técnicas são atualmente aplicadas é o controlo de robôs móveis, incluindo Veículos Guiados Autonomamente (AGV).

Este trabalho tem como objetivo explorar e implementar técnicas de redes neuronais para o controlo de um robô móvel. Optou-se pelo desenvolvimento de um robô seguidor de linha, uma aplicação prática e relevante que permite demonstrar a eficácia das técnicas de *Deep Learning* no controlo de robôs. O controlo de robôs móveis é um desafio significativo devido à necessidade de navegar em ambientes dinâmicos e imprevisíveis, onde a precisão e a rapidez na tomada de decisões são cruciais.

Para alcançar esse objetivo, o trabalho será dividido nas seguintes etapas: uma pesquisa bibliográfica detalhada sobre redes neuronais e *Deep Learning* aplicados ao controlo de robôs, o desenvolvimento de várias estruturas de controlo. Além disso, será feita uma revisão de vários automóveis capazes de este género de aplicações com a escolha, aquisição e estudo de um deles.

Por fim, serão desenvolvidas várias estruturas de controlo, testes e validação das redes neuronais propostas, e uma análise comparativa dos resultados obtidos.

## 1.1 Contextualização

Nos últimos anos, os avanços em inteligência artificial, especialmente em *Machine Learning* e *Deep Learning*, têm transformado diversas áreas, incluindo a robótica. Robôs móveis, como os AGV, são essenciais em indústrias como manufatura e logística, exigindo controlo preciso e eficiente para navegar autonomamente em ambientes complexos.

Tradicionalmente, técnicas como o controlo proporcional-integral-derivativo (PID) e algoritmos de planeamento de trajetória têm sido usadas no controlo de robôs móveis. No entanto, essas abordagens enfrentam limitações em ambientes dinâmicos e não estruturados. As técnicas de *Deep Learning* oferecem uma abordagem baseada em dados que pode melhorar a adaptabilidade e a inteligência desses sistemas de controlo.

Neste contexto, optou-se pelo desenvolvimento de um robô seguidor de linha utilizando técnicas de *Deep Learning*. Essa aplicação prática e relevante permite demonstrar a eficácia das redes neuronais no controlo de robôs, proporcionando uma validação sólida para novas abordagens de controlo. Ao comparar técnicas tradicionais e modernas, este projeto busca identificar as vantagens e limitações do *Deep Learning*, contribuindo para o desenvolvimento de sistemas robóticos mais inteligentes e eficientes.

## 1.2 Definição do problema

A crescente demanda por automação em ambientes industriais tem impulsionado o desenvolvimento de AGV. No entanto, a eficiência e a adaptabilidade desses robôs ainda enfrentam limitações significativas devido ao uso de técnicas tradicionais de controlo, que muitas vezes não conseguem lidar adequadamente com ambientes dinâmicos e não estruturados.

Técnicas tradicionais, como o controlo PID, têm demonstrado eficácia limitada em situações onde a dinâmica do ambiente muda rapidamente. Esses métodos muitas vezes falham em proporcionar uma navegação suave e precisa, especialmente em ambientes complexos com obstáculos móveis.

Desta maneira o problema proposto é o de desenvolvimento de aplicações de *Deep Learning* e redes neuronais aplicáveis a AGV de forma a controlá-lo de forma dinâmica. Para isso a escolha recaiu num robô seguidor de linha.

### 1.2.1 Objetivos

O objetivo principal deste projeto é desenvolver e validar um sistema de controlo baseado em *Deep Learning* e redes neuronais para um robô seguidor de linha. Os objetivos específicos incluem:

- Pesquisar e analisar técnicas de redes neuronais aplicadas ao controlo de robôs;
- Pesquisar e analisar diferentes robôs escolhendo um deles;
- Desenvolver várias estruturas de controlo baseadas em *Deep Learning*;
- Implementar, testar e validar essas estruturas num robô seguidor de linha;
- Realizar uma análise comparativa entre técnicas tradicionais e modernas.

### 1.2.2 Resultados esperados

Com o desenvolvimento deste projeto é esperado gerar avanços na aplicação de técnicas de *Deep Learning* no controlo de robôs móveis. Especificamente, os resultados esperados incluem:

- **Desenvolvimento e implementação de estruturas de controlo baseadas em *Deep Learning*:** Espera-se criar várias estruturas de controlo utilizando redes neuronais que otimizem a precisão e eficiência do robô seguidor de linha.
- **Validação experimental das estruturas de controlo:** Através de experimentos práticos, os modelos desenvolvidos deverão demonstrar uma capacidade melhorada de seguir linhas em condições variáveis, validando a eficácia das técnicas propostas.
- **Análise comparativa de desempenho:** Uma comparação detalhada entre as novas técnicas e as tradicionais deverá evidenciar as melhorias em termos de adaptabilidade e precisão, reforçando a relevância do uso de *Deep Learning* no controlo de AGV.

## 1.3 Plano de trabalho

O plano de trabalho é dividido em várias etapas, cada uma com objetivos e atividades específicas. A Figura 1.1 apresenta um cronograma detalhado para garantir a execução eficiente do projeto.

| Tarefa | Data de início | Data de fim |
|---|---|---|
| Análise e definição do problema | 22/01/2024 | 29/01/2024 |
| Pesquisa de diferentes plataformas de robôs móveis | 30/01/2024 | 15/02/2024 |
| Estado de arte | 30/01/2024 | 10/05/2024 |
| Introdução e Hardware | 13/05/2024 | 10/06/2024 |
| Desenvolvimento de uma plataforma PID | 14/06/2024 | 28/06/2024 |
| Desenvolvimento de rede LSTM para o controlo do robô | 28/06/2024 | 13/07/2024 |
| Documentar a primeira experiência | 13/06/2024 | 26/07/2024 |
| Desenvolvimento de rede CNN para o controlo do robô | 29/07/2024 | 26/08/2024 |
| Documentar a segunda experiência | 26/08/2024 | 09/09/2024 |
| Conclusões e resumo | 09/09/2024 | 15/09/2024 |
| Correções e detalhes finais | 15/09/2024 | 22/09/2024 |

Figura 1.1: Cronograma do projeto.

## 1.4 Organização da dissertação

Esta dissertação está organizada em seis capítulos, conforme descrito a seguir:

**Capítulo 1: Introdução**
Este capítulo apresenta o contexto do projeto, a importância das técnicas de *Deep learning* no controlo de robôs móveis, e define os objetivos e a estrutura do projeto.

**Capítulo 2: Estado da arte**
Revisa a literatura sobre inteligência artificial, redes neuronais e *Deep Learning*, com foco nas aplicações na robótica. Analisa comparativamente técnicas tradicionais de controlo e AGV existentes, e discute desafios futuros.

**Capítulo 3: Plataformas de robôs móveis**
Analisa diferentes plataformas de robôs móveis já disponíveis no mercado, com as suas diferenças, vantagens e desvantagens, terminando com a escolha justificada de um destes para a realização do projeto.

**Capítulo 4: Desenvolvimento do sistema**
Expõe numa primeira parte os componentes físicos do veículo escolhido realçando as características técnicas de cada um. E numa segunda parte as ferramentas e *software* utilizados para o desenvolvimento das experiências.

**Capítulo 5: Implementação e resultados**
Mostra de forma justificada as experiências realizadas neste projeto. Foram implementadas redes neuronais no controlo de um robô seguidor de linha utilizando os diferentes sensores do robô e diferentes redes neuronais. As partes essenciais são destacadas com o código desenvolvido junto da obtenção e discussão de resultados.

**Capítulo 6: Conclusões**
Por fim, na conclusão são expostas as vantagens e desvantagens das duas experiências realizadas junto da comparação delas com meios tradicionais de controlo. Além disso também é exposto potencial trabalho que poderia ser feito para enriquecer os resultados desta dissertação.

# Capítulo 2

# Estado da arte

Neste capítulo apresenta-se o estudo do estado da arte com a introdução dos conceitos de inteligência artificial, redes neuronais e *Deep Learning*, a aplicação, e desafios, destes e de outros sistemas de controlo na robótica. É também realizada uma análise comparativa de várias plataformas de robôs móveis já existentes no mercado.

## 2.1 Inteligência artificial

As redes neuronais e o *Deep Learning* são conceitos de extrema relevância na área da Inteligência Artificial (IA). A inteligência artificial tem por base disciplinas como ciência da computação, psicologia, linguística, matemática e engenharia. É desenvolvida através do estudo do funcionamento do cérebro humano, tendo especial foco os métodos de aprendizagem e de decisão face à necessidade de resolver um problema. O resultado obtido é, depois, utilizado para desenvolver sistemas de *software* inteligentes [1, 2].

Dentro da IA existem diversas sub disciplinas, tais como:

- *Natural Language Processing* (NLP) - Combina linguística computacional e modelação baseada em regras da linguagem humana com modelos estatísticos e de *Machine Learning* para permitir que computadores e dispositivos digitais reconheçam, compreendam e gerem texto e fala [3, 4]. A Figura 2.1 apresenta a *pipeline* de um destes algoritmos;

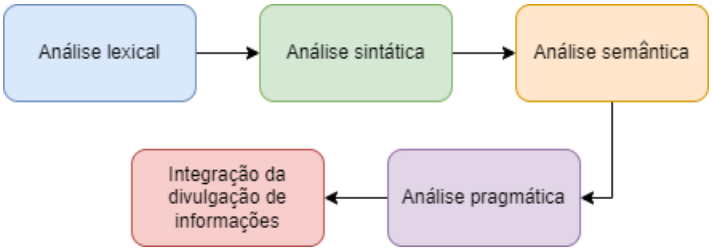



Figura 2.1: *Pipeline* de um algoritmo NLP.

- *Pattern Recognition* (PR) - é um método de análise de dados que usa algoritmos para reconhecer automaticamente padrões e regularidades nos dados. Esses dados podem ser qualquer natureza, desde texto e imagens até sons ou outras qualidades definíveis. Os sistemas de reconhecimento de padrões podem reconhecer padrões familiares com rapidez e precisão [5, 6]. A Figura 2.2 apresenta a *pipeline* de um destes algoritmos;

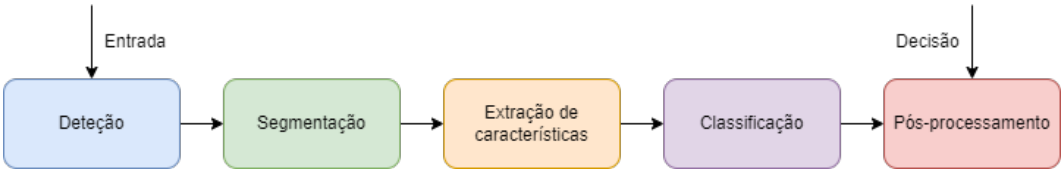



Figura 2.2: *Pipeline* de um algoritmo de PR.

- *Machine Learning* (ML) - Trata do estudo, análise e construção de algoritmos para fazer uma máquina aprender a tomar decisões por conta própria. Os algoritmos de ML usam como entrada dados passados, ou seja, especificamente chamados de dados de treino [1, 7];

- *Artificial Neural Networks* (ANN) - Uma rede neuronal artificial é um programa ou modelo de ML que toma decisões de forma semelhante ao cérebro humano, recorrendo a processos que imitam a maneira como os neurónios biológicos trabalham juntos para identificar fenómenos, considerar opções e chegar a conclusões [8, 9]. Na Figura 2.3 é ilustrado um exemplo de uma destas redes.

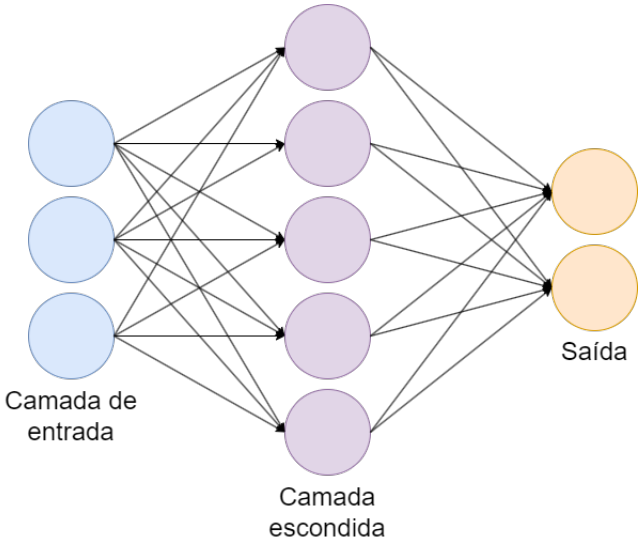



Figura 2.3: Um exemplo de uma rede neuronal artificial.

É importante entender que estes conceitos não são concretamente separados entre si visto que estas sub divisões podem usar métodos e algoritmos de outras.

### 2.1.1 *Machine learning*

O ML é uma tecnologia que permite aos computadores aprender autonomamente a partir de dados e é hoje amplamente utilizada em várias aplicações, como sistemas de recomendação, filtragem de correio eletrónico, identificação de imagens e reconhecimento de voz. Distingue-se dos métodos tradicionais devido aos avanços na tecnologia informática, permitindo que os computadores aprendam sem programação explícita. Ao criar modelos matemáticos a partir de dados históricos, os algoritmos de ML fazem previsões ou tomam decisões. Estes modelos melhoram com mais dados, permitindo previsões mais exatas. O ML simplifica tarefas complexas através da análise de grandes quantidades de dados, o que a torna essencial em vários domínios, como os carros autónomos, a deteção de fraudes e os sistemas de recomendação utilizados por empresas como a Netflix e a Amazon [10].

O ML é normalmente separado em três categorias que se diferem no seu tipo de aprendizagem dos seus algoritmos, estas categorias são:

#### Aprendizagem supervisionada

Aprendizagem supervisionada é a técnica mais popular e amplamente utilizada. Na aprendizagem supervisionada, a máquina aprende sob alguma orientação ou supervisão. Todos os tipos de atividades devem ser realizados ou levados a cabo sob supervisão. Inicialmente deve-se fornecer o conjunto de dados de entrada e rótulos conhecidos de acordo com os dados. Estes dados podem incluir qualquer

formato, como dados de imagem, dados de texto, etc. Em seguida, constrói-se um modelo, separando o conjunto de dados em conjuntos de dados de treino e de teste tendo uma variável de saída no conjunto de dados de treino que precisa de ser prevista ou armazenada [11]. A Figura 2.4 mostra um um diagrama exemplificativo da classificação de objetos por aprendizagem supervisionada.

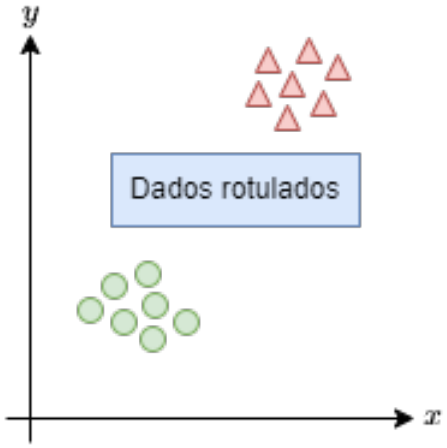



Figura 2.4: Diagrama ilustrativo da classificação de objetos por aprendizagem supervisionada.

Todos os algoritmos aprendem algum tipo de tendência a partir do conjunto de dados de treino e aplicam-na ao conjunto de dados de teste para previsão ou classificação. Quando a variável de saída é categórica, o que significa que existem duas classes, tais como Sim-Não, Masculino-Feminino, Verdadeiro-Falso, deteção de spam, etc., são utilizados métodos de classificação. Existem vários algoritmos de classificação [11, 7]:

- **Classificação binária:** Como o nome sugere, a classificação binária classifica os dados em duas classes. As classes podem ter a forma Sim/Não ou Verdadeiro/Falso ou 1/0, o que significa que uma classe representa o estado normal e outra classe representa o estado anormal;
- **Classificação multi-classe:** A classificação multi-classe classifica os dados em mais de duas classes. Estas classes não são do tipo Sim/Não, Verdadeiro/-Falso ou 1/0. Os dados multi-camada podem ser classificados em categorias especiais como desporto, entretenimento e política, análise de sentimentos, etc;
- **Classificação multi-rótulo:** A classificação multi-rótulo é uma forma geral de classificação multi-classe. Nesta classificação, cada classe pode ser classificada em subclasses, isto é, cada variável de entrada pode ser mapeada para mais de duas ou mais instâncias ou subclasses de uma única classe. Não existe qualquer restrição quanto ao número de classes que podem ser afetadas num

problema multi-rótulo; quanto mais classes existirem, mais complexo se torna o problema [11].

Além dos métodos de classificação, a regressão é uma técnica que tem por objetivo modelar a relação entre várias características e uma variável-alvo contínua. A regressão é uma técnica de ML em que um modelo prevê resultados que são números contínuos. As áreas de domínio em que a regressão é geralmente utilizada são as finanças, o investimento, a previsão, a modelação de séries temporais e a descoberta da relação de causa e efeito entre variáveis. A relação entre a condução imprudente e o número de acidentes rodoviários pode ser melhor analisada utilizando a regressão, etc [11]. A Figura 2.5 compara esta técnica com as técnicas de classificação.

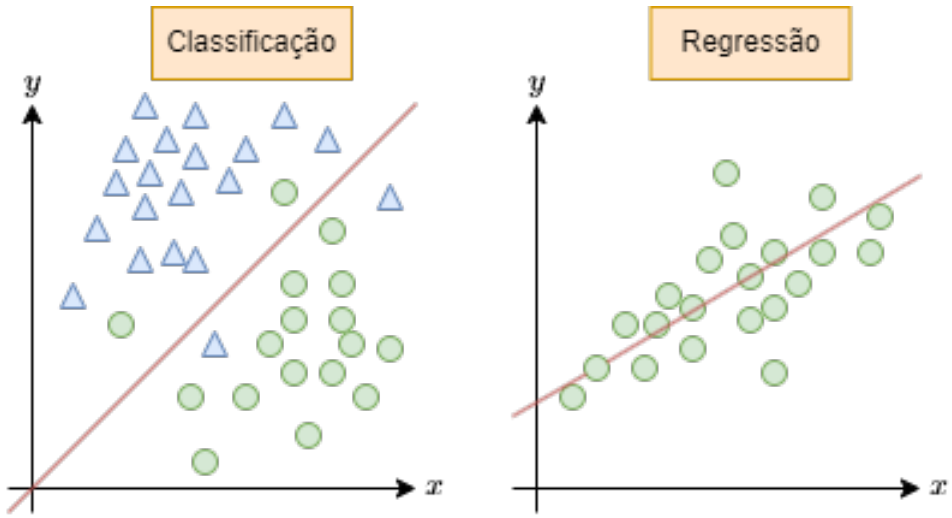



Figura 2.5: Contraste ilustrativo entre métodos de classificação e regressão na aprendizagem supervisionada.

### Aprendizagem não supervisionada

A aprendizagem não supervisionada é um tipo de aprendizagem em que um computador recolhe informações sem qualquer caracterização humana. A máquina é treinada utilizando um conjunto de dados não rotulados, não classificados ou não categorizados, e o algoritmo deve responder de forma independente a esses dados. O objetivo da aprendizagem não supervisionada é reorganizar os dados de entrada em novas características ou numa coleção de objetos com padrões relacionados. Não existe um resultado predefinido na aprendizagem não supervisionada [10].

Quando os dados de formação não são categorizados nem rotulados, são utilizadas técnicas de aprendizagem automática não supervisionada. A aprendizagem não supervisionada investiga a forma como os sistemas podem extrapolar uma função a partir de dados não rotulados para descrever uma estrutura oculta. O sistema nunca pode ter a garantia de que o resultado está correto. Em vez disso, infere qual

deve ser o resultado com base em conjuntos de dados [10]. A Figura 2.6 mostra um um diagrama exemplificativo da categorização de objetos por aprendizagem não supervisionada.


Figura 2.6: Diagrama ilustrativo da categorização de objetos por aprendizagem não supervisionada.

### Aprendizagem por reforço

Um agente de aprendizagem num sistema de aprendizagem por reforço recebe uma recompensa por cada ação correta e recebe uma penalização por cada atividade incorreta. Com a ajuda deste *feedback*, o agente aprende automaticamente e tem um melhor desempenho. O agente explora e interage com o ambiente durante a aprendizagem por reforço. Um agente tem um melhor desempenho porque o seu objetivo é acumular o maior número de pontos de recompensa. Os algoritmos de aprendizagem por reforço interagem com o meio envolvente, realizando ações e identificando sucessos ou fracassos. A aprendizagem por tentativa e erro e as recompensas diferidas são duas das características mais importantes da aprendizagem por reforço. Com a ajuda desta técnica, as máquinas e os agentes de *software* podem selecionar automaticamente o melhor curso de ação numa determinada situação para melhorar o desempenho. O sinal de reforço, que é o *feedback* direto da recompensa, é necessário para que o agente descubra qual a melhor ação [10]. A Figura 2.7 mostra um um diagrama simplificado do funcionamento da aprendizagem por reforço.

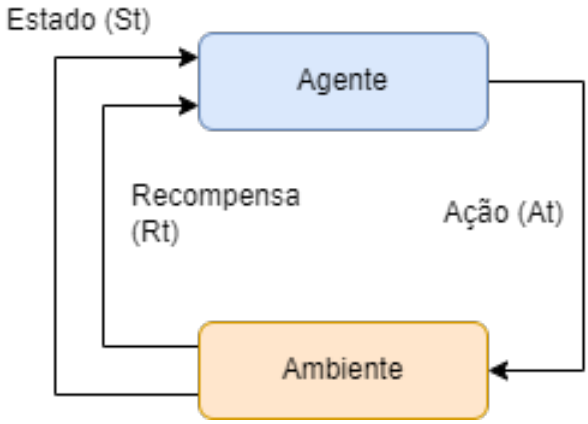



Figura 2.7: Diagrama simplificado do funcionamento de aprendizagem por reforço.

Atualmente, as linguagens de programação mais usadas para aplicações de ML são Python, R, Java e JavaScript, Julia e Lisp [10].

### 2.1.2 *Deep learning*

O *Deep Learning* permite que modelos computacionais compostos por várias camadas de processamento aprendam representações de dados com vários níveis de abstração. Estes métodos melhoraram drasticamente aplicações em reconhecimento da fala, no reconhecimento visual de objetos, na deteção de objetos e em muitos outros domínios, como a descoberta de medicamentos e genoma [12]. O *Deep Learning* descobre estruturas complexas em grandes conjuntos de dados, utilizando o algoritmo de retropropagação para indicar como uma máquina deve alterar os seus parâmetros internos que são utilizados para calcular a representação em cada camada a partir da representação na camada anterior [13].

#### CNN

A *Convolutional Neural Network* (CNN) é uma arquitetura popular de *Deep Learning* discriminativa que aprende diretamente a partir da entrada sem necessidade de extração de características por humanos. Como resultado, a CNN melhora a conceção das ANN tradicionais, como as redes *Multilayer Perceptron* (MLP) regularizadas. Cada camada na CNN tem em conta parâmetros ótimos para uma saída significativa, bem como reduz a complexidade do modelo [14].

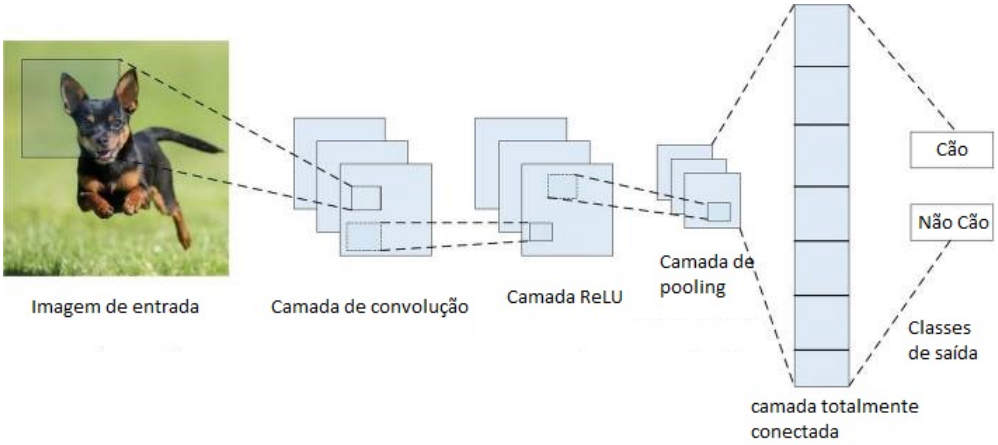



Figura 2.8: Um exemplo de arquitetura CNN para classificação de imagens [15].

As CNN destinam-se especificamente a lidar com uma variedade de formas 2D, pelo que são amplamente utilizadas no reconhecimento visual, na análise de imagens médicas, na segmentação de imagens, no processamento de linguagem natural e em muitos outros domínios. A capacidade de descobrir automaticamente características essenciais a partir da entrada sem necessidade de intervenção humana torna-a mais poderosa do que uma rede tradicional. Existem várias variantes de CNN na área, incluindo o grupo de geometria visual (VGG), AlexNet, Xception, Inception, ResNet, etc., que podem ser utilizadas em vários domínios de aplicação de acordo com as suas capacidades de aprendizagem [14]. Na Figura 2.8 é ilustrado um exemplo de arquitetura CNN para classificação de imagens.

### RNN

As *Recurrent Neural Network* (RNN) são um algoritmo comummente utilizado e conhecido na disciplina de *Deep Learning*. As RNN são principalmente aplicadas no domínio do processamento da fala e em contextos de NLP. Ao contrário das redes convencionais, as RNN utilizam dados sequenciais na rede. Uma vez que a estrutura incorporada na sequência dos dados fornece informações valiosas, esta caraterística é fundamental para uma série de aplicações diferentes. Por exemplo, é importante entender o contexto da frase para determinar o significado de uma palavra específica nela contida. Assim, é possível considerar a RNN como uma unidade de memória de curto prazo, em que $x$ representa a camada de entrada, $y$ é a camada de saída e $s$ representa a camada de estado (oculta). Para uma dada sequência de entrada, um diagrama típico de uma RNN desdobrada é ilustrado na Figura 2.9 [15].

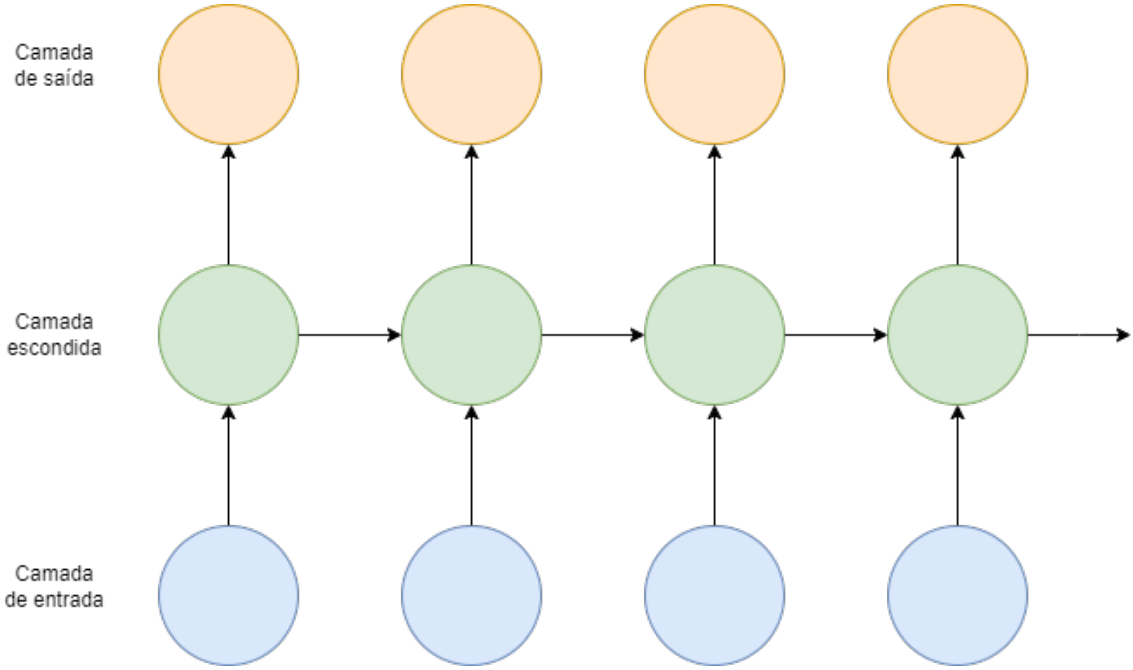



Figura 2.9: Diagrama RNN desdobrado típico.

### LSTM

A *Long Short-Term Memory* (LSTM) é um tipo avançado de RNN desenvolvido por Hochreiter e Schmidhuber [16]. As RNN tradicionais têm dificuldade em aprender dependências de longo prazo devido ao problema do gradiente de desaparecimento. As LSTM resolvem este problema introduzindo uma célula de memória que pode reter informações durante períodos prolongados [16, 17].

#### Principais características do LSTM

- **Célula de memória**: atua como um recipiente para armazenar informação durante longos períodos de tempo;
- **Portas**: Controlam o fluxo de informação para dentro e para fora da célula [16];
  - **Porta de entrada**: Determina que nova informação é armazenada na célula;
  - **Porta de esquecimento**: Controla que informação é descartada da célula;
  - **Porta de saída**: Regula que informação é usada para calcular a saída.

A Figura 2.10 apresenta uma célula mostrando a arquitetura LSTM.

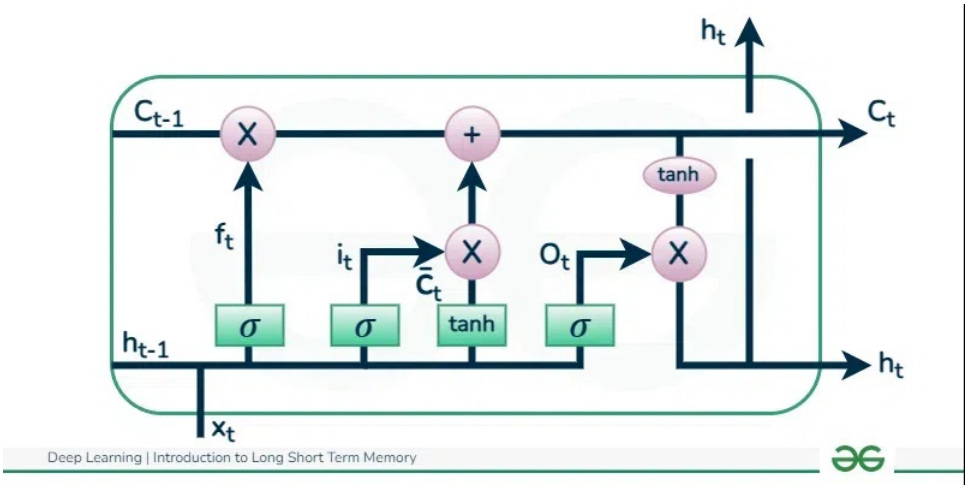



Figura 2.10: Célula LSTM [16].

**Aplicações das redes LSTM**

- **Modelação de linguagem**: As redes LSTM são excelentes em tarefas de processamento de linguagem natural, como modelação de linguagem, tradução automática e resumo de texto. Aprendem as dependências entre palavras para gerar frases coerentes e gramaticalmente corretas;
- **Reconhecimento da fala**: São eficazes na conversão da fala em texto e no reconhecimento de comandos falados através da aprendizagem de padrões nos dados da fala;
- **Previsão de séries temporais**: Estas redes preveem eventos futuros em dados de séries temporais, como preços de ações, condições meteorológicas e consumo de energia, identificando padrões subjacentes. Controladores PID cabem nesta situação;
- **Deteção de anomalias**: Detetam anomalias nos dados, como a deteção de fraudes e intrusões na rede, aprendendo padrões normais e identificando desvios;
- **Sistemas de recomendação**: As LSTM fornecem recomendações personalizadas para filmes, música e livros, aprendendo os padrões de comportamento do utilizador;
- **Análise de vídeo**: Frequentemente combinados com redes CNN, são utilizados em tarefas de análise de vídeo como a deteção de objetos, o reconhecimento de atividades e a classificação de ações [16].

As redes LSTM, com a sua capacidade de captar dependências a longo prazo e processar dados sequenciais de forma eficiente, revolucionaram muitos domínios, incluindo o processamento de linguagem natural, o reconhecimento de voz e a previsão de séries temporais. A sua arquitetura avançada, que envolve células de memória e portas, permite-lhes aprender e recordar padrões complexos ao longo do tempo, tornando-as uma ferramenta essencial nas aplicações modernas de ML e inteligência artificial [16, 18].

#### *Transfer learning*

O *transfer learning* é uma técnica que permite utilizar eficazmente o conhecimento de modelos previamente aprendidos para resolver uma nova tarefa com um mínimo de formação ou de afinação. Em comparação com as técnicas típicas de ML, *Deep Learning* (DL) requer uma grande quantidade de dados de formação. Consequentemente, a necessidade de um volume substancial de dados rotulados constitui um obstáculo significativo à resolução de algumas tarefas essenciais de um domínio específico, em especial no setor médico, em que a criação de conjuntos de dados médicos ou de saúde anotados em grande escala e de alta qualidade é difícil e dispendiosa [19]. Além disso, o modelo DL padrão exige muitos recursos computacionais, como um servidor com GPU. Por conseguinte, o *Transfer learning*, um método de aprendizagem por transferência baseado em DL, pode ser útil para resolver este problema. Atualmente, é especialmente popular em DL, uma vez que permite treinar redes neuronais profundas com muito poucos dados [14]. Um diagrama simplificado deste processo é ilustrado na Figura 2.11.

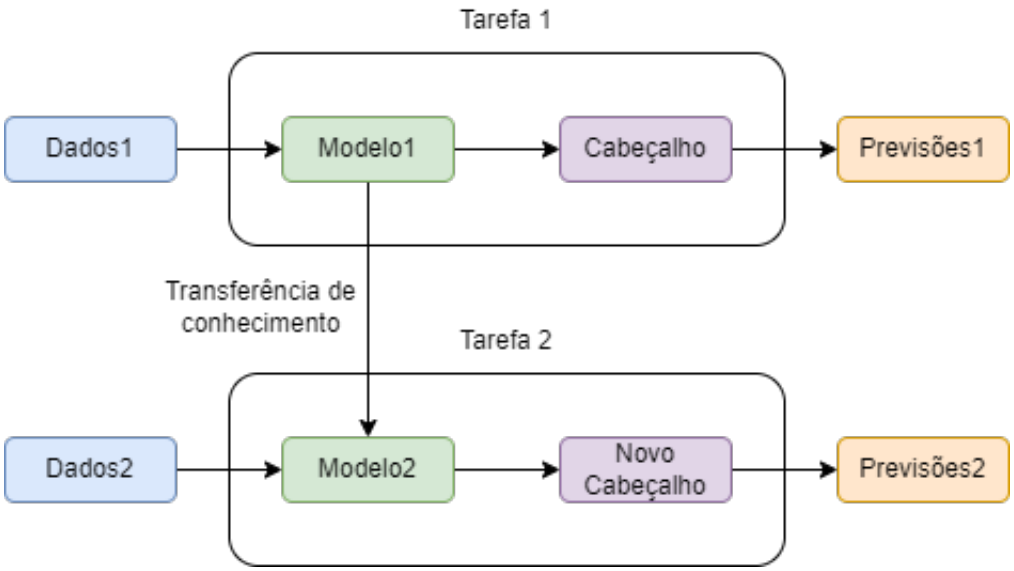



Figura 2.11: Diagrama simplificado do processo de *transfer learning*.

### Arquiteturas CNN

Na última década, as arquiteturas CNN evoluíram significativamente, com avanços importantes na reformulação estrutural, regularização e otimizações de parâmetros. No entanto, as melhorias mais notáveis resultam da reorganização das unidades de processamento e da introdução de novos blocos. Entre esses desenvolvimentos, o aumento da profundidade da rede tem sido fundamental [20].

A Tabela 2.1 apresenta uma panorâmica concisa das arquiteturas CNN mais populares, desde a AlexNet em 2012 até ao modelo de alta resolução (HR) em 2020. A compreensão das suas características, como o tamanho da entrada, a profundidade e a robustez, ajuda os investigadores a selecionar a arquitetura mais adequada para as suas tarefas [15].

Tabela 2.1: Tabela das arquiteturas CNN mais conhecidas e as suas principais características [15].

| **Modelo** | **Principais resultados** | **Profundidade** | ***Dataset*** | **Taxa de erro** | **Tamanho da entrada** | **Ano** |
|---|---|---|---|---|---|---|
| AlexNet | Utiliza Dropout e ReLU | 8 | ImageNet | 16.4 | 227×227×3 | 2012 |
| VGG | Profundidade aumentada, tamanho do filtro reduzido | 16, 19 | ImageNet | 7.3 | 224×224×3 | 2014 |
| GoogLeNet | Aumento da profundidade, conceito de bloco, diferentes tamanhos de filtro , conceito de concatenação | 22 | ImageNet | 6.7 | 224×224×3 | 2015 |
| Inception-V3 | Utiliza um tamanho de filtro pequeno, melhor representação das características | 48 | ImageNet | 3.5 | 229×229×3 | 2015 |
| ResNet | Robustez contra o sobreajuste devido a ligações de saltos baseadas no mapeamento de simetria | 152 | ImageNet | 3.57 | 224×224×3 | 2016 |
| Inception-ResNet-v2 | Introduziu o conceito de ligações residuais | 164 | ImageNet | 3.52 | 229×229×3 | 2016 |
| Xception | Uma convolução em profundidade seguida de uma convolução pontual | 71 | ImageNet | 0.055 | 229×229×3 | 2017 |
| DenseNet | Blocos de camadas, camadas ligadas umas às outras | 201 | CIFAR-10, CIFAR-100, ImageNet | 3.46, 17.18, 5.54 | 224×224×3 | 2017 |
| HRNetV2 | Representações de alta resolução | - | ImageNet | 5.4 | 224×224×3 | 2020 |

### Aplicações de *Deep Learning*

Atualmente, várias aplicações de DL estão generalizadas ao redor do mundo. Estas aplicações incluem os cuidados de saúde, a análise de redes sociais, o processamento

de áudio e de fala (como o reconhecimento e o melhoramento), os métodos de processamento de dados visuais (como a análise de dados multimédia e a visão por computador) e a NLP (tradução e classificação de frases), entre outras. Estas aplicações foram classificadas em cinco categorias: classificação, localização, deteção, segmentação e registo. Embora cada uma destas tarefas tenha o seu próprio objetivo, existe uma sobreposição fundamental na implementação das condutas destas aplicações.

#### *Frameworks* de *Deep Learning*

Nos últimos anos, foram desenvolvidas várias estruturas e conjuntos de dados de DL. Também foram utilizadas várias estruturas e bibliotecas para acelerar o trabalho com bons resultados. Graças à sua utilização, o processo de formação tornou-se mais fácil. Algumas *frameworks* mais conhecidas são:

- Torch para as linguagens de programação C e Lua;
- DL4j para Java;
- MatConvNet para MATLAB;
- TensorFlow, Caffe, MXNet, CNTK e Gluon para C++;
- TensorFlow, Keras, Theano e OpenDeep para Python.

## 2.2 AGV

Os *Automated Guided Vehicle* (AGV), como o nome indica, tratam-se de veículos guiados automaticamente com pouco, se nenhum, controlo humano. Estes podem abranger desde pequenos robôs para fins educacionais a ferramentas de suporte ao fabrico como *Manufacturing Execution System* (MES) e *Transport Management System* (TMS) e até a carros automaticamente guiados [1].

Na década de 1950, foram utilizados pela primeira vez AGV em aplicações industriais. Um exemplo considerado por alguns autores como sendo o primeiro AGV foi produzido em 1953 pela *Barrett Electronics* e foi apelidado de "*Guide-O-Matic*"(Figura 2.12). Tratou-se de uma modificação de um trator de reboque que foi reconfigurado de modo a seguir um fio suspenso num armazém de mercearias. O seu inventor, A.M. Barrett Jr, não lhe atribuiu na época o nome de AGV, mas ainda assim, é creditado na literatura por ter inventado o primeiro veículo automaticamente guiado [21] [22].

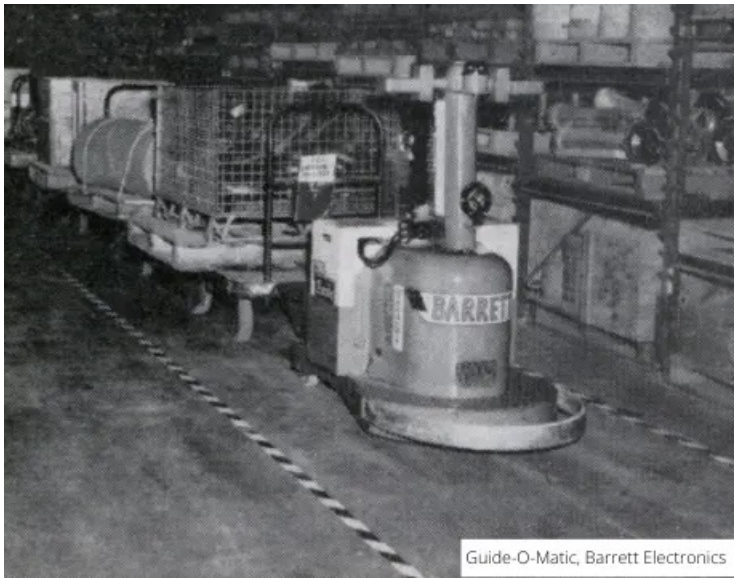



Figura 2.12: *Guide-O-Matic* da *Barrett Electronics* [23].

### 2.2.1 Tipos de sensores e navegação

Os AGV têm que ser capazes de percecionar o ambiente à sua volta através de vários sensores como sonares, LiDAR, infravermelho, câmaras, etc. Para que, junto da incorporação de sistemas de controlo, possam efetuar o objetivo para que foram projetados. Nesta subsecção, vão-se expor os principais sensores e métodos de navegação usados neste tipo de dispositivos.

#### Seguidor de linha

Os AGV seguidores de linha guiam-se através de uma linha física no, ou sob, o chão. Esta linha pode ser uma fita magnética, pintada no chão ou um fio indutivo embutido no solo. Estas diferentes alternativas requerem também diferentes sensores. Com o uso de uma fita magnética ou fio indutivo, o veículo será equipado com sensores capazes de medir as diferenças no campo magnético. Já com uma fita pintada este poderá usar sensores infravermelhos de forma a detetar diferenças na refletividade da radiação infravermelha [22]. Na Figura 2.13 ilustra-se o funcionamento de um AGV seguidor de linha.

**Vantagens**:

- Esta tecnologia de navegação é fiável e precisa;
- Não é afetada por alterações dinâmicas no ambiente, como o aparecimento ou a remoção de obstáculos ou outros veículos [22].

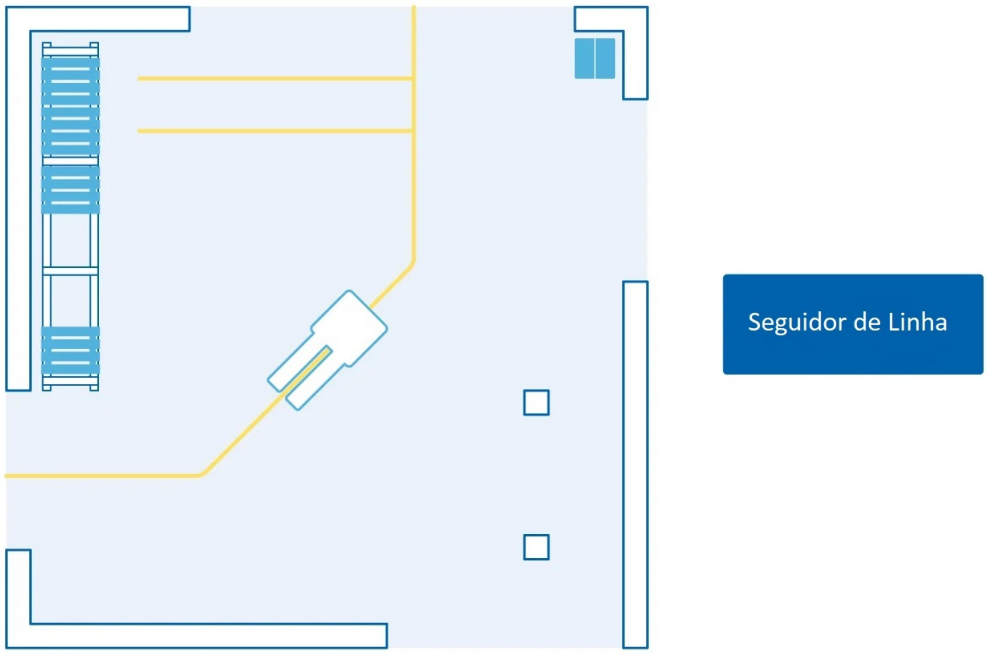



Figura 2.13: Imagem exemplificativa de um AGV seguidor de linha [22].

**Desvantagens**:

- Estes sistemas são demorados de serem instalados e, por consequência, a sua instalação pode ser dispendiosa (a inserção de uma linha física no pavimento é especialmente perturbadora), e este processo terá de ser repetido se for necessário alterar os itinerários;
- A fita magnética ou a tinta podem sofrer desgaste ao longo do tempo, levando ao risco de rotas interrompidas e, portanto, a erros nos veículos;
- A gestão de frotas com veículos que seguem linhas é difícil, um único AGV que segue linhas físicas pode funcionar bem, mas conseguir que vários veículos trabalhem em conjunto, por exemplo, em cruzamentos, requer o desenvolvimento de algoritmos complexos adicionais, para garantir a sincronização e/ou comunicação entre AGV [22].

### Navegação laser

Os veículos guiados por laser calculam a sua posição com a ajuda de feixes de laser. Estes são emitidos em torno do ambiente utilizando *scanners* laser LiDAR posicionados no veículo e refletidos a partir de alvos refletores cuidadosamente instalados no ambiente. Desde que um mínimo de três alvos possam ser detetados em qualquer

altura, o veículo pode calcular a sua posição de forma precisa e fiável por triangulação [22]. Na Figura 2.14 é ilustrado o funcionamento de um AGV com navegação laser.

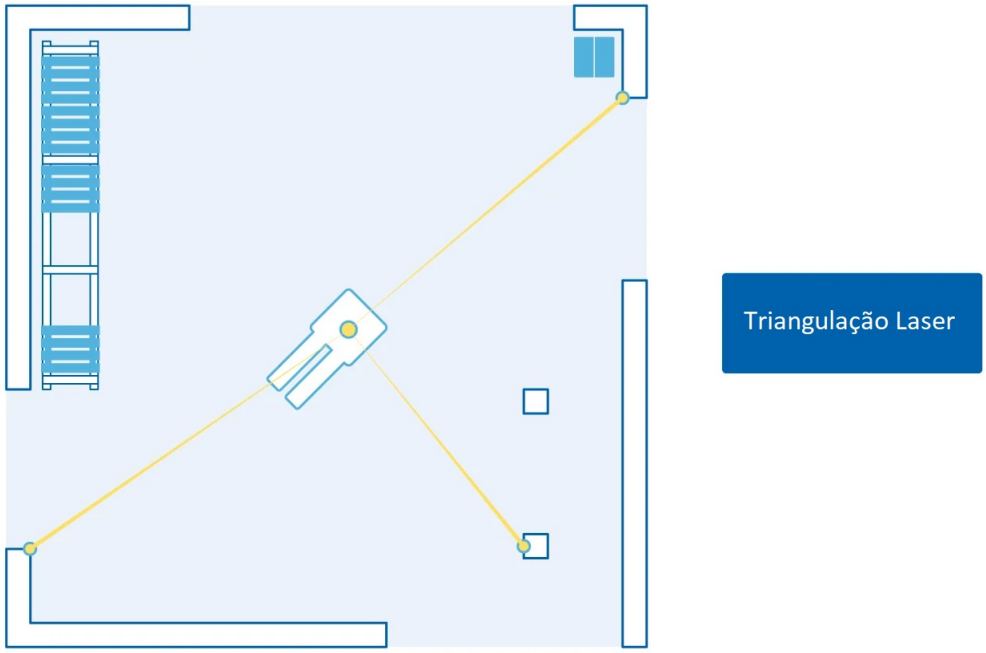



Figura 2.14: Imagem exemplificativa de um AGV de navegação laser [22].

**Vantagens**:

- A triangulação laser é fiável e precisa;
- Uma vez que é utilizado um servidor central para gerir os veículos que seguem percursos virtuais, a gestão de uma frota de vários veículos automatizados é eficaz e eficiente [22].

**Desvantagens**:

- O processo de instalação de refletores e, em seguida, a validação da sua posição pode ser demorado e dispendioso [22];
- Uma vez que as rotas dos veículos são digitais, modificá-las quando uma operação evolui é rápido e simples. A menos que seja necessário alterar a localização das instalações dos refletores, o que pode ser moroso e dispendioso, uma vez que os refletores têm de ser instalados primeiro por um técnico e depois as suas posições medidas por um topógrafo profissional [22].

### Navegação visual

O AGV guiado por visão adquire e processa a textura da imagem obtida pela instalação de uma ou mais câmeras para construir um mapa 3D e localizar-se. É combinado com a navegação inercial e pode ser implementado sem infra-estruturas de orientação. No entanto, a sua precisão é um inconveniente para aplicações industriais complexas [24]. Na Figura 2.15 é demonstrado o seu funcionamento.

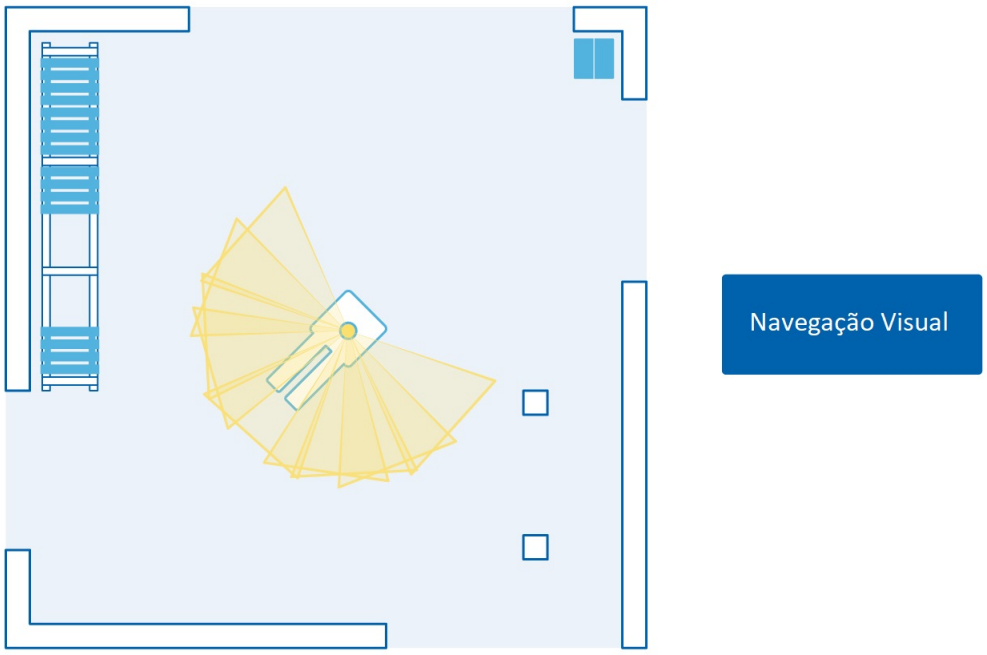



Figura 2.15: Imagem exemplificativa de um AGV de navegação visual [22].

**Vantagens**:

- São rápidos e simples de instalar, sem precisar mudanças na infraestrutura [24].

**Desvantagens**:

- A sua falta de precisão é a maior inconveniência, o que não permite grandes aplicações industriais [24].

### Navegação natural (SLAM)

A navegação natural é fiável, precisa e os seus veículos são rápidos de instalar e modificar. Esta abordagem utiliza características (ou referências) permanentes e estáticas no ambiente, como paredes, colunas e maquinaria fixa para calcular a posição do veículo. Junto dos AGV guiados por laser estes dois tipos são os mais comuns para aplicações industriais [25].

A metodologia mais usada é a navegação *Simultaneous Localization and Mapping* (SLAM) que significa que um AGV com Navegação SLAM é capaz de mapear o seu ambiente e localizar onde se encontra graças à informação recebida do ambiente à sua volta. Estes podem usar sensores como câmeras, lasers e LiDAR e é realizado através de um mapeamento prévio do ambiente para que o veículo seja depois capaz de comparar os dados obtidos com o mapa construído navegando-se desta forma [25]. Por fim, é possível observar na Figura 2.16 uma ilustração do seu funcionamento.

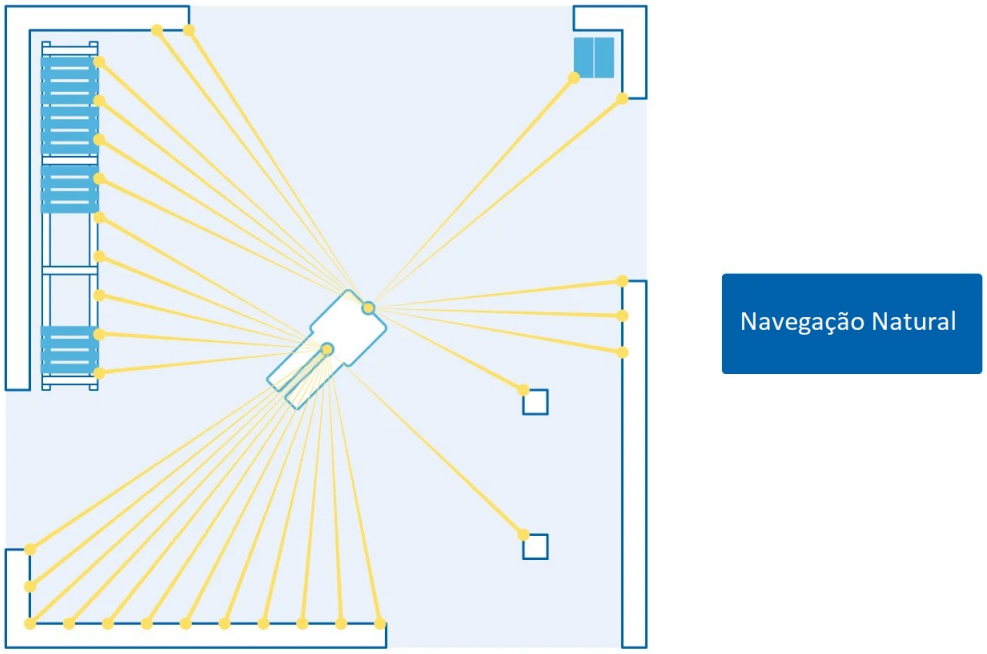



Figura 2.16: Imagem exemplificativa de um AGV de navegação natural [22].

**Vantagens**:

- Veículos SLAM são rápidos de instalar e modificar;
- Uma vez que as trajetórias dos veículos são virtuais, a gestão da frota através de um servidor central é eficiente;
- Não são necessárias alterações permanentes nas infra-estruturas. Ocasionalmente, pode ser necessário adicionar autocolantes refletores (em áreas com poucas características naturais visíveis) mas, ao contrário da triangulação a laser, estes não precisam de ser validados por um inspetor profissional [22].

**Desvantagens**:

- São mais sensíveis a alterações no ambiente (como o aparecimento ou deslocação de equipamentos e materiais). Esta sensibilidade pode fazer com que a precisão e a robustez de um veículo sejam afetadas ao longo do tempo;

- Exigem frequentemente a instalação de um scanner laser adicional na parte superior do veículo para maximizar a visibilidade das características permanentes. Este facto limita o número de tipos de veículos disponíveis com base nesta tecnologia, para além de aumentar os custos [22].

## 2.3 Métodos de controlo

Depois da escolha dos sensores para o AGV é necessário utilizar um método de controlo adequado. Este método de controlo será o responsável por receber as informações obtidas por estes sensores e por calcular um novo valor de saída a ser aplicado ao motores ou direção do veículo. Nas próximas subsecções irão ser discutidos alguns métodos de controlo para este tipo de veículos.

### 2.3.1 Controladores PID

Os controladores de *feedback* são sistemas capazes de controlar uma variável desejada através do uso da própria saída que é medida. O controlador PID começou a ser largamente usado em 1939 e continua a ser um dos métodos mais utilizados em aplicações industriais atualmente devido à sua eficácia e estrutura simples [26, 27].

A nomenclatura Proporcional, Integral e Derivativo (PID) refere-se aos três termos que constituem estes controladores, com isto, entende-se como sendo capazes de tomar em consideração o passado, presente e futuro do erro da leitura da saída para determinar o novo sinal de entrada a aplicar ao sistema [26]. A Figura 2.17 ilustra um diagrama de blocos do funcionamento de um controlador PID, em que os seus componentes são:

#### Termo proporcional

O uso do termo proporcional depende do erro atual, o integrativo do erro passado e o derivativo da previsão do erro futuro. Estes três termos serão depois somados para calcular a nova saída a aplicar ao sistema [27].

Este primeiro termo, o proporcional ($P$), é o mais simples dos restantes e fornece à saída um valor proporcional ao erro, determinado através da multiplicação do erro atual com o ganho proporcional $K_p$. Este termo é utilizado com o intuito de aproximar a variável de controlo com o valor de referência pretendido. Um elevado valor deste ganho provocará uma grande mudança na saída e diminuirá o erro em regime permanente ($e_{ss}$), no entanto, nunca o anulando e pode tornar o sistema instável [27, 26, 28]. Por fim, este pode ser calculado da seguinte maneira:

$$P = K_p.e(t)$$

### Termo integrativo

O termo integrativo depende tanto da magnitude do erro atual como da sua duração [27], representando a variação do erro ao longo do tempo [26]. É calculado através do integral do erro, isto é, através da soma dos erros anteriores, este valor é depois multiplicado pelo ganho integrativo ($K_i$) e somado à saída [26]. Um elevado valor deste ganho pode provocar oscilações ao sistema piorando a resposta transitória [27] mas anulando o erro em regime permanente [26, 29]. O uso mais comum deste termo é em conjunto com o termo proporcional, correspondendo a um controlador PI [28]. Por fim, o termo integrativo pode ser calculado da seguinte maneira:

$$I = K_i \int_0^t e(\tau)\, d\tau$$

### Termo derivativo

Por fim, o termo derivativo é obtido através do declive da resposta ao longo do tempo e multiplicando-o ao ganho derivativo ($K_d$) [26]. Este irá abrandar a taxa de variação na saída do controlador, desta maneira tornando-o mais estável, reduzindo o *overshoot* e melhorando a resposta transitória [27]. Este termo pode ser calculado da seguinte maneira:

$$D = K_d . \frac{d}{dt} e(t)$$

Estes controladores podem ser representados através da seguinte equação:

$$u(t) = K_p(t) \left( e(t) + \frac{1}{T_i} \int_0^t e(\tau)\, d\tau + T_d \frac{de(t)}{dt} \right)$$

Onde $u$ representa o valor de saída do controlador, $e$ o erro, que por si é a diferença entre o valor de referência desejado e o valor atual da variável a controlar, $K_p$ o ganho proporcional, $T_i$ a constante de tempo integral, $\tau$ o parâmetro de integração (assume valores de 0 a $t$) e $T_d$ a constante de tempo derivativa [30, 28].

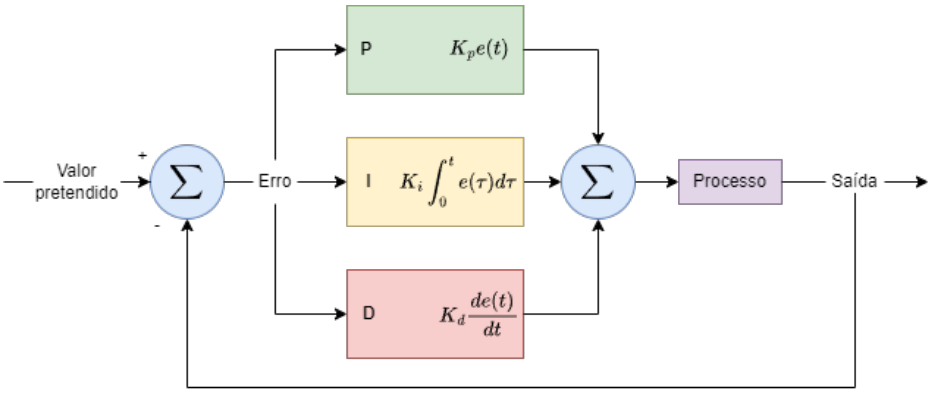



Figura 2.17: Funcionamento de um controlador PID.

A saída de controlo de um controlador PID é a soma dos seus três componentes: P+I+D. Essa saída composta garante uma resposta diferenciada e adaptável, tornando os controladores PID altamente versáteis para uma infinidade de aplicações [30, 28, 31].

### Métodos de sintonia

Como explicado anteriormente, cada termo de um controlador PID tem a sua função e influência no sistema, devido a este facto, escolher os valores dos ganhos $K_p$, $K_i$ e $K_d$ corretos para um funcionamento do sistema desejado pode ser um desafio. A Tabela 2.2 mostra como é que as propriedades do sistema se alteram dependendo do ganho introduzido.

Tabela 2.2: Resposta do sistema alterando os diferentes ganhos.

| **Parâmetro** | **Tempo de subida** | ***Overshoot*** | **Tempo de estabilização** | **Erro em regime estacionário** |
|---|---|---|---|---|
| $K_p$ | Diminui | Aumenta | Pequena alteração | Diminui |
| $K_i$ | Diminui | Aumenta | Aumenta | Diminui significativamente |
| $K_d$ | Diminui ligeiramente | Diminui ligeiramente | Diminui ligeiramente | Sem efeito |

Para a sintonia destes controladores existem bastantes métodos diferentes que permitem uma rápida configuração destes parâmetros.

### Manual

Como o nome indica, este método de sintonia baseia-se na escolha manual destes parâmetros. Mesmo que este seja o método mais simples de execução visto que não tem nenhuma análise matemática envolvida, se não for feito por alguém experiente pode precisar de bastante tentativa e erro para chegar a um resultado satisfatório [27].

### Ziegler-Nichols

Outros métodos de sintonia podem fornecer uma calibração rápida do sistema sem necessidade de alguém experiente, no entanto, estes necessitam de uma análise prévia do sistema para determinar os parâmetros deste, tal como o ganho unitário ($K$), a constante de tempo ($\tau$) e o tempo de atraso ($T_{delay}$) [32].

O método de Ziegler-Nichols é um dos mais usados. Este é executado definindo os ganhos $I$ (integral) e $D$ (derivativo) para zero. O ganho $P$ (proporcional), $K_p$,

é então aumentado (a partir de zero) até atingir o ganho final $K_u$, no qual a saída da malha de controlo oscila com uma amplitude constante e período ($P_u$) [27]. A Tabela 2.3 representa como determinar os valores de $K_p$, $T_i$ e $T_d$ dependendo do controlador e especificações do utilizador.

Tabela 2.3: Método Ziegler-Nichols.

| Método de Ziegler-Nichols | | | |
|---|---|---|---|
| **Método de controlo** | $K_p$ | $T_i$ | $T_d$ |
| **P** | $K_u$ / 2 | - | - |
| **PI** | $0{,}45K_u$ | $P_u/1.2$ | - |
| **PID** | $0{,}60K_u$ | $P_u/2$ | $P_u/8$ |

### 2.3.2 Cohen-Coon

De igual modo ao método de Ziegler-Nichols, o método de sintonia de Cohen-Coon funciona somente para sistemas de primeira ordem, isto é, simples sistemas lineares que respondem a mudanças de forma exponencial. No entanto, este precisa de mais análise matemática do sistema em questão e tem significativamente melhor desempenho devido ao processo usar mais informação [27]. A Tabela 2.4 indica como determinar o $K_p$, $T_i$ e $T_d$ para os diferentes controladores possíveis onde $\tau_{del}$ é o tempo de atraso e $r$ como sendo $\tau_{del}/\tau$ [33].

Tabela 2.4: Método Cohen-coon.

| | $K_c$ | $\tau_{Int}$ | $\tau_{Der}$ |
|---|---|---|---|
| P | $\frac{1}{rK}\left(1+\frac{r}{3}\right)$ | | |
| PI | $\frac{1}{rK}\left(0.9+\frac{r}{12}\right)$ | $\tau_{del}\frac{30+3r}{9+20r}$ | |
| PID | $\frac{1}{rK}\left(\frac{4}{3}+\frac{r}{4}\right)$ | $\tau_{del}\frac{32+6r}{13+8r}$ | $\tau_{del}\frac{4}{11+2r}$ |

Alguns exemplos de aplicações reais são:

- **Sistemas de controlo de temperatura**: Os controladores PID são amplamente utilizados em sistemas de aquecimento, ventilação e ar condicionado

(HVAC) para manter uma temperatura consistente dentro de um espaço. Regulam a potência fornecida aos elementos de aquecimento ou resfriamento com base nas variações de temperatura;

- **Processo industrial**: Os processos de fabrico geralmente utilizam controladores PID para controlar variáveis como pressão, escoamento e nível nas indústrias química e petroquímica. Esses controladores garantem uma operação precisa e estável;
- **Robótica e controlo de movimento**: Os controladores PID são utilizados em sistemas robóticos e controlo de motores para obter movimentos precisos e suaves. Ajudam na manutenção da posição, velocidade e controlo de binário, contribuindo para a eficiência de robôs industriais e máquinas automatizadas [30, 28].

### 2.3.3 Controlo por malha de controlo cinemático

O controlo cinemático em malha é uma abordagem de controlo utilizada na robótica que se baseia na modelação da cinemática do robô. A cinemática de um robô descreve a relação entre as velocidades das rodas ou articulações do robô e o movimento resultante do corpo deste, sem considerar as forças ou os binários envolvidos no seu movimento.

Esta abordagem é frequentemente utilizada em robôs móveis, como carros autónomos, veículos subaquáticos, *drones* e robôs terrestres, em que o principal objetivo é controlar o movimento do robô para alcançar uma determinada trajetória ou posição desejada no ambiente.

Existem diferentes métodos para implementar o Controlo Cinemático em Malha, mas a ideia geral é calcular comandos de velocidade para as rodas ou articulações do robô com base na cinemática do movimento. Por exemplo, num carro autónomo, o sistema de controlo pode calcular as velocidades individuais das rodas necessárias para virar o veículo na direção desejada e movê-lo para a frente ou para trás.

Uma vantagem do controlo cinemático em malha é a sua simplicidade e eficiência computacional, uma vez que não é necessário considerar a dinâmica interna do robô. No entanto, esta abordagem também tem as suas limitações, especialmente em ambientes complexos ou situações dinâmicas em que as condições podem mudar rapidamente. A Figura 2.18 demonstra um esquema simplificado do controlo cinemático [34].

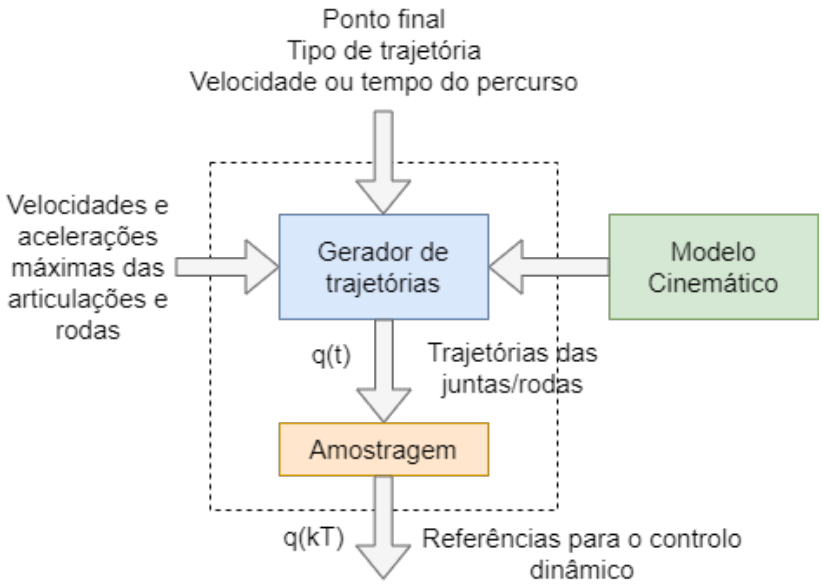



Figura 2.18: Esquema simplificado do controlo cinemático.

### 2.3.4 Controlo Baseado em lógica fuzzy

A lógica *fuzzy* proporciona uma capacidade de gerir melhor com a imprecisão e incerteza do que através da lógica tradicional. Este método utiliza variáveis linguísticas pertencentes a conjuntos *fuzzy* para descrever os 'valores' que uma variável pode tomar. Sendo depois tomada uma decisão de regras base previamente definidas [35, 36, 37].

Estes controladores são especializados em lidar com informação imprecisa ou em tentar implementar um 'raciocínio humano' ao controlo, porém, estes requerem um grande conhecimento no domínio no qual vão ser implementados [35, 36]. Devido ao facto de esta técnica tentar simular ou aproximar ao raciocínio e linguagem humana este método é considerado também uma aplicação de inteligência artificial. Na Figura 2.19 podemos ver o seu funcionamento [38].

Abaixo são apresentados os principais processos e conceitos relacionados aos controladores *Fuzzy*:

- **Conjuntos *Fuzzy***: Em vez de ter conjuntos com fronteiras nítidas, permitem que os elementos tenham graus de pertinência. Por exemplo, em um conjunto de "Altura Baixa", um indivíduo pode ter uma pertinência de 0,3, indicando uma baixa adesão a esse conjunto.

- **Funções de Pertinência**: Cada conjunto *fuzzy* é caracterizado por uma função de pertinência que atribui graus de pertinência aos elementos. Essas funções definem o quão bem um elemento se encaixa num conjunto específico.

- **Regras *Fuzzy***: Definem relações entre conjuntos *fuzzy* de entrada e saída. São expressas na forma "Se (condição), então (ação)", onde as condições e ações envolvem conjuntos *fuzzy*.
- **Fuzzificação**: O processo de traduzir entradas nítidas (valores exatos) em conjuntos *fuzzy*. Isso permite que as variáveis de entrada lidem com imprecisões e incertezas.
- **Inferência *Fuzzy***: Utilizando as regras *fuzzy*, a inferência determina que regras são ativadas com base nos valores *fuzzy* das variáveis de entrada.
- **Combinação de Regras**: Quando várias regras são ativadas, os resultados são combinados para produzir uma saída *fuzzy* agregada.
- **Defuzzificação**: O processo de converter as saídas *fuzzy* agregadas em valores nítidos. Isso é essencial para a implementação prática do controlo [35, 36].

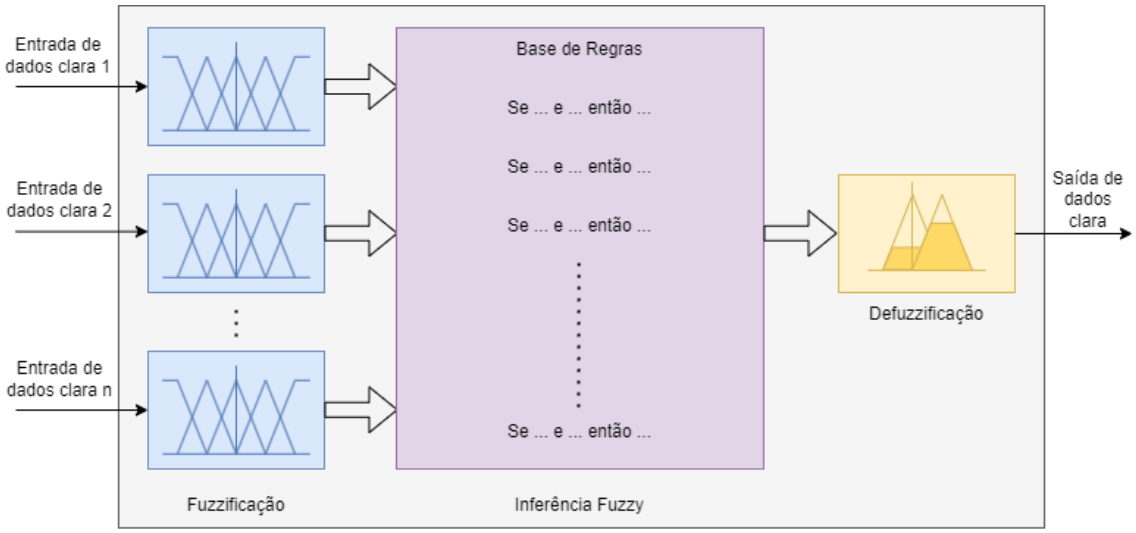



Figura 2.19: Funcionamento de um controlador Fuzzy [39].

Entre outros exemplos de aplicação, temos:

- Controlo de qualidade e tratamento da água;
- Controlo do *cruise control* de um automóvel;
- Controlo de eletrodomésticos inteligentes.

### 2.3.5 Algoritmos de planeamento de trajetória

O planeamento de trajetórias é um processo crucial para que um robô consiga determinar a melhor rota entre dois pontos. Essa rota pode ser escolhida com base em

diversos critérios, como a distância mais curta, a redução de curvas ou a minimização do consumo energético, dependendo das necessidades específicas da aplicação.

Ao longo do tempo, diversos algoritmos foram desenvolvidos e melhorados para lidar com essa tarefa, adaptando-se tanto ao avanço tecnológico quanto às características do ambiente e do próprio robô. A introdução destes algoritmos em áreas como biologia computacional, inteligência artificial em jogos e simulações dinâmicas contribuiu significativamente para o progresso desses métodos.

O planeamento de trajetórias envolve a representação geométrica ou matemática dos caminhos possíveis entre o ponto de partida e o destino, garantindo a evasão de colisões. Inicialmente, é necessário definir um espaço de configuração que abrange todas as possíveis posições e orientações do robô.

Em seguida, a escolha do algoritmo adequado para a busca do melhor caminho é essencial. Existem algoritmos com e sem heurística, cada um com suas características, com heurísticas refere-se a métodos com regras organizadas de forma a chegar a encontrar uma resposta organizada. Entre os algoritmos sem informação estão a Pesquisa em Largura, a Pesquisa em Profundidade e o algoritmo de Dijkstra. Enquanto entre os algoritmos com heurística destacam-se o algoritmo guloso e o A*, juntamente com suas variantes [40].

Em suma, o planeamento de trajetórias é um processo complexo e fundamental para a navegação eficiente de robôs, exigindo a escolha cuidadosa do algoritmo mais adequado às necessidades específicas de cada situação [41].

A Figura 2.20 mostra um exemplo de um algoritmo gerador de trajetórias para determinar o trajeto a tomar pelo robô móvel utilizado por alunos da Universidade de Brasília de forma a encontrar a solução para o labirinto [42].

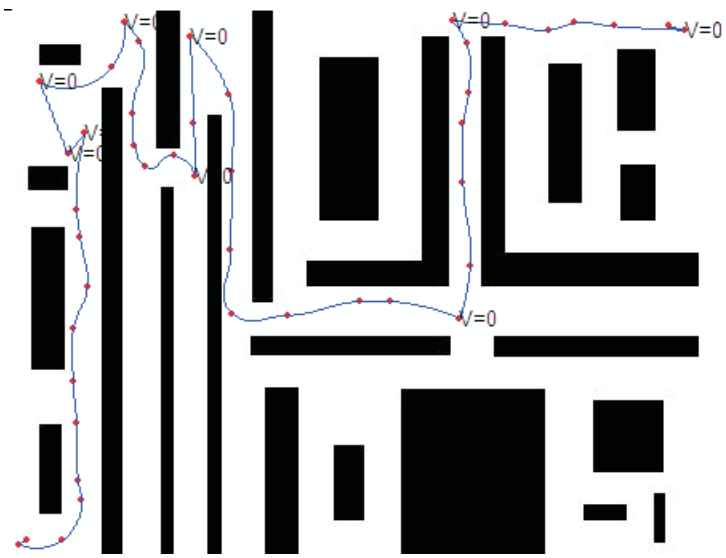



Figura 2.20: Trajetória calculada para solucionar o labirinto [42].

## 2.4 Uso de inteligência artificial em veículos autónomos

O uso de inteligência artificial para o controlo de AGV é um tópico de recente popularidade devido aos avanços atuais de inteligência artificial e os potenciais benefícios dela para o controlo de robôs móveis. De facto, já são bastantes os AGV existentes que beneficiam das potencialidades desta tecnologia. No entanto, a maioria destes não apresenta documentação que explique o seu processo de desenvolvimento, normalmente por se tratarem de veículos desenvolvidos por empresas que naturalmente não partilham esse género de informação.

Nesta secção irão ser explorados alguns projetos que implementam IA para o controlo de AGV junto de alguns desafios inerentes a estas implementações.

### 2.4.1 AGV controlado por navegação visual

O primeiro exemplo foi desenvolvido por Wong Chun Xiang e Soon Chin Fhong na *Universiti Tun Hussein Onn Malaysia* e trata-se de um veículo que pode ser guiado automaticamente ou por *bluetooth* [43].

Este veículo foi construído para o uso durante a pandemia de COVID-19 de forma a impedir o contacto entre superfícies com o objetivo de minimizar o contágio do vírus. Trata-se de um AGV a ser usado em ambientes como hospitais, áreas de escritórios, industrias, escolas, supermercados entre outros e tem o propósito de carregar pequenas cargas de até 10 kg.

Este funciona de forma a navegar atrás do seu operador medindo a sua proximidade com um objeto segurado por este, como indica a Figura 2.21.

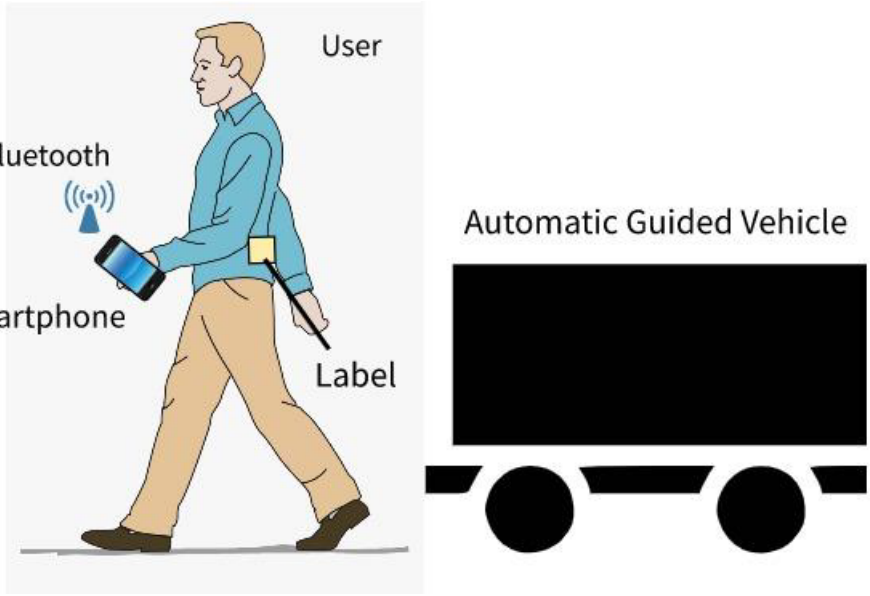



Figura 2.21: Modelo gráfico do AGV controlado por navegação visual.

Para isto, a câmera utilizada foi a HuskyLens com funcionalidades de IA embutidas como reconhecimento facial, rastreio de objetos, reconhecimento de objetos, rastreio de linhas, reconhecimento de cores, reconhecimento de *tags* e classificação de objetos. Este sensor, quando o AGV é configurado para funcionar no modo automático, irá calcular as coordenadas do objeto e dependendo delas irá virar para a esquerda ou direita, ou ir para a frente e para trás. As coordenadas são depois enviadas para o microcontrolador Arduino Uno onde é feito o tratamento de dados e controlo do veículo (Figura 2.22).

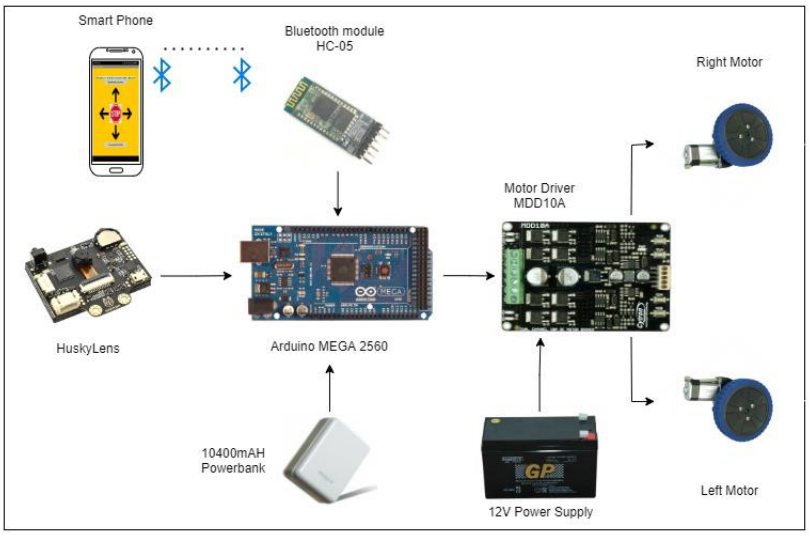



Figura 2.22: Hardware do AGV controlado por navegação visual.

O protótipo funcional é composto pelo circuito de potência e controlo e de uma estrutura que permite que itens sejam carregados (Figura 2.23).

Uma das maiores desvantagens deste projeto é o facto que o alcance do sensor frontal é de apenas 80 cm, o que obriga o operador a estar bastante perto para o comandar. Além disso, este não tem capacidade de detetar o ambiente ao redor pelo que poderia acabar por colidir com paredes. Por fim, ao usar as funcionalidades de IA embutidas do sensor os autores falham em explicar como é que a leitura das coordenadas funciona ao nível da IA utilizada.

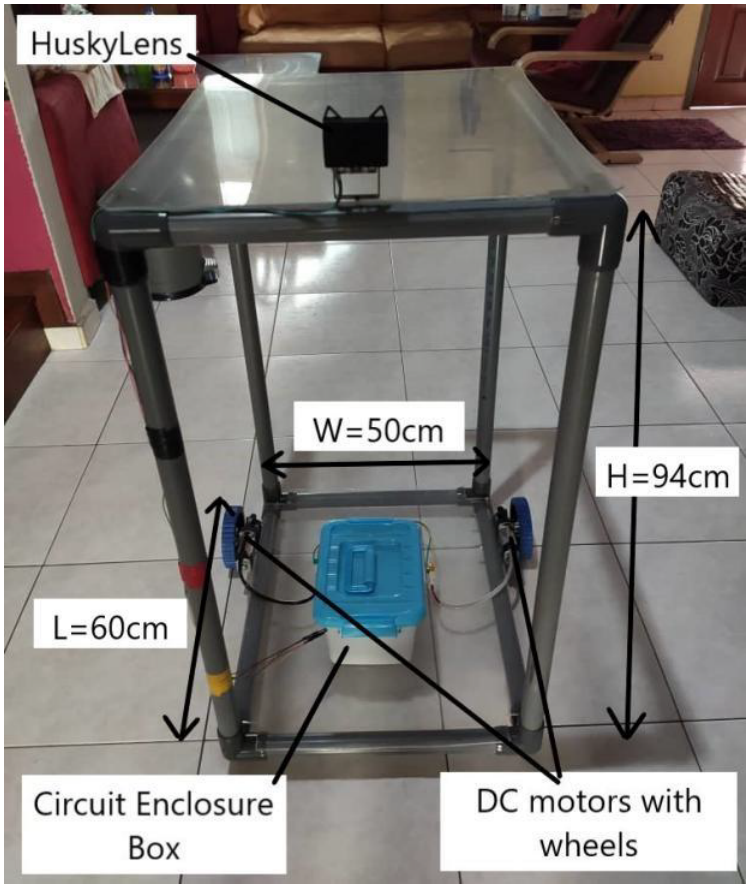



Figura 2.23: Protótipo do AGV controlado por navegação visual.

### 2.4.2 AGV controlado por navegação LIDAR

Este segundo veículo foi desenvolvido por alunos da universidade de National Formosa na Tailândia. O AGV foi desenvolvido de forma a ser usado em supermercados onde iria guiar os clientes a um produto que desejassem ou em aeroportos para levar os passageiros até às respetivas portas de embarque [44]. A Figura 2.24 mostra um possível cenário de funcionamento deste veículo num supermercado.

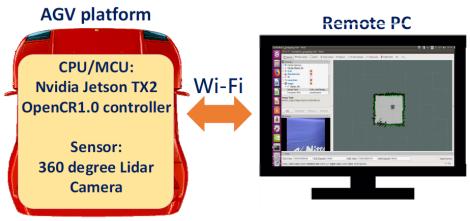
O componente mais importante deste sistema é o sensor LiDAR usado para mapeamento SLAM e modelação por varrimento. Além disso, o veículo também está equipado de uma lente fotográfica que permite obter imagens instantâneas da sua plataforma. Por fim, o *robot operating system* (ROS) é usado para desenvolver e completar a navegação do sistema.

Desta forma, o computador de bordo do veículo, utilizando as informações do LiDAR e da lente fotográfica junto de um mapa do ambiente prévio faz uso de um algoritmo de *Machine Learning* para se movimentar, localizar-se no ambiente e evitar obstáculos e, assim, navegar de forma autónoma [44].

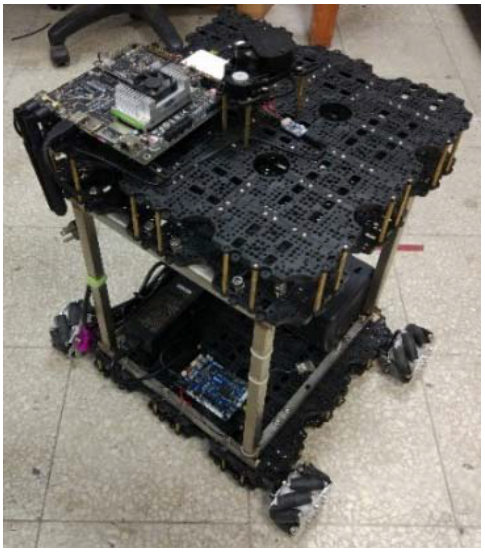
Para isto foi utilizado o robô Turtlebot 3 Waffle Pi robot que é bastante rico em recursos. Este está incluído no dispositivo Nvidia Jetson TX2 e controlador

OpenCR 1.0 (Figura 2.25a) que foi depois montado num formato de mesa para que seja acessível a futuros clientes (Figura 2.25b).

Figura 2.24: Cenário exemplificativo de funcionamento do AGV controlado por navegação LiDAR.




(a) Estrutura do sistema Turtlebot 3 Waffle Pi robot



(b) Protótipo baseado Turtlebot 3 Waffle Pi robot

Figura 2.25: Estrutura (a) e protótipo (b) baseado no robô Turtlebot 3 Waffle Pi robot [44].

Tal como no exemplo anterior, no desenvolvimento deste veículo foram utilizadas as capacidades de IA embutidas no próprio *hardware* obtido, neste caso o Turtlebot 3 Waffle Pi robot, pelo que também falham em explicar em pormenor a sua implementação.

### 2.4.3 AGV auto-evolutivo multimodal orientado para a perceção

Como último exemplo vai ser estudado o AGV desenvolvido por Jamie Roche, Varuna De-Silva e Ahmet Kondoz na Universidade de Loughborough em Londres [45].

Segundo os autores, quando um AGV utiliza redes neuronais convolucionais na deteção de espaço livre, se o conjunto de dados usado para treinar a rede não tiver diversidade suficiente isto pode induzir a riscos de segurança. Além disso, ainda é afirmado que apesar da maioria destes veículos terem um bom desempenho em ambientes estruturados, a necessidade de intervenção humana aumenta significativamente quando confrontados com ambientes não estruturados. Com isso foi desenvolvido um AGV para navegação interior e exterior sem falhas para recolher fluxos de dados multimodais realistas.

O AGV utilizado é composto de vários sensores como LiDAR, câmera frontal e traseira e sonares (Figura 2.26).


Figura 2.26: Loughborough London plataforma autónoma [45].

A navegação é feita através de fotografar o caminho à frente utilizando um algoritmo ML para encontrar espaço livre, depois é utilizado o LiDAR e o sonar para confirmar se o caminho está, de facto, livre. A Figura 2.27 mostra a comparação entre dois algoritmos utilizados onde a verde se mostra o que ambos os algoritmos entendem como sendo espaço livre.

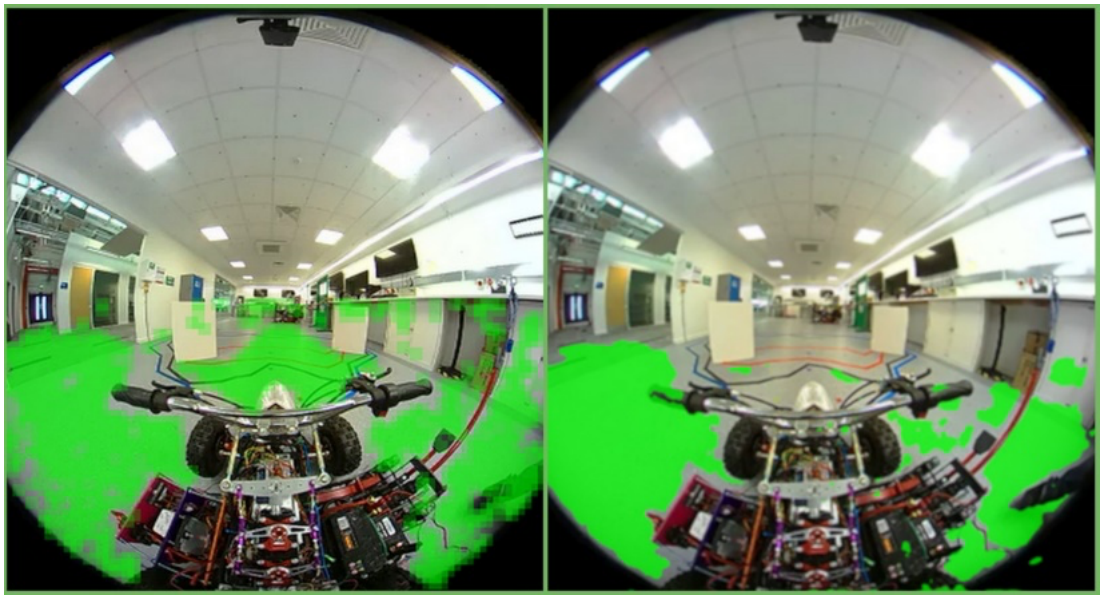


Figura 2.27: Dois algoritmos de espaço livre comparados [45].

### 2.4.4 Dificuldades na implementação de IA em AGV

Para a futura implementação de algoritmos de IA em veículos autónomos é necessário conhecer os desafios inerentes a esta implementação.

#### Problemas no planeamento de rotas e da assistência à navegação

A navegação de AGV em ambientes industriais utiliza vários métodos, cada um com as suas próprias vantagens e limitações. A navegação com fita magnética ou fio indutivo é simples mas inflexível, baseando-se em sensores que seguem rotas designadas marcadas por bandas magnéticas. Os sistemas baseados na visão utilizam câmeras para detetar estas fitas, proporcionando maior flexibilidade, mas ainda dependentes de marcadores físicos.

A navegação por GPS proporciona um posicionamento global, mas sofre de uma menor precisão, especialmente em espaços interiores. A navegação natural baseada em LiDAR, utiliza sensores laser para analisar os arredores, comparando as características detetadas com um ambiente mapeado. No entanto, enfrenta desafios como a confiança limitada devido a novos objetos que não pertençam ao mapa e a diminuição da precisão com a distância e ambientes de baixa refletividade.

O acoplamento a estações de produção requer uma maior precisão, que pode ser alcançada através de sistemas de medição adicionais ou orientação externa. Sensores adicionais, como réguas óticas ou ultrassons, ajudam a uma acoplagem precisa, embora possam ser necessários métodos de fusão de dados para uma precisão ótima. As verificações da posição vertical e horizontal após o acoplamento garantem o alinhamento com as estações de produção, mas aumentam a complexidade em comparação com outras tarefas de navegação [46].

### Problema da não repetibilidade dos dados gerados por um AGV

A mudança do fabrico em massa para a produção de pequenas séries introduz desafios nas operações de AGV devido à variabilidade das tarefas de fabrico. Os sistemas tradicionais baseavam-se em processos estáveis, mas na produção de pequenas séries, as frequentes mudanças tecnológicas exigem que os AGV se adaptem dinamicamente.

Os compromissos entre o tempo, a qualidade e o custo, juntamente com as despesas de integração, afetam a implementação do AGV. Os AGV devem ser flexíveis e auto-adaptáveis para suportar ambientes de fabrico em mudança, oferecendo serviços de transporte abertos que interagem com os sistemas de produção.

No entanto, as tarefas dos AGV na produção de pequenas séries não podem ser facilmente categorizadas devido a fatores ambientais e à autonomia do sistema. A natureza não determinística da navegação dos AGV leva a trajetórias variáveis, mesmo para tarefas semelhantes [46].

Os sistemas de IA devem aprender durante a operação para lidar eficazmente com novas variantes na produção. As anomalias detetadas pela IA exigem uma análise cuidadosa.

### O problema da multiplicidade de algoritmos de IA que processam os dados dos AGV utilizados pelos sistemas de fabrico

A análise da IA dos dados do AGV apoia tanto a operação individual do AGV como todo o sistema de logística interna. Deteta problemas técnicos, otimiza os percursos de navegação e melhora a eficiência energética. Isto inclui autodiagnóstico, manutenção preditiva e seleção dinâmica de caminhos para evitar o congestionamento do tráfego.

No entanto, garantir a partilha ideal de dados para vários algoritmos de IA coloca desafios, levando à multiplicação de dados, inconsistência e sobrecarga do sistema. Para resolver este problema, os fluxos de dados podem ser agrupados e anotados com base no seu significado técnico e contexto de utilização. O *middleware* de comunicação atua como uma ponte entre os dados brutos e a informação estruturada para os algoritmos de IA.

O *middleware* de comunicação baseado em modelos suporta a organização de características e a normalização de dados para a entrada de IA, permitindo a comparação entre diferentes fontes de dados.

Os dados agregados formam registos de produção discretos, descrevendo percursos ou tarefas de transporte. Estes dados, enriquecidos com conhecimentos tecnológicos, servem como uma fonte conveniente para os algoritmos de IA [46].

# Capítulo 3

# Plataformas de robôs móveis

O panorama tecnológico dos AGV está em rápida evolução, com diversas opções disponíveis para aplicações industriais. A escolha do AGV certo é fundamental para aumentar a eficiência e reduzir custos, sendo influenciada por fatores como as tarefas a realizar, o ambiente de operação e as caraterísticas tecnológicas.

Neste capítulo, é realizada uma avaliação de vários AGV no mercado, com o objetivo de selecionar a opção mais adequada para a implementação de um algoritmo de controlo autónomo baseado em IA. Os critérios de seleção incluem a versatilidade, o conjunto de sensores e a compatibilidade com abordagens avançadas de IA.

A análise técnica de cada AGV visa identificar a plataforma mais adequada para o projeto, garantindo uma base robusta para o desenvolvimento e teste do sistema de controlo autónomo.

## 3.1 IADIY Smart Video RC Robot Car

A IADIY é uma empresa profissional de engenharia e integração de tecnologias, centrada no desenvolvimento e fabrico de produtos de fotónica, optoelectrónica, sensores robóticos, sensores IOT e aplicações inovadoras.

O IADIY Smart Video RC Robot Car é um dos produtos disponíveis e este está equipado com uma câmera PTZ de dois eixos, que pode ser utilizada para o controlo remoto de vídeo em tempo real através do telemóvel ou do computador. Com sensores ultra-sónicos e infravermelhos, pode evitar obstáculos, seguir rotas, planear percursos e outras funções, sendo originalmente adequado como material

de aprendizagem para principiantes. Além disso, este modelo é compatível com os processadores Arduino UNO, Raspberry Pi 4B-2G e Raspberry Pi 4B-4G pelo que a escolha terá que ser selecionada na compra do mesmo. Uma imagem deste robô é apresentada na Figura 3.1.

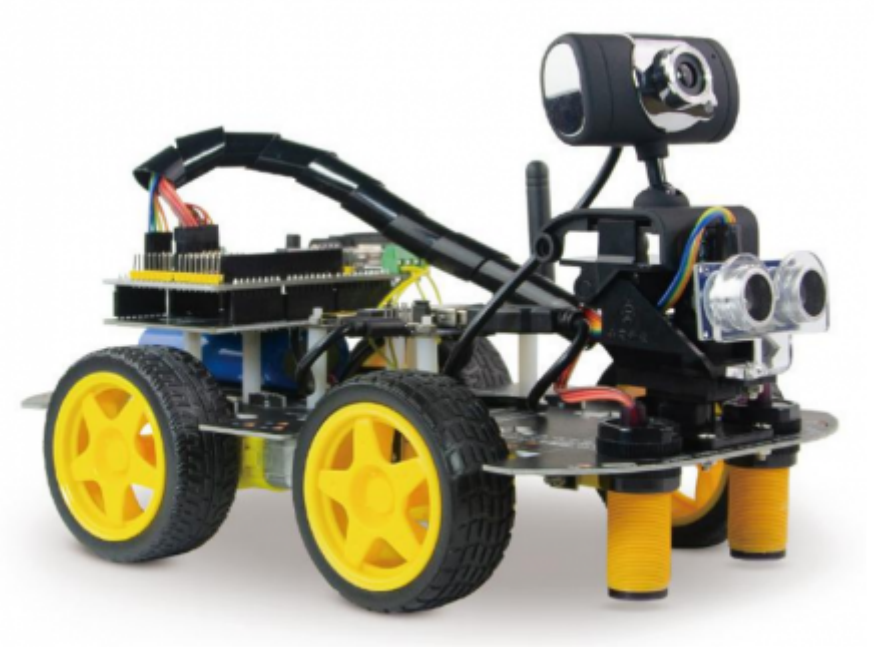


Figura 3.1: Robô IADIY Smart Video RC Robot Car.

Apesar de este produto ser um *kit* compacto com quase tudo o que é necessário incluído, alguns componentes como baterias não estão presentes. Além disso, os sensores infravermelhos não são os comumente utilizados neste tipo de aplicações.

## 3.2 Yahboom Raspbot AI Vision Robot Car

A Shenzhen Yahboom Technology foi criada em 2015 e trata-se de um fornecedor de soluções de inteligência artificial e educação de robôs. É uma empresa chinesa de alta tecnologia que integra pesquisa e desenvolvimento independentes, produção em massa e vendas globais. Com a visão e a missão de reduzir o limiar de criação e popularizar a educação de robôs, a empresa foca-se em desenvolver e inovar a tecnologia de robôs e as aplicações de ensino.

O Raspbot foi especialmente concebido para os principiantes em IA aprenderem IA ao mais baixo custo. É adequado para a placa de desenvolvimento Raspberry Pi 4B (2G/4G/8G). Com uma placa de expansão multi funcional como chassis, equipado com quatro motores TT, uma câmera de alta definição, sensor infravermelho de quatro canais, módulo ultrassónico, etc. Pode ser utilizado como um carro de

controlo remoto ou um *kit* de aprendizagem de IA. À semelhança de outros automóveis inteligentes no mercado, pode executar funções normais de rastreio, usar ultra-sons e infravermelhos para evitar obstáculos e outras funções. A diferença é que este modelo tem incluída uma série de funções de visão de IA para o Raspbot com base no CV de código aberto através da programação Python. O robô pode assim ser controlado por APP, IR e Jupter Lab web. Na Figura 3.2 pode-se ver a estrutura deste veículo.

Figura 3.2: Robô Yahboom Raspbot AI Vision Robot Car.

Tal como o exemplo anterior este trata-se de um veículo *Four-wheel Drive* (4WD), ou seja, é propulsionado por 4 rodas, pelo que dispõe de um motor para cada roda. Além disso também está equipado com o mesmo género de sensores, sendo estes a câmera, sonar e infravermelhos. No entanto, ao contrário do exemplo anterior, este vem incluído com baterias, 4 sensores infravermelhos e uma câmera capaz de resolução até 1920×1080 pixeis. Assim, o microprocessador terá que ser obtido em separado.

## 3.3 SunFounder Smart Video Robot Car PiCar-X

A SunFounder é uma empresa centrada na educação STEAM com produtos como robôs de código aberto, *kits* Arduino e Raspberry Pi, ecrãs de visualização e dispositivos inteligentes. A SunFounder serve o mercado global a partir da sua sede em Shenzhen, China.

O SunFounder Smart Video Robot Car PiCar-X é um dos produtos que esta empresa fornece, é originalmente desenhado para apresentar uma introdução à robótica, programação e eletrónica sendo também adequado a entusiastas e investigadores. Na próxima Figura 3.3 pode-se ver o seu aspeto.

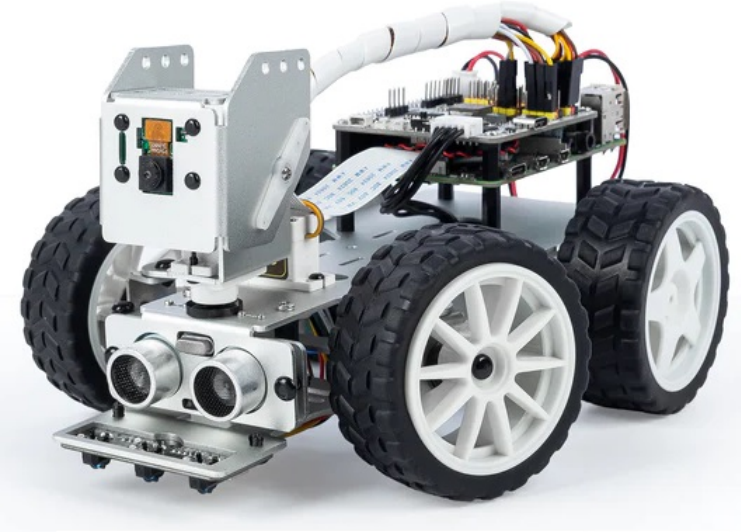


Figura 3.3: Robô SunFounder Smart Video Robot Car.

Trata-se de um modelo bastante semelhante aos dos exemplos anteriores pelo que tem o mesmo género de sensores (câmera, sonar e infravermelhos), além disso é compatível com os microprocessadores Raspberry Pi 4/3B+/3B/Zero W. Adicionalmente é capaz de implementações como uso de IA para deteção de objetos, rostos, cores, gestos, entre outros, evitar obstáculos com o sensor ultrassónico e seguir linhas e evitar penhascos com os sensores infravermelhos. Inclui também um altifalante permitindo funções de discurso texto para fala.

Este modelo contem bastantes tutoriais exemplificativos das diferentes funcionalidades do mesmo, o que facilitaria, num estado inicial, o teste e a validação do funcionamento do veículo. No entanto, este apenas contém três sensores infravermelhos, diminuindo a precisão aquando seguir uma linha, não existe muitas informações sobre a câmera (apenas que esta tem 5 Megapixels) e, por fim, mesmo que o veículo tenha 4 rodas, só existe dois motores pelo que apenas as rodas de trás são alimentadas pelos motores.

## 3.4 OSOYOO Robotic Car for Raspberry Pi

Com sede no Canadá, a OSOYOO concebe e fabrica séries de *kits* de aprendizagem de eletrónica e *kits* de robôs para ajudar os jovens a adquirir experiência prática em *Science-Technology-Engineering-Math* (STEM) utilizando *hardware open-source* como o Arduino e o Raspberry Pi. Os produtos contêm tutoriais pormenorizados, vídeos e códigos de exemplo, o que facilita à aprendizagem.

O OSOYOO Robotic Car V4.0 é um *kit* de aprendizagem versátil, adequado tanto para os entusiastas do Arduino como do Raspberry Pi pelo que se trata da quarta iteração do mesmo. Enquanto as configurações baseadas em Arduino são

óptimas para tarefas básicas, o sistema operativo Linux e o poder de computação do Raspberry Pi permitem projetos mais avançados, como visão computacional e IoT. O *kit* inclui tutoriais para configurar o Raspberry Pi OS, aprender noções básicas de Linux e controlar o carro através de um navegador Web utilizando Python. Também estão disponíveis projetos opcionais para praticar a visão computacional robótica com OpenCV. Em geral, é uma plataforma ideal para estudantes principiantes e intermédios explorarem a robótica. Na Figura 3.4 podemos ver uma imagem do robô referido.

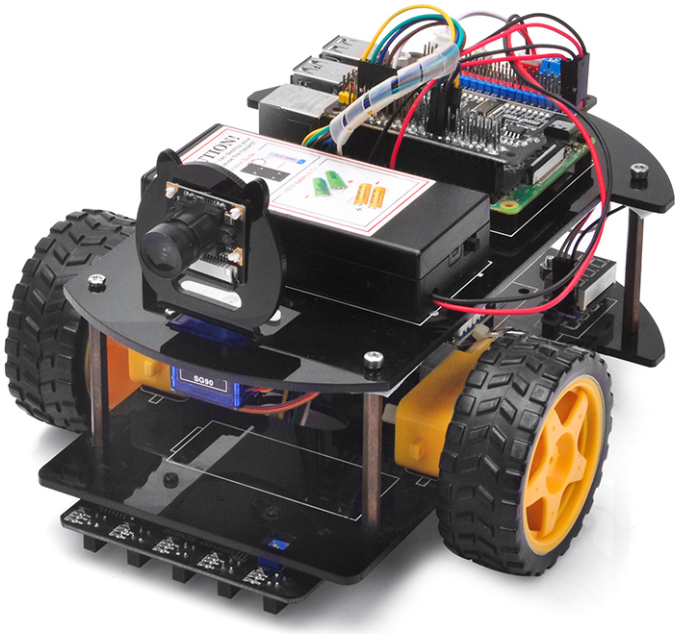


Figura 3.4: Robô OSOYOO Robotic Car V4.0.

Este modelo está equipado com um módulo de 5 sensores infravermelhos, um sensor ultrassónico e uma câmera CSI 1080p de 5 megapixels. Além disso tem também um *step motor* que permite mover a câmera ou o sensor ultrassónico com 180º de liberdade.

Apesar deste exemplo ter um módulo infravermelho de 5 sensores, o que aumentaria a precisão na implementação de seguidor de linha, este tem apenas duas rodas, sendo cada uma alimentada pelo respetivo motor. Além disso, o tamanho mais reduzido faz com que apenas um sensor entre a câmera e o ultrassónico possa ser equipado ao mesmo tempo.

## 3.5 Análise dos robôs móveis

A Tabela 3.1 apresenta as diferentes plataformas de robôs móveis estudados resumindo as principais características de cada um.

Tabela 3.1: Tabela comparativa dos diferentes AGV estudados.

| | **IADIY** | **Yahboom** | **SunFounder** | **OSOYOO** |
|---|---|---|---|---|
| **Sensores** | Câmera<br>Infravermelhos inferiores<br>Ultrassons | Câmera<br>Infravermelhos inferiores<br>e laterais<br>Ultrassons | Câmera<br>Infravermelhos inferiores<br>Ultrassons | Câmera ou Ultrassons<br>Infravermelhos |
| **Módulo infravermelhos** | 2 | 4 | 3 | 5 |
| **Nº de rodas** | 4 | 4 | 4 | 2 |
| **Nº de motores** | 4 | 4 | 2 | 2 |
| **Microcontrolador** | Arduino UNO/<br>Raspberry Pi 4B | Raspberry Pi 4B | Raspberry Pi 4B<br>ou 3B | Raspberry Pi 4 ou 3<br>A+ B+ |
| **Câmera** | 480P USB | 5 Megapixeis 1080P@30FPS/<br>720P@60FPS/480P@90FPS<br>interface CSI | 5 Megapixeis interface CSI | 1080p de 5 megapixels<br>interface CSI |
| **Gimbal câmera** | Sim | Sim | Sim | Sim |
| **Gimbal Ultrassom** | Sim | Não | Não | Sim |

O robô escolhido foi o Raspbot AI Vision Robot Car da YahBoom. Este foi selecionado por representar o *kit* mais completo dos exemplos estudados. Apesar de não ter incluído o microprocessador Raspberry Pi 4, pelo que será também necessário obter separadamente. Contém os sensores necessários para a implementação de um AGV seguidor de linha, sendo estes os sensores infravermelhos e câmera, pelo que também contém um sensor ultrassónico. Adicionalmente, também está incluído de bateria, e carregador, para a alimentação de todos os componentes do veículo. Além disso, trata-se de um veículo 4WD, pelo que permitirá um melhor controlo sobre o mesmo. Apesar de este modelo não apresentar um módulo de 5 sensores infravermelhos igualmente espaçados como o robô da OSOYOO, o módulo de 4 sensores, a distribuição de dois centrais e dois nos extremos, poderá fornecer um controlo mais preciso ao seguir uma linha desenhada no solo.

# Capítulo 4

# Desenvolvimento do sistema

Neste capítulo é realizado um levantamento do *hardware* utilizado, incluindo as características técnicas do AGV e de todos os seus componentes e de como é que estes componentes estão ligados entres si.

Como referido, o robô escolhido foi o Raspbot AI Vision Robot Car da YahBoom. Para complementar este robô foi também escolhido o mini-computador compatível Raspberry Pi modelo 4.

## 4.1 Arquitetura do robô

Nesta secção, vamos explorar a arquitetura do robô Raspbot utilizado, detalhando a forma como estes componentes estão interligados e os tipos de ligações utilizadas. Compreender a arquitetura é crucial para perceber a funcionalidade do robô, o fluxo de dados e a interação entre os componentes de *hardware* e *software*. A Figura 4.1 mostra um diagrama de blocos da arquitetura do sistema.

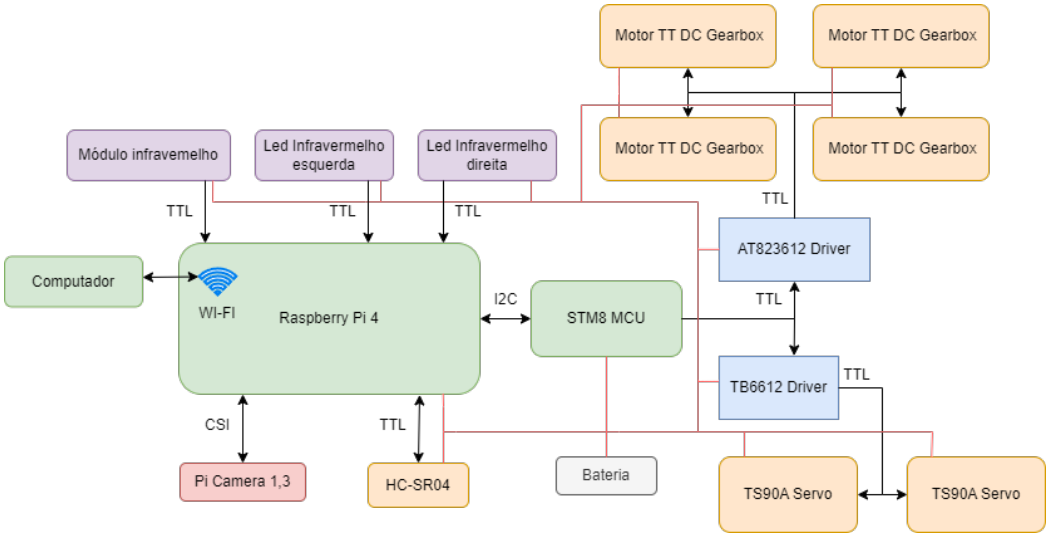



Figura 4.1: Diagrama de blocos da arquitetura do sistema.

Os blocos com a cor verde representam os controladores do sistema com o principal sendo a Raspberry Pi 4. A Raspberry será responsável por controlar todos os componentes do sistema. Para isto é utilizado um conector de 40 pinos sendo que os diferentes componentes, exceto a câmera, serão mapeados para estes *General Purpose Input/Output* (GPIO) controlados através de lógica *Transistor-Transistor Logic* (TTL). Desta maneira o módulo infravermelho de 4 sensores está conectado nos GPIO 0, 2, 3 e 7 (um pino para cada sensor do módulo), os módulos infravermelhos na parte lateral do veículo no 27 e para o sensor ultrassónico o eco é recebido pelo pino 5 e o sinal de *trigger* pelo 4.

Para o controlo dos 6 motores (4 motores TT DC Gearbox e 2 servo motores) a Raspberry Pi irá comunicar através de $I^2C$ com um outro microcontrolador, o STM8. Desta forma a Raspberry Pi irá comunicar com o STM8, que em sua vez irá controlar o *driver* AT823612 para os 4 motores TT DC Gearbox e o *driver* TB6612 para os dois servo motores.

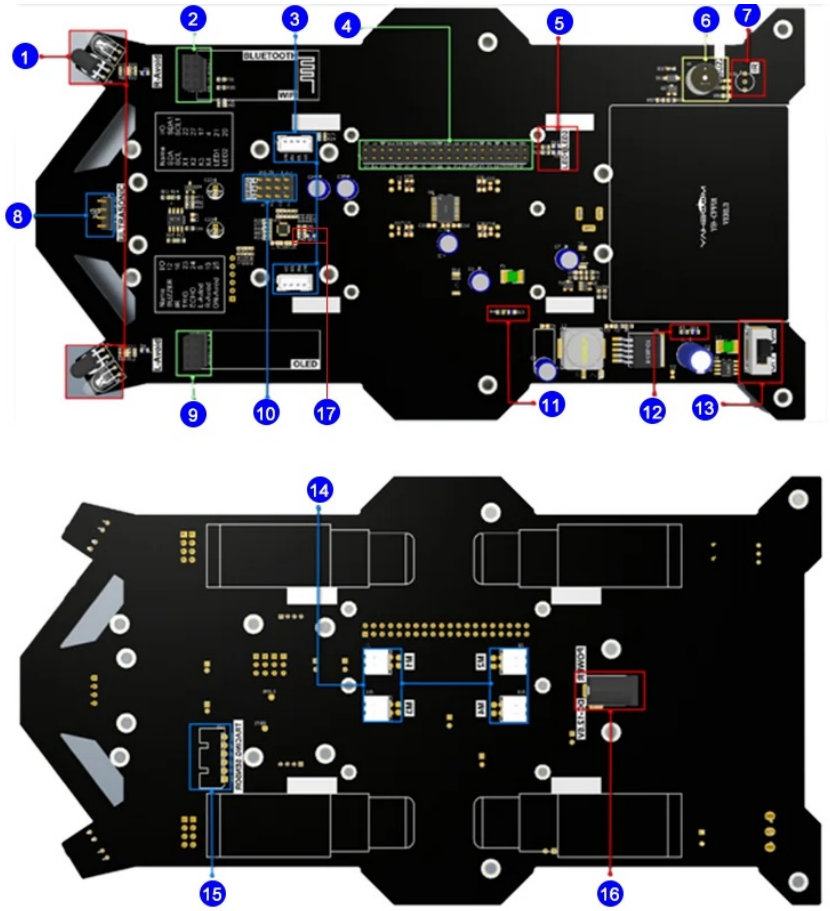
A câmera é alimentada e controlada pelo um conector separado, pela porta da câmera da Raspberry Pi comunicando através do protocolo CSI (*camera serial interface*). Por fim, de forma a programar a Raspberry, transferir e receber dados dela sem ter que a conectar a periféricos como um teclado e monitor é conectado um computador por *wi-fi* dando assim acesso remoto ao sistema operativo da Raspberry Pi através da plataforma JupyterLab.

### 4.1.1 *Board* do Raspbot

O robô Raspbot AI Vision Robot Car, ou apenas Raspbot, é uma opção da Yahboom que permite ao utilizador obter conhecimentos sobre eletrónica, sistemas embebidos,

sensores e atuadores, mas especialmente de conhecimentos sobre *computer vision* e as suas aplicações.

O pacote inclui uma placa de expansão, que serve como chassis do veículo, permitindo suporte para os motores, rodas, sensores e computador, e como placa de circuito, contendo as ligações elétricas entre os vários componentes do veículo junto de alguns elementos passivos e interfaces para os diferentes módulos. Na Figura 4.2 é ilustrado a face superior e inferior deste chassis, respetivamente, junto de uma legenda indicando os diferentes componentes.




① Sensor infravermelho de desvio de obstáculos x2
② Porta série
③ Interface 3I2C PH2.0 x2
④ Interface de 40 pinos para Raspberry Pi
⑤ Can driver LED1 (vermelho), LED2 (azul)
⑥ Buzina passiva Ultrasonic module interface
⑦ Recetor infravermelho
⑧ Interface para o módulo ultrassónico
⑨ Interface OLED
⑩ Interface PWM para servos x4
⑪ Indicador de voltagem de 5V
⑫ Indicador de entrada da bateria
⑬ Interruptor ON/OFF
⑭ Interface para motores DC x4
⑮ Interface do módulo de deteção de linha
⑯ Interface da fonte de alimentação DC
⑰ Indicador do estado do MCU

Figura 4.2: Placa de expansão para o robô Raspbot.

Outras caraterísticas gerais do robô podem ser enumeradas da seguinte forma:

- Proteção contra ligação inversa, sobrecorrente, baixa tensão, proteção e curto-circuito;
- Material composto de fibra de vidro epóxida;
- Dimensões após montagem de 240x150x150 mm;
- Peso do veículo (sem a Raspberry Pi) de 528 g.

### 4.1.2 Motores e rodas

Como mencionado anteriormente, o veículo apresenta um sistema 4WD, significando que apresenta 4 rodas e 4 motores, um motor para cada roda, permitindo um elevado grau de liberdade no seu controlo.

O motor TT DC Gearbox com uma relação de transmissão de 1:48 é uma solução económica para alimentar projetos robóticos. Com fios de 2x200 mm, permite uma fácil integração em circuitos. Funcionando numa gama de tensões de 3 VDC a 6 VDC, a velocidade do motor varia com a tensão, atingindo 120 RPM, 185 RPM e 250 RPM a 3 VDC, 4.5 VDC e 6 VDC, respetivamente, em condições de vazio. A Figura 4.3 mostra o motor em questão.

Figura 4.3: Motor TT DC Gearbox com uma relação de transmissão de 1:48.

Outras características técnicas deste motor são:

- Tensão nominal: 3∼6 V;
- Corrente contínua sem carga: 150 mA +/- 10 %;

- Velocidade mínima de funcionamento (3 V): 90 +/- 10 % RPM;
- Velocidade mínima de funcionamento (6 V): 200 +/- 10 % RPM;
- Binário de paragem (3 V): 0.4 kg.cm;
- Binário de paragem (6 V): 0.8 kg.cm;
- Relação de transmissão: 1:48;
- Dimensões do corpo: 70x22x18 mm;
- Peso: 30,6 g.

Além disso, o veículo contém também dois servo motores TS90A (Figura 4.4), estes motores estão colocados na base da câmera do veículo atribuindo-lhe 180 graus de liberdade tanto horizontalmente como verticalmente, permitindo assim que a câmera possa captar o ambiente ao redor do AGV.


Figura 4.4: Servo motor TS90A.

As especificações destes motores são as seguintes:

- Velocidade padrão: 360 graus;
- Dimensões: 23x12,2x22 mm;
- Peso: 9 g;
- Tensão de funcionamento: 4,8∼6 V;
- Temperatura de trabalho: -17∼+55º C;
- Corrente sem carga: 60 mA;
- Corrente do rotor de bloqueio: 750 mA;
- Sinal de controlo: PWM 50 Hz;
- Velocidade sem carga: 0,09 s/60 graus a 4,8 V;
- Binário de paragem: 1,6 kg-cm (4,8 V).

Já as rodas de robô de 65 mm para motor anterior exposto foi concebida para oferecer maior tração e durabilidade com o pneu de borracha e jante de *nylon*. Com um revestimento de esponja para maior resistência e um *design* de piso para maior fricção, esta roda tem como objetivo melhorar o desempenho em diferentes superfícies. A Figura 4.5 mostra esta roda.

Figura 4.5: Roda de robô de 65 mm para motor utilizado.

Outras características técnicas destas rodas são:

- Capacidade de carga: 2,5 kg;
- Peso: 34 g;
- Diâmetro da roda: 65 mm;
- Largura da roda: 27 mm;
- Material do corpo: Plástico;
- Material do pneu: Borracha.

### 4.1.3 Sensores

O robô está equipado com diferentes sensores como infravermelhos para deteção de obstáculos e precipícios tal como para seguir uma linha desenhada no chão. Um sensor ultrassónico frontal para também desviar de obstáculos e seguir um objeto mantendo uma certa distância. E uma câmera frontal para implementações de inteligência artificial como reconhecimento e rastreio facial, objetos, gestos, cores, etc.

#### Sensores infravermelhos

O veículo apresenta dois módulos de infravermelhos diferentes, dois laterais para deteção de obstáculos na periferia do robô (indicado com o número 1 na Figura 4.2)

e um módulo na parte frontal e inferior para deteção de precipícios e para seguir uma linha desenhada no chão.

Os módulos laterais são compostos de um LED emissor IR e um fototransistor recetor IR como os da Figura 4.6.

Figura 4.6: LED emissor IR e fototransistor recetor IR.

As especificações do LED emissor IR são as seguintes:

- Diâmetro: 5 mm;
- Longitude de onda: 940 nm;
- Tipo de lente: Transparente;
- Ângulo visão: 15-30º;
- Corrente máxima: 20 mA;
- Tensão de trabalho: 1,2 – 1,5V;
- Temperatura de funcionamento: -25 a 80 ºC.

Já o fototransistor recetor IR apresenta as seguintes características:

- Diâmetro: 5 mm;
- Longitude de onda ótima: 940 nm;
- Tipo de lente: Negra;
- Ângulo visão: 15-30º;
- Corrente máxima: 20 mA;
- Tensão de trabalho: 1,4 - 1,5 V;
- Temperatura de funcionamento: -25 a 80 ºC.

Por sua vez, o módulo de rastreio inferior adota quatro sondas de rastreamento e um *layout* de espaçamento otimizado. As duas sondas interiores podem identificar com precisão a linha preta. As duas sondas exteriores podem ajudar a sonda interior a identificar a linha preta e fornecer um aviso prévio. O robô pode assim reconhecer a estrada com curvas de 90 graus e outras pistas complexas. As resistências ajustáveis do tipo botão e os circuitos de amplificação operacionais profissionais permitem ajustar a sensibilidade do sensor. É fabricado pelo próprio fabricante do robô, a Yahboom e foi desenhado para ser utilizado nos próprios veículos. A Figura 4.7 mostra o módulo referido.


Figura 4.7: Sensor de rastreio por infravermelhos de 4 canais da Yahboom.

Outros parâmetros técnicos podem ser enumerados da seguinte forma:

- Tensão de funcionamento: 3,3 - 5 V;
- Temperatura de funcionamento: -10 a + 50 ºC;
- Corrente de funcionamento: 10 - 50 mA;
- Distância de deteção: 1 mm - 10 cm ajustável (recomendado cerca de 0,8 cm);
- Tamanho: 70x29 mm;
- Interface de saída: Interface de 6 pinos, 1234 é um terminal de saída de sinal de 4 canais, + é fonte de alimentação positiva, - é fonte de alimentação negativa; item Sinal de saída: Nível TTL.

**Sensor ultrassónico**

Para determinar a distância de um objeto à frente do veículo foi utilizado o sensor ultrassónico HC-SR04 (Figura 4.8). O HC-SR04 é um sensor de distância ultrassónico composto por um emissor e um recetor, com capacidade de medir distâncias de 2 cm até 4 m, com uma precisão de aproximadamente 3 mm. Este sensor emite sinais ultrassónicos que refletem no objeto a ser atingido e retornam ao sensor, indicando a distância do alvo.


Figura 4.8: Sensor de distância ultrassónico HC-SR04.

As características do sensor HC-SR04 podem ser resumidas da seguinte forma:

- Alimentação: 5V DC;
- Corrente da Operação: 2 mA;
- Ângulo de efeito: 15 graus;
- Alcance: 2 cm $\sim$ 4 m;
- Precisão: 3 mm;
- Frequência de funcionamento: 40 kHz;
- Dimensões: 45x20x15 mm;
- Sinal de entrada de disparo: Pulso TTL de 10 $\mu$s.

### Câmera

Para a implementação das funcionalidades de *computer vision* e inteligência artificial o robô está equipado da câmera Raspberry Pi Camera Rev 1,3 (Figura 4.9). Esta câmera é fabricada pelo Raspberry Pi para os próprios microcomputadores, especificamente para os modelos A e B. Este módulo pode ser diretamente conectado ao modelo utilizado através da porta *Camera Serial Interface* (CSI). Esta câmera possui lente de foco fixo, capaz de fornecer resolução de 2592 x 1944 pixeis para imagens estáticas e 1080p30, 720p60 e 640x480p60/90 para vídeos.

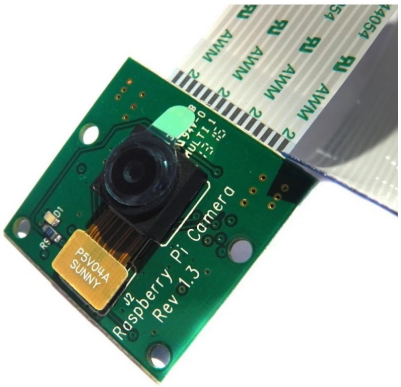



Figura 4.9: Raspberry Pi Camera Rev 1,3.

Mais características técnicas são as seguintes:

- Dimensões: 25×24×9 mm;
- Peso: 3 g;
- Resolução estática: 5 Megapixels;
- Modos de vídeo: 1080p30, 720p60 and 640 × 480p60/90;
- Sensor: OmniVision OV5647;
- Resolução do sensor: 2592 × 1944 pixels;
- Área de imagem do sensor: 3,76×2,74 mm;
- Tamanho do pixel: 1.4 µm × 1.4 µm;
- Foco: Fixo;
- Profundidade de campo: Aproximadamente 1 m a ∞;
- Distância focal: 3.60 mm +/- 0.01;
- Campo de visão horizontal: 53.50 +/- 0.13 graus;
- Campo de visão vertical: 41.41 +/- 0.11 graus.

### 4.1.4 Bateria

A bateria incluída será responsável por alimentar todos os componentes do veículo sendo que a própria placa de expansão está incluída de diferentes conversores para assegurar que a tensão correta seja fornecida aos diferentes componentes. A Figura 4.10 mostra a bateria utilizada.

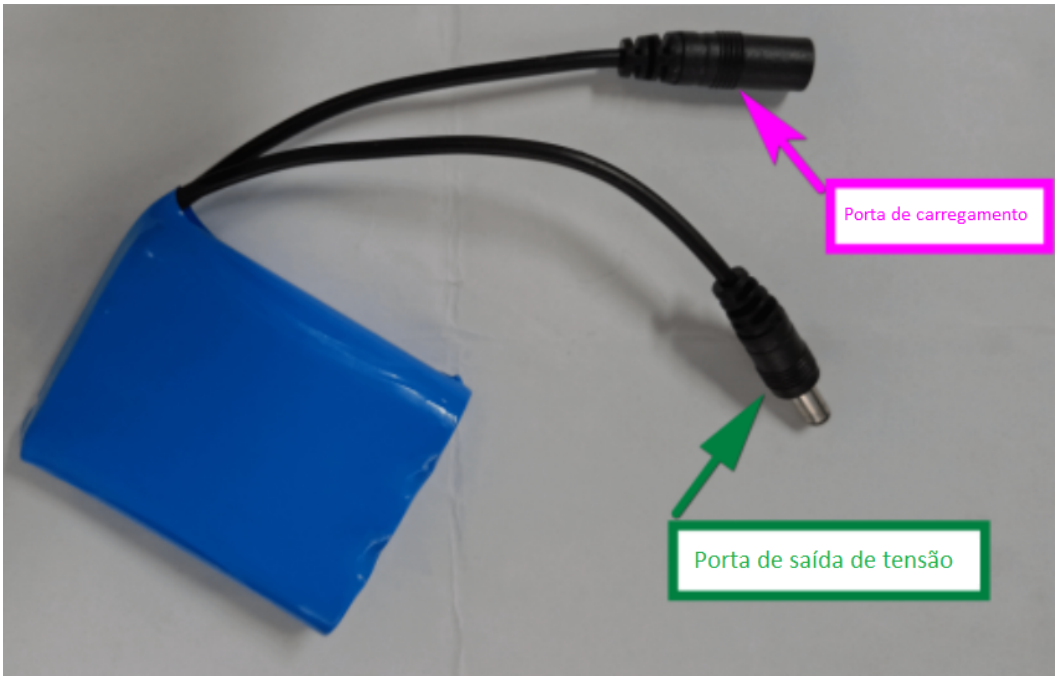



Figura 4.10: Raspberry Pi Camera Rev 1,3.

Poucas informações acerca da bateria estão disponíveis, no entanto as características principais são as seguintes:

- Tensão de saída: 12,6 V;
- Corrente máxima de saída: 6 A;
- Capacidade da bateria: 2200 mAh.

### 4.1.5 Raspberry Pi

Para controlar o veículo, medir e analisar os dados produzidos pelos sensores, determinar novas instruções a serem dadas e enviá-las para os respetivos módulos foi escolhido o minicomputador Raspberry Pi 4 Modelo B.

O Raspberry Pi 4 Modelo B é um computador de placa única do tamanho de um cartão de crédito desenvolvido pela fundação Raspberry Pi. É a quarta geração da série Raspberry Pi e oferece desempenho e recursos melhorados em comparação com os seus antecessores. Com um processador mais rápido, mais opções de memória e suporte para dois ecrãs 4K, o Raspberry Pi 4 Modelo B é um dispositivo versátil e poderoso para vários projetos e aplicações. A Figura 4.11 mostra o computador utilizado.

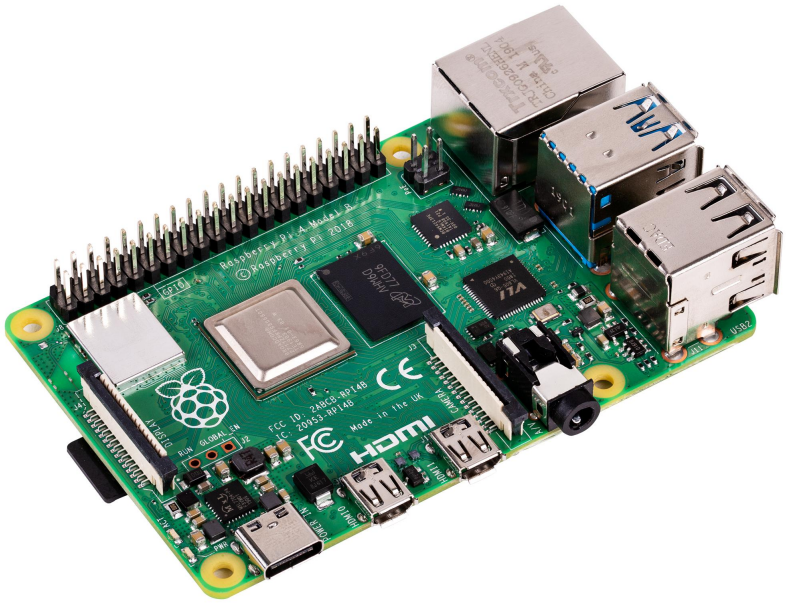


Figura 4.11: Raspberry Pi 4 modelo B.

As especificações deste computador são as seguintes:

- Processador: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1,5GHz;
- Memória: 1GB, 2GB, 4GB ou 8GB LPDDR4 (consoante o modelo) com ECC no disco, para este projeto foi escolhido o modelo de 4 GB por apresentar um bom balanço entre memória e preço;
- Conectividade: LAN sem fios IEEE 802.11b/g/n/ac de 2,4 GHz e 5,0 GHz, Bluetooth 5.0, BLE, Gigabit Ethernet, 2 × portas USB 3.0, 2 × portas USB 2.0;
- GPIO: Cabeçalho GPIO standard de 40 pinos (totalmente compatível com as placas anteriores);
- Vídeo e som: 2 × portas micro HDMI (suportam até 4Kp60), porta de ecrã MIPI DSI de 2 vias, porta de câmera MIPI CSI de 2 vias, porta de áudio estéreo de 4 pólos e de vídeo composto;
- Multimedia: H.265 (descodificação 4Kp60), H.264 (descodificação de 1080p60, codificação de 1080p30), Gráficos OpenGL ES, 3,0;
- Suporte para cartão SD: Ranhura para cartão Micro SD para carregar o sistema operativo e armazenamento de dados;
- Potência de entrada: 5 V DC através do conetor USB-C (mínimo 3A1), 5 V DC através do cabeçalho GPIO (mínimo 3A1), Power over Ethernet (PoE) (requer um HAT PoE separado);
- Ambiente: Temperatura de funcionamento 0-50 ºC.

## 4.2 Ferramentas utilizadas

Nesta secção é mostrado o *software* utilizado para a realização deste trabalho, apresentado aspetos como o sistema operativo instalado no robô, os ambientes de trabalho ou *Application Programming Interface* (API) e as principais bibliotecas utilizadas.

### 4.2.1 Sistema operativo e aplicação

Em sistemas embebidos baseados em microcontroladores menos complexos, como os da família ATMEGA, não é possível instalar um sistema operativo. A Raspberry Pi 4, devido às suas melhores capacidades já expostas, com a sua arquitetura assemelhando-se mais a de um computador convencional, é capaz de suportar diversos sistemas operativos.

A utilização de um *Operating System* (OS) é benéfica pois fornece uma interface de fácil utilização, gere o *hardware* e os recursos de forma eficiente e oferece ferramentas e bibliotecas de desenvolvimento robustas. Simplifica a ligação em rede, aumenta a segurança através da gestão de utilizadores e de atualizações regulares, e suporta a gestão do sistema de ficheiros. Um OS também permite que o Raspberry Pi execute várias aplicações em simultâneo e é personalizável para necessidades específicas, tornando-o uma plataforma versátil e poderosa para várias aplicações, como servidores *Web*, centros multimédia e domótica.

Para este trabalho, foi utilizado um OS distribuído pelo fabricante do próprio robô utilizado, que por si trata-se de uma versão personalizada do sistema operativo distribuído e suportado pela empresa Raspberry para os próprios dispositivos, o Raspberry Pi OS (Figura 4.12), ou antigamente chamado de Raspbian.

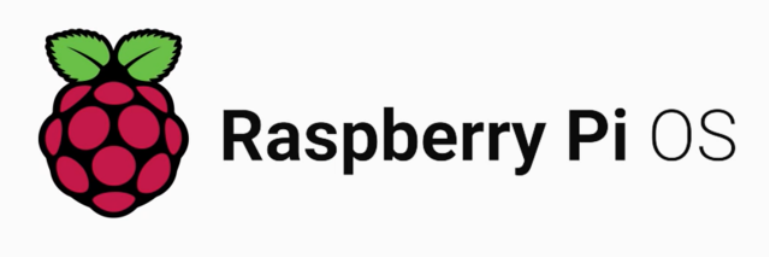



Figura 4.12: Logótipo do Raspberry Pi OS.

O Raspberry Pi OS é um sistema operativo do tipo Unix baseado na distribuição Debian GNU/Linux para a família Raspberry Pi de computadores compactos de placa única. O Raspbian foi desenvolvido de forma independente em 2012, tornou-se o principal sistema operativo para estas placas desde 2013, foi originalmente otimizado para o Raspberry Pi 1 e distribuído pela Raspberry Pi Foundation. A Raspberry Pi Foundation substituiu-o em 2020 pelo Raspberry Pi OS.

O Raspberry Pi OS é suportado em todos os Raspberry Pi exceto no microcontrolador Pico. O Raspberry Pi OS utiliza um ambiente de trabalho LXDE modificado com o gestor de janelas *Openbox stacking*, juntamente com um tema único. A distribuição padrão é fornecida com uma cópia do sistema de álgebra computacional Wolfram Mathematica, VLC, e uma versão leve do navegador *web* Chromium.

Esta versão personalizada do Raspberry Pi OS permite ao utilizador conectar a Raspberry Pi a uma rede Wi-fi sem que seja necessário, posteriormente, conectar esta a um monitor e teclado. Esta conexão é feita através da aplicação de telemóvel YahboomRobot (Figura 4.13) definindo-se a rede a se ligar e a sua palavra-passe, codificando esses dados em formato de um código QR e utilizando a câmera instalada no robô para transmitir e ler esses dados (Figura 4.14).


Figura 4.13: Aplicação de telemóvel YahboomRobot.

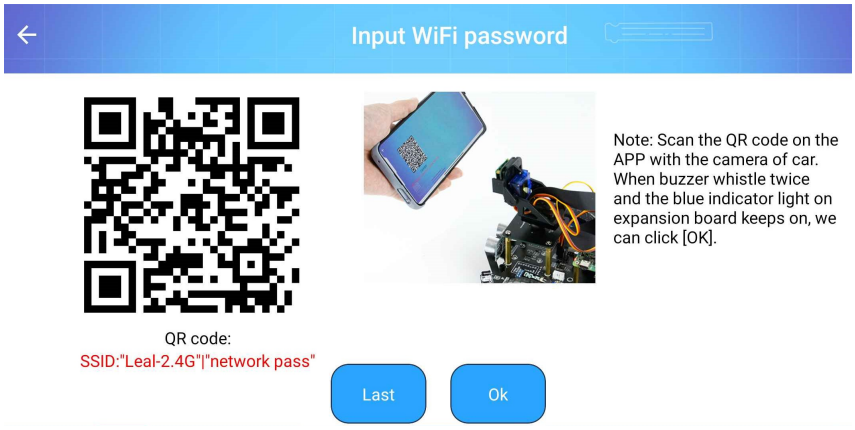



Figura 4.14: Conexão Wi-fi do Raspbot.

Após conectado, ainda através da aplicação, o utilizador pode experimentar várias aplicações práticas disponíveis no OS distribuído. Além disso, com a conexão feita, é também possível aceder de forma remota ao sistema através de um servidor de JupyterLab (Figura 4.15), desta forma o OS permite que o utilizador aceda ao

sistema sem ter que, em qualquer ponto da sua instalação, conectar um monitor, teclado ou outro periférico à Raspberry Pi.

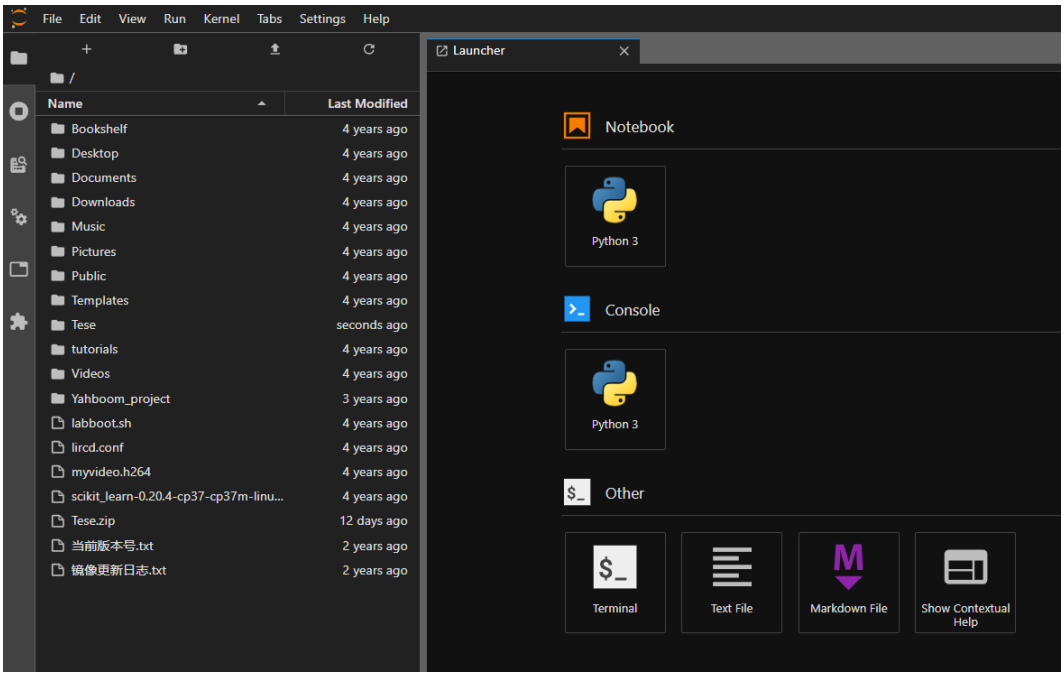



Figura 4.15: Ambiente Jupyterlab.

### 4.2.2 API e outras ferramentas utilizadas

Outra vantagem de um OS é a possibilidade de usar linguagens de programação de alto nível fornecendo bibliotecas e ferramentas robustas. Desta forma, tal como está também presente na figura 4.15, foi utilizado API Jupyter Notebook por estar já incluído no OS.

Jupyter Notebook é uma aplicação Web de código aberto que permite criar e partilhar documentos que contêm código em tempo real, equações, visualizações e texto narrativo. É amplamente utilizado para limpeza e transformação de dados, simulação numérica, modelação estatística, visualização de dados, *machine learning* e muito mais. O Jupyter suporta mais de 40 linguagens de programação, incluindo Python, R e Julia, e integra-se com muitas bibliotecas de ciência de dados para proporcionar um ambiente interativo e de fácil utilização para análise de dados e computação científica. Uma outra grande vantagem desta aplicação é a possibilidade de separar o código em diferentes células, esta capacidade oferece benefícios significativos, como modularidade, interatividade e eficiência. Permite aos utilizadores organizar o código de forma lógica, executar e testar secções individuais de forma independente e documentar o seu trabalho juntamente com o código para uma melhor legibilidade. A Figura 4.16 apresenta o aspeto do API Jupyter Notebook utilizado, onde é possível observar esta divisão entre células.

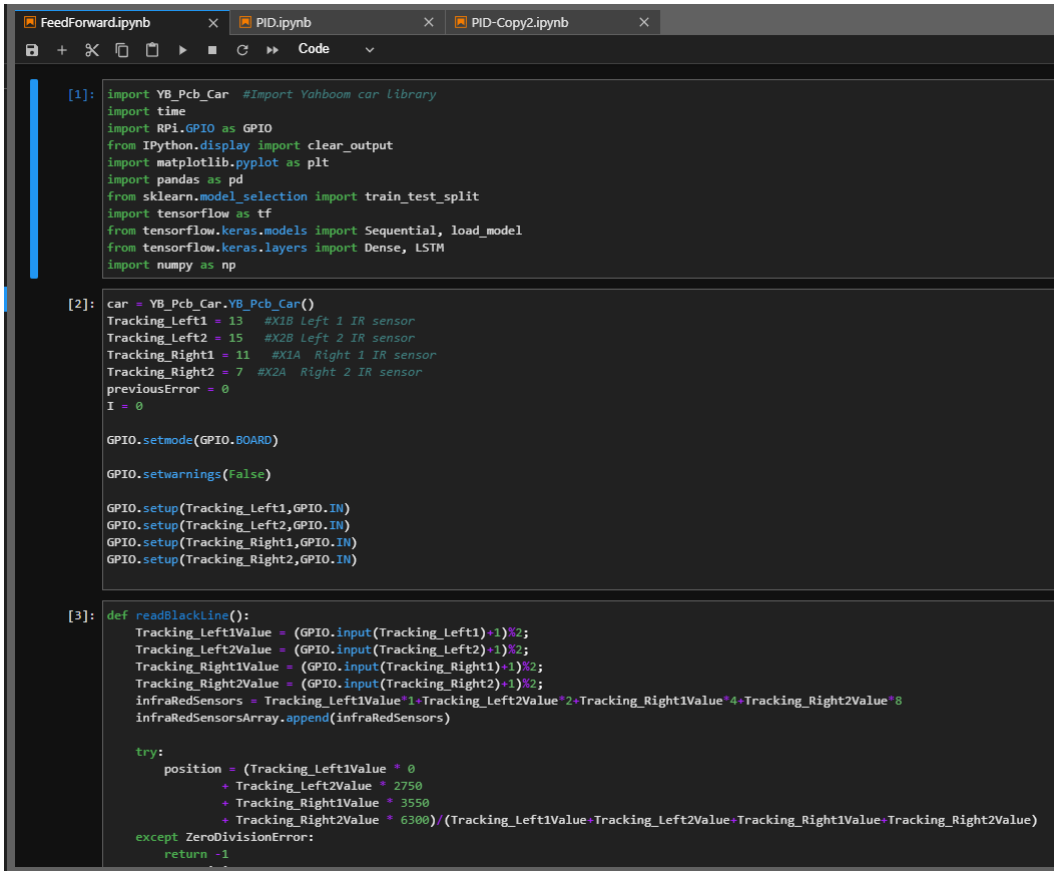



Figura 4.16: Ambiente Jupyter Notebook.

Além disso, para treinar as redes neuronais desenvolvidas, foram transferidos os dados de treino obtidos pelo robô para um outro computador com melhores especificações. Isto permite que o treino da rede se faça mais rapidamente permitindo também que diferentes configurações da rede sejam testadas.

Para isso também é necessário o uso de uma API no computador. Para facilitar a instalação das bibliotecas necessárias para a definição e treino das redes neuronais utilizadas foi utilizado o *software* Anaconda. O Anaconda é uma distribuição popular de código aberto das linguagens de programação Python e R para ciência de dados e aprendizagem automática. Inclui um gestor de pacotes e um gestor de ambiente, facilitando a instalação, gestão e utilização de diferentes versões de Python, R e respetivas bibliotecas. Com isto, foi também instalado o JupyterLab Notebook para o desenvolvimento do código necessário.

### 4.2.3 Linguagem de programação e principais bibliotecas

Para a escolha da linguagem de programação utilizada é importante ter em consideração existentes bibliotecas e recursos de redes neuronais disponíveis para a plataforma escolhida. A escolha recaiu sobre C/C++, MATLAB ou Python pelo facto de estas três serem as mais utilizadas para este género de aplicações.

(a) Logótipo do Python. (b) Logótipo do TensorFlow. (c) Logótipo do Keras.

Figura 4.17: *Frameworks* utilizadas.

Foi escolhida a linguagem Python (Figura 4.17a) pelo maior grau de familiaridade já presente e pela facilidade de visualizar os dados obtidos o que ajudará também no processo de *debugging* durante o desenvolvimento do trabalho.

Junto da linguagem Python, foram também usadas várias bibliotecas que irão permitir a realização do projeto, destas as principais são:

- **YB_Pcb_Car**: Esta biblioteca é da própria Yahboom e está incluída com o OS instalado. Esta irá estabelecer a conexão $I^2C$ necessária para comunicar com os motores e de os controlar, este controlo é feito através de um valor booleano (0 ou 1) para o sentido que o motor deve rodar junto de um valor de 8 bits, ou seja, de 0 a 255 para o valor de PWM do motor;
- **TensorFlow**: O TensorFlow (Figura 4.17b) é uma *framework* de ML de código aberto desenvolvida pela Google, concebida para criar e implementar modelos de ML. Fornece um ecossistema abrangente para a criação de modelos de aprendizagem profunda, suportando uma variedade de arquiteturas de redes neuronais e tarefas de ML. O TensorFlow oferece ferramentas flexíveis para a criação, formação e implementação de modelos em diferentes plataformas, incluindo móveis e Web.
- **Keras**: O Keras (Figura 4.17c) é uma API de redes neuronais de alto nível, escrita em Python e capaz de ser executada sobre o TensorFlow. Simplifica o processo de construção de modelos de *deep learning*, fornecendo componentes modulares e fáceis de utilizar, facilitando a experimentação e o desenvolvimento rápido de modelos.

De notar que devido às limitações da Raspberry Pi e do seu sistema operativo, foi necessário utilizar uma versão antiga do TensorFlow, a versão 1.14.0, pelo que é preciso certificar que algumas das suas dependências tenham versões compatíveis, como as bibliotecas `numpy` e `pandas`, onde foram instaladas as versões 1.16.4 e 1.1.5, respetivamente.

# Capítulo 5

# Implementação e resultados

Neste capítulo, é apresentado o código e as experiências que validam a funcionalidade do robô. Irão ser explicandos os principais *scripts*, bibliotecas e funções, e como funcionam em conjunto. Serão destacadas áreas-chave como a inicialização, o processamento de dados dos sensores, o controlo do motor, treino e uso das redes neuronais.

De seguida, é descrito a configuração experimental e os resultados, apresentando o desempenho do robô.

## 5.1 Controlo através dos sensores infravermelhos

Como primeira experiência, pretende-se controlar o robô para seguir uma linha desenhada no chão através de uma rede neuronal utilizando apenas como entrada da rede os sensores infravermelhos. Para tal é necessário obter primeiramente dados de treino desta rede pelo que surgiram três opções para os adquirir.

A primeira sugestão seria o controlo manual do robô, onde o utilizador será responsável por controlar a velocidade das rodas de forma a manter o veículo centrado com a linha preta. Durante este processo os dados dos sensores seriam guardados para que possam ser utilizados para treinar a rede neuronal. Desta forma a rede neuronal iria imitar o controlo feito pelo utilizador.

A vantagem principal desta abordagem seria a da potencialidade da rede neuronal ser treinada com os próprios comandos do utilizador, de certa forma, imitando o controlo deste, evitando a necessidade futura de um controlo remoto manual. Com

isto, seria apenas necessário o utilizador controlar manualmente o veículo por alguns minutos, de forma a obter os dados de treino, pelo que a rede neuronal se adaptaria a qualquer outro percurso após treinada. No entanto, para o controlo manual do robô, seria necessário o desenvolvimento duma plataforma que permitisse controlar o robô com um grau de liberdade de 360 graus e a diferentes velocidades, para tal, poderia ser criada uma aplicação por *bluetooth* ou *wi-fi*, porém, a complexidade do desenvolvimento de tal aplicação levou a considerar outras opções de aquisição dos dados de treino.

Além disso, o controlo feito por um ser humano não estaria ausente de viés, isto é, distorções ou tortuosidades no modo de conduzir o veículo devido a estímulos exteriores. De facto, seria difícil ao utilizador tomar apenas em consideração a informação dos sensores infravermelhos em tempo real para conduzir o veículo, pelo que teria que observar o circuito enquanto o faz. Com isto, por exemplo, seria natural que o utilizador controlasse o veículo não tomando em grande consideração se os módulos infravermelhos estão a captar, ou não, a linha preta, pelo que se focaria em apenas tentar manter o veículo aproximadamente centrado, ou que, podendo observar o circuito, o condutor poderia se antecipar do trajeto adiante, começando uma curva antes que essa fosse captada pelos sensores. Estas perturbações poderiam levar a que o modelo fosse indevidamente treinado.

A segunda sugestão seria de estabelecer um conjunto de regras utilizadas para a navegação do veículo, regras do género "se e só se o sensor da esquerda for acionado, vira para a esquerda a uma velocidade $x$"e treinar a rede neuronal com os valores dos sensores e velocidade das rodas. Esta é uma opção válida, no entanto, trata-se de um método pouco robusto e bastante simples, pelo que o modelo iria somente aprender tais simples regras. Tal experiência poderia ser realizada com uma rede *feed forward* visto que a saída da rede, a velocidade das rodas, dependeria apenas do valor atual (e possivelmente do valor anterior) dos sensores. Por estas razões procurou-se um controlador mais robusto.

Por fim, pode ser aplicado um controlador já abordado, como um controlador PID ou *Fuzzy*, ao controlo do veículo. De certa forma, o uso destes controladores enquadra-se no mesmo dilema da opção anterior, onde, independentemente do controlador programado, por mais complexo, é essencialmente um conjunto de instruções predefinidas pelo que a rede neuronal iria treinar e simular tais instruções. Entretanto, a utilização de um controlador trará mais flexibilidade e complexidade ao sistema, sendo valioso analisar os resultados da rede neuronal treinada com dados coletados desta forma. Assim, o controlador PID foi o método escolhido para esta primeira experiência. Desta forma, na Figura 5.1 podemos ver um esquema simplificado desta abordagem.

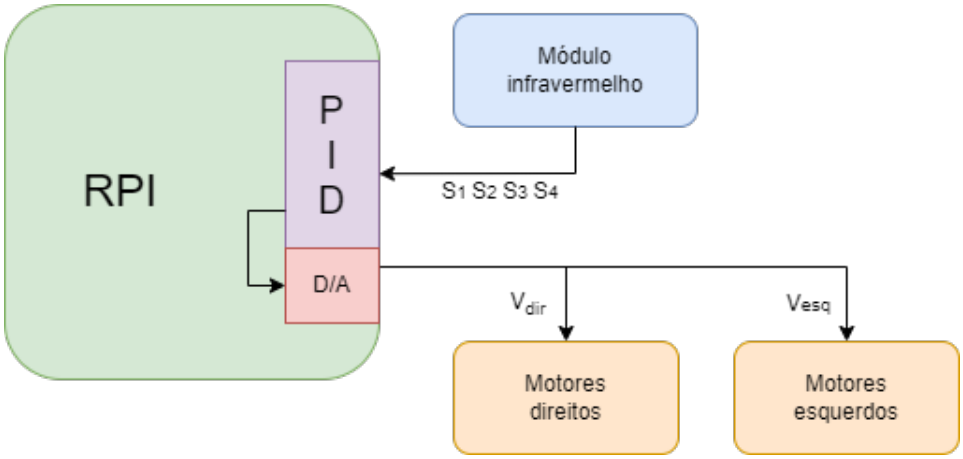



Figura 5.1: Esquema do sistema com o controlador PID.

### 5.1.1 Controlador PID

Os controladores PID tratam-se de controladores de *feedback*, isto é, utilizam a saída anterior para calcular o novo valor da saída. Para isto, estes servem-se de um erro calculado através da leitura dos sensores do veículo e calcula uma nova saída, esta que é separada em três termos, proporcional (P), integrativo (I) e derivativo (D).

Atualmente, os controladores digitais são usados em aplicações de controlo de grande e pequena escala. É agora prática comum implementar controladores PID digitais, o que significa que ele operam no domínio de tempo discreto [27].

Na sua versão digital, o integral torna-se num somatório e o diferencial numa diferença. Para esta implementação é necessário ler o valor da variável a controlar e de aplicar o novo valor de saída do controlador [27].

Através desta implementação, como este passa a trabalhar no domínio de tempo discreto, é preciso alterar a fórmula para o domínio discreto [28]:

$$u[n] = K_p\left(e[n] + \frac{T}{T_i}\sum_{k=0}^{n} e[k] + \frac{T_d}{T}(e[n] - e[n-1])\right) \tag{5.1}$$

onde $T_i$ é a constante integrativa, $T_d$ a constante derivativa e $T$ o período de amostragem. Sendo o período de amostragem constante ($T = 1/80$ segundos) foi implementada a seguinte equação simplificada:

$$u[n] = K_p.e[n] + K_i\sum_{k=0}^{n} e(k) + K_d.(e[n] - e[n-1]) \tag{5.2}$$

em que $K_i = \frac{K_p T}{T_i}$ e $K_d = \frac{T_d K_p}{T}$.

Devido ao facto que os motores elétricos utilizados têm uma resposta temporal bastante rápida não se tornaria prático fazer os testes experimentais para determinar as características do sistema, pelo que os motores também carecem de um *encoder* capaz de medir a velocidade destes. Por esta razão, optou-se pelo método

de sintonia manual em que foram testados bastantes conjuntos de valores até que fossem encontrados ganhos satisfatórios para o controlador PID.

### Cálculo do erro

Como observado na Figura 2.17, primeiramente é necessário calcular um valor atual do erro. Este erro será a subtração de um valor pretendido, com o valor atual medido pelos sensores. Desta forma, para calcular o valor do erro, vai-se admitir que este é a distância a que o centro do módulo infravermelho está da linha preta. Ou seja, quão mais afastado o sensor estiver da linha, maior será o módulo do erro.

Para calcular a posição aproximada da linha preta debaixo dos sensores vai ser utilizada a função *readBlackLine()* (Figura 5.3).

A estimativa é feita utilizando uma média ponderada dos índices dos sensores multiplicada por um certo ganho. Somente para exemplo, assumindo esse ganho como 1000, desta forma, um valor de retorno de 0 indica que a linha está diretamente abaixo do sensor 0, um valor de retorno de 1000 indica que a linha está diretamente abaixo do sensor 1, 2000 indica que está abaixo do sensor 2, etc. Valores intermédios indicam que a linha se encontra entre dois sensores. A fórmula é (onde $v_0$ representa o valor do primeiro sensor):

$$Position = \frac{(0 \times v_0) + (1000 \times v_1) + (2000 \times v_2) + ...}{v_0 + v_1 + v_2 + ...} \tag{5.3}$$

Desde que os sensores não estejam demasiado afastados em relação à linha, este valor devolvido foi concebido para ser monotónico, o que o torna excelente para utilização no controlo PID em circuito fechado.

No exemplo anterior, por motivos de clareza, utilizaram-se os ganhos como sendo 1000, no entanto, estes valores do ganho devem ser justificadamente escolhidos. De facto, os ganhos utilizados poderiam ser escolhidos de forma arbitrária pelo que seria necessário ajustar os ganhos do controlador correspondentemente, no entanto, como os 4 sensores não estão espaçados igualmente entre si, a solução que foi encontrada foi a de corresponder os ganhos à distância entre os diferentes sensores. Medindo esta distância como observado na Figura 4.7, adotou-se os ganhos como 0, 2750, 3550 e 6300, conforme a Figura 5.2.

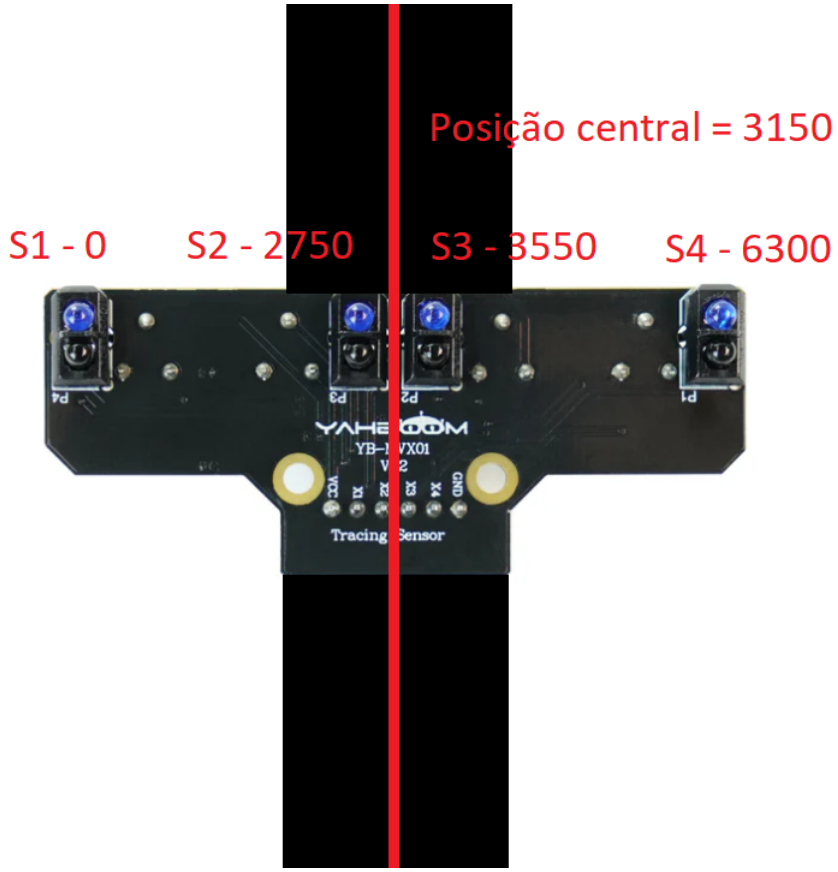



Figura 5.2: Esquema da média ponderada utilizada pelos sistemas e o cálculo do erro.

Além disso é importante realçar o facto que os sensores infravermelhos têm uma saída digital. Uma saída analógica resultaria em um valor de posição mais preciso, no entanto, também estaria mais sujeito a ruído.

Na Figura 5.3 podemos ver um fluxograma da sua implementação.

No caso onde todos os sensores medem o valor 0, isto é, os sensores "perderam"a linha, é necessário evitar o erro de divisão por 0, retornando o valor -1.

Finalmente, e com a distância calculada, o erro pode ser determinado através da subtração do valor pretendido com esta posição. O erro é calculado ao subtrair o valor central dos ganhos escolhidos, neste caso, 3150 com o valor do erro. Desta forma, se o erro for 0, o sensor estará centrado com a linha, se for negativo, a linha está à sua direita, e quando positivo, à sua esquerda. Além disso, se a função anterior retornar -1, isto é, o sensor "perder"a linha, o erro será alterado tendo em conta o erro anterior. Desta forma, se o erro anterior for negativo, o erro será alterado para o valor -6000, e se positivo, 6000, como mostra a implementação da Figura 5.4. As equações utilizadas para o cálculo da posição e erro são dadas, respetivamente, por:

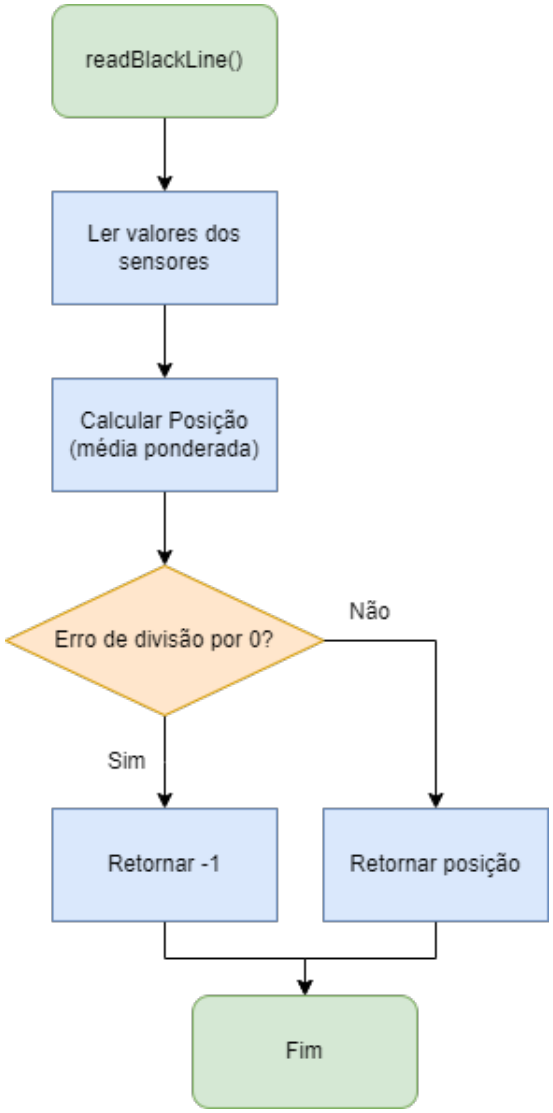



Figura 5.3: Fluxograma da função *readBlackline*.

$$Position = \frac{(0 \times S1) + (2750 \times S2) + (3550 \times S3) + (6300 \times S4)}{S1 + S2 + S3 + S4} \tag{5.4}$$

$$Erro = 3150 - Position \tag{5.5}$$

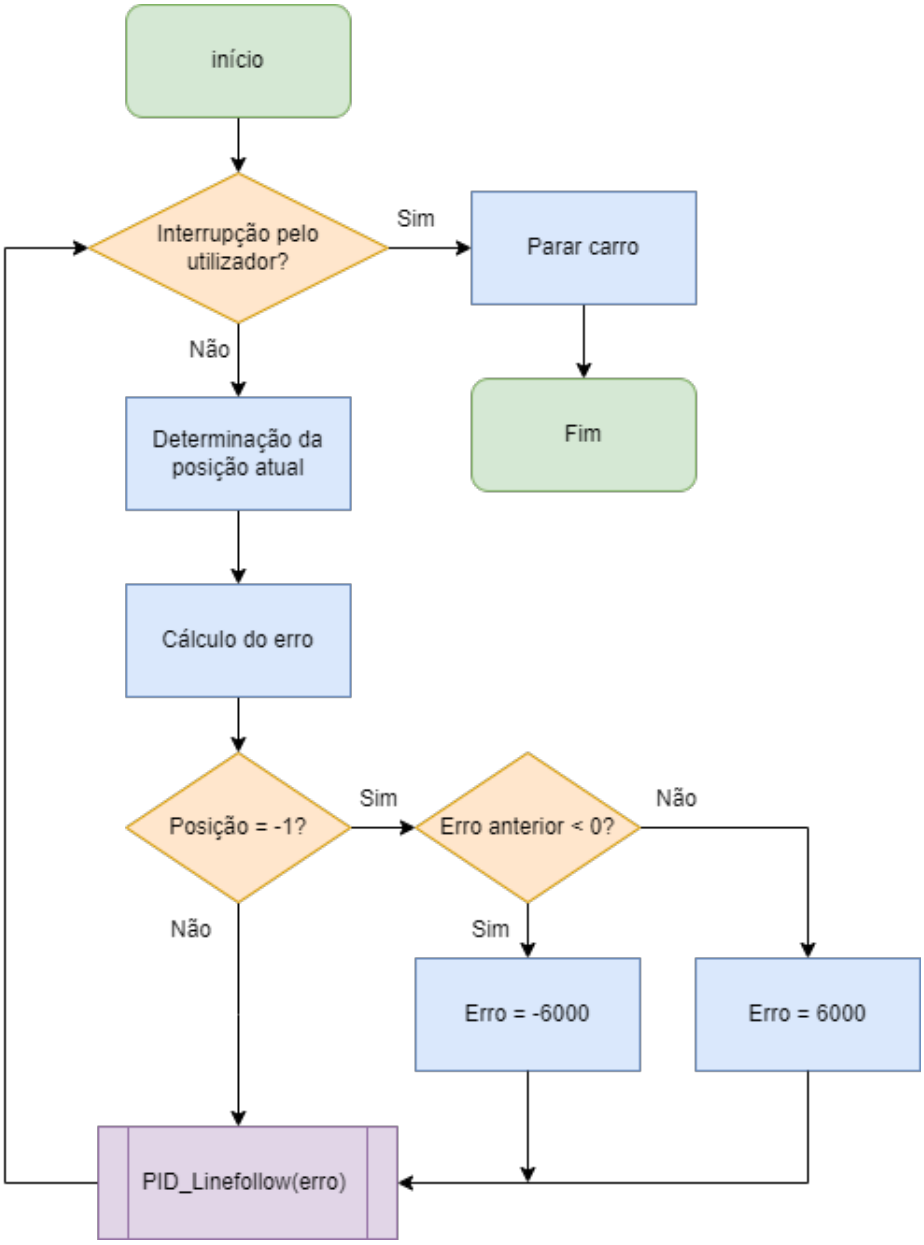



Figura 5.4: Fluxograma do ciclo principal do controlador PID.

Como é possível observar, após o cálculo o erro, este é dado como parâmetro para a função "PID_Linefollow"que será responsável por calcular o novo valor da velocidade das rodas e de guardar esses valores em vetores. Na Figura 5.5 é ilustrado um fluxograma resumindo as etapas desta função.

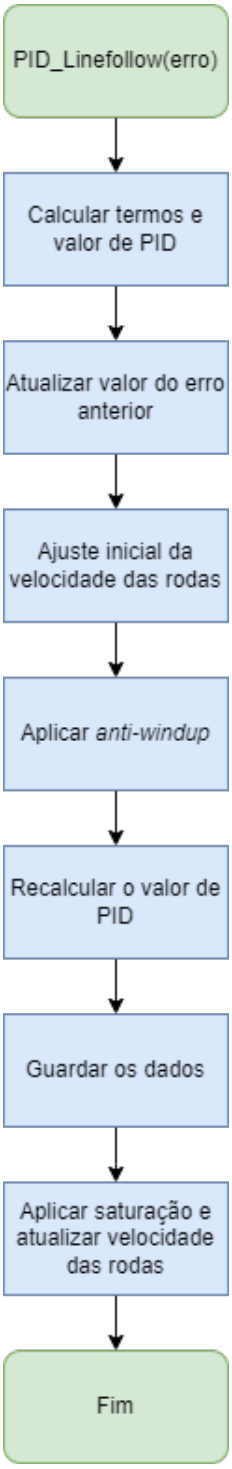



Figura 5.5: Fluxograma do controlador PID.

Esta função é dividida em várias etapas, sendo estas:

1. **Cálculo dos termos** proporcional ($P = error$), integrativo ($I = I + error$) e derivativo ($D = error - previousError$), multiplicá-los pelos respetivos ganhos e somá-los para calcular o valor de PID.

Os ganhos foram obtidos experimentalmente, tentando vários até obter valores satisfatórios, neste caso os ganhos são $K_p = 0.04$, $K_i = 0.00002$ e $K_d = 0.035$ (Listagem 5.1);

```
    # Proportional term
    P = error
    # Integrative term
    Ivalue = I + integrativeGain * error
    # Derivative term
    D = error - previousError

    # Calculate the individual PID components
    Pvalue = P * proportionalGain
    Dvalue = D * derivativeGain

```

Listagem 5.1: Cálculo dos termos P, I e D.

2. **Ajuste inicial da velocidade das rodas**, é definido um valor padrão para os motores rodarem, neste caso 70, chamado de velocidade de base. O ajuste vai somar o resultado do PID aos motores direitos e subtrair aos do lado esquerdo, desta forma é mantido o valor médio de 70 na direção da frente do veículo (Listagem 5.2);

```
    # Calculate the total PID value
    PIDvalue = Pvalue + Ivalue + Dvalue
    # Update the previous error
    previousError = error

    # Adjust the motor speeds based on PID value
    leftSpeed = defaultSpeed - PIDvalue
    rightSpeed = defaultSpeed + PIDvalue

```

Listagem 5.2: Ajuste inicial da velocidade das rodas.

3. **Aplicar o mecanismo de *anti-windup***, este ajuda a que a ação integrativa não atinja valores muito elevados, a que poderia levar ao robô, ao perder a linha, a rodar indefinidamente incapaz de voltar a seguir o percurso.

   Desta forma, caso as velocidades das rodas ultrapasse o limite aceitável (menos que -255 ou mais que +255), a quantidade a que estas velocidades ultrapassam os limites vai ser adicionado ao valor de *anti-windup*. Neste passo é preciso ter atenção aos sinais utilizados tomando em consideração se o valor das rodas é

inferior a -255 ou superior a +255 e se são os motores direitos ou esquerdos, visto que o valor de PID é somado à velocidade dos motores direitos e subtraído aos esquerdos (Listagem 5.3);

```
    # Anti-windup mechanism to prevent integral windup
    antiWindUp = 0
    antiWindUpGain = 0.25
    if leftSpeed > 255: antiWindUp -= leftSpeed - 255
    if leftSpeed < -255: antiWindUp -= leftSpeed + 255
    if rightSpeed > 255: antiWindUp += rightSpeed - 255
    if rightSpeed < -255: antiWindUp += rightSpeed + 255

    # Adjust the integral term to account for windup
    Ivalue = Ivalue - (antiWindUp * antiWindUpGain)

```

Listagem 5.3: Aplicação do mecanismo de *anti-windup*.

4. **Recalcular o valor PID**, devido ao facto que o *anti-windup* altera o valor da ação integrativa é necessário recalcular a saída do PID e a velocidade das rodas (Listagem 5.4);

```
    # Recalculate the total PID value with adjusted integral
    term
    PIDvalue = Pvalue + Ivalue + Dvalue

    # Recalculate the motor speeds based on the new PID value
    leftSpeed = defaultSpeed - PIDvalue
    rightSpeed = defaultSpeed + PIDvalue

```

Listagem 5.4: Recalcular o valor PID.

5. **Guardar os dados** para análise, *debugging* e, posteriormente, treinar a rede neuronal (Listagem 5.5);

```
    # Log data for analysis and debugging
    simulationTime.append(time.time() - startTime)
    errorArray.append(error)
    pidOutput.append(PIDvalue)
    proportionalTermArray.append(Pvalue)
    integrativeTermArray.append(Ivalue)
    derivativeTermArray.append(Dvalue)
    leftSpeedArray.append(leftSpeed)
    rightSpeedArray.append(rightSpeed)
```

```
10      antiWindUpArray.append(antiWindUp * antiWindUpGain)
11
```

Listagem 5.5: Guardar os dados do controlador PID.

6. **Aplicar saturação** às velocidades dos motores para que estas não excedam os limites possíveis (-255 a 255) e finalmente aplicar estas velocidades aos motores (Listagem 5.6).

```
1       # Clamp the motor speeds to be within the valid range
2       if leftSpeed > 255: leftSpeed = 255
3       if leftSpeed < -255: leftSpeed = -255
4       if rightSpeed > 255: rightSpeed = 255
5       if rightSpeed < -255: rightSpeed = -255
6
7       # Send the control signals to the car's motors
8       car.Control_Car(leftSpeed, rightSpeed)
9
```

Listagem 5.6: Aplicação da saturação.

Com isto, foi obtido um controlo como o indicado na Figura 5.6.

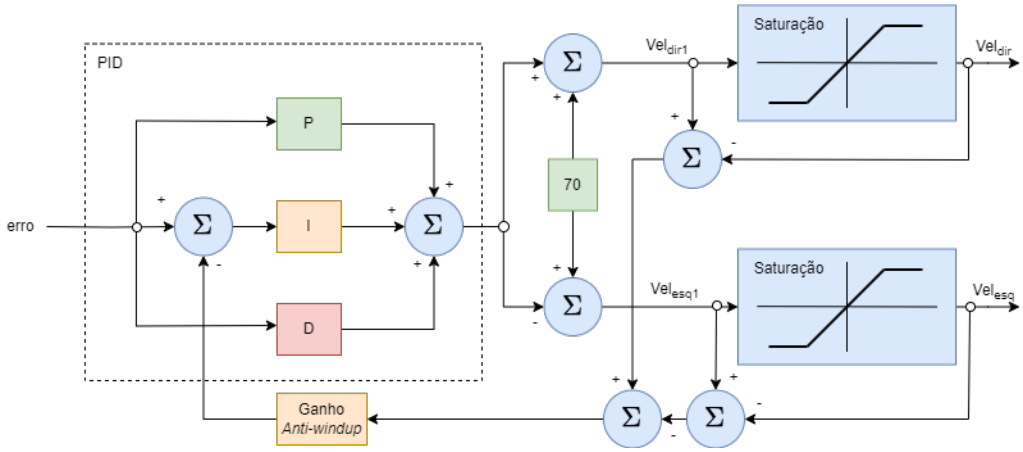



Figura 5.6: Diagrama de blocos do controlo PID implementado.

Deste modo foi desenvolvido um controlador PID capaz de controlar o veículo de forma a este se manter e navegar por uma linha preta no chão. Será difícil mostrar o desempenho do controlador sem um vídeo deste a funcionar, no entanto, é possível observar um gráfico do erro, valor da saída do PID com os seus termos, e o *anti-windup* ao longo de uma volta do percurso, como mostrado na Figura 5.7.

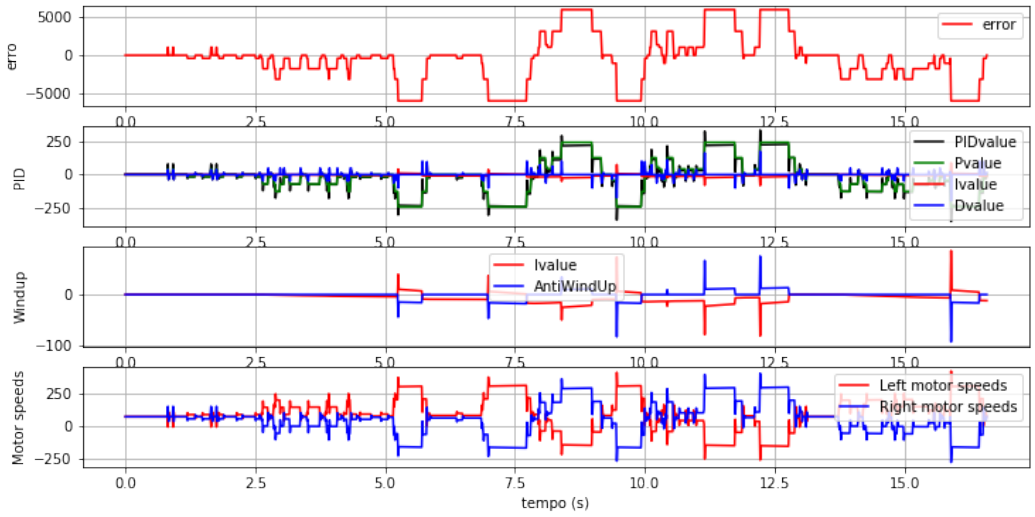



Figura 5.7: Gráfico do erro, termos e saída do PID e *anti-windup* ao longo de uma volta do circuito.

Por fim, com os dados guardados, para que estes possam ser utilizados para treinar a rede neuronal, é necessário convertê-los num formato compatível para essa utilização, pelo que os dados foram convertidos para o formato CSV, conforme a Listagem 5.7.

```
# Create a DataFrame using the error, left speed, and right speed arrays
data = pd.DataFrame({
    'error': errorArray,          # Array of error values
    'new_left_speed': leftSpeedArray,   # Array of left motor speeds
    'new_right_speed': rightSpeedArray  # Array of right motor speeds
})

# Save the DataFrame to a CSV file named 'robot_LSTM_data.csv' without including the index
data.to_csv('robot_LSTM_data.csv', index=False)
```

Listagem 5.7: Guardar dados do controlador PID em formato CSV.

A Figura 5.8 mostra o circuito utilizado para a realização destes testes.

Figura 5.8: Circuito utilizado para a primeira experiência.

### 5.1.2 LSTM

Com os dados obtidos é agora possível treinar uma rede neuronal *Long Short-Term Memory* (LSTM) para que esta simule o comportamento do controlador PID. Para isto é necessário escolher um modelo adequado para esta implementação.

Devido ao facto do controlador PID tomar em consideração valores anteriores para calcular o novo valor de saída, é importante considerar utilizar um modelo que tenha a capacidade de utilizar valores passados para calcular novas velocidades para os motores, pelo que foi escolhido uma rede LSTM.

### Implementação

A implementação da rede neuronal LSTM pode-se dividir em várias etapas:

1. **Preparar os dados de treino**: com os dados no formato CSV é necessário identificar quais são as entradas e saídas, criar sequências e separá-las entre dados de treino e teste;
2. **Definir a rede a ser usada**: definindo a sua profundidade, tipo e tamanho de cada camada;
3. **Treinar a rede**: testando diferentes configurações, como diferentes camadas, maior ou menor profundidade comparando o erro de cada uma;
4. **Guardar o modelo treinado**: para que possa ser transferido de volta para a Raspberry Pi do veículo e utilizado para o controlar.

Carregando os dados do ficheiro CSV criado, é primeiramente necessário definir as sequências e o tamanho destas. Para a utilização de camadas LSTM é fundamental estabelecer quantos valores anteriores a rede terá em consideração para definir uma nova saída, $n_{steps}$. Foi definido este valor para 80 por apresentar uma boa relação entre o aumento do tempo de treino da rede e a diminuição do erro desta. Este número representa que a rede irá ter em consideração as 80 amostras anteriores (1 segundo) para calcular a nova saída. A implementação da criação destas sequências pode ser vista na Listagem 5.8.

```
# Define the number of time steps in each sequence
n_steps = 80

# Function to create sequences from the data
def create_sequences(data, n_steps):
    sequences = []
    for i in range(len(data) - n_steps):
        seq_x = data.iloc[i:i + n_steps, 0].values.astype('float32')  # Features (errors)
        seq_y = data.iloc[i + n_steps, 1:3].values.astype('float32')  # Labels (left and right speeds)
        sequences.append((seq_x, seq_y))
    return sequences

# Create sequences
sequences = create_sequences(data, n_steps)
```

Listagem 5.8: Criar sequências de 80 valores.

Com as sequências criadas é agora necessário dividi-las entre dados de treino que vão ser utilizadas para efetivamente treinar a rede neuronal, e dados de teste que vão ser utilizados para comparar à rede treinada e avaliar o seu desempenho.

Para isto foi utilizado um rácio de 0,2, ou seja, 80 % dos dados vão servir de treino e os restantes 20 % de teste, como é possível observar na Listagem 5.9.

```
# Split the sequences into input (X) and output (y)
X, y = zip(*sequences)

# Convert evaluated tensors to NumPy arrays
X = np.array(X)
y = np.array(y)

# Reshape X to have three dimensions if it doesn't already
if X.ndim == 2:
    X = X.reshape((X.shape[0], n_steps, 1))

# Verify the shapes
print(f'X shape: {X.shape}')  # Should be (num_sequences, n_steps,
    num_features)
print(f'y shape: {y.shape}')  # Should be (num_sequences,
    num_outputs)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
```

Listagem 5.9: Preparar dados de treino para a rede LSTM.

Com isto os dados estão configurados corretamente para serem utilizados para treinar a rede neuronal. O próximo passo é definir as suas camadas e profundidade com a Listagem 5.10.

```
model = Sequential()
model.add(LSTM(50, input_shape=(n_steps, 1), return_sequences=True
    ))
model.add(Dropout(0.1))  # Add Dropout layer with 10% dropout rate
model.add(LSTM(75, input_shape=(n_steps, 1), return_sequences=
    False))
model.add(Dropout(0.1))  # Add Dropout layer with 10% dropout rate
model.add(Dense(75, activation='relu'))  # Additional Dense layer
    with ReLU activation
model.add(Dense(2))  # Output layer for left and right speeds
model.compile(optimizer='adam', loss='mse')
model.summary()
```

Listagem 5.10: Rede LSTM utilizada.

O modelo LSTM foi concebido para fornecer com precisão as velocidades esquerda e direita novas aos motores, atingindo um equilíbrio entre a complexidade e a regularização para otimizar o desempenho e, ao mesmo tempo, atenuar o *overfitting* (Figura 5.9).

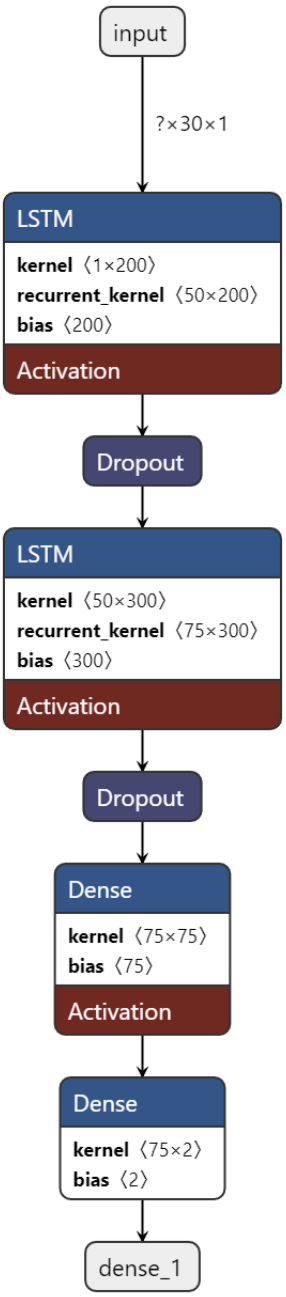



Figura 5.9: Esquema da rede LSTM utilizada.

Aqui está uma análise detalhada:

1. **Primeira camada LSTM (50 unidades, Sequências de retorno)**:
   - **Objetivo**: Esta camada é crucial para captar as dependências temporais nos dados de entrada. Com 50 unidades, extrai padrões significativos dos dados sequenciais ao longo do tempo;
   - **Sequências de retorno**: Ao definir return_sequences=True, a camada devolve a sequência completa de resultados, assegurando que as camadas subsequentes podem continuar a processar estas sequências.
2. **Primeira camada de *dropout* (taxa de *dropout* de 0,1)**:
   - **Objetivo**: O *dropout* é uma técnica de regularização que ajuda a evitar o *over-fitting*. Ao eliminar aleatoriamente 10% das unidades durante o treino, ela força a rede a aprender características mais robustas que se generalizam melhor para dados não vistos.
3. **Segunda camada LSTM (75 unidades, sem sequências de retorno)**:
   - **Objetivo**: Esta camada baseia-se nas características temporais extraídas pela primeira camada LSTM, com 75 unidades que lhe permitem captar padrões e interações mais complexos nos dados;
   - **Sem sequências de retorno**: Ao definir return_sequences=False, a camada devolve apenas a saída final da sequência, resumindo a informação de toda a sequência num único vetor.
4. **Segunda camada de *dropout* (taxa de *dropout* de 0,1)**:
   - **Objetivo**: Semelhante à primeira camada de abandono, esta camada ajuda a evitar ainda mais o *over-fitting*, adicionando outro passo de regularização.
5. **Camada densa (75 unidades, ativação ReLU)**:
   - **Objetivo**: As camadas densas são utilizadas para transformar o resultado das camadas LSTM numa forma mais adequada para a previsão final. Com 75 unidades e ativação ReLU, esta camada introduz a não linearidade, permitindo ao modelo aprender relações e padrões mais complexos nos dados.
6. **Camada densa de saída (2 unidades)**:
   - **Objetivo**: Esta última camada produz as previsões para as velocidades esquerda e direita. Com 2 unidades, corresponde diretamente às duas variáveis-alvo, fornecendo os valores previstos.

Com a rede definida podemos agora treiná-la de acordo com a Listagem 5.11.

```
# Train the model
history = model.fit(X_train, y_train, epochs=60, batch_size=32,
    validation_data=(X_test, y_test))
```

Listagem 5.11: Treinar a rede LSTM com 60 épocas.

Foram escolhidas 60 *epochs*, isto é, o número de vezes que todo o conjunto de dados de treino é passado pela rede. O treino com mais *epochs* permite que o modelo aprenda melhor, mas um número excessivo pode levar a *over-fitting*.

Podemos ver a evolução do erro do modelo ao longo que este é treinado na Figura 5.10. Já a Tabela 5.1 mostra os parâmetros da rede utilizada.

Tabela 5.1: Hiperparâmetros do Modelo LSTM

| **Hiperparâmetro** | **Valor** |
|---|---|
| Unidades LSTM (Camada 1) | 50 |
| Forma de Entrada | $(n\_steps, 1)$ |
| Retornar Sequências (Camada 1) | Verdadeiro |
| Dropout (Camada 1) | 0.1 (10%) |
| Unidades LSTM (Camada 2) | 50 |
| Retornar Sequências (Camada 2) | Falso |
| Dropout (Camada 2) | 0.1 (10%) |
| Unidades Densas (Oculta) | 50 |
| Função de Ativação (Oculta) | ReLU |
| Unidades Densas (Saída) | 2 |
| Otimizador | Adam |
| Função de Perda | MSE (Erro Quadrático Médio) |
| *Batch size* | 32 |
| Épocas | 60 |
| Dados de Validação | $(X\_test, y\_test)$ |

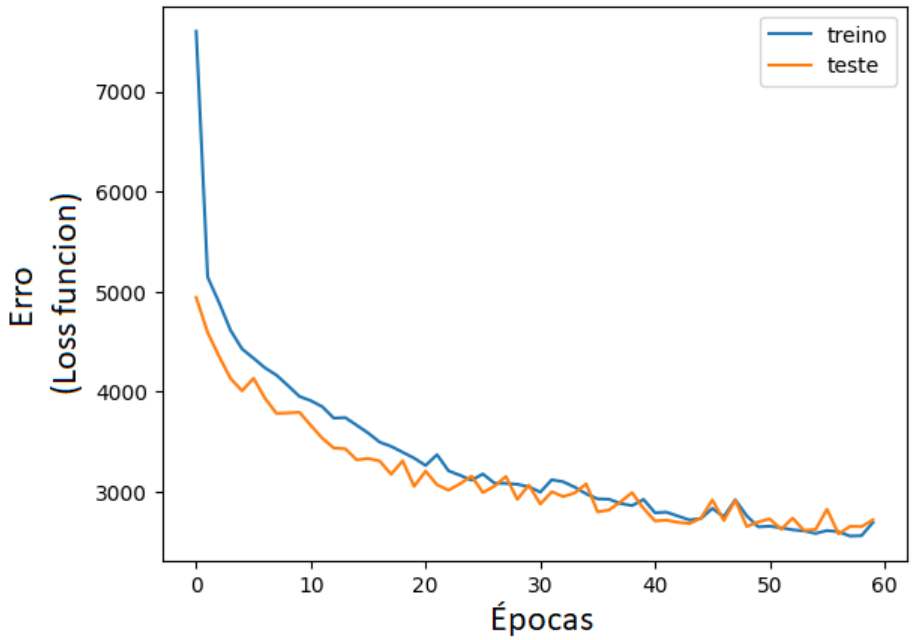



Figura 5.10: Evolução do erro da rede LSTM ao longo do seu treino.

Por fim, o modelo pode ser guardado para ser transferido de volta para a Raspberry Pi onde será utilizado para controlar o veículo (Listagem 5.12).

```
model.save(r'my_lstm_model4.h5')
```

Listagem 5.12: Guardar modelo LSTM.

### 5.1.3 Controlo do veículo através da rede LSTM

Como último passo, o modelo é descarregado para a Raspberry Pi para ser utilizado substituindo o controlador PID. Para isto, parte de código desenvolvido pode ser reutilizado pelo que após o cálculo da posição do veículo e do seu erro, o código terá que ser alterado para remover o controlador PID e utilizar o modelo para calcular as novas velocidades das rodas.

No entanto, de forma a utilizar o modelo, é necessário atribuir como parâmetro as 80 últimas leituras dos sensores. Para este efeito, utilizou-se um vetor junto da função *roll* da biblioteca `numpy` para atualizar a última posição do vetor com o valor atual dos sensores (Listagem 5.13).

```
        # Shift all elements to the left
        inputErrorArray = np.roll(inputErrorArray, -1)
        # Update the last value
        inputErrorArray[-1] = error

```

```
6         # Prepare the input data in the same format as the
      training data
7         input_data = np.array([[inputErrorArray]]).reshape((1, 80,
       1))  # Ensure input_data has the shape (1, 3)
```

Listagem 5.13: Atualizar vetor de 80 leituras.

Com isto podemos carregar e utilizar de forma a extrair a nova velocidade das rodas (Listagem 5.14).

```
1 # Load the trained model
2 model = load_model('my_lstm_model4.h5')
3
4         # Prepare the input data in the same format as the
      training data
5         input_data = np.array([[inputErrorArray]]).reshape((1, 80,
       1))  # Ensure input_data has the shape (1, 3)
6
7         # Use the model to predict new speeds
8         new_speeds = model.predict(input_data)
9
10        # Extract the predicted speeds
11        new_left_speed, new_right_speed = new_speeds[0]
```

Listagem 5.14: Utilizar rede LSTM para obter novas velocidades.

Por fim, aplica-se saturação da mesma forma que o controlador PID para finalmente alterar a velocidade das rodas (Listagem 5.15).

```
1         if new_left_speed > 255: new_left_speed = 255
2         if new_left_speed < -255: new_left_speed = -255
3         if new_right_speed > 255: new_right_speed = 255
4         if new_right_speed < -255: new_right_speed = -255
5
6         # Control the car with the new speeds
7         car.Control_Car(new_left_speed, new_right_speed)
```

Listagem 5.15: Aplicar saturação dos valores da rede LSTM e controlar o AGV.

Deste modo, obtém-se um controlo do sistema como o da Figura 5.11. Em suma, foi retirado o controlador PID e o sistema de *anti-windup* e foi substituindo em seu lugar a rede LSTM.

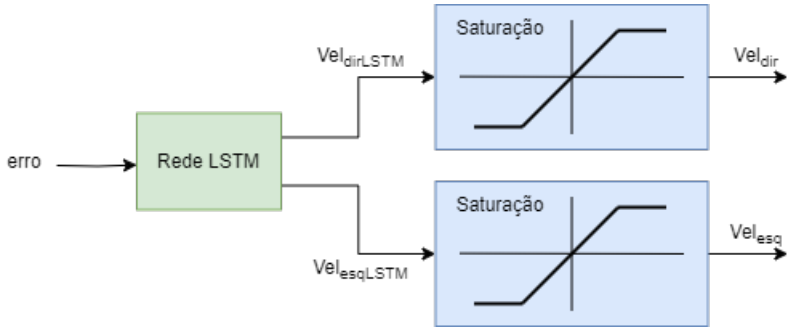



Figura 5.11: Diagrama de blocos do controlo com rede LSTM implementado.

Com isto, foi possível o controlo do veículo com um rede neuronal utilizando para isso apenas como entradas os sensores infravermelhos. O controlo com este método assemelha-se bastante ao do controlo por PID, o que é de esperar visto que foi utilizado os dados desse controlador como treino para a rede neuronal. Por outro lado, as similaridades existentes validam que a rede LSTM foi capaz, de facto, aprender os padrões associados e dependências temporais do controlador PID. A Figura 5.12 mostra o gráfico, incluindo o erro das leituras do sensores infravermelhos, ao longo de uma volta do circuito. O segundo apresenta o resultado da previsão feita pela rede LSTM. De notar que ao contrário do controlador PID, onde após ser calculado o novo valor este seria somado e subtraído às velocidades das rodas, a rede LSTM já tem estas velocidades como saída. De forma a comparar o desempenho de ambos os controlos, foi realizada a operação inversa, onde, com as novas velocidades das rodas, foi calculado o valor aproximado do que seria o valor de PID. No entanto, é importante realçar que este valor não irá traduzir completamente para o valor de PID, mas será um valor aproximado visto que a rede está treinada para simular o comportamento de tal controlador. De facto, comparando os dois gráficos, é possível observar que ambos são bastante similares.

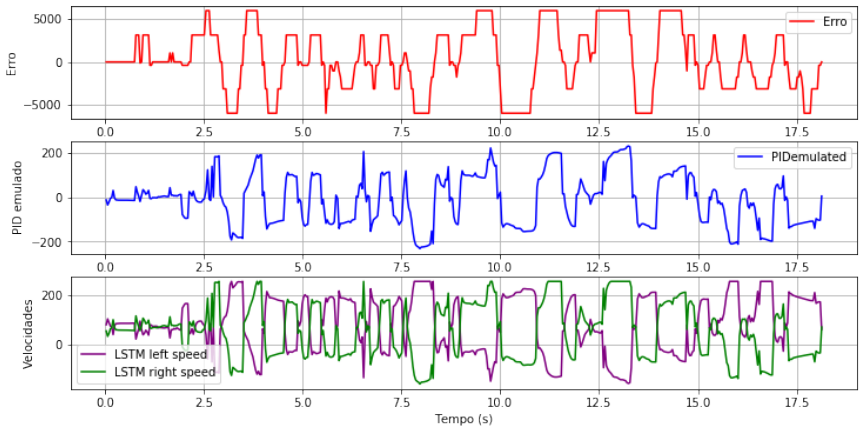



Figura 5.12: Gráficos do controlo com a rede LSTM e as velocidades dos motores ao longo de uma volta.

A Figura 5.13 mostra uma comparação entre o controlo por PID e através do LSTM ao longo de uma volta do circuito. Mesmo que algumas semelhanças possam ser observadas, o facto que, mesmo com o circuito idêntico, as condições iniciais e ao longo da experiência são bastante diferentes, o que causa bastantes diferenças entre os dois gráficos.

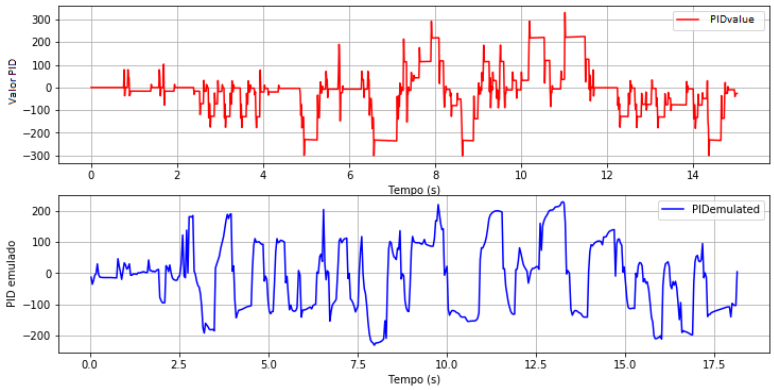



Figura 5.13: Comparação da saída dos controladores PID e LSTM.

## 5.2 Controlo através da câmera

Numa segunda experiência, pretende-se desenvolver uma plataforma para que o robô siga uma linha desenhada no chão através de redes neuronais, como na experiência anterior, mas desta vez utilizando para isso a câmera. Desta forma, a câmera Raspberry Pi montada no robô irá ser apontada para baixo para que possa tirar imagens sucessivas do chão à frente deste. Estas imagens serão depois alimentadas a uma rede neuronal que irá controlar o robô ao longo do percurso.

Com esta estratégia, a utilização de uma rede neuronal LSTM teria benefícios limitados comparando com o uso de uma rede CNN, visto que, a rede LSTM foi escolhida para a experiência anterior para que esta possa simular o comportamento do controlador PID, o qual necessita de informações passadas para calcular a nova saída. A utilização de imagens em tempo real para o controlo do robô não apresenta essa dependência temporal do controlador PID. Por outro lado, o uso de uma rede LSTM neste caso poderia facilmente, se configurada incorretamente, levar à rede a memorizar o percurso a que foi treinada, levando a que o robô tivesse comportamentos anormais ou errados se colocado num circuito diferente. Por estas razões foi escolhida uma rede CNN.

### 5.2.1 Obtenção de imagens de treino

Tal como na primeira experiência, é necessário obter bastantes dados para treinar a rede neuronal, neste caso, estes dados serão um grande número de imagens obtidas

pelo próprio robô, pelo que é necessário desenvolver um método para obter tais imagens.

Como mencionado, o primeiro passo será obter várias fotografias do percurso e guardá-las por forma a que possam ser utilizadas para treinar a rede neuronal. Para isso irão ser seguidos os passos presentes no fluxograma da Figura 5.14.

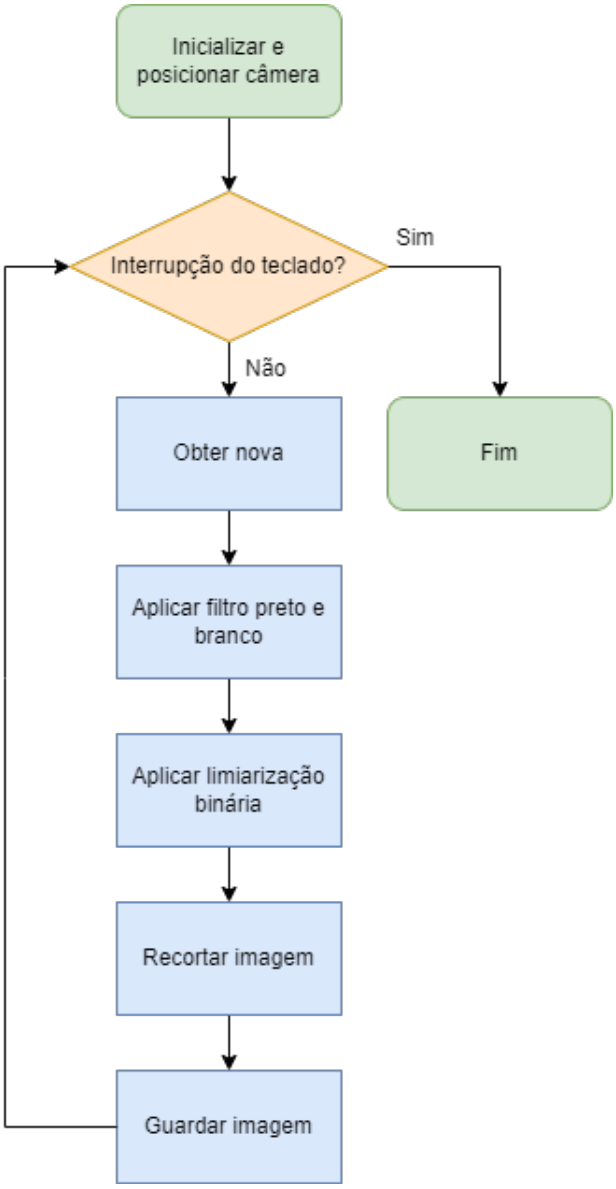



Figura 5.14: Fluxograma retratando o processo de captação, tratamento, e arquivamento de imagens.

Onde:

- **Inicialização e posicionamento da câmera**: Durante a inicialização, a câmera é configurada para capturar imagens e posicionada corretamente para a tarefa. O código começa por inicializar o objeto PiCamera e configurar a

resolução (640x480) e a taxa de fotografias (32 fps). Um breve atraso (0,1 segundos) permite que a câmera aqueça antes de capturar dados, garantindo que está pronta e a funcionar corretamente.

Além disso, é ajustada a posição dos servos ligados à câmera utilizando comandos como `car.Ctrl_Servo(1, 83)` e `car.Ctrl_Servo(2, 180)`. Estes comandos garantem que a câmera está alinhada apontada diretamente para baixo para uma captura de imagem precisa.

A Listagem 5.16 apresenta o código desenvolvido para esta implementação;

```
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(640, 480))
time.sleep(0.1)  # Wait for the camera to warm up
car = YB_Pcb_Car.YB_Pcb_Car()
car.Ctrl_Servo(1, 83)  # Position the camera
car.Ctrl_Servo(2, 180)

```

Listagem 5.16: Inicialização e posicionamento da câmera.

- **Obtenção de uma nova imagem**: Em cada iteração do ciclo, é capturada uma nova imagem da câmera (Figura 5.15, armazenada na variável 'image'). Esta captura contínua é tratada por `camera.capture_continuous()`, o que significa que o sistema está a capturar fotogramas em tempo real. A aquisição contínua de imagens é crucial para o processamento de imagens em tempo real.

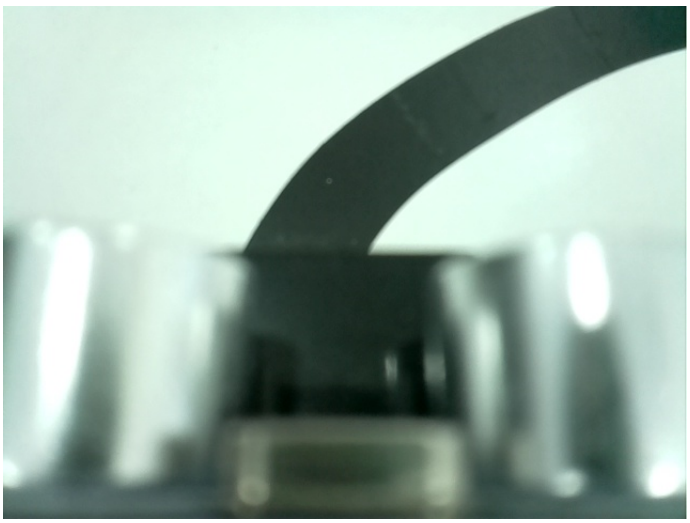


Figura 5.15: Exemplo de uma imagem capturada no início do ciclo.

A Listagem 5.17 apresenta o código desenvolvido para esta implementação;

```
1 for frame in camera.capture_continuous(rawCapture, format="bgr
    ", use_video_port=True):
2     image = frame.array  # Capture the image in RGB format as
    a NumPy array
3
```

Listagem 5.17: Obter uma nova imagem no início de cada ciclo.

- **Aplicação de filtro preto e branco**: Depois de capturar a imagem a cores, esta é convertida para escala de cinzentos utilizando a função `cv2.cvtColor()` (Figura 5.16). Isto reduz a complexidade da imagem, uma vez que o próximo algoritmo de limiarização funciona melhor com imagens em escala de cinzentos de um canal do que com imagens RGB de vários canais (Listagem 5.18);

```
1 gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  #
    Convert to grayscale
2
```

Listagem 5.18: Aplicação do filtro preto e branco.

Figura 5.16: Exemplo de uma imagem com o filtro preto e branco aplicado.

- **Aplicação de limiarização binária**: A imagem em escala de cinzentos é então processada utilizando a limiarização binária (Figura 5.17). Isto transforma a imagem em tons de cinzento numa imagem binária com apenas duas cores (preto e branco). A função `cv2.threshold()` é utilizada para aplicar

um valor de limiar (valor limiar = 70), em que os pixels acima deste limiar são definidos como brancos (255) e os pixels abaixo são definidos como pretos (0). Este processo simplifica a imagem para facilitar a deteção de formas ou contornos. Este tratamento de dados é válido para esta implementação visto que o objetivo é apenas realçar o que é, ou não, a linha preta (Listagem 5.19);

Figura 5.17: Exemplo de uma imagem com a limiarização binária aplicada.

```
1 gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  #
      Convert to grayscale
2
```

Listagem 5.19: Aplicação de limiarização binária.

- **Recorte da imagem**: Para otimizar o processamento e concentrar-se nas partes relevantes da imagem (Figura 5.18), a metade inferior da imagem é cortada pelo que, como é possível observar nas imagens anteriores, estas contêm parte do robô. Numa alternativa, os motores servo poderiam ser configurados de forma a inclinar a câmera ligeiramente para cima, evitando este passo, no entanto, experiências realizadas mostraram que captar a linha perto do robô é essencial para o bom funcionamento deste método de controlo. Neste caso, o código corta a imagem, limitando-a aos 230 pixels superiores e descartando a parte inferior desnecessária. Isso poderá também ajudar a reduzir o tempo de processamento (Listagem 5.20);

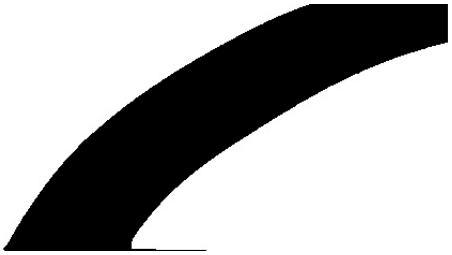


Figura 5.18: Exemplo de uma imagem recortada para eliminar partes desnecessárias.

```
binary_image = binary_image[:230, :]  # Crop the lower half of
    the image

```

Listagem 5.20: Recortar imagem.

- **Guardar imagem**: Após o processamento (conversão da escala de cinzentos, limiarização e corte), a imagem binária é guardada num ficheiro. O caminho do ficheiro é gerado dinamicamente e a imagem é guardada no formato .jpg utilizando `cv2.imwrite()`. Cada imagem é guardada sequencialmente com um nome de ficheiro numerado na pasta especificada (`images_path`) (Listagem 5.21).

```
image_path = os.path.join(images_path, f"{start}.jpg")  #
    Define the file path
cv2.imwrite(image_path, binary_image)  # Save the binary image

```

Listagem 5.21: Guardar imagem processada.

Desta forma foram obtidas diversas imagens do percurso em diferentes locais, posições e inclinações para garantir que a rede neuronal seja treinada para qualquer situação que o robô possa enfrentar. O *dataset* tem um tamanho de 1800 imagens. Idealmente seriam utilizadas mais imagens, mas um maior *dataset* leva a um tempo de classificação maior. A Figura 5.19 mostra o aspeto do *dataset* antes do processamento e a Figura 5.20 o aspeto depois. Por exemplo, nestas imagens a imagem 20 está classificada como '*left_spin*' e a imagem 12 como '*hard_left*'.

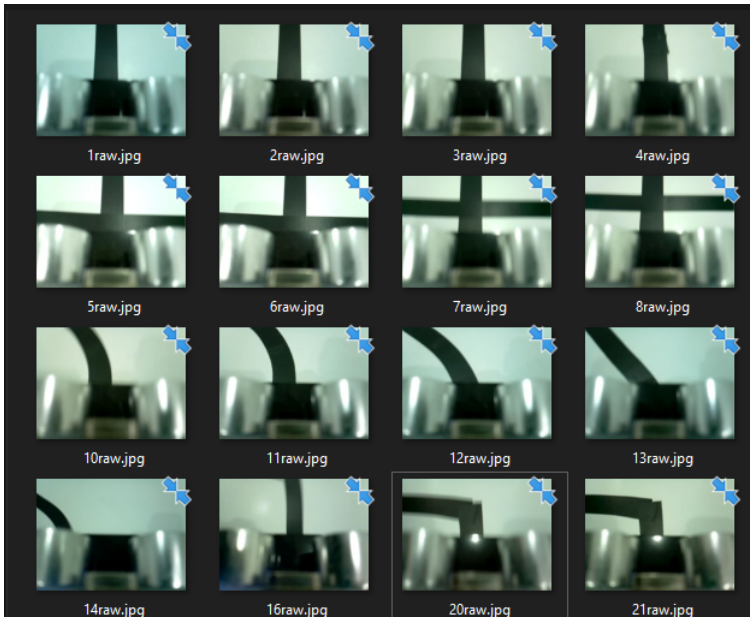



Figura 5.19: Aspeto do *dataset* utilizado antes do processamento das imagens.

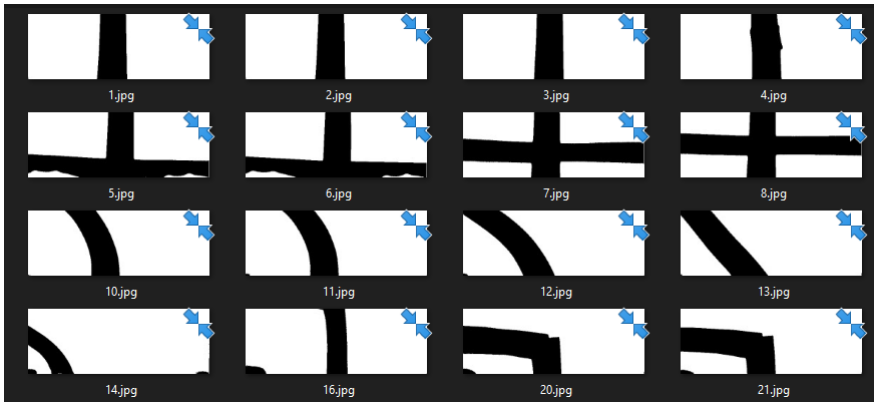



Figura 5.20: Aspeto do *dataset* utilizado depois do processamento das imagens.

### 5.2.2 Classificação das imagens

Com a obtenção das imagens, o próximo passo será de as classificar para que possam ser utilizadas para servirem de treino da rede neuronal, isto é, atribuir a cada imagem a ação que o robô terá que fazer. Esta ação terá um valor específico para cada motor, como na experiência anterior, ou uma categoria como por exemplo, 'frente', 'esquerda' e 'direita'.

#### Classificação numérica através da biblioteca Pygame

Para isto foram testados dois métodos. Num primeiro método, foi experimentado atribuir valores numéricos das velocidades dos motores esquerdos e direitos, sendo

desenvolvida uma plataforma para facilitar este processo.

Para tal, foi utilizada a biblioteca 'pygame'. Pygame é uma biblioteca Python de código aberto utilizada para o desenvolvimento de jogos de vídeo. Fornece módulos e funções para lidar com vários aspetos do desenvolvimento de jogos. Neste caso, a sua utilização não será para o desenvolvimento de um jogo, mas sim de uma interface interativa de forma a facilitar o processo de classificação.

Com isto, a solução encontrada foi a de criar uma interface onde as imagens obtidas serão carregadas e apresentadas uma de cada vez no plano de fundo por uma fração de segundo, onde o utilizador terá a opção de definir as velocidades das rodas indiretamente através de um *joystick*, como indicado na Figura 5.21.

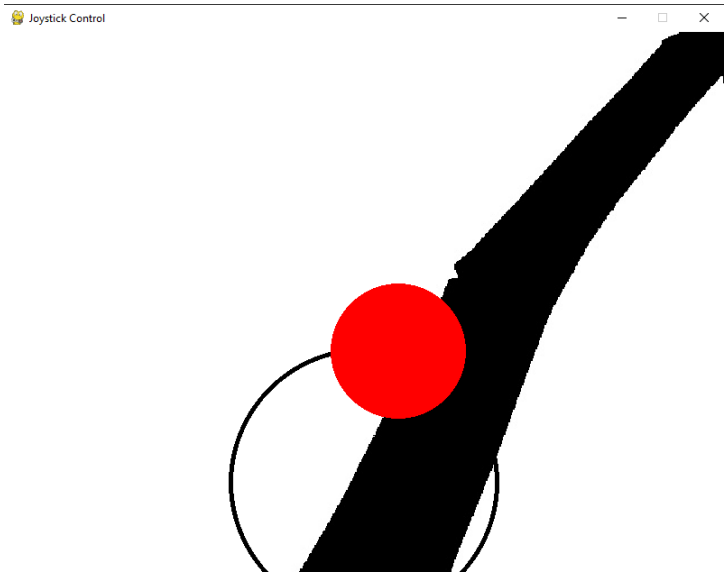



Figura 5.21: Plataforma Pygame para a categorização de imagens.

O fluxograma da Figura 5.22 mostra o funcionamento desta aplicação.

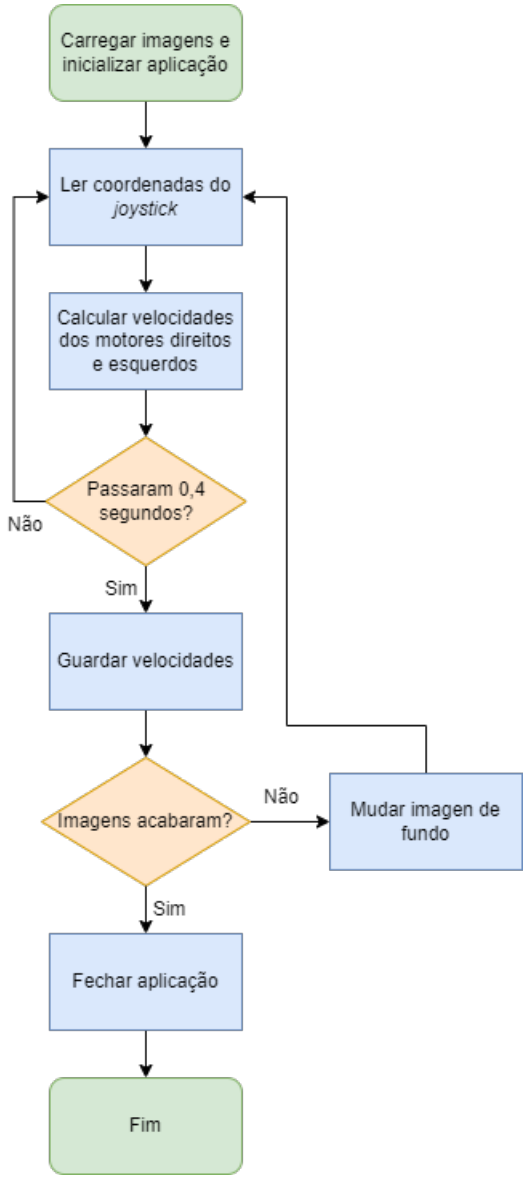



Figura 5.22: Plataforma Pygame para a categorização de imagens.

Primeiro, o Pygame é inicializado para configurar uma janela de 800x600 pixéis. As imagens de fundo são carregadas a partir de uma pasta especificada e ordenadas numericamente, permitindo transições suaves entre imagens. As variáveis de controlo do *joystick* e do carro, como a posição do *joystick*, o raio e a velocidade do carro, também são definidas, bem como um ficheiro CSV para armazenar os dados.

O *loop* principal "ouve"continuamente os eventos do utilizador, como cliques e libertações do rato. Se o utilizador clicar dentro do círculo exterior do *joystick*, este fica ativo e a sua posição segue o cursor do rato. A distância e o ângulo entre a posição atual do *joystick* e o seu centro são utilizados para calcular a velocidade e a direção do carro. O eixo das abcissas do *joystick* controla a direção (ajustando a diferença entre as velocidades do motor esquerdo e direito), enquanto o eixo das

ordenadas controla o movimento para a frente e para trás. As velocidades do carro são limitadas a um máximo definido.

Durante cada ciclo, a imagem de fundo atual é apresentada atrás do *joystick*. O código também armazena os valores de velocidade atuais do carro juntamente com o nome de ficheiro da imagem numa lista, que é periodicamente escrita num ficheiro CSV para análise posterior. A cada 0,4 segundos, a imagem de fundo muda para a seguinte, e os dados correspondentes do *joystick* são registados e impressos.

Quando todas as imagens de fundo tiverem sido apresentadas, o programa sai do ciclo, para a interação com o *joystick* e fecha a janela do Pygame. Os valores de velocidade e direção do carro para cada imagem são gravados, tornando esta configuração útil para gravar movimentos de carros controlados por *joystick* em sincronização com as imagens de fundo.

Preferencialmente, este método poderia funcionar em tempo real, ou seja, em vez de retirar várias imagens do percurso, guardá-las, transferi-las para o computador e classificá-las deste método, utilizar esta interface para controlar diretamente o robô ao longo deste percurso, guardando a informação de cada imagem e velocidades das rodas. No entanto, a interface do JupyterLab utilizada para remotamente configurar e controlar o sistema operativo do Raspbot não é compatível com a biblioteca Pygame, pelo que não suporta janelas *popup*. Este método direto sugerido poderia eventualmente levar a melhores resultados visto que o utilizador poderia receber *feedback* em tempo real e certificar-se que estava a controlar devidamente o robô ao longo do percurso. De facto, quando estes dados foram utilizados para treinar uma rede neuronal, e esta utilizada para controlar o robô ao longo do percurso, esta apresentou um mau desempenho, pelo que resultou de movimentos erráticos não sendo capaz de seguir a linha no chão. Com isto, procuraram-se outros métodos para classificar as imagens.

### Classificação categórica das imagens

Numa segunda tentativa para classificar corretamente as imagens de treino, optou-se por uma abordagem categórica. Com isto, cada imagem vai ser colocada sobre uma categoria pelo que a rede neuronal será treinada de forma a atribuir à imagem a categoria correta.

Desta forma adotou-se 8 categorias diferentes:

- ***'forward'***: Uma imagem com esta categoria irá ser configurada por forma a fazer o AGV andar em linha reta a uma velocidade definida;
- ***'soft_left'***: Imagens classificadas nesta categoria irão indicar ao AGV que este deve realizar uma curva suave à esquerda. O veículo deverá ajustar a velocidade de modo a curvar ligeiramente sem perder o alinhamento com a trajetória;

Tabela 5.2: Velocidade dos motores direitos e esquerdos de cada categoria.

| **Categoria** | **Velocidade esquerda** | **Velocidade direita** |
|---|---|---|
| forward | 60 | 60 |
| soft_left | 50 | 70 |
| soft_right | 70 | 50 |
| hard_left | 30 | 90 |
| hard_right | 90 | 30 |
| spin_left | -50 | 70 |
| spin_right | 70 | -50 |
| lost_line | - | - |

- ***'soft_right'***: Similar à categoria anterior, esta corresponde a uma curva suave à direita.
- ***'hard_left***: Nesta categoria, o AGV deverá realizar uma curva acentuada à esquerda, o que exige uma maior diferença de velocidade entre as rodas direita e esquerda, resultando numa rotação mais rápida para essa direção;
- ***'hard_right'***: A categoria de curva acentuada à direita. O AGV ajusta a sua velocidade de maneira a realizar uma curva rápida e mais fechada para a direita;
- ***'spin_left'***: Esta categoria será aplicada quando o AGV precisar de girar sobre si mesmo no sentido anti-horário. Ambas as rodas giram em direções opostas para realizar uma rotação no lugar;
- ***'spin_right'***: Semelhante à categoria anterior, mas desta vez o AGV roda no sentido horário.
- ***'lost_line'***: Esta categoria corresponde ao cenário em que o AGV perdeu a linha guia, e a imagem será utilizada para indicar ao sistema que deve entrar num modo de busca para encontrar novamente a linha.

A Tabela 5.2 mostra as velocidades definidas a cada motor para a respetiva categoria.

Para tal, foram criadas 8 pastas diferentes, uma com o nome de cada categoria, onde foram classificadas manualmente todas as imagens retiradas no passo anterior. Apesar de este método de classificação ser mais simples do que o método tentado anteriormente, este é um processo significativamente mais demorado. Além disso, este método é menos robusto do que o anterior, visto que com 8 categorias, o robô irá ter apenas 8 modos de operações diferentes. No entanto, utilizar mais categorias levaria também a uma maior dificuldade a classificar o *dataset* devido à maior ambiguidade da distinção dessas mesmas categorias.

### 5.2.3 Configuração e treino da rede neuronal

Com a informação obtida e classificada, esta pode finalmente ser utilizada para treinar a rede CNN (Figura 5.23).

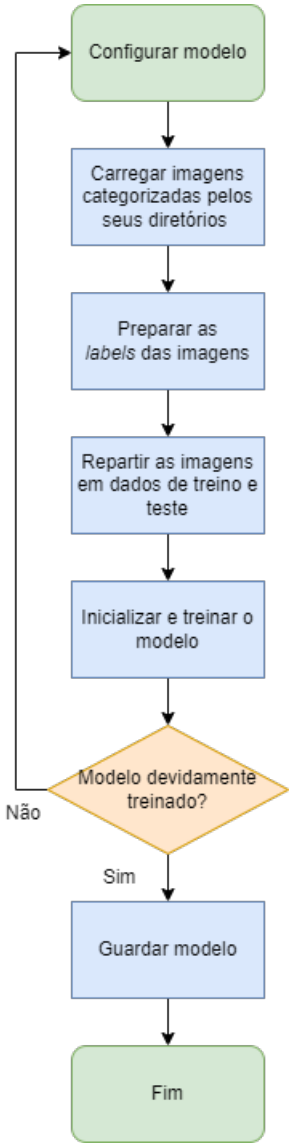



Figura 5.23: Fluxograma mostrando o processo de treino da rede neuronal CNN.

Onde:

- **Configurar modelo**: Na primeira etapa, a rede CNN é configurada utilizando a arquitetura LeNet, que é adequada para tarefas de classificação de imagens. O modelo foi configurado para receber imagens em escala de cinzentos de 36x36 pixeis e classificá-las numa de oito categorias, cada uma representando um comando de condução diferente. O modelo é compilado com o otimizador Adam, garantindo uma aprendizagem eficiente (Listagem 5.22);

```
model = LeNet.build(width=36, height=36, depth=1, classes=8)
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])

```

Listagem 5.22: Inicializar modelo CNN.

- **Carregar imagens categorizadas e atribuir *labels***: O conjunto de dados que contém imagens é carregado a partir de um diretório especificado. Cada imagem é lida, convertida para escala de cinzentos e redimensionada para 36x36 pixels para corresponder às dimensões de entrada da CNN. As imagens são armazenadas numa lista para processamento posterior. As *labels* são derivadas da estrutura do diretório: cada pasta corresponde a um comando de condução (por exemplo, '*forward*', '*soft_left*'), que é depois convertido em etiquetas numéricas (Listagem 5.23);

```
# Load images from the dataset, resize them, and convert to
    grayscale
image = cv2.imread(imagePath, cv2.IMREAD_GRAYSCALE)
image = cv2.resize(image, (36, 36))
data.append(image)
label = os.path.basename(os.path.dirname(imagePath))
# Assign numerical labels based on folder names

```

Listagem 5.23: Carregar imagens e atribuir *labels*.

A Figura 5.24 apresenta a estrutura dos diretórios criada.

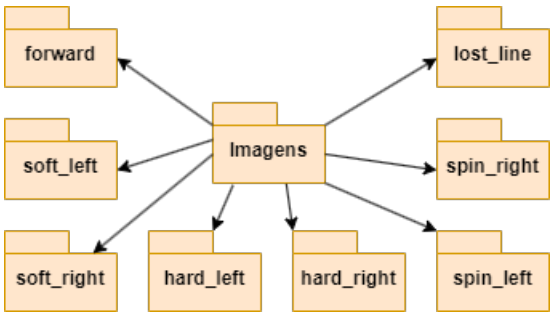



Figura 5.24: Estrutura dos diretórios do *dataset*.

- **Dividir o *dataset* em dados de treino e teste**: Para garantir uma boa generalização do modelo, o conjunto de dados é dividido em conjuntos de treino e de teste. A divisão é configurada de forma a que 75% dos dados sejam

utilizados para treinar o modelo e os restantes 25% sejam reservados para testes. Esta separação permite que o desempenho do modelo seja avaliado em dados não vistos após o treino (Listagem 5.24);

```
(trainX, testX, trainY, testY) = train_test_split(data, labels
    , test_size=0.25, random_state=42)

```

Listagem 5.24: Divisão das imagens em dados de treino e teste.

- **Treinar o modelo**: A CNN é treinada utilizando os dados de treino, com 100 épocas e um tamanho de lote de 32. Para evitar o *overfitting*, é implementado o *Early Stopping*. Esta técnica monitoriza a perda de validação e interrompe o treino se o modelo deixar de melhorar após um determinado número de épocas, garantindo que o modelo não se ajusta demasiado aos dados de treino e generaliza melhor (Listagem 5.25);

```
# Define early stopping criteria
early_stopping = EarlyStopping(monitor='val_loss', patience=4,
     restore_best_weights=True, verbose=1)

# Train the model
H = model.fit(trainX, trainY, validation_data=(testX, testY),
    batch_size=BS, epochs=EPOCHS, callbacks=[early_stopping])

```

Listagem 5.25: Treinar rede CNN.

A Tabela 5.3 mostra a lista completa dos hiperparâmetros utilizados pelo modelo.

Tabela 5.3: Hiperparâmetros do modelo LeNet

| **Hiperparâmetro** | **Valor** |
|---|---|
| Formato de Entrada | (altura, largura, profundidade) |
| Número de Camadas Conv | 2 |
| Filtros da Primeira Camada Conv | 20 |
| Filtros da Segunda Camada Conv | 50 |
| Tamanho do Filtro (Ambas as Camadas Conv) | (5, 5) |
| Função de Ativação | ReLU |
| Tamanho do *Pooling* | (2, 2) |
| Stride do *Pooling* | (2, 2) |
| *Dropout* (Após Camadas de *Pooling*) | 0,25 |
| *Dropout* (Antes das Camadas de Classificação) | 0,5 |
| Número de Camadas Densas | 2 |
| Unidades da Primeira Camada Densa | 500 |
| Unidades da Segunda Camada Densa | 500 |
| Unidades da Camada de Saída | Número de classes |
| Função de Ativação da Saída | *Softmax* |

- **Avaliação do modelo**: Após o treino, o desempenho do modelo é avaliado através da representação gráfica da perda de treino e validação ao longo do tempo. Estes gráficos fornecem informações sobre a forma como o modelo aprendeu a generalizar. Se a perda de validação continuar a diminuir, isso indica que o modelo está a melhorar (Listagem 5.26);

```
plt.plot(H.history['loss'], label='Train Loss')
plt.plot(H.history['val_loss'], label='Validation Loss')

```

Listagem 5.26: Avaliação gráfica do modelo.

- **Guardar o modelo**: Finalmente, depois de o modelo ter sido treinado com sucesso, é guardado no disco. Isto permite a utilização futura do modelo (Listagem 5.27).

```
model.save("image_model8.h5")

```

Listagem 5.27: Guardar modelo.

### 5.2.4 Rede CNN utilizada

Antes de apresentar os pormenores da arquitetura da CNN, é essencial fornecer algum contexto. A CNN aqui utilizada foi especificamente configurada para classificar imagens em categorias predefinidas, extraindo e refinando progressivamente as caraterísticas das imagens de entrada. Cada camada da rede tem um papel diferente, desde a deteção de padrões básicos nas fases iniciais até à identificação de estruturas mais complexas nas camadas mais profundas. Segue-se uma explicação detalhada da arquitetura da rede (Figura 5.25), descrevendo a função e a importância de cada camada.

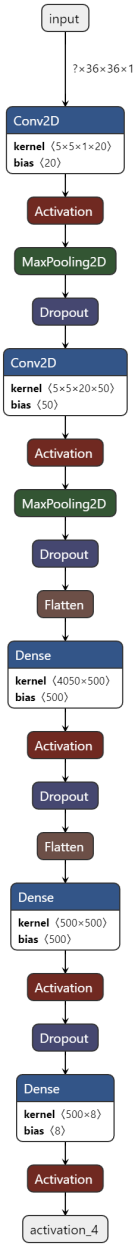



Figura 5.25: Estrutura da rede CNN utilizada.

1. **Camada de entrada**:
    - O modelo começa com o parâmetro *inputShape*, que define as dimensões da imagem de entrada (largura, altura, profundidade). Espera imagens em escala de cinzentos.
2. **Primeira camada convolucional**:
    - **Conv2D(20, (5, 5))**: Aplica 20 filtros de tamanho 5x5 à imagem de entrada, extraindo caraterísticas de baixo nível, como arestas;
    - **ReLU**: Introduz a não-linearidade, permitindo que o modelo aprenda padrões mais complexos;
    - **MaxPooling2D(2x2)**: Reduz as dimensões espaciais para metade, mantendo as caraterísticas essenciais e reduzindo a carga computacional;
    - ***Dropout* (0,25)**: Elimina aleatoriamente 25% dos neurónios para evitar o *overfitting*.
3. **Segunda camada convolucional**:
    - **Conv2D(50, (5, 5))**: Adiciona 50 filtros, extraindo padrões mais complexos dos mapas de caraterísticas da camada anterior;
    - **ReLU**: Continua a introduzir a não-linearidade;
    - **MaxPooling2D(2x2)**: Reduz ainda mais as dimensões espaciais;
    - ***Dropout* (0,25)**: Aplica a regularização, eliminando novamente 25% dos neurónios.
4. **Camada *Flatten***
    - Converte os mapas de caraterísticas 2D num vetor 1D para entrada nas camadas totalmente ligadas.
5. **Primeira camada totalmente conectada:**
    - ***Dense* (500):** Liga totalmente 500 neurónios, combinando todas as caraterísticas para uma maior abstração;
    - **ReLU:** Ativação não linear para limites de decisão mais complexos;
    - ***Dropout* (0,5):** Elimina 50% dos neurónios para melhorar a generalização.
6. **Segunda camada totalmente conectada:**
    - ***Dense* (500):** Outra camada de 500 neurónios para combinação e abstração adicional de caraterísticas;

- **ReLU:** Introduz novamente a não-linearidade;
- ***Dropout* (0,5):** Regularização para evitar o *overfitting*.

7. **Camada de saída:**

- ***Dense* (classes):** Uma camada com tantos neurónios como classes (neste caso 8), responsável pela classificação;
- ***Softmax*:** Converte a saída numa distribuição de probabilidade, atribuindo a imagem de entrada a uma das classes.

As classes foram escolhidas de forma a abranger as possíveis direções e comportamentos que o veículo autónomo (AGV) poderá assumir em diferentes cenários. Cada classe reflete uma ação necessária para manter ou corrigir a trajetória do veículo. Além disso, pelo facto de a CNN demorar consideravelmente menos tempo a treinar do que o modelo da experiência anterior (cerca de 20 s) foi possível ajustar os parâmetros do modelo até que se chegasse a resultados satisfatórios.

Como mencionado, foi configurado um *early stopping* ao treino do modelo, ou seja, ao longo do treino, este vai automaticamente avaliar o desempenho do modelo, e, caso o erro de validação subir durante 3 épocas sucessivas, o treino do modelo será interrompido. Isto foi necessário visto que, mesmo com as camadas de *dropout*, facilmente ocorria *overfitting* ao decorrer do treino. O desempenho do modelo pode ser visto da Figura 5.26.

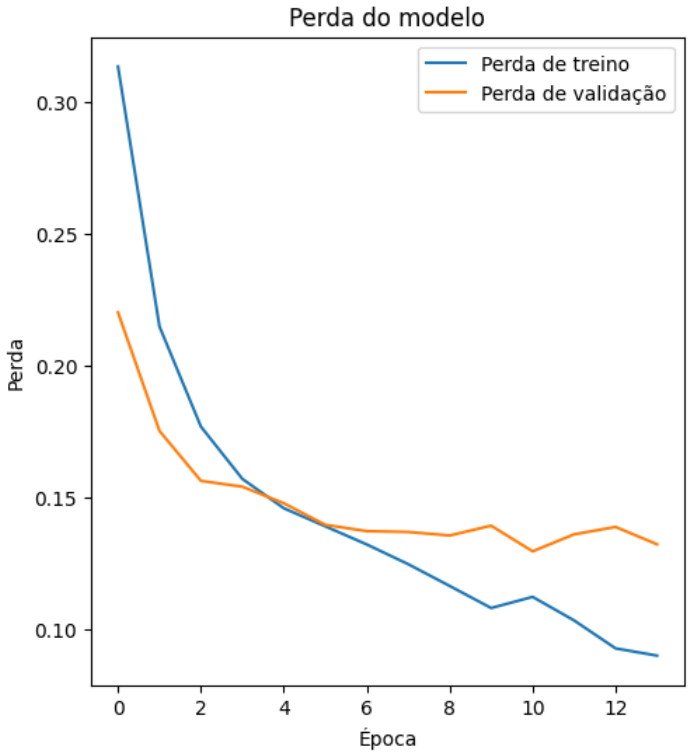



Figura 5.26: Evolução do erro da CNN ao longo do seu treino.

A Figura 5.27 mostra a matriz de confusão do modelo treinado. Nesta consegue-se ver quantas vezes o modelo foi capaz de categorizar corretamente os dados de teste. Assim, a diagonal principal mostra o número de vezes que o modelo previu corretamente a classe, enquanto que fora da diagonal principal mostra as vezes que o modelo errou, classificando uma classe como outra. Pode-se ver que o desempenho da rede é satisfatório sendo que esta é capaz de, na maior parte dos casos, de categorizar corretamente os dados com uma exatidão (o rácio entre as instâncias corretamente previstas (tanto positivas como negativas) e o número total de previsões) de 79.17%, precisão (o rácio entre as observações positivas corretamente previstas e o total de observações positivas previstas) de 74.92%, *recall* (o rácio entre as observações positivas corretamente previstas e todas as observações positivas reais) de 0.7375 e *F1-score* (a média harmónica da precisão e do *recall*. Proporciona um equilíbrio entre os dois) de 0.7360.

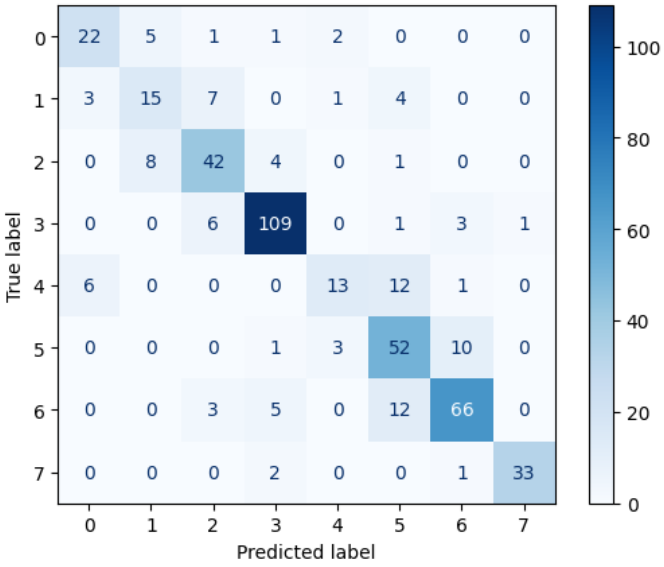



Figura 5.27: Matriz de confusão do modelo CNN.

O número correspondente a cada classe pode ser encontrado na Tabela 5.4.

Tabela 5.4: Número correspondente a cada classe da CNN

| Classe | Número correspondente |
|---|---|
| *forward* | 0 |
| *soft_left* | 1 |
| *hard_left* | 2 |
| *spin_left* | 3 |
| *soft_right* | 4 |
| *hard_right* | 5 |
| *spin_right* | 6 |
| *lost_line* | 7 |

### 5.2.5 Utilização da rede neuronal

Com o modelo treinado este finalmente pode ser utilizado para controlar o robô em tempo real. Para esta implementação, parte do código da captação das imagens pode ser reutilizado, visto que, em vez das imagens serem guardadas em ficheiros, estas serão alimentadas ao modelo. Desta forma o modelo irá categorizar as imagens em tempo real e controlar o robô, conforme a Listagem 5.28.

```
#function to control direction of robot based on prediction from
    CNN
prevLeftSpeed = 0
prevRightSpeed = 0
def control_robot(image):
  prediction = np.argmax(model.predict(image))
  global prevLeftSpeed, prevRightSpeed
  if prediction == 0:
    print("forward")
    car.Control_Car(60,60)
    prevLeftSpeed = 60
    prevRightSpeed = 60
  elif prediction == 1:
    print("soft_left")
    car.Control_Car(50,70)
    prevLeftSpeed = 50
    prevRightSpeed = 70
  elif prediction == 2:
    print("hard_left")
    car.Control_Car(30,90)
    prevLeftSpeed = 30
    prevRightSpeed = 90
  elif prediction == 3:
    print("spin_left")
```

```
24      car.Control_Car(-40,60)
25      prevLeftSpeed = -60
26      prevRightSpeed = 60
27    elif prediction == 4:
28      print("soft_right")
29      car.Control_Car(70,50)
30      prevLeftSpeed = 70
31      prevRightSpeed = 50
32    elif prediction == 5:
33      print("hard_right")
34      car.Control_Car(90,40)
35      prevLeftSpeed = 90
36      prevRightSpeed = 40
37    elif prediction == 6:
38      print("spin_right")
39      car.Control_Car(60,-40)
40      prevLeftSpeed = 60
41      prevRightSpeed = -60
42    else:
43      print("Lost line")
44      if prevLeftSpeed > prevRightSpeed: car.Control_Car(60,-40)
45      else: car.Control_Car(-40,60)
```

Listagem 5.28: Avaliação gráfica do modelo.

Nesta função, a cada categoria é associada velocidades dos motores direitos e esquerdos de forma a controlar o veículo ao longo do percurso. Estas velocidades foram obtidas depois de um processo de calibração manual, pelo que foram testadas diferentes velocidades até se chegar a valores satisfatórios.

Com isto, foi possível criar uma plataforma que permite o robô ser controlado autonomamente seguindo uma linha desenhada no chão através da câmera instalada neste. Para esta experiência foi utilizado o circuito da Figura 5.28.

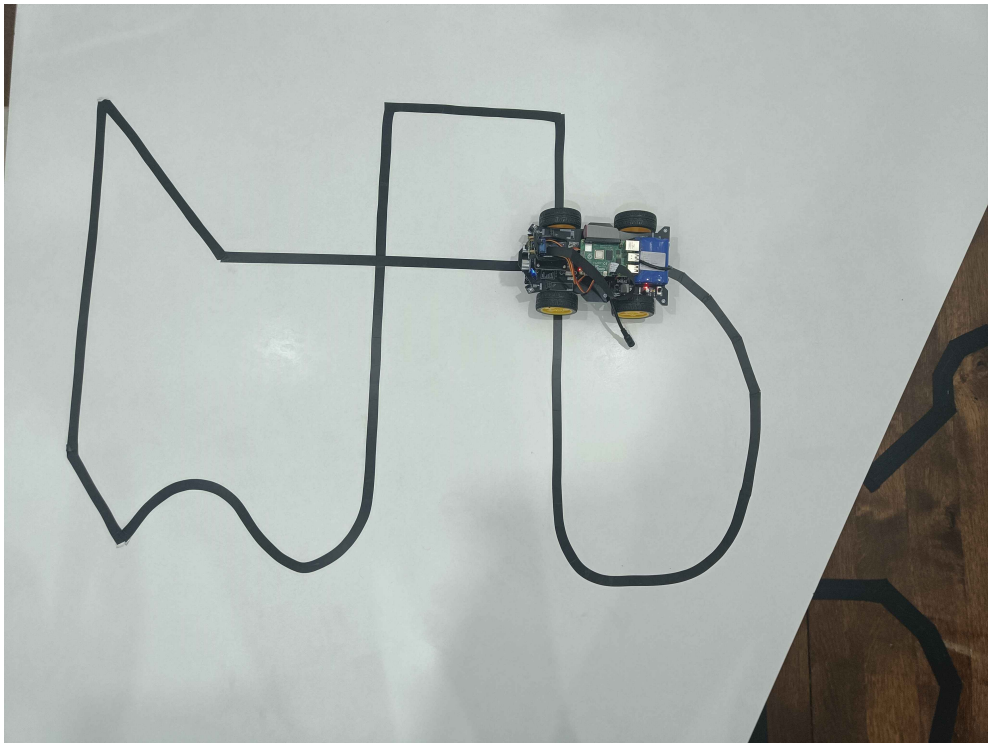


Figura 5.28: Circuito utilizado para a segunda experiência.

### 5.2.6 Desempenho do modelo

O desempenho do modelo foi influenciado por vários fatores ambientais e operacionais, que exigiram vários ajustes para garantir um comportamento ótimo. Uma das primeiras modificações foi a necessidade de mudar a pista original, que estava colocada em cima de um chão de madeira com brilho, para uma pista a preto e branco. A superfície de madeira introduziu um ruído significativo nos dados da imagem devido à sua textura e reflexos, dificultando a identificação exata do caminho pelo modelo. A pista a preto e branco, com o seu contraste mais claro, reduziu este ruído, permitindo que o modelo funcionasse de forma mais fiável.

Outra consideração importante foi a iluminação. O modelo necessita de boas condições de iluminação para detetar e processar corretamente a via. No entanto, uma iluminação excessiva provoca reflexos que interferirem com os dados da imagem, complicando a capacidade do modelo para classificar corretamente as imagens. Este equilíbrio entre muita e pouca luz revelou-se crucial para garantir um desempenho preciso.

A elevada carga computacional associada ao processamento de imagens em tempo real também afetou o sistema. Devido às exigências acrescidas do processamento de imagens, a taxa de amostragem (a frequência com que o robô processa e responde a novos dados) foi reduzida para cerca de 25 imagens por segundo. Como resultado, a

velocidade do robô teve de ser reduzida para garantir que tinha tempo suficiente para processar cada imagem com precisão, mantendo o controlo sobre os seus movimentos.

A calibração foi outro desafio fundamental ao longo deste processo. A necessidade de calibração deve-se em grande parte aos ajustes manuais necessários para as velocidades das rodas. O ajuste fino destas velocidades foi essencial para conseguir uma navegação suave e reativa. Além disso, foi necessário classificar manualmente as imagens de treino várias vezes para melhorar o desempenho do modelo, o que implicou tentativas e erros para atingir a precisão desejada.

Apesar destes desafios, o modelo teve um bom desempenho em condições ótimas. Quando devidamente calibrado, com a pista e a iluminação certas, demonstrou fiabilidade e foi capaz de navegar através de obstáculos mais complexos. A sua capacidade de adaptação a estas condições demonstra o potencial desta abordagem em cenários mais exigentes.

No entanto, devido à natureza da operação do robô, é difícil mostrar o desempenho do modelo eficazmente sem a apresentação em vídeo. No entanto, a Figura 5.29 mostra os gráficos das velocidades dos motores e da saída da CNN ao longo de uma volta do circuito.

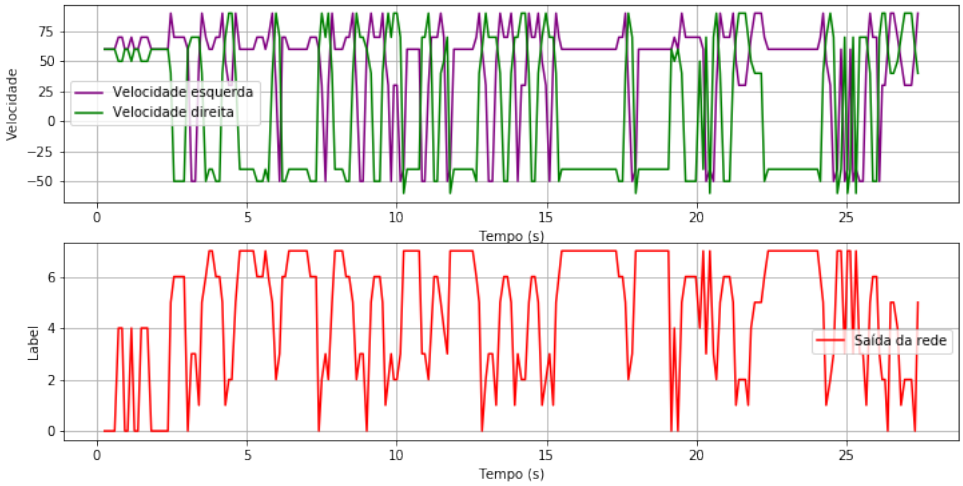



Figura 5.29: Gráficos das velocidades dos motores e da saída da CNN ao longo de uma volta do circuito.

## 5.3 Análise dos resultados

No geral, todos os métodos utilizados foram capazes de controlar eficientemente o robô ao longo de uma linha desenhada no chão. Com isto, tanto o controlador PID como a rede LSTM e a rede CNN são possíveis abordagens para este tipo de controlo. No entanto, devido à simples natureza do objetivo proposto, um robô seguidor de linha, o primeiro método é o mais eficaz pelo que os outros são também alternativas viáveis.

Observou-se que, mesmo que o controlador PID seja o método mais eficiente, este não é perfeito, visto que não consegue superar alguns obstáculos da pista, tal como ângulos agudos, algo que a rede CNN o consegue fazer. Isto mostra que, apesar de no exercício do seguidor de linha o PID tenha mais vantagens em relação aos outros métodos, as redes neuronais e o *Deep Learning* são ferramentas poderosas capazes de serem utilizadas em cenários mais complexos.

Como os métodos de controlo realizados são bastante diferentes entre si, torna-se difícil obter métricas comuns para comparar o desempenho entre métodos. A Tabela 5.5 mostra os índices de erro *Mean Squared Error* (MSE), *Mean Absolute Error* (MAE), *Root Mean Squared Error* (RMSE), tempo que cada método leva para percorrer uma volta e oscilações do robô ao longo do percurso. Estes dados foram obtidos ao executar cada método 5 vezes, obtendo os valores médios e desvios padrões de cada um.

Tabela 5.5: Comparação dos métodos

| **Método** | **MSE** | **MAE** | **RMSE** | **Tempo de circuito (s)** | **Oscilação** |
|---|---|---|---|---|---|
| PID | $10684111 \pm 515855$ | $2257 \pm 71$ | $3267 \pm 79$ | $16.40 \pm 0.22$ | + |
| LSTM | $17158701 \pm 1598305$ | $3435 \pm 242$ | $4137 \pm 192$ | $20.53 \pm 1.04$ | +++ |
| CNN | $21040166 \pm 1186665$ | $3857 \pm 188$ | $4585 \pm 128$ | $25.96 \pm 0.29$ | ++ |

As expressões dos índices de desempenho são dadas pelas seguintes equações, onde $N$ é o número total de amostras utilizadas:

$$MSE = \frac{1}{N}\sum_{i=1}^{N} erro_i^2 \tag{5.6}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |erro_i| \tag{5.7}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} erro_i^2} \tag{5.8}$$

onde o *erro* é calculado de acordo com a equação (5.5).

Como referido, numa posição central, o erro da leitura dos sensores é 0, já quando esta perde a linha, o erro passa a ser $\pm 6000$, com qualquer valor neste intervalo definindo um valor de posição intermédio. Com isto, os valores de erro, especialmente do MAE, do controlador PID são bastante satisfatórios. O método LSTM apresenta um maior erro pelo que o desempenho não é tão bom, no entanto, ainda satisfatório.

De notar que os índices do método CNN não refletem com precisão o desempenho da rede devido que este não utiliza os sensores infravermelhos para o controlo, que foram utilizados para o cálculo do erro, mas sim a câmera, pelo que os sensores infravermelhos podem não alinhar com a linha. Além disso, como a linha do segundo

circuito é menos larga, numa posição central, nenhum sensor estaria a detetar a linha, o que provocará um aumento do erro.

# Capítulo 6

# Conclusões

Concluindo, os objetivos propostos foram atingidos com sucesso. Inicialmente, foram pesquisadas e analisadas técnicas de redes neuronais aplicadas ao controlo de robôs, incluindo CNN e LSTM. Em seguida, diferentes plataformas de robôs foram avaliadas, culminando na escolha de um robô seguidor de linha adequado ao projeto.

Foram desenvolvidas e implementadas várias estruturas de controlo baseadas em *deep learning* e métodos tradicionais, que foram testadas e validadas no robô selecionado. Por fim, uma análise comparativa entre técnicas tradicionais e modernas foi realizada, demonstrando as vantagens e desafios das abordagens de *deep learning* em relação aos métodos clássicos, como o controlador PID.

Comparando os dois métodos utilizados, isto é, utilizando uma rede LSTM para controlar o Raspbot através dos sensores infravermelhos e numa segunda experiência uma CNN para o controlar utilizando a câmera é possível perceber que ambos têm as suas vantagens e desvantagens.

Por um lado, a primeira experiência, utilizando os sensores infravermelhos, por permitir uma maior taxa de amostragem (80 Hz) devido à menor complexidade do processamento de dados, permite uma maior velocidade do robô ao longo do percurso. Além disso, este é menos dependente das boas condições do meio e do circuito, desde que a diferença entre a linha preta e a superfície sejam distintas o suficiente e os sensores sejam corretamente calibrados, o método implementado deve conseguir navegar pelo circuito, mesmo com algumas falhas e desgaste da linha. No entanto, devido ao modo que o modelo foi treinado (utilizando um controlador PID) e à posição dos sensores infravermelhos, é preciso uma linha bastante larga para que,

numa linha reta, os dois sensores centrais sejam capazes de detetar a linha, o que é necessário para o bom funcionamento do modelo. Adicionando, esta implementação tem mais dificuldade para navegar circuitos mais complexos, especialmente ângulos agudos da linha.

Por outro lado, a segunda experiência, utilizando a câmara para deteção da linha, demonstrou-se mais eficaz em circuitos complexos, mas foi bastante sensível às condições ambientais. A pista de alto contraste (preto e branco) foi crucial para reduzir o ruído causado pelo piso de madeira, e o equilíbrio da iluminação foi essencial (muita luz criava reflexos que prejudicavam o processamento). Devido à elevada carga computacional, a taxa de amostragem foi reduzida, o que exigiu uma velocidade menor do robô.

A calibração foi desafiadora, exigindo ajustes manuais na velocidade das rodas e a classificação repetida das imagens para melhorar o desempenho. Ainda assim, quando corretamente configurado, o modelo demonstrou ser fiável e capaz de navegar por obstáculos mais difíceis.

Ao comparar os resultados entre os métodos com redes neuronais e o método tradicional baseado no controlador PID, algumas diferenças importantes foram observadas. As redes neuronais demonstraram a capacidade de navegar por trajetos mais complexos, lidando com curvas acentuadas e condições de pista mais desafiadoras de maneira mais eficiente que o PID. No entanto, essa eficiência não veio sem desvantagens. O método com redes neuronais foi muito mais sensível a fatores ambientais, como iluminação e textura do piso, exigindo uma calibração mais minuciosa e ajustes constantes.

Além disso, o uso de redes neuronais envolveu uma carga computacional significativamente maior, resultando em uma taxa de amostragem mais baixa (25 Hz no caso da rede CNN) e, por consequência, uma velocidade reduzida do robô. Enquanto o controlador PID, com sua simplicidade e rápida resposta, foi mais robusto e eficiente em ambientes controlados e trajetos menos complexos, as redes neuronais mostraram-se mais poderosas, mas em cenários específicos e não tão estruturados.

No entanto, o uso de redes neuronais no controlo de AGV em geral oferece um grande potencial. A capacidade das redes de aprender e adaptar-se a ambientes dinâmicos e não estruturados, além de lidar com obstáculos mais complexos, torna evidente suas vantagens para cenários mais avançados e menos previsíveis do que um simples trajeto de linha. Assim, embora para robôs seguidores de linha o PID continue a ser uma solução prática e eficiente, o futuro dos AGV claramente aponta para a aplicação de técnicas de inteligência artificial como as redes neuronais.

## 6.1 Trabalho Futuro

Para trabalho futuro, uma das primeiras possibilidades a considerar seria a implementação simulada do robô físico. A criação de um ambiente de simulação permitiria realizar testes exaustivos dos modelos de controlo sem depender da presença física do robô, o que oferece várias vantagens. Com a simulação, seria possível testar diferentes cenários, trajetos e configurações de forma mais rápida e em condições controladas, reduzindo tempo e recursos. Além disso, qualquer erro ou ajuste necessário poderia ser feito de forma mais segura e eficiente, antes de aplicar os modelos ao robô físico.

Outra vertente promissora seria a utilização de redes pré-treinadas, como redes treinadas em grandes conjuntos de dados de navegação autónoma ou de deteção de trajetos. Implementar essas redes e compará-las com os modelos desenvolvidos poderia fornecer *insights* valiosos sobre o desempenho e eficiência dessas abordagens em comparação com os métodos desenvolvidos utilizados. Além disso, o uso de *transfer learning* (adaptar uma rede já treinada para novas tarefas) poderia reduzir significativamente o tempo de treino e melhorar a precisão do modelo em determinadas condições.

Finalmente, um passo importante seria expandir as aplicações do robô para além do seguimento de linha. Poderiam ser exploradas outras tarefas, como deteção e desvio de obstáculos, navegação em ambientes dinâmicos, ou mesmo tarefas de transporte de objetos em trajetos previamente desconhecidos. A implementação de algoritmos de visão computacional mais avançados, como reconhecimento de padrões ou objetos, poderia abrir um novo campo de estudo e aplicação. Testar o robô em ambientes mais complexos, simulando cenários industriais ou urbanos, também permitiria validar a robustez dos modelos em condições mais realistas e exigentes.

# Referências


[1] A. AJuhi and S. Kumar, "A survey on artificial intelligence overview," 2020. Available at `https://api.semanticscholar.org/CorpusID:243405529`. [Citado nas páginas 7, 8 e 19]

[2] J. P. F. de Souza Allain Teixeira, "O que é inteligência artificial," 2009. Available at `https://api.semanticscholar.org/CorpusID:188971519`. [Citado na página 7]

[3] IBM, "What is natural language processing (nlp)?." Available at `https://www.ibm.com/topics/natural-language-processing/`, 2023. (Last accessed in 05/03/2024). [Citado na página 7]

[4] Vanita, "An extant of natural language processing," *International Journal For Multidisciplinary Research*, 2024. [Citado na página 7]

[5] ARM, "Pattern recognition." Available at `https://www.arm.com/glossary/pattern-recognition`, 2023. (Last accessed in 05/03/2024). [Citado na página 8]

[6] S. J. Patel and N. Lokwani, "An introduction to pattern recognition," 2015. Available at `https://api.semanticscholar.org/CorpusID:56346923`. [Citado na página 8]

[7] "A review paper on machine learning and its applications," *International Research Journal of Modernization in Engineering Technology and Science*, 2022. [Citado nas páginas 8 e 10]

[8] IBM, "What is a neural network?." Available at `https://www.ibm.com/topics/neural-networks`, 2023. (Last accessed in 05/03/2024). [Citado na página 8]

[9] "Artificial neural networks (anns, also shortened to neural networks (nns) or neural nets)," *International Research Journal of Modernization in Engineering Technology & Science*, 2023. [Citado na página 8]

[10] Shaveta, "A review on machine learning," *International Journal of Science and Research Archive*, vol. 9, pp. 281–285, 05 2023. [Citado nas páginas 9, 11, 12 e 13]

[11] N. Jain and R. Kumar, "A review on machine learning and it's algorithms," *International Journal of Soft Computing and Engineering*, vol. 12, pp. 1–5, 11 2022. [Citado nas páginas 10 e 11]

[12] M. Hall, "Deep learning," 2011. Available at `https://api.semanticscholar.org/CorpusID:204088288`. [Citado na página 13]

[13] B. Y. . H. G. LeCun Y., "Deep learning," *Nature*, p. 7, 5 2015. Available at `https://www.nature.com/articles/nature14539`. [Citado na página 13]

[14] I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *SN Computer Science*, p. 14, 8 2021. [Citado nas páginas 13, 14 e 17]

[15] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-dujaili, Y. Duan, O. Al-Shamma, J. I. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, cnn architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, 2021. [Citado nas páginas ix, xiii, 14 e 18]

[16] aakarshachug, "What is lstm – long short term memory?." `https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/`, 2024. (Last accessed in 23/07/2024). [Citado nas páginas ix, 15, 16 e 17]

[17] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Interspeech*, 2014. [Citado na página 15]

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997. [Citado na página 17]

[19] A. Ali, M. G. Yaseen, M. Aljanabi, S. A. Abed, and C. Gpt, "Transfer learning: A new promising techniques," *Mesopotamian Journal of Big Data*, 2023. [Citado na página 17]

[20] A. Upreti, "Convolutional neural network (cnn): A comprehensive overview," *International Journal of Multidisciplinary Research and Growth Evaluation*, 2022. [Citado na página 18]

[21] I. Conferences, "Automated guided vehicles." Available at `https://www.conferenceseries.com/Automated-Guided-Vehicles.php`, 2019. (Last accessed in 06/03/2024). [Citado na página 19]

[22] R. Rayner, "The pros and cons of different agv navigation systems." Available at `https://insights.antdriven.com/agv-navigation-systems-pros-cons`. (Last accessed in 08/03/2024). [Citado nas páginas ix, 19, 20, 21, 22, 23, 24 e 25]

[23] P. Denton, "Thouzer agv – the easiest agv you'll ever see." `https://jlcrobotics.com/thouzer-agv-the-easiest-agv-youll-ever-see/`, 2024. (Last accessed in 15/04/2024). [Citado nas páginas ix e 20]

[24] A. Moshayedi, J. Li, and L. Liao, "Agv (automated guided vehicle) robot: Mission and obstacles in design and performance," vol. 12, pp. 5–0018, 11 2019. [Citado na página 23]

[25] I. Robot, "4 types of agv (automated guided vehicles) navigation systems." Available at `https://ist-robot.com/types-of-agv-navigation-systems/`, 6 2021. (Last accessed in 08/03/2024). [Citado nas páginas 23 e 24]

[26] Z. H. Abdullahi, B. A. Danzomo, and Z. S. Abdullahi, "Design and simulation of a pid controller for motion control systems," *IOP Conference Series: Materials Science and Engineering*, vol. 344, 2018. [Citado nas páginas 25 e 26]

[27] K. A. Tehrani and A. Mpanda, "Pid control theory," 2012. Available at `https://api.semanticscholar.org/CorpusID:9908243`. [Citado nas páginas 25, 26, 27, 28 e 67]

[28] M. T. Inc, "Avr221: Discrete pid controller on tinyavr and megaavr devices." `https://www.ni.com/docs/en-US/bundle/labview/page/fuzzy-controllers.html`, 7 2016. (Last accessed in 12/03/2024). [Citado nas páginas 25, 26, 27, 29 e 67]

[29] R. C. Dorf and R. H. Bishop, *Sistemas de Controle Modernos*. Rio de Janeiro: LTC, 13ª ed., 2018. [Citado na página 26]

[30] A. Aboelhassan, M. A. Abdel-Geliel, E. E. Zakzouk, and M. Galea, "Design and implementation of model predictive control based pid controller for industrial applications," *Energies*, 2020. [Citado nas páginas 26, 27 e 29]

[31] F. F. França, "Controlador pid para controle de temperatura de uma carga resistiva ac," 2018. [Citado na página 27]

[32] J. Ziegler and N. Nichols, "Optimum settings for automatic controllers," *Transactions of the ASME*, vol. 64, pp. 759–768, 1942. Archived from the original on 2017-09-18. [Citado na página 27]

[33] F. Isdaryani, F. Feriyonika, and R. Ferdiansyah, "Comparison of ziegler-nichols and cohen coon tuning method for magnetic levitation control system," *Journal of Physics: Conference Series*, 02 2020. [Citado na página 28]

[34] C. H. Nikolaus Correll, Bradley Hayes and A. Roncone, eds., *Introduction to Autonomous Robots Mechanisms, Sensors, Actuators, and Algorithms*. Colorado: The MIT Press, 2022. [Citado na página 29]

[35] I. S. Jesus and R. S. Barbosa, "Application of fuzzy fractional pd+i controllers tuned by a genetic algorithm," in *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, pp. 3335–3340, 2013. [Citado nas páginas 30 e 31]

[36] R. S. Barbosa, "Educational platform for modeling and control," in *2020 XIV Technologies Applied to Electronics Teaching Conference (TAEE)*, pp. 1–9, 2020. [Citado nas páginas 30 e 31]

[37] S. Autsou, K. Kudelina, T. Vaimann, A. Rassõlkin, and A. Kallaste, "Principles and methods of servomotor control: Comparative analysis and applications," *Applied Sciences*, 2024. [Citado na página 30]

[38] R.-E. Precup, A. Nguyen, and S. Blazic, "A survey on fuzzy control for mechatronics applications," *International Journal of Systems Science*, 2023. [Citado na página 30]

[39] N. Instruments, "Fuzzy controllers." `https://ww1.microchip.com/downloads/en/Appnotes/Atmel-2558-Discrete-PID-Controller-on-tinyAVR-and-megaAVR_ApplicationNote_AVR221.pdf`, 2023. (Last accessed in 13/03/2024). [Citado nas páginas ix e 31]

[40] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path planning and trajectory planning algorithms: A general overview," 2015. [Citado na página 32]

[41] W. R. B. F. e. J. C. M. C. Carlos Eduardo Fanti, "Planejamento de trajetórias robóticas definidas por segmentos de reta concordantes por polinomiais," 2011. [Citado na página 32]

[42] B. Adorno, C. Rocha, and G. Borges, "Planejamento de trajetória para o robô omni utilizando o algoritmo mapa de rotas probabilístico," 01 2005. [Citado nas páginas ix e 32]

[43] C. X. Wong and C. F. Soon, "A bluetooth and vision controlled automatic guided vehicle," *Emerging Advances in Integrated Technology*, vol. 3, p. 53–63, Jul. 2022. [Citado na página 33]

[44] K.-H. Lin, H.-M. Chen, G.-J. Li, J.-C. Tu, and S.-S. Huang, "Automatic guided vehicle with artificial intelligence navigation," in *2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, pp. 1–2, 2019. [Citado nas páginas 35 e 36]

[45] J. Roche, V. De-Silva, and A. Kondoz, "A multimodal perception-driven self evolving autonomous ground vehicle," *IEEE Transactions on Cybernetics*, vol. 52, no. 9, pp. 9279–9289, 2022. [Citado nas páginas x, 37 e 38]

[46] R. Cupek, J. C.-W. Lin, and J. H. Syu, "Automated guided vehicles challenges for artificial intelligence," in *2022 IEEE International Conference on Big Data (Big Data)*, pp. 6281–6289, 2022. [Citado nas páginas 38 e 39]

# Anexo A

# Código e modelos

Os ficheiros referentes ao trabalho realizado (código e modelos obtidos) podem ser encontrados no repositório GitHub: `https://github.com/HugoMLeal/TEDI1190660`