



ISEL – INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA  
ADEETC – ÁREA DEPARTAMENTAL DE ENGENHARIA DE  
ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

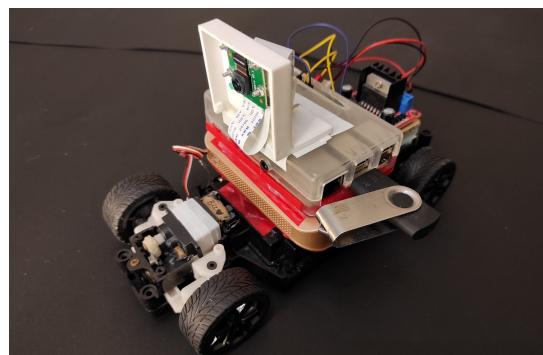
---

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA  
UNIDADE CURRICULAR DE PROJETO

---

## Carro Autónomo



Frederico Oliveira (43807)

---

*Orientador*

---

*Professor [Doutor]* Gonçalo Marques

---

*Setembro, 2021*



# Abstrato

Neste projeto é proposta uma solução de lane keeping para carros autónomos.

Ao longo deste projeto serão detalhadas a concepção, implementação e os resultados de duas soluções para o tema proposto. O objetivo principal é demonstrar que as tecnologias utilizadas podem ser aplicadas em situações reais, assim como comparar o desempenho das duas soluções propostas para o problema, nomeadamente um modelo MLP (Multilayer Perceptron) e uma CNN (Convolutional Neural Network).



# Agradecimentos

Este foi o projeto mais ambicioso e trabalhoso que tive o prazer de desenvolver até à data. Dito isto, é importante referir que não percorri este longo caminho sozinho. Quero manifestar o meu agradecimento às pessoas que me apoiaram ao longo deste projeto, sendo que as mesmas foram essenciais ao sucesso desta jornada. A minha profunda gratidão está direcionada especialmente às seguintes pessoas:

À minha mãe Ana Carvalho que sempre me motivou, me transmitiu a sua calma e que com toda a sua certeza me assegurou que eu era capaz de fazer tudo a que me dedicasse, mesmo nas fases mais difíceis. Obrigado por tudo mãe.

Ao meu pai Luis Oliveira que sempre demonstrou o seu interesse no meu projeto, que me ajudou com os aspectos mecânicos do projeto assim como arranjou todos os componentes necessários para este projeto, obrigado por tudo pai.

À minha irmã Beatriz Oliveira que me permitiu trabalhar no quarto dela e que me acompanhou durante a construção e desenvolvimento do Hardware deste projeto, o teu nome está escrito na frente do carro como símbolo da minha gratidão.

À minha namorada Joana Tomás que esteve presente em todas as fases deste projeto, que viveu e festejou comigo todas as vitórias e me apoiou durante os dissabores desta jornada e ainda, que imortalizou com um vídeo o momento em que o carro se ligou e funcionou pela primeira vez. Obrigado por tudo esquilinha, este projeto não acontecia sem o teu apoio incondicional.

Ao meu grande amigo Diogo Andrade que esteve também presente em todos os aspetos deste projeto, que generosamente me emprestou o seu RaspberryPi, que me ajudou a compilar o OpenCV para o Raspbian e que aguentou de bom grado longas conversas onde se discutia os mais variados aspectos técnicos e ideias para o projeto. Obrigado Diogo, grande abraço!

Ao meu orientador Professor Doutor Gonçalo Marques que sempre se demonstrou disponível face a quaisquer necessidades do projeto. A sua experiência e brilhante direção foram essenciais ao sucesso deste meu projeto de final de curso. Obrigado professor.

Este trabalho é tão meu como vosso, sem vocês este projeto não teria sido possível.

*Quero dedicar este projeto a 4 pessoas muito importantes para mim, embora já não estejam neste mundo o seu apoio e amor incondicional marcou-me profundamente e em conjunto com os meus pais formaram a pessoa que sou hoje, por isto estou-lhes eternamente grato.*

*Avô Zecas, Avó Zita, Avô Bino e Avó Ana  
Este projeto é dedicado em honra da vossa memória.*

*O meu eterno obrigado, espero que estejam orgulhosos de mim . . .*



# Índice

<b>Abstrato</b>	<b>i</b>
<b>Agradecimentos</b>	<b>iii</b>
<b>Índice</b>	<b>vii</b>
<b>Lista de Figuras</b>	<b>ix</b>
<b>1 Introdução</b>	<b>1</b>
1.0.1 Motivação . . . . .	2
1.0.2 Definição do problema . . . . .	2
<b>2 Hardware</b>	<b>3</b>
2.1 Introdução . . . . .	3
2.2 Alterações mecânicas . . . . .	4
2.2.1 Direção . . . . .	4
2.2.2 Motor . . . . .	5
2.3 Circuito e eletrónica . . . . .	6
<b>3 Software</b>	<b>7</b>
3.1 Introdução . . . . .	7
3.2 Controlo do carro . . . . .	8
3.2.1 Classe CarController . . . . .	8
3.2.2 Controlo remoto . . . . .	9
3.3 Construção do Dataset . . . . .	11
3.3.1 DatasetBuilder . . . . .	11
3.3.2 DatasetParser . . . . .	12
3.3.3 Data Augmentation . . . . .	13

3.4	Modelo MLP . . . . .	15
3.4.1	Observações . . . . .	16
3.4.2	Funcionamento geral . . . . .	16
3.4.3	Algoritmo de Visão . . . . .	17
3.5	Modelo CNN . . . . .	23
3.5.1	Funcionamento geral . . . . .	24
3.5.2	Topologia e Hiperparâmetros do modelo . . . . .	24
3.6	Resultados . . . . .	28
<b>4</b>	<b>Conclusões e Trabalho Futuro</b>	<b>31</b>
	<b>Bibliografia</b>	<b>33</b>

# **Lista de Figuras**

1.1	Estrada onde o carro conduz . . . . .	2
2.1	Carro original . . . . .	3
2.2	Carro após alterações . . . . .	3
2.3	Direção após alterações . . . . .	4
2.4	Motor após alterações . . . . .	5
2.5	Diagrama do circuito . . . . .	6
3.1	Funcionamento geral da comunicação . . . . .	9
3.2	Ferramenta de debug do comando . . . . .	10
3.3	Funcionamento geral da recolha de dados . . . . .	11
3.4	Variações do nível de luz . . . . .	13
3.5	Espelho da imagem original e controlos . . . . .	14
3.6	Representação de um modelo MLP . . . . .	15
3.7	Abordagem com modelo MLP . . . . .	16
3.8	Ferramenta de calibração . . . . .	17
3.9	Imagen recebida . . . . .	18
3.10	Resultado . . . . .	18
3.11	Imagen recebida . . . . .	18
3.12	Resultado . . . . .	18
3.13	Imagen recebida . . . . .	19
3.14	Resultado . . . . .	19
3.15	Imagen recebida . . . . .	19
3.16	Resultado . . . . .	19
3.17	Imagen recebida . . . . .	20
3.18	Resultado . . . . .	20
3.19	Imagen recebida . . . . .	21

3.20 Resultado . . . . .	21
3.21 Imagem recebida . . . . .	22
3.22 Características extraídas . . . . .	22
3.23 Representação de uma CNN . . . . .	23
3.24 Abordagem com uma CNN . . . . .	24
3.25 Evolução do erro ao longo do treino (Difícil comparação) . . . .	25
3.26 Visualização de coordenadas em paralelo . . . . .	25
3.27 Comparação das diferentes topologias.	
CNN                  Lite                  -                  verde;	
CNN                  Medium                -                  cinzento;	
CNN Large - laranja . . . . .	27
3.28 Regressões obtidas pelos diferentes modelos . . . . .	28
3.29 Visualização da tomada de decisões . . . . .	29

# Capítulo 1

## Introdução

Este projeto teve como objetivo a construção de um carro telecomandado, que com base na aprendizagem suportada por dados de condução humana, é capaz de se manter nos limites da estrada utilizando apenas a imagem proveniente de uma câmera instalada na frente do veículo.

Para isto foi necessário construir tanto o Hardware como o Software do carro, sendo que ambas as componentes foram construídas de raiz.

A componente de Hardware envolveu a adaptação de um carro telecomandado e dos seus respetivos constituintes (motor, servo, circuito, micro-controlador, etc..) de forma a atingir o objetivo definido. Todas as alterações feitas irão ser descritas ao longo deste relatório tal como a justificação para as mesmas.

A componente de Software deste projeto incluiu a construção do código que controla os diferentes elementos do carro e o protocolo de comunicação entre o computador e o RaspberryPi (instalado no veículo), de modo a permitir o controlo remoto do mesmo. Visto que ambas as soluções de lanekeeping propostas neste projeto são baseadas em Deep Learning, foi necessário construir os modelos em si, bem como a lógica de recolha de dados de modo a construir um dataset que foi utilizado para os treinar. O último elemento deste projeto envolveu uma comparação do desempenho de ambas as soluções para o problema proposto.

### 1.0.1 Motivação

A minha motivação para este projeto surgiu da paixão que eu tenho pela interseção das áreas de Machine Learning e Computer Vision. Sempre me fascinou a possibilidade de ensinar agentes a percepcionar o ambiente em que se inserem e tomar ações no mesmo. Dito isto, a tarefa de condução autónoma é um problema muito interessante devido à sua elevada complexidade e objetivo bem definido. Esta é uma área que espero poder impactar de alguma forma no futuro. No entanto, com os conhecimentos que tenho atualmente tomei como meta final uma versão mais simples deste problema.

### 1.0.2 Definição do problema

Como descrito anteriormente, o objetivo deste projeto foi a construção de um carro capaz de se manter dentro dos limites de uma estrada (construída usando cartolina preta e fita branca), utilizando apenas a câmera instalada na frente do veículo. Mais concretamente, este objetivo foi definido como um problema de regressão, para uma dada imagem é pedido que o carro produza os valores de aceleração e direção apropriados para esse input, sendo que o que se considera "condução apropriada" é a forma de um humano controlar o veículo ao longo da estrada.



Figura 1.1: Estrada onde o carro conduz

# Capítulo 2

## Hardware

### 2.1 Introdução

A componente de Hardware deste projeto consistiu na construção do carro em si, envolvendo tanto o aspeto mecânico como eletrónico. O projeto foi construído sobre um carro telecomandado, do qual só foi mantido o chassis original devido às alterações necessárias de modo a acomodar o novo hardware. As mesmas serão descritas em detalhe de seguida.



Figura 2.1: Carro original

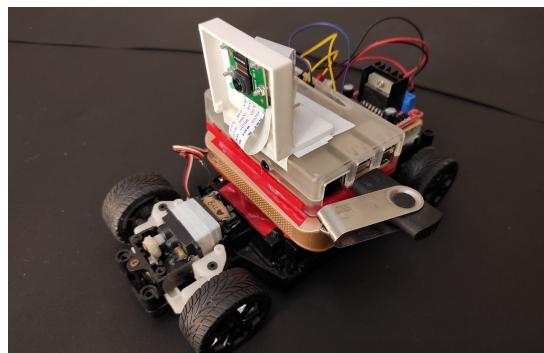


Figura 2.2: Carro após alterações

## 2.2 Alterações mecânicas

### 2.2.1 Direção

Originalmente a direção era controlada utilizando um motor DC ligado a uma roda dentada que, ou virava as rodas totalmente para a direita ou totalmente para a esquerda. O meu objetivo era que a direção pudesse ser gradual, permitindo assim um controlo suave da direção. Para isto, removi o motor DC original (este tipo de motores não permite um controlo preciso do seu ângulo de rotação) e substituí-o por um servo, adaptando também o mecanismo adjacente do eixo de direção.

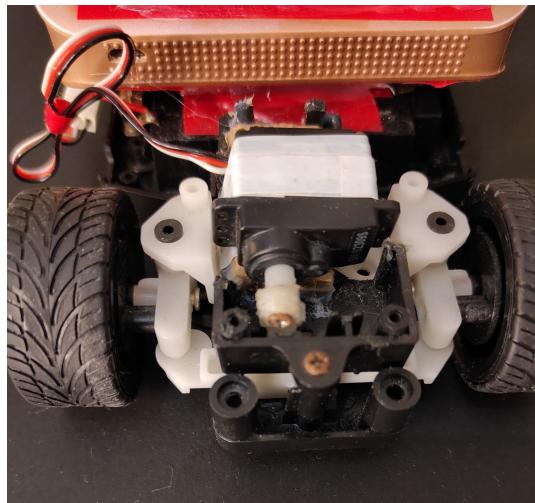


Figura 2.3: Direção após alterações

### 2.2.2 Motor

No carro original o motor responsável por movimentar o veículo era um pequeno motor DC, no entanto, após um teste de peso, reparei que este não teria potência suficiente para guiar o carro com o peso adicional dos novos componentes. Sendo assim, adaptei o chassis de modo a poder usar um motor DC mais potente.

Devido às novas dimensões do motor foi necessário fazer um corte e criar suportes no chassis para acomodar o mesmo.

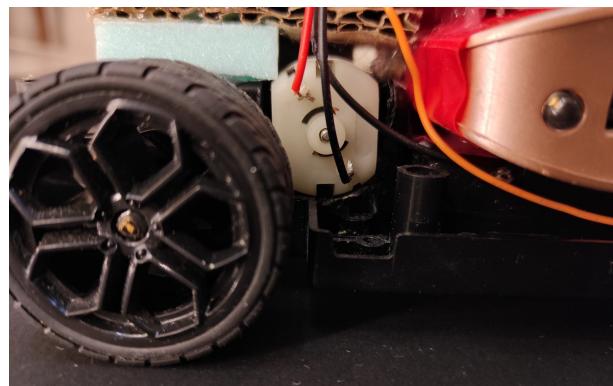


Figura 2.4: Motor após alterações

## 2.3 Circuito e eletrônica

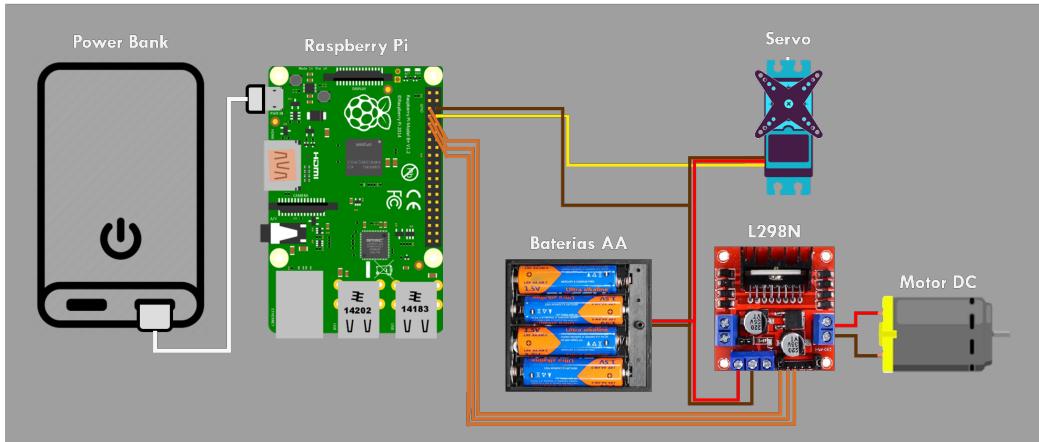


Figura 2.5: Diagrama do circuito

Todo o circuito que vinha de fábrica foi removido, tendo assim de se construir um circuito de raiz para acomodar o novo Hardware.

Para além do servo e do motor DC mencionados anteriormente, foram também adicionados:

- Um RaspberryPi3 para controlar todos os componentes programaticamente;
- Uma PowerBank para alimentar o RaspberryPi;
- Uma placa L298N, esta placa é uma dual H-Bridge, permite a inversão da polaridade do circuito que alimenta o motor, possibilitando assim que o carro faça marcha atrás;
- Uma Câmera, inicialmente usou-se uma câmera USB, no entanto, devido ao buffer usado neste tipo de câmeras e ao overhead do protocolo USB, a imagem não era fluída acabando por ser substituída por uma PiCamera. Esta câmera liga-se diretamente à slot de câmeras built-in no RaspberryPi.

Note-se que existem duas fontes de energia separadas: a PowerBank que alimenta o RaspberryPi e o alçapão de pilhas que vinha de fábrica (que foi aproveitado e alimenta o motor DC e o servo).

# Capítulo 3

## Software

### 3.1 Introdução

O software foi escrito inteiramente em Python 3.7, fazendo recurso a algumas bibliotecas de modo a simplificar o processo de desenvolvimento. Esta parte do projeto incluiu a construção do código que controla os diferentes elementos do carro, o protocolo de comunicação entre o computador e o RaspberryPi (instalado no veículo), de modo a permitir o controlo remoto do mesmo. Visto que ambas as soluções de lanekeeping propostas neste projeto são baseadas em Deep Learning, foi necessário construir os modelos em si, bem como a lógica de recolha de dados, de modo a construir um dataset que foi utilizado para os treinar. O último elemento deste projeto envolveu uma comparação do desempenho de ambas as soluções para o problema proposto.

#### Principais bibliotecas utilizadas:

- **PyTorch** - framework de machine learning;
- **OpenCV** - biblioteca de computer vision;
- **Numpy** - biblioteca otimizada para aplicar operações matemáticas sobre grandes volumes de dados;
- **Matplotlib** - biblioteca de visualização de dados;
- **PyGame** - biblioteca de criação de jogos e interfaces;
- **RPI.GPIO** - biblioteca usada no controlo da interface GPIO do RaspberryPi.

## 3.2 Controlo do carro

### 3.2.1 Classe CarController

De modo a facilitar o controlo da direção e velocidade do carro foi criada a classe CarController. Esta classe é responsável pelo controlo do servo da direção e do motor DC, através dos métodos setDirection e setSpeed.

#### Método setDirection

Este método recebe um float compreendido entre -1 e 1 que corresponde à direção que o carro deve tomar:

- -1 significa virar o máximo para a esquerda;
- 1 significa virar o máximo para a direita;

Para isto é necessário mapear este float para um número inteiro correspondente ao DutyCycle do sinal PWM enviado para o servo de modo a que este rode para o ângulo desejado.

#### Método setSpeed

Este método recebe um float compreendido entre -1 e 1 que corresponde à aceleração que o carro deve tomar:

- -1 significa acelerar o máximo para trás;
- 1 significa acelerar o máximo para a frente;

Note-se que devido a especificidades da placa L298N usada para controlar o motor do carro, é necessário gerar dois sinais PWM um para andar para a frente e outro para a marcha atrás apenas um destes está ativo num dado momento.

### 3.2.2 Controlo remoto

Devido às limitações computacionais do RaspberryPi todos os algoritmos de processamento de imagem e tomada de decisão são executados no computador, ou seja, a comunicação processa-se da seguinte forma: o RaspberryPi lê e envia a imagem da câmera, o computador recebe a imagem, processa-a e toma decisões com base na mesma, de seguida envia essas decisões (aceleração e direção) para o RaspberryPi e este executa as mesmas.

Para isto foi necessário definir um protocolo de comunicação bem como um formato de troca de mensagens eficiente.

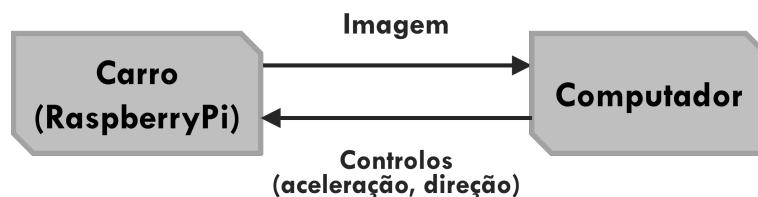


Figura 3.1: Funcionamento geral da comunicação

De modo a permitir o controlo remoto do carro por um humano foi aproveitado o protocolo implementado, a diferença é que os controlos que o carro deve tomar passam a ser gerados por um humano em vez de um algoritmo.

## Computador

Do lado do computador foram criadas as classes RemoteController e VideoStreamReceiver.

A classe RemoteController é responsável pela leitura do input proveniente de um comando da PS4 (DualShock4) e o envio desse input para o RaspberryPi através de TCP. De modo a ler os controlos executados no comando da PS4(Dualshock4) é necessário ligar o mesmo ao computador por Bluetooth visto que este tem integração nativa com o Windows. De seguida utilizou-se a biblioteca PyGame para ler os analog sticks do comando.

Foi criada também uma ferramenta de debug gráfico para assegurar o correto funcionamento do comando.

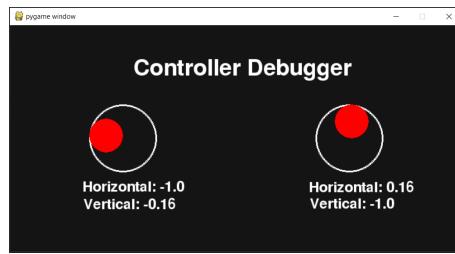


Figura 3.2: Ferramenta de debug do comando

A classe VideoStreamReceiver é responsável pela receção das imagens enviadas pelo RaspberryPi e transformação das mesmas em Numpy arrays de modo a que possam ser processadas pelo resto do código.

## Carro

Do lado do carro a lógica de envio das imagens da câmera e receção dos controlos está implementada na classe CarController nos métodos sendVideoFrame e receiveAndParseCtrls respetivamente.

### 3.3 Construção do Dataset

Para treinar os modelos foi necessário recolher dados que representam como um condutor humano controla o veículo ao longo da estrada. Com este objetivo, o condutor teve de conduzir remotamente o carro ao longo da estrada de modo a exemplificar o que é ”condução perfeita”. Neste projeto assumiu-se que a condução feita por um humano é a condução ideal, ou o que se considera o comportamento perfeito. De modo a facilitar a construção e utilização do dataset foram criadas duas classes, DatasetBuilder e DatasetParser.

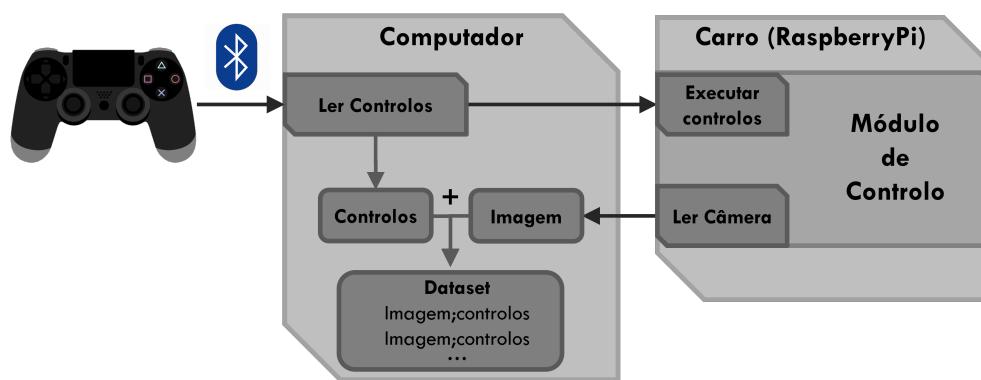


Figura 3.3: Funcionamento geral da recolha de dados

#### 3.3.1 DatasetBuilder

A classe DatasetBuilder é responsável pela construção do dataset, este irá associar uma imagem da câmera do veículo aos controlos tomados por um condutor humano nesse instante. Dada a natureza dos dados recolhidos e ao grande volume de informação gerado não é possível manter o dataset em memória e apenas no final escrever tudo para o ficheiro, em vez disto foi implementado um buffer cujo tamanho pode ser definido no construtor, desta forma os dados recolhidos vão ser guardados neste buffer, quando este estiver cheio, os dados irão ser escritos para o ficheiro que contém o dataset e o buffer será limpo, estando assim pronto para receber mais dados.

### Estrutura do ficheiro que contém o Dataset

A primeira linha é metadata, esta linha descreve a shape dos inputs e dos outputs bem como os respetivos tipos de variável, esta linha será usada pelo DatasetParser na reconstrução do Dataset.

As restantes linhas contêm os dados recolhidos, cada linha associa uma imagem flattened aos controlos tomados por um condutor humano, estes controlos são a aceleração e a direção.

#### 3.3.2 DatasetParser

A classe DatasetParser é responsável por fazer parse do ficheiro que contém o dataset e carregá-lo para memória facilitando assim a utilização do mesmo. Esta classe possui o método parseDataset, este método recebe o caminho para o ficheiro que contém o dataset que se pretende usar. Quando executado, irá ler a primeira linha do ficheiro (metadata) e utilizará esta informação para reconstruir os dados com a shape bem como os respetivos tipos de variáveis corretos.

### 3.3.3 Data Augmentation

Os modelos de DeepLearning usados neste projeto (MLP e CNN) necessitam de um grande volume de dados, estes dados necessitam de ser variados para que estes modelos sejam capazes de generalizar. Com isto em mente foi implementada a classe DataAugmenter, esta classe possui métodos que recebem uma imagem e geram variações da mesma.

#### Variação do nível de luz

Esta transformação está implementada no método generateLightVariations, esta função gera um número inteiro aleatoriamente compreendido entre -200 e 200, este inteiro é então subtraído à imagem original utilizando a função subtract do OpenCV.

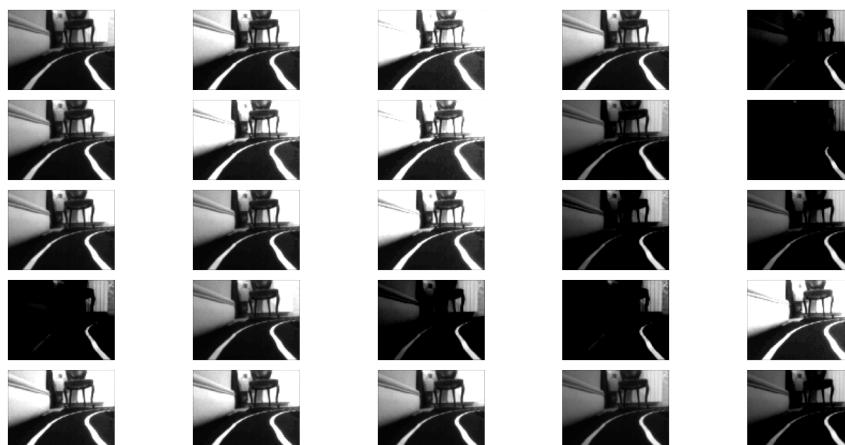


Figura 3.4: Variações do nível de luz

### Espelhar a imagem

Esta transformação está implementada no método mirrorData, este método não transforma apenas a imagem, os controlos tomados pelo condutor também são alterados de modo a que sejam congruentes com o novo sentido da estrada, a aceleração mantém-se igual mas a direção é multiplicada por -1.

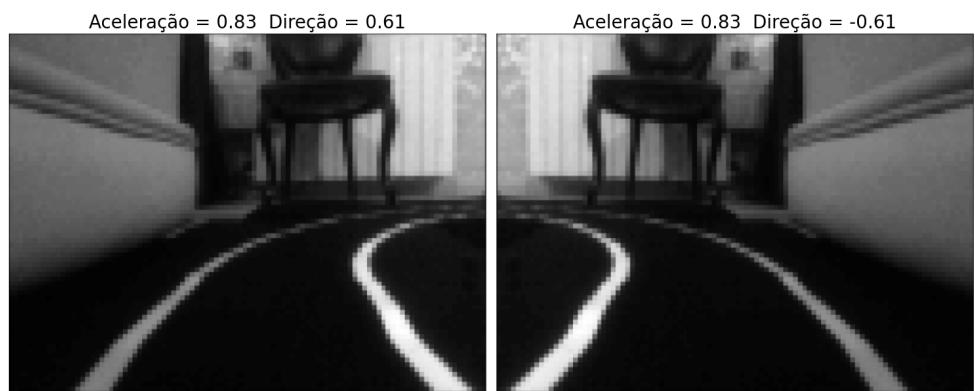


Figura 3.5: Espelho da imagem original e controlos

### 3.4 Modelo MLP

Um modelo MLP (também conhecido como MultiLayer Perceptron) é um tipo de rede neuronal composta por várias camadas de neurónios, nestes modelos, cada neurónio de uma camada está ligado a todos os neurónios da camada seguinte.

Estes modelos não estão desenhados para receber imagens, pois seria muito pesado computacionalmente sendo assim necessário um passo intermédio de processamento de imagem que extrai características da imagem como: curvatura da estrada, desvio do centro da estrada e a forma da mesma.

Estas características são então passadas ao modelo e este retorna os valores da aceleração e direção que o carro deve tomar.

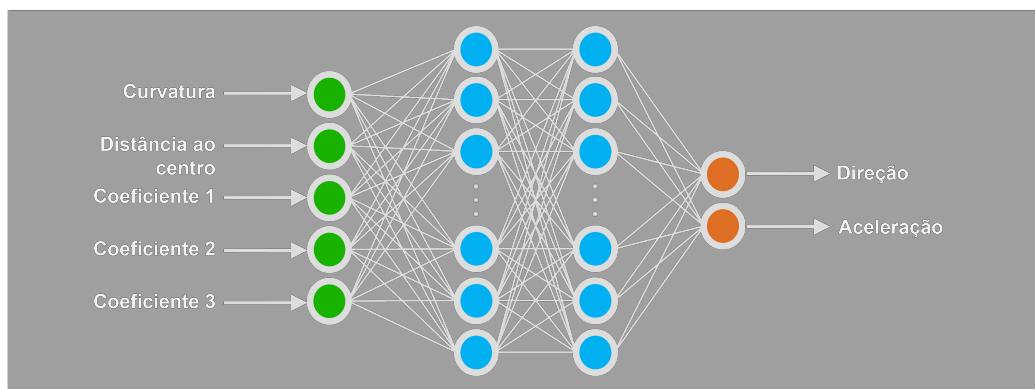


Figura 3.6: Representação de um modelo MLP

### 3.4.1 Observações

A implementação da abordagem que será descrita de seguida, não foi terminada. O algoritmo de visão criado não produziu resultados suficientemente coerentes, logo, seria impossível atingir o comportamento desejado treinando um modelo MLP sobre dados que são erróneos por natureza. No entanto, com algumas melhorias ao algoritmo de visão, nomeadamente uma suavização temporal das características calculadas pelo mesmo, esta abordagem tornar-se-ia viável. Ainda assim, dificilmente seria melhor do que uma abordagem que faz recurso a uma CNN.

### 3.4.2 Funcionamento geral

O ciclo de condução segue os seguintes passos:

- O carro lê a imagem da câmera;
- Envia a imagem para o computador;
- A imagem é passada ao algoritmo de visão e são retornadas as características extraídas pelo mesmo;
- As características são passadas ao modelo MLP e são retornados os controlos que o carro deve assumir;
- Os controlos são enviados para o carro;
- O carro executa os controlos recebidos.

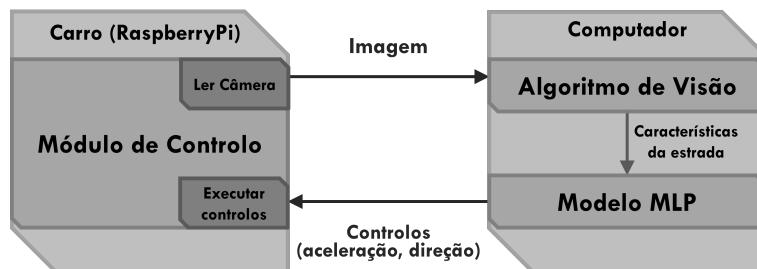


Figura 3.7: Abordagem com modelo MLP

### 3.4.3 Algoritmo de Visão

Como explicado anteriormente, este algoritmo é responsável pela extração de características da imagem relevantes à tarefa de condução. A inspiração para este algoritmo veio do repositório [Malhotra, 2017] citado na bibliografia.

#### Calibração

O algoritmo de visão construído necessita de várias constantes que terão de ser calibradas consoante as condições em que o carro está a conduzir. Para isto foi criado um script de calibração que demonstra o que o carro está a ver e permite calibrar estas constantes e observar o seu efeito no funcionamento do algoritmo em tempo real.

As constantes que têm de ser calibradas são as seguintes:

- O threshold para a segmentação;
- O y que define onde o horizonte começa de modo a ser cortado;
- Os 4 vértices que definem a região de interesse;
- Os 4 vértices que definem o trapézio usado na transformação de perspectiva;

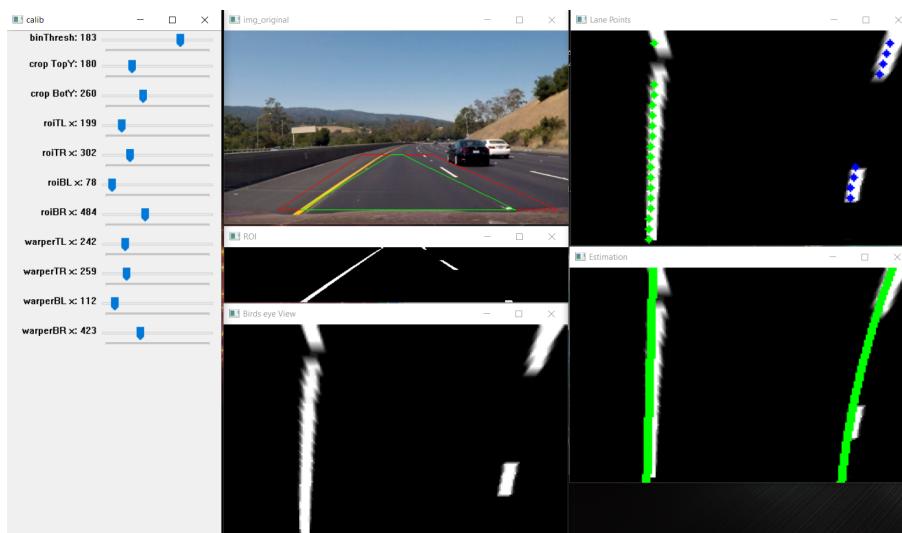


Figura 3.8: Ferramenta de calibração

### Corte do horizonte

Esta operação irá cortar o horizonte da imagem original mantendo apenas a estrada visível.



Figura 3.9: Imagem recebida



Figura 3.10: Resultado

### Segmentação e operações morfológicas

A segmentação irá transformar a imagem greyscale numa imagem binária, todos os pixeis cujo o valor está acima de um dado threshold passam a ter o valor de 1 e os restantes passam a ter o valor de 0. De seguida são aplicadas operações morfológicas à imagem binária de modo a eliminar possível ruido presente na imagem e evidenciar as linhas da estrada.

Operações morfológicas aplicadas:

- Dilatação: máscara 3x5, 1 iteração;
- Erosão: máscara 3x3, 1 iteração;



Figura 3.11: Imagem recebida



Figura 3.12: Resultado

### Região de Interesse

A máscara da região de interesse é composta por um trapézio de 4 pontos, as coordenadas destes pontos são definidas no script de calibração. Todos os pixeis que estão fora da região de interesse passam a ter o valor 0. Esta operação permite filtrar toda a informação irrelevante à tarefa de condução mantendo apenas as linhas da estrada visíveis.



Figura 3.13: Imagem recebida



Figura 3.14: Resultado

### Homografia

Este passo tem como objetivo aplicar uma transformação de perspetiva da estrada de modo a que esta passe a ser vista de cima (bird's eye view). Para esta transformação foi usado o método warpPerspective do OpenCV, este método requer um trapézio de 4 vértices (script de calibração) estes representam os da imagem transformada.



Figura 3.15: Imagem recebida



Figura 3.16: Resultado

### Identificação de pontos na estrada

Tendo a uma vista de cima da imagem é agora necessário encontrar pontos nas duas linhas da estrada, para isto aplicou-se um algoritmo denominado por "slidding window search", como o nome indica este algoritmo irá deslizar janelas pela imagem até encontrar um certo número de pixels brancos (note-se que o número de janelas bem como o número de pixels brancos necessários para definir um ponto são configuráveis) após encontrar o número de pixels necessários dentro da janela é calculado o centróide dos mesmos e guardadas as coordenadas. Este processo é repetido até se encontrar o número de pontos desejado tanto na linha da esquerda como a da direita.



Figura 3.17: Imagem recebida

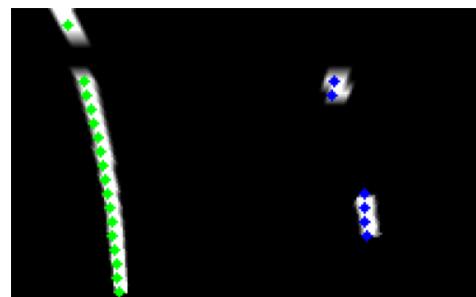


Figura 3.18: Resultado

### Estimação da forma da estrada

Após termos encontrado os centróides que definem a estrada podemos fazer uma regressão quadrática nestes pontos de modo a obter uma estimativa da forma da estrada (são feitas 2 regressões, uma para cada linha da estrada). A regressão é feita através do método polyfit do Numpy. Obtendo assim 2 funções que definem ambas as linhas da estrada.

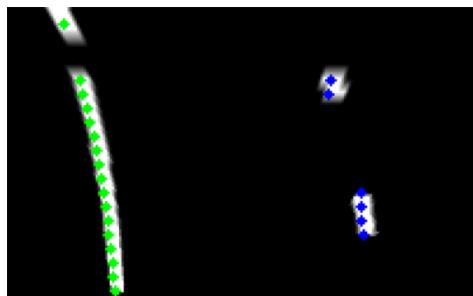


Figura 3.19: Imagem recebida

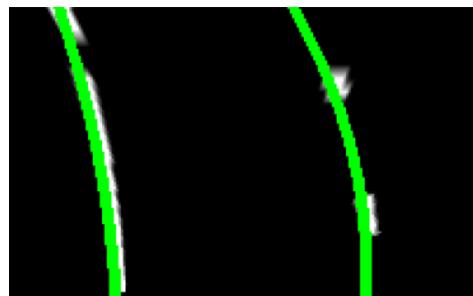


Figura 3.20: Resultado

### Extração de características

Tendo uma estimativa da forma da estrada podemos passar à extração de características da mesma.

As características extraídas são as seguintes:

- Curvatura da estrada (x do ponto mais acima da linha - x do ponto mais abaixo da linha);
- Desvio do centro da estrada (centro das linhas da estrada - centro da imagem);
- Coeficientes da regressão feita para estimar a forma da estrada.

Estas características são o output final do algoritmo de visão, estando assim prontas para serem utilizadas pelo modelo MLP.

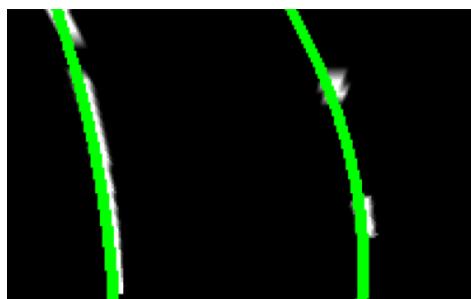


Figura 3.21: Imagem recebida



Figura 3.22: Características extraídas

## 3.5 Modelo CNN

Foi também implementada uma CNN(Convolutional Neural Network), este tipo de rede neuronal está desenhado para receber imagens diretamente eliminado assim a necessidade de um algoritmo de extração de características da imagem. Estas redes têm várias camadas e cada camada é composta por filtros ou kernels que extraem características relevantes da imagem. As ultimas camadas desta rede são fully connected permitindo assim ter dois neurónios de output que retornam o valor da aceleração e direção que o carro deve assumir como resposta à imagem recebida.

A intuição neste tipo de rede é que as primeiras camadas extraem características mais simples como linhas horizontais ou verticais e as camadas mais profundas extraem características mais complexas como uma curva à direita ou uma curva à esquerda.

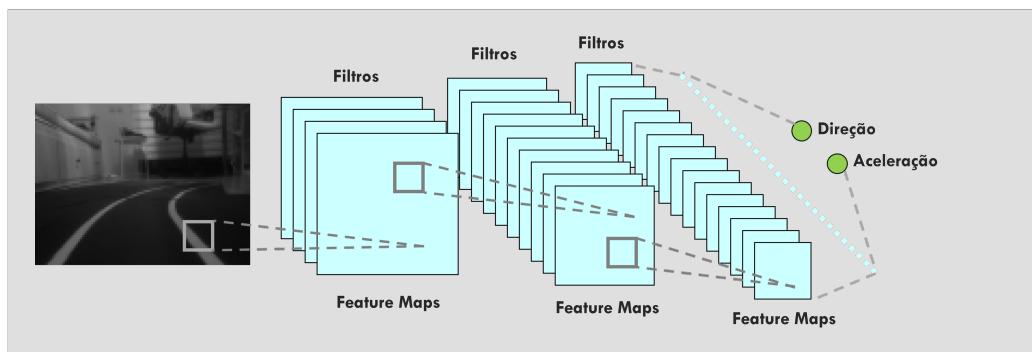


Figura 3.23: Representação de uma CNN

### 3.5.1 Funcionamento geral

Como explicado anteriormente, o diagrama de funcionamento para a condução autónoma com uma CNN é mais simples.

O ciclo de condução segue os seguintes passos:

- O carro lê a imagem da câmara;
- Envia a imagem para o computador;
- A imagem é passada à CNN e são retornados os controlos que o carro deve assumir;
- Os controlos são enviados para o carro;
- O carro executa os controlos recebidos.

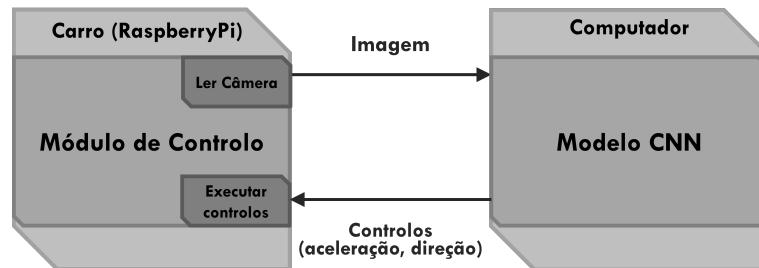


Figura 3.24: Abordagem com uma CNN

### 3.5.2 Topologia e Hiperparâmetros do modelo

De modo a treinar a CNN face à tarefa proposta foi escolhida a função de erro MSE (Mean Squared Error). Como função de erro, foi escolhido o algoritmo Adam para afinar os pesos do modelo. Como métrica MSE (Mean Squared Error) pois garante que o modelo não irá fazer previsões que produzam erros muito grandes face a um outlier.

Resumo dos parâmetros de treino:

- Otimizador: Algoritmo Adam
- Função de erro:  $MSE = \frac{1}{n} \sum_{t=1}^n e_t^2$

### Batch size e Learning Rate

De modo a encontrar pares de Batch size e Learning Rate que funcionem bem em conjunto, foi criado um pequeno script que percorre uma lista de possíveis valores para cada um deles e vai guardando a redução da perda ao longo do treino bem como um erro médio correspondente a cada par gerado. Devido ao elevado número de pares testados torna-se difícil avaliar que pares funcionam bem, optando-se assim por uma visualização de coordenadas em paralelo (implementada no tensorboard).

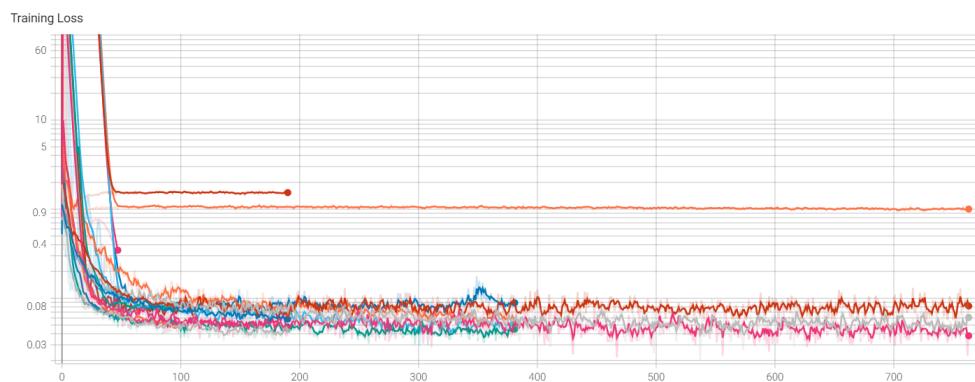


Figura 3.25: Evolução do erro ao longo do treino (Difícil comparação)

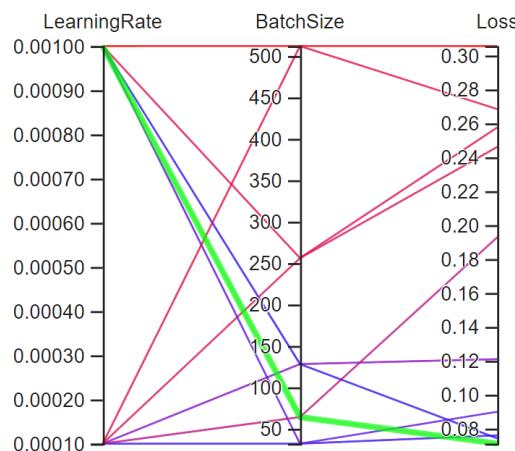


Figura 3.26: Visualização de coordenadas em paralelo

Como podemos verificar na figura acima, entre os pares testados, o que produz os melhores resultados é um learning rate de 0.001 e um batch size de 64. Estes serão os valores usados para treinar os modelos.

### Número de camadas e Filtros

De modo a encontrar uma topologia capaz de minimizar o erro da melhor forma foram testados e comparados modelos com diferentes números de camadas e filtros.

Foram implementados 3 modelos (CNN):

CNN Lite			
Layer	Nº of Kernels	Kernel Size	Activation Function
Conv2D	16	(3, 3)	ReLU
Avg Pooling 2D	-	(2, 2)	-
Conv2D	32	(3, 3)	ReLU
Avg Pooling 2D	-	(2, 2)	-
Conv2D	64	(3, 3)	ReLU
Avg Pooling 2D	-	(2, 2)	-
Linear In=1280 Out=2			

CNN Medium			
Layer	Nº of Kernels	Kernel Size	Activation Function
Conv2D	16	(3, 3)	ReLU
Conv2D	16	(3, 3)	ReLU
Avg Pooling 2D	-	(2, 2)	-
Conv2D	32	(3, 3)	ReLU
Conv2D	32	(3, 3)	ReLU
Avg Pooling 2D	-	(2, 2)	-
Conv2D	64	(3, 3)	ReLU
Avg Pooling 2D	-	(2, 2)	-
Linear In=640 Out=2			

CNN Large			
Layer	Nº of Kernels	Kernel Size	Activation Function
Conv2D	16	(3, 3)	ReLU
Avg Pooling 2D	-	(2, 2)	-
Conv2D	32	(3, 3)	ReLU
Avg Pooling 2D	-	(2, 2)	-
Conv2D	64	(3, 3)	ReLU
Avg Pooling 2D	-	(2, 2)	-
Linear In=576 Out=100			
Linear In=100 Out=2			

A comparação da evolução do erro ao longo do treino destes modelos foi feita através da seguinte figura:

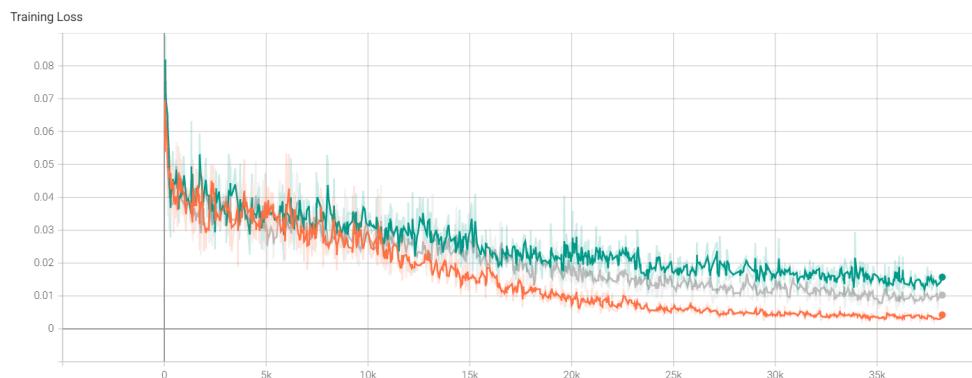


Figura 3.27: Comparação das diferentes topologias.

- CNN Lite - verde;
- CNN Medium - cinzento;
- CNN Large - laranja

Após o treino, como podemos verificar pelo gráfico, quanto maior ou mais complexo o modelo menor o erro produzido nos dados de treino. Isto não significa que o modelo é melhor, iremos explorar esta afirmação em maior detalhe na secção **Resultados**.

## 3.6 Resultados

De modo a avaliar a performance dos diferentes modelos implementados foram criadas duas visualizações que permitem uma avaliação clara dos resultados obtidos.

### Comparação das regressões

Sendo este problema definido como um problema de regressão, foi criada uma visualização que permite uma avaliação bastante intuitiva da performance dos modelos tanto na regressão da aceleração como da direção. Nos gráficos demonstrados, as linhas verdes representam os valores objetivo, as linhas azuis representam as previsões feitas pelos modelos. Note-se que cada linha (conjunto de gráficos da aceleração e direção) desta visualização representa cada modelo, neste caso foram avaliados 3 modelos.

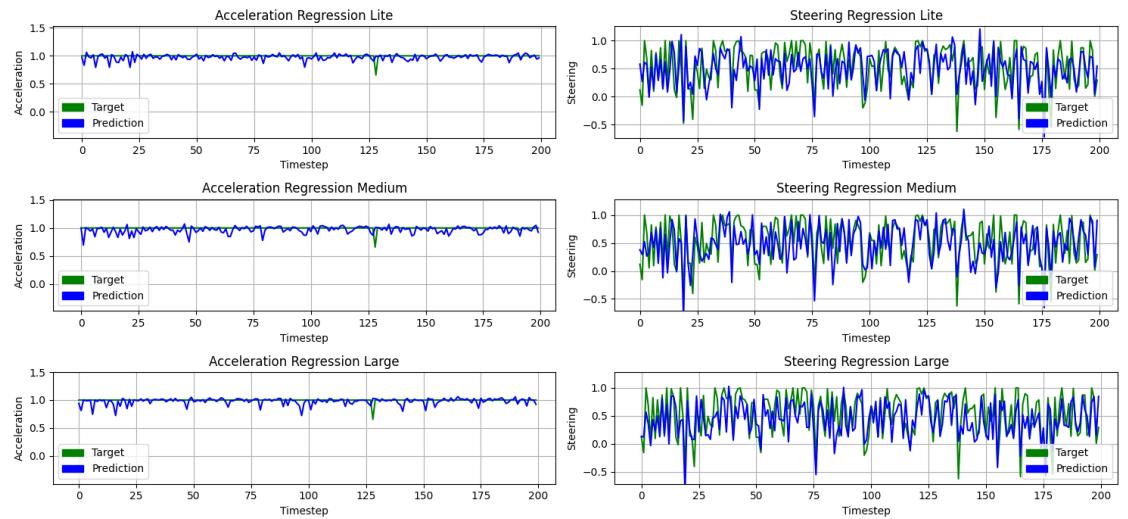


Figura 3.28: Regressões obtidas pelos diferentes modelos

**Tomada de decisões em tempo real**

Foi criada também uma ferramenta que permite visualizar as decisões tomadas pelo modelo face a uma dada imagem. À esquerda podemos ver o valor da aceleração produzida pelo modelo como resposta à imagem recebida, no centro vemos uma representação da direção.

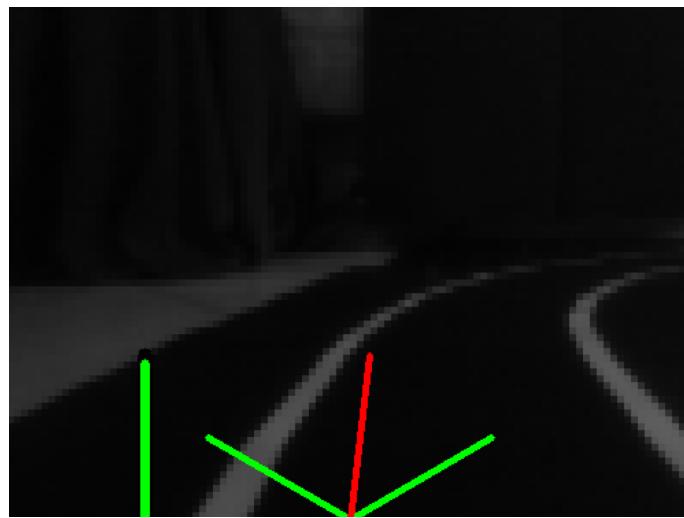


Figura 3.29: Visualização da tomada de decisões

### Análise dos resultados

Após uma análise dos resultados obtidos verificou-se que de forma geral os modelos testados parecem ter uma performance adequada face ao problema proposto. No entanto de modo a avaliar de forma mais objetiva os diferentes modelos, foi feito um teste onde foi registado o erro médio (usando a mesma métrica utilizada no treino dos modelos  $MSE = \frac{1}{n} \sum_{t=1}^n e_t^2$ ), fazendo recurso a dados que não constam no dataset de treino. Os resultados obtidos foram os seguintes:

<b>Avaliação dos modelos no dataset de teste (10475 entradas)</b>	
<b>Model</b>	<b>MSE</b>
CNN Lite	0.0464122
CNN Medium	0.0522229
CNN Large	0.0449630

A partir da tabela acima, podemos verificar que apesar dos modelos "Medium" e "Large" serem mais complexos, o erro que produzem nos dados de teste é muito próximo do modelo mais simples (no caso do modelo "Medium" o erro produzido é maior). Esta observação em conjunto com o facto de nos dados de treino se verificar que quanto mais complexo o modelo menor o erro produzido, chegamos à conclusão que os modelos Medium e Large estão a fazer "overfit" dos dados de treino, ou seja, estão a perder a capacidade de generalizar para dados novos. Optando-se assim por usar como modelo final a CNN Lite, pois entre os modelos testados, este modelo é o menos computacionalmente exigente e produz resultados equivalentes aos restantes.

## Capítulo 4

# Conclusões e Trabalho Futuro

Em suma o objetivo do projeto foi atingido com sucesso, utilizando uma CNN foi possível conduzir o carro de forma autónoma na estrada construída, utilizando apenas a câmera montada na frente do veículo.

Após uma comparação do desempenho das duas abordagens (MLP e CNN), podemos verificar que a CNN é uma solução mais robusta para este problema pois não é tão susceptível a cometer erros face a ruído na imagem, este resultado deve-se ao facto das previsões não estarem a ser feitas com base em características predefinidas calculadas a partir de um algoritmo de visão (que também é falível). As CNN's aprendem os filtros que extraem as características da imagem que são realmente importantes para a tarefa proposta.

Como trabalho futuro, gostaria de terminar a implementação da abordagem que faz recurso ao algoritmo de visão em conjunto com o modelo MLP, seria necessário fazer uma suavização temporal dos resultados do algoritmo de visão bem como a implementação de um ciclo de treino para o modelo MLP (as validações seriam idênticas às usadas para a CNN).

Penso que também seria interessante explorar a biblioteca Optuna, esta biblioteca pode ser usada para fazer uma procura automatizada dos hiperparâmetros de ambos os modelos, permitindo assim encontrar a topologia ótima para a resolução do problema proposto. Com mais tempo seria interessante recolher dados de diferentes estradas pois os modelos iriam desenvolver uma política mais geral para o problema proposto. Gostaria também de explorar a possibilidade de correr os modelos preditivos no carro, obtendo-se assim um sistema totalmente independente. Para isto talvez seja necessária

a utilização de um TPU (Tensor Processing Unit) como o que está integrado no acelerador USB Coral da empresa Coral.ai, este tipo de Hardware está especificamente dimensionado para operações com tensors, reduzindo assim o tempo necessário para um passo de inferência dos modelos implementados.

# Bibliografia

- [Albawi et al., 2017] Albawi, S., Mohammed, T. A., e Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, p. 1–6. Ieee.
- [Bojarski et al., 2016] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.
- [Malhotra, 2017] Malhotra, S. (2017). Advancedlanefinding. <https://github.com/raskolnikov-reborn/CarND-P4-AdvancedLaneFinding>.
- [Noriega, 2005] Noriega, L. (2005). Multilayer perceptron tutorial. *School of Computing. Staffordshire University*.
- [Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037.
- [Persson, 2020] Persson, A. (2020). Pytorch tensorboard tutorial.