Pontifícia Universidade Católica
do Rio de Janeiro

Leonardo Gurgel Gomes

Navegação de Robôs Móveis baseada em Rastreamento de Marcos por Visão Computacional

Projeto de Graduação

Projeto de Graduação apresentado ao Departamento de Engenharia Mecânica da PUC–Rio.

Orientador: Marco Antonio Meggiolaro

Rio de Janeiro
Julho de 2019

Agradecimentos

Ao Professor Marco Antonio Meggiolaro, orientador deste trabalho, que já orientou pacientemente tantos outros projetos meus me ajudando a polir cada vez mais minhas habilidades em projetos.

Ao meus grandes amigos da equipe RioBotz, pelos vários anos de felicidade e conquistas que partilhamos juntos.

Aos meus pais, Luiz e Angela pelo apoio durante o curso.

Aos meus colegas da empresa CyberLabs pelo apoio e conhecimento cedidos.

A todos os demais heróis e heroínas que tanto me ajudaram ao longo da minha vida e cujos nomes são tantos que não cabem nesta breve nota de agradecimentos.

Muitíssimo Obrigado.

Resumo

Navegação de Robôs Móveis baseada em Rastreamento de Marcos por Visão Computacional

No presente trabalho será projetado e construído um robô autônomo capaz de navegar através de diversos locais do campus da Pontifícia Universidade Católica seguindo marcações visuais colocadas ao longo do trajeto as quais funcionarão como *checkpoints* do trajeto e transmitirão as instruções para as ações seguintes do robô. Os motores, sensores, controladores eletrônicos de velocidade, e o microcontrolador empregados no robô são selecionados de forma que o sistema consiga realizar a missão que lhe é designada mantendo os custos do projeto no menor patamar possível. Além da seleção dos materiais físicos, também é treinado um modelo de rede neural para a detecção das marcações e é escrito um algoritmo para determinar as ações do robô. O robô é testado fazendo-o percorrer um conjunto de trajetos em diversos locais da universidade e verificando o quanto ele se desvia do trajeto proposto.

Palavras–chave

Navegação; rastreamento de marcos; robô; redes neurais;

Abstract

Mobile Robot Navigation based on Tracking Frames by Computational Vision

The goal of this work is to project and build a small autonomous robot that is capable of moving around several locations in the Pontifical Catholic University following some visual landmarks that should work as checkpoints to be found by the robot and that should tell the robot the next move that should be performed. The motors, sensors, electronic speed controllers and controllers are selected in such a way that the robot will be able to complete it's mission and keep the costs as small as possible. Also, a neural network model will be trained to detect those landmarks and a computational algorithm will be written to determine actions taken by the robot. The robot is tested by having it traverse a set of paths in several places of the university and checking how much it deviates from the proposed path.

Keywords

Navigation; landmark tracking; robot; newral network.

Sumário

Lista de Abreviaturas **8**

1 Introdução **9**
1.1 Motivação 9
1.2 Objetivo 10
1.3 Revisão Bibliográfica 10
1.4 Metodologia 11
1.5 Estrutura do Trabalho 11

2 Fundamentos Teóricos **12**
2.1 O problema de detecção e classificação de objetos 12
2.2 A rede neural artificial 15
2.3 As redes convolucionais 17
2.4 Processo de treinamento 18
2.5 O detector de objetos baseado em SSD 22

3 Seleção de componentes **23**
3.1 O kit XiaoR Geek Wi-fi Robot 23
3.2 Microcontrolador 24
3.3 Controlador eletrônico de velocidade 26

4 Lógica de programação **27**
4.1 Organização do *dataset* 27
4.2 Treinamento do detector de objetos 29
4.3 Teste do modelo 30
4.4 *Loop* de execução do programa 31

5 Resultados e Discussão **33**
5.1 Teste do modelo de detecção de objetos 33
5.2 Teste dos códigos de controle 34

6 Conclusões e Sugestões **38**

Lista de figuras

2.1 Placa de pare obtida do *dataset* 13
2.2 Foto de exemplo do detector YOLO 14
2.3 Foto de exemplo do detector YOLO com os *bounding boxes* 14
2.4 Representação muito simplificada de uma rede neural. 15
2.5 Ilustração de um classificador baseado em uma rede convolucional. Obtida do site easy-tensorflow. 17
2.6 Estrutura básica do SSD300 rodando em cima do VGG-16 conforme descrito por [1] 22

3.1 Pinos GPIO do Raspberry PI e suas funcionalidades 25
3.2 Vista superior do controlador de velocidade utilizado no projeto. 26

4.1 Exemplos de placas contidas no *dataset* 28
4.2 Gráfico mostrando o erro na saída do modelo 29
4.3 Gráfico suavizado mostrando o erro na saída do modelo 30
4.4 Funcionamento do código de controle do robô. 32

5.1 Teste para verificar se o robô detecta a placa de pare adequadamente. 35
5.2 Teste para verificar se o robô chega até a primeira placa e então localiza a segunda. 36

Lista de tabelas

5.1 Acurácia do sistema para um modelo de 4194 épocas de treino. 33
5.2 Acurácia do sistema para um modelo de 4230 épocas de treino. 33

Lista de Abreviaturas

ESC – Eletrinoc speed controller
GPIO – General Purpose Input and Output
SSD — Single Shot Detector

1
Introdução

Neste capítulo são apresentadas as principais motivações para a realização deste trabalho, bem como uma breve revisão bibliográfica.

1.1
Motivação

Desde o advento da primeira revolução industrial, tem-se ocorrido uma demanda crescente pelo emprego de máquinas em atividades anteriormente desempenhadas por humanos. À medida que o desenvolvimento tecnológico avança, surgem demandas cada vez maiores por máquinas capazes de desempenhar tarefas complexas, tais como observar o ambiente ao seu redor e tomar decisões sobre quais ações deve fazer de forma autônoma.

Dado que praticamente todos os ambientes da sociedade são feitos considerando a navegação em caráter visual, bem como os elevados custos ou mesmo a impossibilidade de se preparar um ambiente específico para um robô navegar com segurança, surge a necessidade de integrar a computação visual com a robótica de forma que as máquinas possam "enxergar"o ambiente ao seu redor.

Um veículo robótico autônomo capaz de navegar através de qualquer ambiente garante uma enorme redução nos custos de implementação e operação, dado que pode ser rapidamente empregado em praticamente qualquer ambiente habitado ou não por humanos utilizando apenas a visão, de forma similar a como a pessoas fazem.

Existem inúmeras aplicações de robôs autônomos que tiram proveito da computação visual, podendo citar:

- Veículos autônomos,
- robôs de inspeção,
- robôs para resgate de vítimas de acidentes em locais de difícil acesso.

Dessa forma, surge a inspiração para este trabalho de se criar um robô capaz de localizar algum ponto de interesse, dirigir-se até ele e executar uma determinada ação.

1.2 Objetivo

O presente trabalho tem como objetivo projetar e construir um pequeno robô autônomo estilo *rover* que seja capaz de navegar pelo ambiente utilizando como referência apenas pequenos marcadores visuais que serão colocados ao longo do seu trajeto. Dessa forma, o robô deve ser capaz de simular uma situação próxima da encontrada na realidade onde ele terá que navegar em um ambiente estranho a ele utilizando somente as referencias visuais que encontrar pelo caminho. Cada tipo diferente de referência será usada pelo robô para ajudá-lo a encontrar o próximo marco.

1.3 Revisão Bibliográfica

Tariq Rashid [2] estudou o processo de formulação e treinamento de modelos simples de redes neurais para propósitos diversos. Sua obra se foca na criação de arquiteturas simples de rede e no treinamento das mesmas para a realização de operações de inferência.

Adrian Rosebrock [3] descreve em maiores detalhes os problemas de detecção e classificação de imagens com foco primariamente na aplicação de redes neurais convolucionais. Sua obra também foca na aplicação da linguagem python junto com diversas bibliotecas para trabalho com os modelos de detecção de objetos, a saber:

- OpenCV: Utilizado para captura e tratamento de imagens,
- TensorFlow: Para cálculos numéricos, treinamento e inferência com modelos de redes neurais,
- Keras: Para projetar arquiteturas de redes neurais.

As contribuições destes autores serviram como base para a realização deste trabalho, o qual faz amplo emprego de sistemas de detecção de objetos e programação.

1.4 Metodologia

O presente trabalho descreve todo o processo de construção do robô autônomo utilizado na navegação pelo campus da universidade.

Primeiramente, realiza-se uma pesquisa de mercado para descobrir o preço e as especificações dos principais modelos de componentes eletrônicos disponíveis no mercado, começando com o microcontrolador que executará a lógica de programação, seguindo para os controladores eletrônicos de velocidade compatíveis com o mesmo e finalmente os motores e demais periféricos auxiliares que compõe o robô.

Uma vez que o robô esteja montado e os componentes individuais testados, procede-se à organização do conjunto de dados utilizados para treinar o modelo adequado de detecção de objetos para identificar as marcações que o robô seguirá. A seguir, escreve-se o código de controle do robô que utilizará o modelo para detectar os marcos, decidir os próximos movimentos e acionar os motores.

Finalmente, procede-se à etapa de testes. Os primeiros testes são feitos separadamente no modelo e nos módulos do código de controle para certificar-se de que cada um está funcionando dentro do esperado. Uma vez que o desempenho dos componentes seja considerado aceitável, procede-se ao teste em campo com o robô completo.

1.5 Estrutura do Trabalho

O presente trabalho está dividido em seis capítulos. No Capítulo 2 são apresentados os fundamentos teóricos sobre os sistemas de controle do robô, bem como os principais componentes eletrônicos. No Capítulo 3 são apresentados os componentes eletrônicos empregados, bem como os procedimentos para a seleção dos mesmos. No Capítulo 4 discute-se o programa de controle do robô, bem como os procedimentos para o treinamento da rede neural para detecção de objetos. No Capítulo 5 são apresentados os resultados dos testes em campo do robô. Finalmente, no Capitulo 6 são apresentadas as conclusões sobre o projeto bem como são apresentadas algumas ideias de trabalhos futuros para melhorá-lo.

2
Fundamentos Teóricos

Neste Capítulo é apresentada uma breve introdução aos principais componentes eletrônicos utilizados no robô, bem como aos sistemas de controle e *frameworks* de programação utilizados para controlá-lo.

2.1
O problema de detecção e classificação de objetos

O principal desafio para o robô consiste em analisar o ambiente ao seu redor e determinar onde estão os marcadores e de quais tipos eles são. Este é um caso conhecido como detecção de objetos.

Primeiramente, cabe citar a principal diferença entre a detecção e a classificação: no problema de classificação, estamos interessados apenas em dizer qual o principal item exibido na imagem, sem preocupações com relação ao seu posicionamento, tamanho ou forma. Por exemplo, se exibirmos a seguinte imagem para o classificador:

Figura 2.1: Placa de pare obtida do *dataset*

Ele dirá que a imagem contem uma placa de pare e nada além disso.

No problema da detecção, se, por exemplo, mostrarmos a seguinte imagem para o detector de objetos:

Figura 2.2: Foto de exemplo do detector YOLO

Ele citará os principais objetos contido na imagem, bem como fornecerá as coordenadas da região de interesse que contém os tais objetos. Esta região é conhecida como *bounding box*. Ao desenhar os *bounding boxes* na imagem 2.3, obtemos:

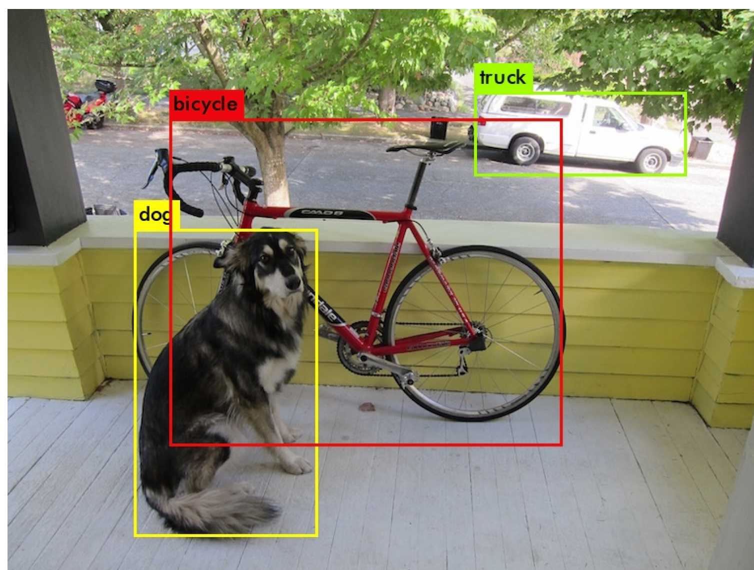

Figura 2.3: Foto de exemplo do detector YOLO com os *bounding boxes*

2.2 A rede neural artificial

As redes neurais são modelos matemáticos não lineares que visam mimetizar o comportamento do sistema nervoso humano. Tais modelos tem sido amplamente empregados para solucionar os mais diversos problemas em múltiplas áreas do conhecimento. Os modelos são dados por uma combinação de operações elementares chamadas neurônios que essencialmente recebem os sinais de entrada dos neurônios anteriores. Os sinais são então ponderados pelos pesos das conexões e somados.

Assim como acontece também no sistema nervoso, o sinal resultante é passado por uma função de ativação que serve para filtrar sinais menos significativos. Finalmente envia-se um sinal de saída para os neurônios subsequentes. O processo continua até que se chegue aos nós de saída da rede, os quais determinam o resultado da operação [2].

A operação de se alimentar os nós de entrada com os dados de trabalho e coletar os resultados nos nós de saída é chamada de inferência. Tal operação também é muitas vezes chamada de *foward propagation* em alusão ao sentido de cálculo da rede.

Uma representação gráfica simplificada de uma rede neural é dada pela figura 2.4 a seguir:

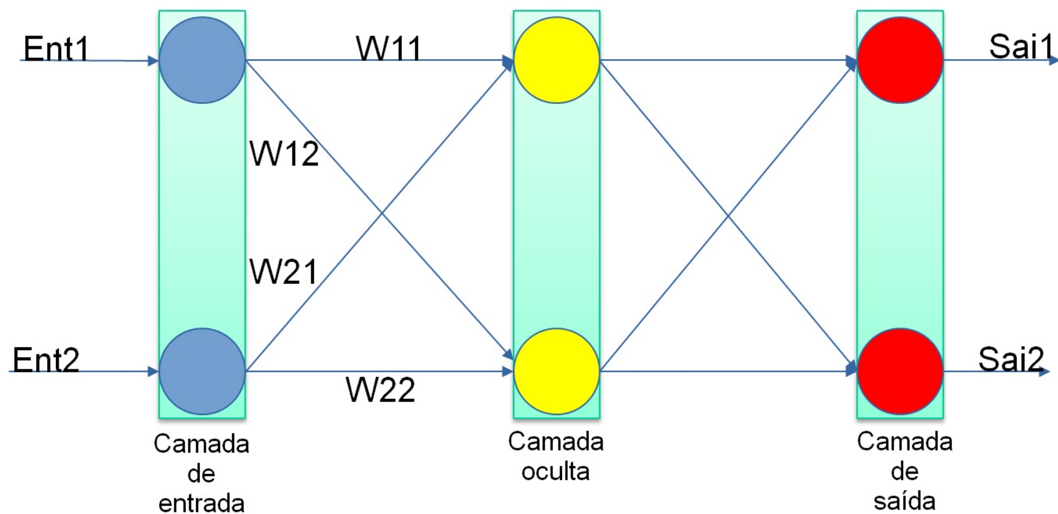

Figura 2.4: Representação muito simplificada de uma rede neural.

Considerando a rede mostrada anteriormente, se aplicarmos os sinais de entrada chamados "Ent1"e "Ent2", nos nós indicados, o sinal resultante passado para a segunda camada será dado por:

$$\begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix} \begin{pmatrix} Ent1 \\ Ent2 \end{pmatrix} = \begin{pmatrix} Sai1 \\ Sai2 \end{pmatrix} \tag{2-1}$$

Presumindo-se a aplicação de uma função de ativação sigmoidal, basta que os valores de "Sai1"e "Sai2"sejam passados pela função:

$$sai_{novo} = \frac{1}{1 + e^{-sai_{antigo}}} \tag{2-2}$$

Com isso, temos os sinais que serão enviados para as próximas camadas da rede. O processo continua até que que se chegue na última camada, que contém os nós de saída. Os dados contidos na saída são os dados para uso no restante do programa.

Importante frisar que neste exemplo tratou-se de uma rede com apenas duas camadas eu uma função de ativação sigmoidal, sendo que cada neurônio enviava o sinal para todos os neurônios subsequentes. Na prática, os modelos empregados no dia-a-dia podem conter diversas camadas intermediárias entre a entrada e a saída, conhecidas como camadas ocultas, ou escondidas. Também é possível que os nós se retroalimentem ou alimente apenas alguns nós na próxima camada. Com relação às funções de ativação, diversas outras funções podem ser empregadas, tais como a função de passo, a tangente hiperbólica, e funções de rampa. Por fim, cada modelo pode incorporar operações matemáticas mais complexas no transcurso dos cálculos através da rede dependendo do problema em questão.

2.3
As redes convolucionais

Para o uso na detecção de objetos, prefere-se o uso de redes neurais convolucionais, onde a cada camada da rede uma operação de convolução é aplicada nos sinais recebidos.

Tal tipo de rede é preferível com relação às demais devido ao fato de que uma imagem é representada em um computador por uma matriz contendo os dados dos *pixels* a serem exibidos. Para processar esses dados diretamente em uma rede neural comum, teria-se que passar cada pixel como entrada de um neurônio e executar os cálculos um a um, o que seria inviável para imagens de dimensões elevadas. Há ainda a possibilidade de escrever manualmente os filtros a serem aplicados para extrair as características desejadas das imagens e ai sim utilizá-las para alimentar a rede.

Pode-se dizer que a cada operação de convolução, uma parte da imagem original é convertida em uma imagem menor contendo apenas alguns padrões geométricos da secção original. Ao fim de todo o processo, obtém-se um vetor final com apenas as *features* de interesse. Uma nova camada, então, utiliza esses dados para alimentar uma nova camada da rede que realiza apenas uma operação de aplicação de pesos, como a descrita anteriormente, a qual é responsável por encontrar a classe a qual o objeto pertence. A figura 2.5 mostra, de forma simplificada, esse processo.

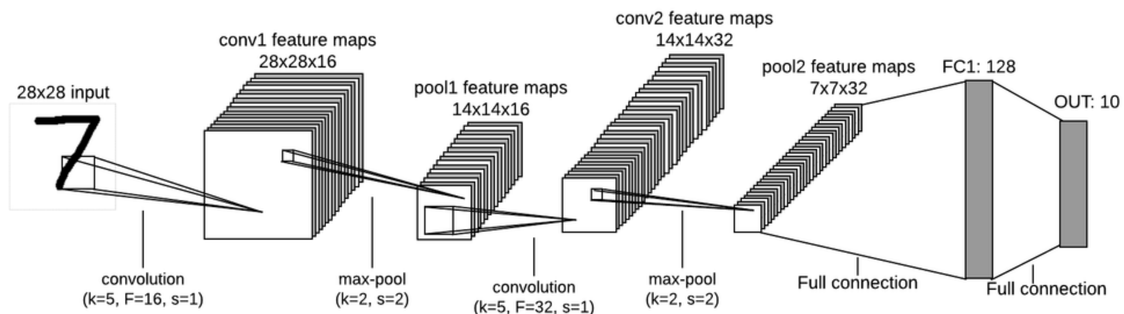

Figura 2.5: Ilustração de um classificador baseado em uma rede convolucional. Obtida do site easy-tensorflow.

Com esse processo a rede convolucional pode analisar a imagem em pequenos blocos e determinar as características mais relevantes para cada classe de objeto que se pretende identificar, eliminando a necessidade implementar filtros manualmente.

2.4 Processo de treinamento

2.4.1 Tipos de treinamento

Uma vez que se tenha a estrutura da rede neural pronta, é necessário descobrir os valores adequados dos pesos das conexões. São os valores dos pesos que definem o conhecimento da rede sobre os objetos a serem detectados. O processo de se ensinar a rede quais são os valores adequados dos pesos se chama treinamento.

No aprendizado de máquinas, o processo de treinamento pode ser classificado em três tipos diferentes dependendo da forma que os dados são trabalhados:

- Supervisionado,
- Semi-supervisionado,
- não supervisionado.

A principal diferença entre as formas de treinamento decorre da forma que o sistema analisa o conjunto de dados para o treinamento, conhecido como *dataset*. No treinamento supervisionado, que é o foco do presente trabalho, além de fornecermos os dados básicos para o treinamento, fornecemos também qual a resposta esperada pela rede para os tais dados. No caso de um detector de objetos, por exemplo, forcemos as imagens com os objetos a serem detectados e as coordenadas dos *bounding box* dos objetos. No treinamento não supervisionado, os dados são fornecidos brutos, e espera-se que o computador encontre um padrão que possa usar para trabalhar com os dados. No treinamento semi-supervisionado, apena parte do *dataset* contém as anotações, devendo o computador complementá-los conforme os padrões que encontrar [3].

2.4.2 Retropropagação e método dos gradientes

Para executar o processo de treinamento propriamente dito, deve-se inicializar a rede com valores arbitrários dos pesos. O processo começa com uma operação comum de inferência alimentando os nós de entrada com os dados de trabalho e verificando os resultados nos nós de saída.

Com os resultados das operações da inferência, compara-se o resultado obtido com o desejado, chegando assim aos erros da operação. Como existem diversos pesos na rede e cada um é responsável por uma parte do erro, chega-se ao problema de descobrir qual a contribuição de cada peso na geração do erro. Um processo comumente utilizado neste caso é conhecido como *backwards propagation*, em que os valores dos erros são passados pela rede no sentido inverso, multiplicando-os pelos pesos. Esta operação é importante devido ao fato de que precisamos encontrar não só o erro no final da rede, mas também os erros nas camadas intermediárias. O erro em cada camada oculta é dado pelo produto entre a transposta da matriz dos pesos pelo erro na camada subsequente. Se tomarmos, por exemplo, a rede mostrada na figura ?? e incluirmos uma camada oculta com dois nós entre as camadas de entrada e saída, podemos calcular os erros reportados pelos nós da camada oculta pela fórmula:

$$\begin{pmatrix} erro_{oculta1} \\ erro_{oculta2} \end{pmatrix} = \begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix} \begin{pmatrix} erro_{sada1} \\ erro_{sada2} \end{pmatrix} \qquad (2\text{-}3)$$

Com os erros em mãos, o método dos gradientes pode ser utilizado para encontrar os valores adequados para os pesos utilizando-se um método conhecido como método dos gradientes. Este método se baseia em tentar descobrir o menor valor de uma dada função que correlaciona o erro com o peso a ser modificado. Uma simples é de tentar encontrar o ponto que a derivada da função com relação ao peso seja o menor possível. O novo valor do peso será dado pelo antigo valor subtraído de uma proporção da sua variação função de erro. Matematicamente, temos:

$$peso_{novo} = peso_{antigo} - \alpha \frac{\partial E}{\partial w_{j,k}} \quad (2\text{-}4)$$

Na equação anterior, $w_{j,k}$ é o peso em questão, E representa a função de erro e α é a taxa de aprendizagem, que mede o quão sensível à variação de erro o ajuste do peso vai ser.

Para encontrar o valor de E, pode-se utilizar o método dos mínimos quadrados, em que obtemos E pela subtração do valor desejado do erro do valor obtido, e então elevamos o resultado ao quadrado. A expressão fica:

$$\frac{\partial E}{\partial w_{j,k}} = \frac{\partial}{\partial w_{j,k}} (t_k - o_k)^2 \quad (2\text{-}5)$$

Onde t_k é o valor desejado e o_k é o valor de erro obtido. Aplicando a regra da cadeia em E e expandindo, obtemos:

$$\frac{\partial E}{\partial w_{j,k}} = -2(t_k - o_k) \frac{\partial}{\partial w_{j,k}} F_a(\sum_j w_{j,k} o_j) \quad (2\text{-}6)$$

Na equação anterior temos que:

- o_k é o erro obtido no nó final,
- o_j é o erro obtido no nó anterior,
- F_a é a função de ativação do nó anterior.

Pode-se perceber pela equação que esse método só é válido quando a função de ativação tem uma derivada definida no ponto em questão [2].

Por fim, ainda sobre o processo de treino, é conveniente ressaltar que:

- O processo descrito anteriormente é apenas uma etapa do processo de treinamento. O processo completo se baseia em fornecer diversos conjuntos de dados e gradativamente ajustar os pesos.
- Os modelos de processamento de imagens são organizados de forma que possam processar múltiplas imagens de uma vez. O processo de agrupar e trabalhar diversas imagens ao mesmo tempo é conhecido como *batching*.
- Durante a comparação dos modelos ajustados com o *dataset* de testes, é comum observar que, após certo ponto no treinamento, o erro começa a crescer. Este fenômeno indica um ajuste excessivo do modelo conhecido como *overfitting*.
- O tamanho do *batch* e os parâmetros de aprendizagem são conhecidos como hiper-parâmetros, e devem ser determinados manualmente antes do processo de treino.

2.5 O detector de objetos baseado em SSD

Para emprego no presente trabalho, o processo de detecção será feito pelo método conhecido como *single shot detection* o qual utilizará uma arquitetura de rede conhecida como *mobilenet*.

O procedimento mais simples para detectar a posição de um objeto em uma dada imagem em uma única etapa consiste em dividir a imagem original em diversas sub-imagens com dimensões e coordenadas conhecidas com relação à imagem original. As novas imagens obtidas podem ser passadas através de uma rede neural de classificação como um único *batch* que retornará, para cada segmento da imagem, as classes e o percentual de confiança. Conhecidas as coordenadas das sub-imagens originais, basta sobrepor as mesmas na imagem original, resultando nos *bounding boxes* desejados.

O *single shot detector* funciona de maneira semelhante, porém ao invés de dividir toda a imagem inicial em imagens menores e aplicar a classificação diretamente nessas imagens, ele extrai um mapa de *features* de cada operação de convolução antes da classificação propriamente dita. Após extrair os mapas, esses informações são levadas diretamente ao classificador que decide a qual classe o objeto pertence. Dessa forma as próprias operações de convolução funcionam como a operação de mapeamento [1].

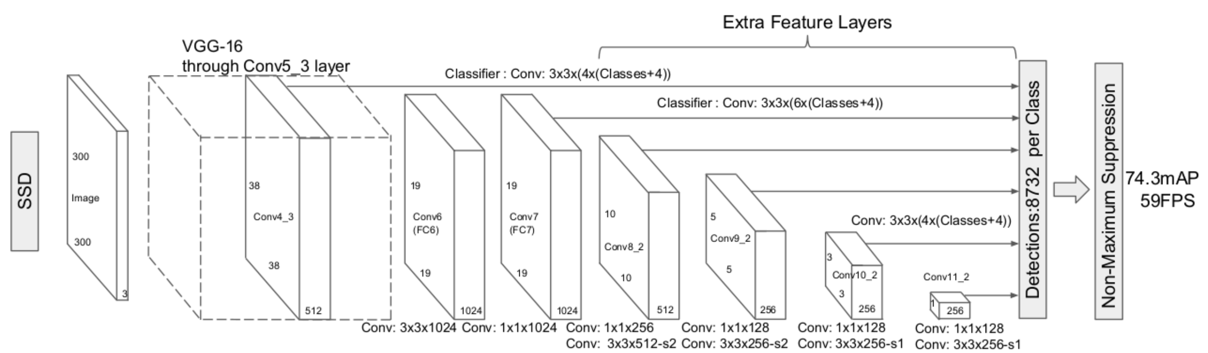

Figura 2.6: Estrutura básica do SSD300 rodando em cima do VGG-16 conforme descrito por [1]

3
Seleção de componentes

Neste capítulo discute-se em maiores detalhes os componentes eletrônicos utilizados, bem como os procedimentos para a seleção dos mesmos. Os principais componentes do robô são:

- Microcontrolador
- Controlador eletrônico de velocidade
- Motores
- Câmera
- Chassis

Cada componente citado acima foi selecionado considerando-se as características descritas no escopo original do projeto.

Uma vez que os componentes principais tenham sido decididos, foi comprado um kit com os componentes eletrônicos para montagem.

3.1
O kit XiaoR Geek Wi-fi Robot

Durante as pesquisar pelos componentes para construir o robô, foram verificados alguns kits com componentes pré-selecionados para robôs autônomos. O emprego do kit foi preferido com relação à ideia de se montar o robô inteiro do zero devido ao fato de que, como existe a necessidade de se encomendar peças do exterior e fabricar alguns componentes estruturais, poderia não haver tempo hábil para se chegar ao final do projeto.

Para a escolha do pacote adequado, primeiro fez-se uma pré-seleção dos componentes eletrônicos desejáveis para o robô considerando as característica descritas no escopo do projeto. Cada componente é melhor descrito em maiores detalhes nas próximas seções.

Com os componentes decididos, procede-se a uma busca no mercado pelos kit que contenham os componentes adequados. O modelo XiaoR Geek Wi-fi Robot foi escolhido devido ao seu baixo custo e extensa documentação, que auxiliou na montagem e modificação dos componentes.

3.2 Microcontrolador

O microcontrolador é a parte responsável por executar os códigos de controle e enviar as instruções para o ESC.

Ressalvadas as placas da série Arduíno, que executa um código compilado em linguagem própria, todos o controladores considerados para serem utilizados são de uma classe de computadores conhecidos como *single board computers* (SBC's). Tais tipos de placas podem ser considerados essencialmente como computadores pessoais comuns criados sob a forma de uma única placa de pequenas dimensões, sendo capazes de executar a maioria dos programas de computador comuns bem como utilizar um sistema operacional de computadores pessoais com pouca necessidade de alteração em seus códigos. Tais características são desejáveis dado que o projeto já foi planejado para tirar proveito de alguns *frameworks* de programação utilizados para computação visual, os quais devem ser executados em cima da linguagem python.

Os principais microcontroladores considerados foram:

- Raspberry PI
- Arduíno
- Asus Tinkerboard
- Nvidia Jetson

Dentre as opções citadas acima, escolheu-se o Raspberry PI 3B como controlador a ser utilizado no projeto. A seleção do controlador foi feita considerando-se o menor custo da placa com relação às demais, a facilidade de uso por ser compatível com as ferramentas de programação escolhidas para esse projeto.

O fato de que mesmo com um poder de processamento mais limitado quando comparado ao Tinkerboard e ao Jetson, o modelo de detecção de objetos escolhido funciona sem maiores problemas no Raspberry PI, por isso não se justifica utilizar uma placa com maior poder de processamento. O Arduíno foi descartado devido à maior dificuldade de se programá-lo.

Para transmitir o sinal para o controlador do motor, o Raspberry PI utiliza um jogo de pinos conhecidos como GPIO, acrônimo para *General Purpose Input and Output.* O sinal de saída de cada pino pode ser programado utilizando-se a biblioteca padrão de programação do Raspberry PI. A figura 3.1 mostra a disposição dos pinos do GPIO.

GPIO#	2nd func.	Pin#		Pin#	2nd func.	GPIO#
	+3.3 V	1		2	+5 V	
2	SDA1 (I²C)	3		4	+5 V	
3	SCL1 (I²C)	5		6	GND	
4	GCLK	7		8	TXD0 (UART)	14
	GND	9		10	RXD0 (UART)	15
17	GEN0	11		12	GEN1	18
27	GEN2	13		14	GND	
22	GEN3	15		16	GEN4	23
	+3.3 V	17		18	GEN5	24
10	MOSI (SPI)	19		20	GND	
9	MISO (SPI)	21		22	GEN6	25
11	SCLK (SPI)	23		24	CE0_N (SPI)	8
	GND	25		26	CE1_N (SPI)	7
(Pi 1 Models A and B stop here)						
EEPROM	ID_SD	27		28	ID_SC	EEPROM
5	N/A	29		30	GND	
6	N/A	31		32		12
13	N/A	33		34	GND	
19	N/A	35		36	N/A	16
26	N/A	37		38	Digital IN	20
	GND	39		40	Digital OUT	21

Figura 3.1: Pinos GPIO do Raspberry PI e suas funcionalidades

Cabe frisar que a tensão elétrica nos pinos do GPIO varia entre 3,3 e 5 volts e a corrente suportada chega a até 50 miliamperes. Como o motor selecionado utiliza tensões variando de 3 a 6 volts e consome uma corrente que varia de 180 a 250 miliamperes, é necessário utilizar um controlador eletrônico de velocidade para controlar os motores.

3.3 Controlador eletrônico de velocidade

Os controladores eletrônicos de velocidade (ESC's) são os dispositivos responsáveis por alimentar os motores elétricos de forma que eles mantenha a velocidade adequada, bem como o sentido de rotação desejados.

A seleção do ESC foi considerada mais restrita, pois foi necessário que este funcionasse bem com o Raspberry PI.

Com base nos modelos sugeridos pela Xiao Geek, selecionou-se o PWR.A53.A como controlador. Tal controlador possui como vantagens o fato de que ele pode ser acoplado diretamente ao GPIO do Raspberry PI, convertendo diretamente o sinal emitido por ele para o motor. Com isso, toda a parte da lógica de programação da ponte H e dos PWM's para controlar os motores pode ser feita diretamente pelo próprio Raspberry PI.

A figura 3.2 mostra a vista superior do controlador utilizado:

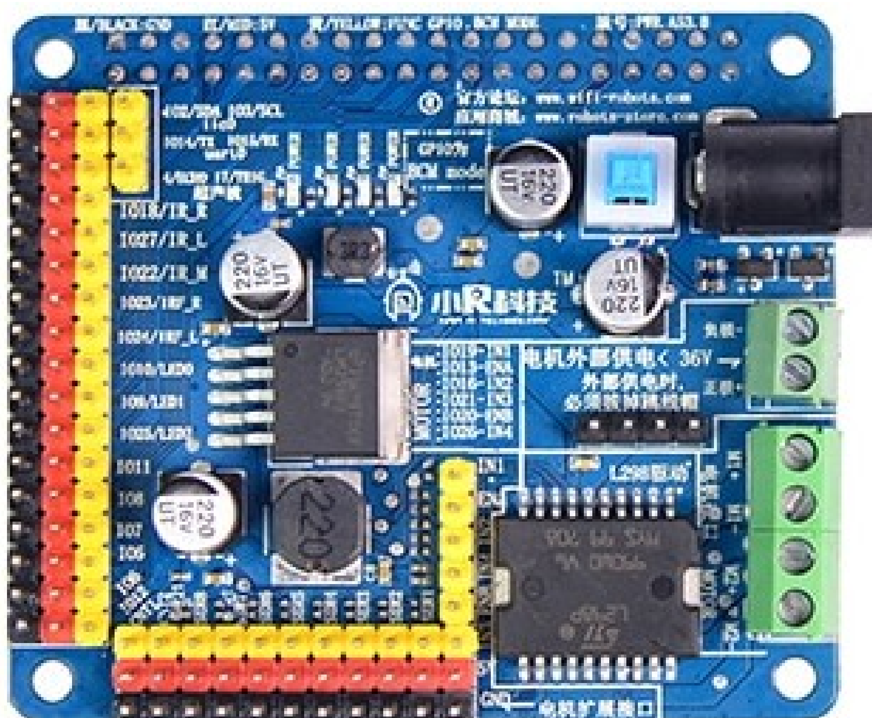

Figura 3.2: Vista superior do controlador de velocidade utilizado no projeto.

Finalmente, cabe ressaltar que o controlador selecionado também funciona como a fonte de alimentação. Dessa forma ele pode alimentar o Raspberry PI diretamente pelo GPIO sem a necessidade de se implementar um circuito próprio apenas para fornecer energia para o PI, ou de utilizar duas baterias separadas.

4
Lógica de programação

Neste capítulo discute-se os principais aspectos relativos à programação dos sistemas de controle do robô.

4.1
Organização do *dataset*

Como mencionado no capítulo 2, uma vez que se tenha a arquitetura da rede neural definida, é necessário "ensinar"ao computador quais são os objetos que ele deve procurar. A primeira etapa para o processo de treino da rede neural é organizar o *dataset* que será utilizado para encontrar os pesos adequados das conexões da rede.

Como existe uma certa liberdade para definir o que são os marcadores visuais que o robô deve encontrar, optou-se pelo uso de um pequeno grupo placas de sinalização de trânsito. O uso de tais tipos de sinalizadores se justifica por algumas características interessantes que eles possuem, a saber:

- As placas de trânsito possuem um formato muito bem definido, o que ajudará no treinamento e na verificação do sistema.
- Já existe um número elevado de *datasets* contendo diversas imagens que podem ser aproveitadas.
- As placas já transmitem uma informação relativa a navegação que pode ser utilizada para o robô decidir seus próximos passos.
- Elas podem ser facilmente impressas em escala adequada. Diferente, por exemplo, de um cone ou outro objeto físico que precisa ser comprado.

Como primeiro objeto de estudo, escolheu-se o Dataset de Placas de Transito da Bélgica para Classificações. Como o nome sugere, trata-se de um *dataset* aberto ao público contendo placas de trânsito da Bélgica e focado em pesquisas para detecção ou classificação de objetos. Este mesmo *dataset* já vem dividido em uma seção de treino e uma seção de teste. A figura 4.1 mostra alguns exemplos de placas do *dataset*.

Figura 4.1: Exemplos de placas contidas no *dataset*

Uma vez que o *dataset* foi decidido, foi selecionado apenas algumas placas de interesse para o projeto. Elas são:

- esquerda — Indica que deve-se buscar a próxima placa à esquerda do robô,
- direita — Indica que deve-se buscar a próxima placa à direita do robô,
- pare — Indica que o robô deve chegar perto da placa e então parar completamente. Ela demarca o fim do percurso.

Além das placas citadas anteriormente, o modelo treinado também detecta as placas "para frente"e "vazia". Tais placas foram incluídas para passar informações adicionais sobre a rota para o robô, mas foram consideradas desnecessárias. Apesar disso, elas ainda aparecem nas métricas relativas ao modelo.

Uma verificação manual das imagens restantes foi feita para assegurar que:

- nomes das placas estavam corretos,
- as regiões demarcadas pelos *bounding boxes* continham a placa em seu inteiro teor,
- era possível distingui-las visualmente de forma clara.

Em uma primeira análise dos dados, percebe-se que as imagens do *dataset* são, em muitos casos repetidas, de forma que haja pouca variação entre as imagens, além de a região de interesse da imagem praticamente coincide com a própria placa, sem deixar espaço para o ruído causado pela região no entorno da placa. Estes fatos já mostram uma possível má qualidade nos resultados do treino.

Com o *dataset* devidamente organizado, prossegue-se à etapa de treinos.

4.2 Treinamento do detector de objetos

Para o processo de treinamento empregou-se a API conhecida como TensorFlow Models, um subprojeto do TensorFlow para treinamento de redes neurais diversas.

O treinamento ocorreu em um computador Dell Inspiron 15R SE 4670 somente em CPU. O parâmetro de *batch size* foi definido como 24 imagens por iteração, sendo este o maior número que se pode obter para não sobrecarregar a memória do computador. Todas as demais configurações foram mantidas no padrão do modelos SSD-Mobilenet do TensorFlow Models.

O treinamento prosseguiu até o erro médio quadrático se estabilizar no seu valor mínimo, conforme os gráficos na figura 4.2:

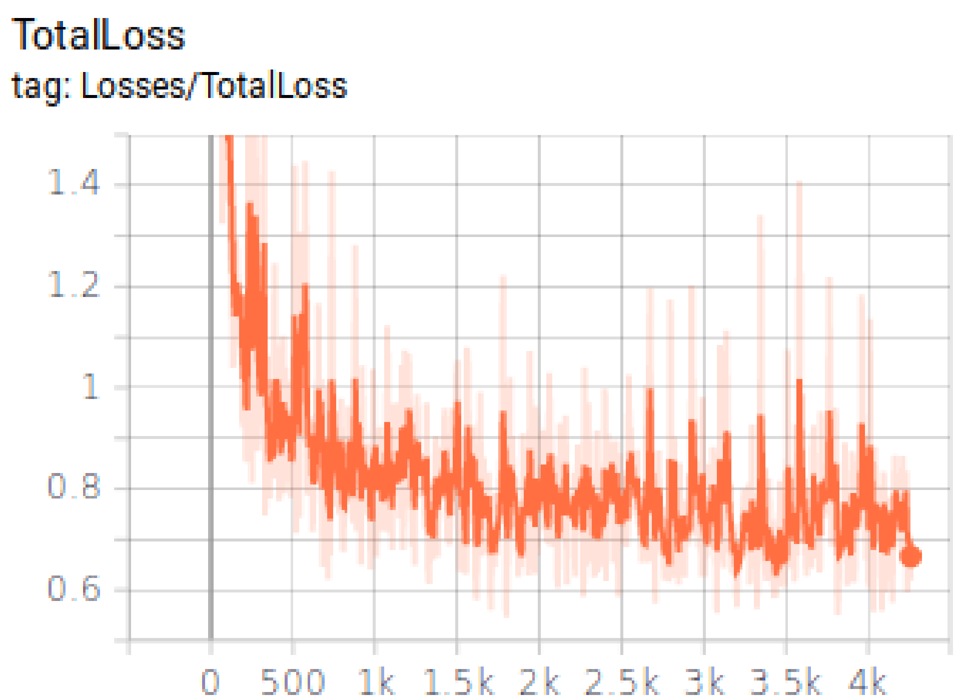

Figura 4.2: Gráfico mostrando o erro na saída do modelo

Para facilitar a visualização do comportamento decrescente do erro, podemos suavizar o gráfico, obtendo:

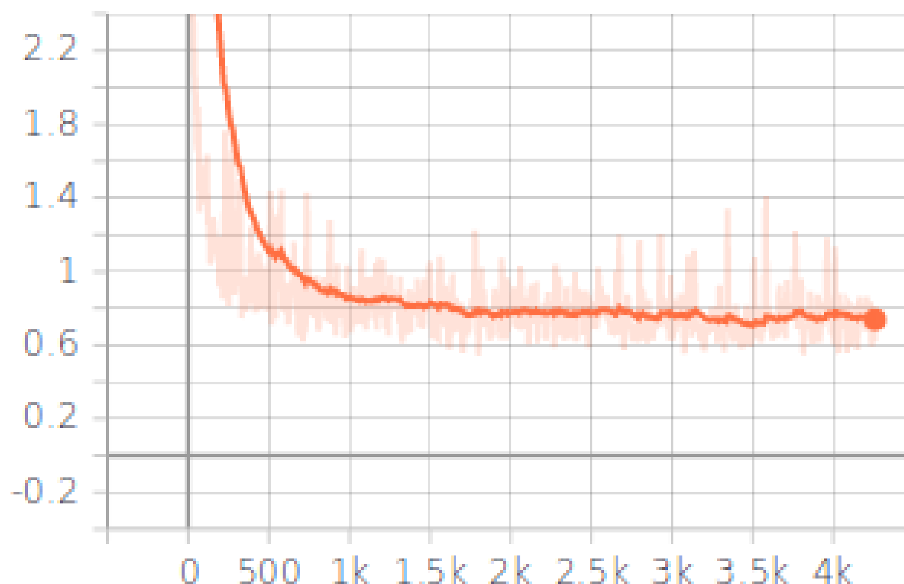

Figura 4.3: Gráfico suavizado mostrando o erro na saída do modelo

Pelo gráfico na figura 4.3, selecionou-se o modelo salvo após 4194 épocas do treinamento.

4.3 Teste do modelo

Para verificar se o modelo treinado atingiu uma acurácia desejável, o modelo treinado foi utilizado em um programa de testes para detectar as placas presentes na seção de testes do *dataset*. O programa consiste em uma rotina que apresenta ao detector as fotos para análise e compara o resultado do detector com o resultado esperado. Ao término da execução, ele apresenta o total de acertos. Além de servir como base para os testes, o programa também foi utilizado para definir, por tentativa e erro, o melhor valor de *threshold* para cada placa.

4.4 *Loop* de execução do programa

O programa de controle do robô funciona na forma de um *loop* que é executado até que a placa de pare seja encontrada. As etapas de execução são:

- Primeiro, captura-se a imagem do ambiente à frente do robô.
- A imagem é então enviada para o detector de objetos que retorna as placas detectadas, suas coordenadas na imagem e os *scores* das mesmas.
- O programa compara os *scores* com o *threshold* da classe do objeto. Se ele for inferior, o objeto é descartado.
- O robô verifica a distância até as placas. O cálculo é feito por semelhança de triângulos considerando conhecido o tamanho original da placa. Se após a filtragem houver mais de uma placa detectada, o robô considera apenas a mais próxima.
- Após a definição da placa, o robô envia as intorções para o controlador no sentido de dirigir até uma distância pré-definida da placa.
- Ao chegar na placa, o robô executa as instruções da mesma.
- Caso o robô não consiga encontrar mais nenhuma placa, ou encontre a placa de pare, ele vai parar de se movimentar.

O fluxograma simplificado do processo é dado pela figura 4.4 abaixo:

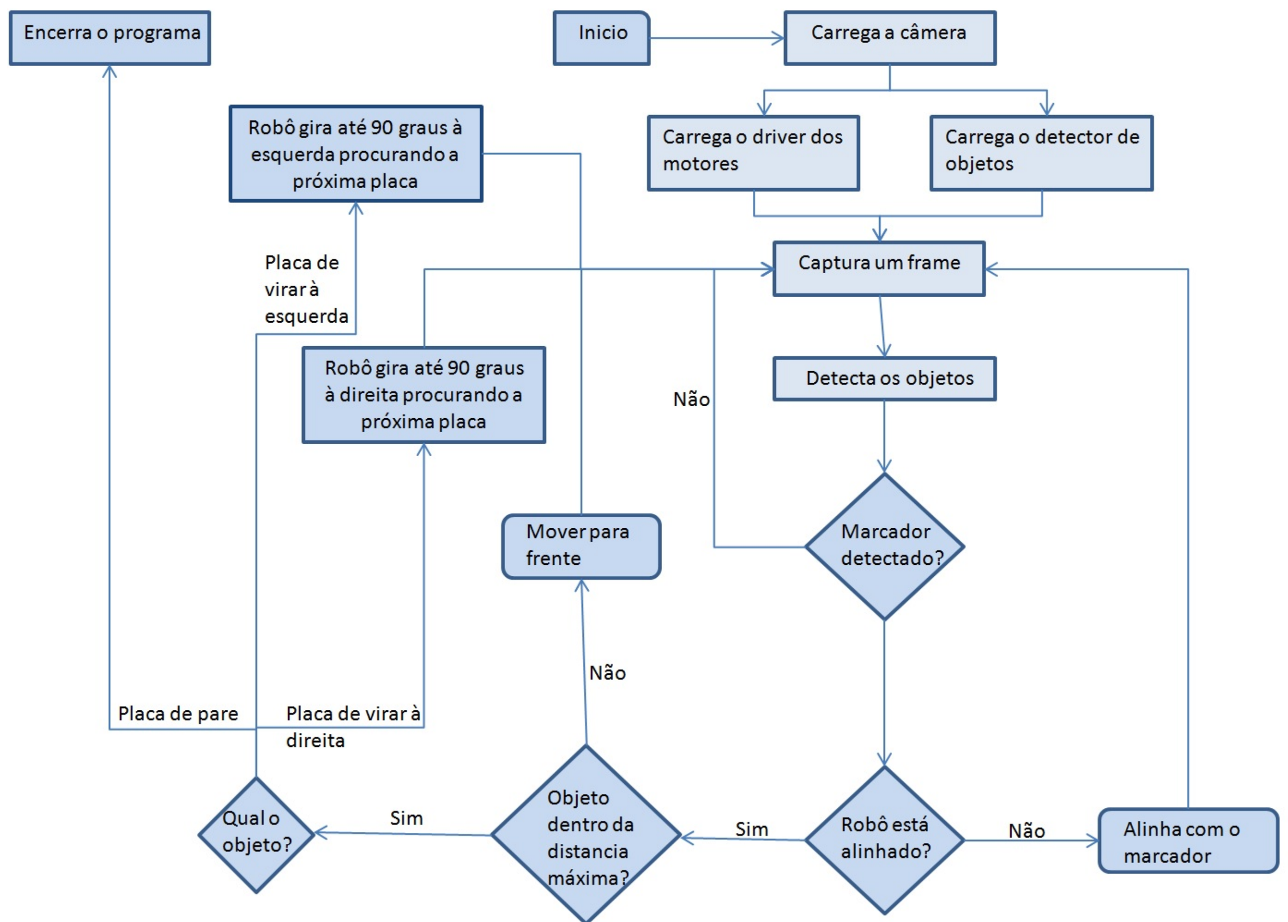

Figura 4.4: Funcionamento do código de controle do robô.

5
Resultados e Discussão

5.1
Teste do modelo de detecção de objetos

Para verificar a acurácia do modelo de detecções, foi escrito um programa que percorre um *dataset* de testes que contem diversas imagens cada uma contendo uma única placa de transito. Para cada tipo de placa executou-se um processo de detecção do objeto contido na imagem e comparou-se com o resultado esperado.

Para ter certeza de que o resultado estava maximizado, o teste foi executado múltiplas vezes utilizando diversos *thresholds* de detecção diferentes. A tabela 5.1 mostra o resultado para 4194 épocas, sendo a época de treino definida como a quantidade de vezes que o programa de treinamento escaneou todo o *dataset*.

threshold	direita (%)	esquerda (%)	frente (%)	pare (%)	vazio (%)
0,01	0	0	0	0	0
0,001	20,62	20,66	16,54	32,41	19,76
0,0001	20,83	20,67	17,03	33,33	20,24

Tabela 5.1: Acurácia do sistema para um modelo de 4194 épocas de treino.

A tabela 5.1 indica que nos melhores casos obteve-se uma taxa de acertos variando entre 20 e 30%. Para verificar se o comportamento se mantinha, selecionou-se um novo *checkpoint* de treino com 4230 épocas e executou-se a comparação com o mesmo programa.

threshold	direita (%)	esquerda (%)	frente (%)	pare (%)	vazio (%)
0,01	0	0	0	0	0
0,001	20,63	20,67	16,54	32,41	19,76
0,0001	20,83	20,67	17,04	33,33	20,24

Tabela 5.2: Acurácia do sistema para um modelo de 4230 épocas de treino.

Percebe-se pela tabela 5.2 que praticamente não há variações quando comparada à tabela 5.1. Com o resultado anterior, seleciona-se o modelo de 4230 épocas, assumindo que não vai mais ocorrer uma variação significativa no desempenho da rede com o avanço dos treinos.

5.2 Teste dos códigos de controle

Para verificar o funcionamento dos códigos de controle, foram planejados inicialmente dois testes principais apenas para o procedimento de aproximação de uma placa, sendo este apenas a sequencia de localizar a placa e se aproximar até a distância desejada. Mais dois testes principais para a navegação, onde além de se aproximar da primeira placa, o robô deve buscar as próximas placas. Diversos outros testes secundários foram executados para melhorar o entendimento dos resultados.

A figura 5.1 mostra um dos testes de aproximação.

Figura 5.1: Teste para verificar se o robô detecta a placa de pare adequadamente.

Já a figura 5.2 mostra a primeira etapa de um dos testes de navegação.

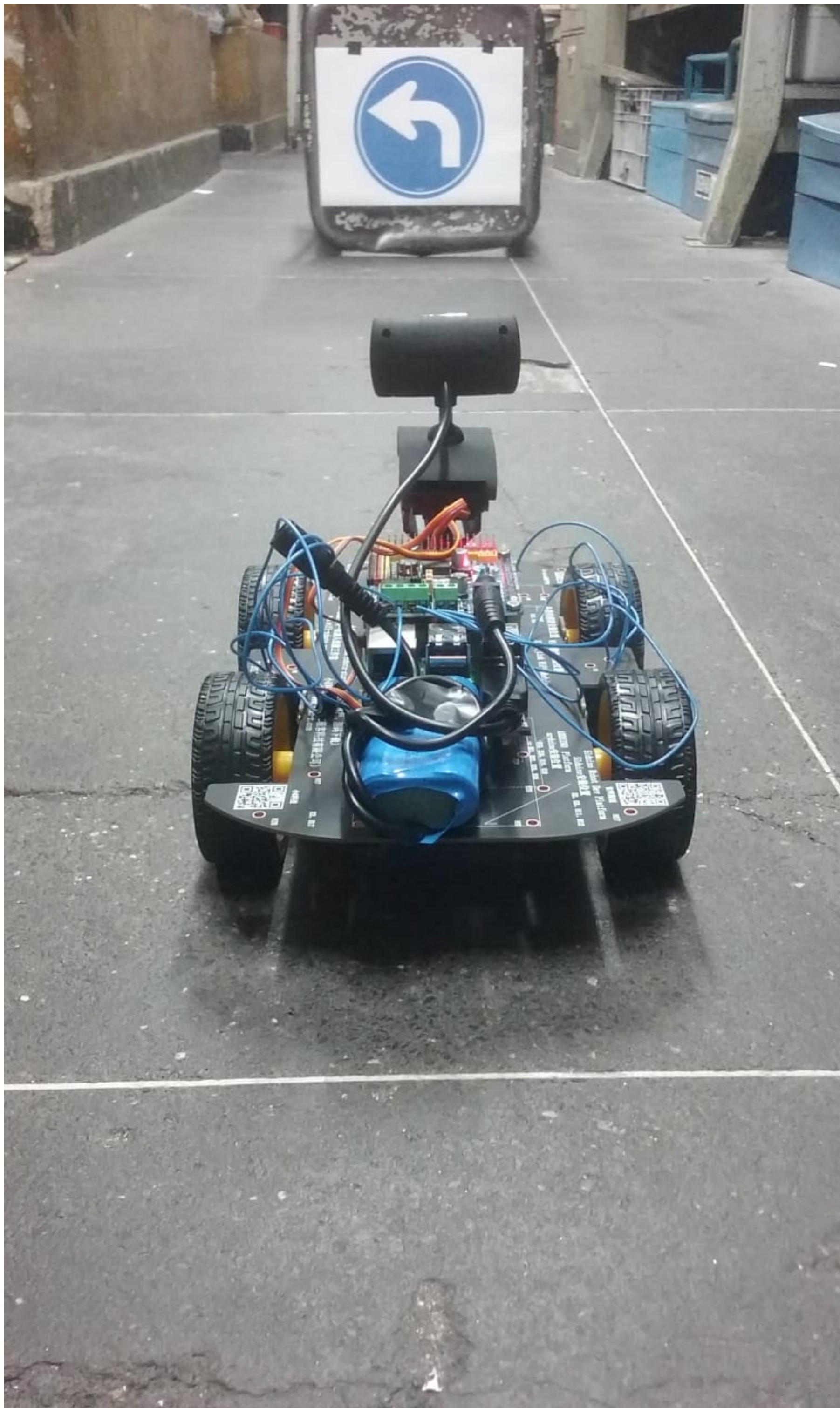

Figura 5.2: Teste para verificar se o robô chega até a primeira placa e então localiza a segunda.

Na maioria dos testes de aproximação, o robô permaneceu imóvel,

indicando a não identificação do marcador. Foram frequentes os casos em que o robô chegou a localizar o marcador, mas se perdeu na etapa de continuar navegando para os próximos pontos.

O pela análise dos arquivos de *log* do robô, e considerando os resultados obtidos no teste para o sistema de detecção, percebeu-se que a maior parte dos erros decorreu do mau ajuste do modelo de detecção. O emprego de um modelo de maior acurácia deve, portanto, aumentar o desempenho do robô.

6
Conclusões e Sugestões

No presente trabalho foi construído e testado um robô autônomo capaz de navegar buscando marcadores espalhados por um percurso pré-definido. O sistema de navegação do robô funciona em cima de um modelo de detecção de objetos que informa para o robô onde fica o próximo marco no trajeto. Baseado na informação contida no marcador encontrado, o robô procura o próximo marco e assim segue até o fim da rota.

Baseado nos resultado obtidos nos testes, percebe-se que o sistema de navegação acabou fortemente prejudicado pela ineficiência apresentada pelo sistema de detecção. Uma sugestão para trabalhos futuros é a de treinar um modelo mai acurado para a detecção dos marcadores. Para isso, pode-se tentar utilizar um *dataset* maior e com imagens mais diferenciadas entre si, bem como elevar o tempo de treinamento ou mesmo trocar a arquitetura do detector empregado.

Por fim, uma vez que o gargalo do detector seja suprido, pode-se investir em novas instruções a serem interpretadas pelo robô, ou novas formas de sensoreamento para adicionar uma confiabilidade maior ao sistema.

Referências Bibliográficas

[1] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[2] Tariq Rashid. *Make Your Own Neural Network*. Amazon Digital Services LLC, 2016.

[3] Adrian Rosebrock. *Deep Learning for Computer Vision with Python*. Pyimagesearch, 2017.