

Universidade Federal de São Carlos  
Centro de Ciências Exatas e de Tecnologias  
Bacharelado em Engenharia de Computação

**Robô seguidor de linha utilizando visão  
computacional e aprendizado por redes neurais  
embarcados em Raspberry Pi**

Igor Felipe Gallon

São Carlos - SP

2017

Igor Felipe Gallon

**Robô seguidor de linha utilizando visão computacional e  
aprendizado por redes neurais embarcados em Raspberry  
Pi**

Monografia apresentada à Universidade Federal de São Carlos como parte dos requisitos exigidos para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Emerson Carlos Pedrino

São Carlos - SP

2017

# Agradecimentos

Primeiramente a Deus por ter me dado força e saúde para superar as dificuldades.

A minha família por todo apoio e amor durante a graduação, me incentivando e ajudando nessa etapa da vida.

A minha namorada Leticia por em todos os momentos ter sido minha companheira, pelo amor, carinho e paciência o tempo todo.

Ao meu orientador Emerson, pelo suporte e incentivo durante a realização deste trabalho.

E a todos aqueles que direta ou indiretamente fizeram parte da minha formação, muito obrigado.

# Resumo

Este trabalho apresenta a discussão e os detalhes da implementação de um robô seguidor de linha capaz de percorrer trajetórias utilizando visão computacional e Redes Neurais Artificiais, embarcadas em Raspberry Pi. Atualmente, a utilização de sensores ópticos para realizarem a tarefa de leitura e interpretação da faixa que serve como guia para o robô possui suas limitações. Essas podem ser contornadas com o emprego de uma câmera e o tratamento de imagens adequado. O tratamento de imagens foi realizado com a biblioteca OpenCV, e a Rede Neural construída contou com a ajuda da biblioteca Keras, ambas em Python. Será apresentado ao decorrer do presente documento os principais conceitos e técnicas utilizadas para a concepção do robô, bem como os resultados obtidos. Um estudo de caso é apresentado, exemplificando a utilização de AGVs em aplicações industriais e discutindo suas limitações.

**Palavras-chaves:** Raspberry Pi. Redes Neurais Artificiais. Visão Computacional. Picamera. Veículo Guiado Automaticamente. OpenCV. Arduino.

## **Resumo**

This document presents the discussion and details about a line-follower robot implementation embedded in Raspberry Pi that uses Computer Vision and Artificial Neural Networks to guide itself by a defined path. Nowadays, the application of optical sensors to read and interpret the path has some limitations. The uses of a camera and the image processing can get around these limitations. In this project, the image processing was design with the OpenCV library and the Artificial Neural Network was design using the Keras library, both in Python language. It will be presented the main concepts and techniques used to construct the robot, as well as the results obtained. A case study that exemplifies the AGVs' uses in industrial applications will be discussed in relation to their limitations.

**Keywords:** Raspberry Pi. Artificial Neural Networks. Computer Vision. Picamera. Automated Guided Vehicle. OpenCV. Arduino.t

# **Lista de ilustrações**

Figura 1 – Representação de um neurônio . . . . .	12
Figura 2 – Funções de ativação . . . . .	13
Figura 3 – Rede Neural Multilayer Perceptron . . . . .	13
Figura 4 – Raspberry Pi 3 Model B . . . . .	14
Figura 5 – Raspberry Pi 3 Model B com Picamera v1.3 conectada . . . . .	15
Figura 6 – Arduino UNO Rev.3 . . . . .	17
Figura 7 – Mapeamento de pinos do Atmega328 . . . . .	18
Figura 8 – Chassis para robô com duas rodas motorizadas . . . . .	19
Figura 9 – Funcionamento da Ponte-H . . . . .	19
Figura 10 – AGVs utilizados na Embraer . . . . .	22
Figura 11 – Painel de controle dos AGVs . . . . .	23
Figura 12 – Carrinho de ferramentas carregado pelos AGVs . . . . .	23
Figura 13 – Trecho desgastado por onde o AGV percorre . . . . .	24
Figura 14 – Diagrama de Estados - Modos de Operação . . . . .	26
Figura 15 – Arquitetura Geral da Aplicação . . . . .	28
Figura 16 – Receptor de IR TSOP 4838 . . . . .	28
Figura 17 – L293D . . . . .	29
Figura 18 – Mapeamento de pinos do L293D . . . . .	30
Figura 19 – Diagrama esquemático da PCB para L293D . . . . .	31
Figura 20 – Layout da PCB para L293D . . . . .	32
Figura 21 – Processo de transferência e corrosão da PCB . . . . .	32
Figura 22 – Mensagem enviada via socket . . . . .	34
Figura 23 – Filtros aplicados no frame . . . . .	35
Figura 24 – Exemplo de frame após aplicação dos filtros . . . . .	35
Figura 25 – Fluxograma do algoritmo de envio de dados de treinamento . . . . .	36
Figura 26 – Fluxograma do algoritmo de recebimento de dados de treinamento . . . . .	37
Figura 27 – Modelo gerado com Keras . . . . .	37
Figura 28 – Fluxograma do algoritmo de treinamento da Rede Neural . . . . .	39
Figura 29 – Fluxograma do algoritmo de predição de novos dados . . . . .	41

# **Lista de tabelas**

Tabela 1 – Sinais decodificados em hexadecimal . . . . .	29
Tabela 2 – Comandos de Controle do L293D . . . . .	30
Tabela 3 – Conversão de comandos em classificação . . . . .	34
Tabela 4 – Relação entre classificação e valores dos neurônios de saída . . . . .	38
Tabela 5 – Custo estimado de cada componente do projeto . . . . .	44

# **Lista de abreviaturas e siglas**

AGV	Veículo Guiado Automaticamente, do inglês: <i>Automated Guided Vehicle</i>
CC	Corrente Contínua
DIP	Encapsulamento em dupla-linha, do inglês: <i>Dual In-Line Package</i>
FPS	Frames por segundo
kHz	Kilo Herts
GPIO	Entrada/Saída de Propósito Geral, do inglês: <i>General Purpose Input/Output</i>
IR	Radiação Infravermelha, do inglês: <i>Infrared Radiation</i>
MP	Megapixels
PCB	Placa de Circuito Impresso, do inglês: <i>Printed Circuit Board</i>
PWM	Modulação por largura de pulso, do inglês: <i>Pulse-Width Modulation</i>
RFID	Identificação por Radiofrequência, do inglês: <i>Radio-Frequency Identification</i>
SO	Sistema Operacional
ANN	Rede Neural Artificial, do inglês: <i>Artificial Neural Network</i>
UART	Transmissor/Receptor Universal Assíncrono, do inglês: <i>Universal Asynchronous Receiver-Transmitter</i>

# Sumário

<b>1</b>	<b>Introdução . . . . .</b>	<b>10</b>
<b>2</b>	<b>Conceitos básicos . . . . .</b>	<b>12</b>
2.1	Redes Neurais Artificiais . . . . .	12
2.2	Raspberry Pi . . . . .	14
2.3	Picamera . . . . .	15
2.4	Biblioteca Keras + TensowFlow . . . . .	16
2.5	Biblioteca OpenCV . . . . .	16
2.6	Arduino . . . . .	17
2.7	Chassis . . . . .	18
2.8	Ponte H . . . . .	18
<b>3</b>	<b>Revisão Bibliográfica . . . . .</b>	<b>20</b>
<b>4</b>	<b>Estudo de caso - AGVs Embraer . . . . .</b>	<b>22</b>
<b>5</b>	<b>Proposta . . . . .</b>	<b>25</b>
<b>6</b>	<b>Implementação da Proposta . . . . .</b>	<b>26</b>
6.1	Arquitetura Geral . . . . .	26
6.2	Receptor IR . . . . .	27
6.3	Controle dos motores - CI L293D . . . . .	29
6.4	Composição do conjunto de dados de treinamento . . . . .	33
6.4.1	Instalação Picamera e OpenCV . . . . .	33
6.4.2	Aquisição e tratamento das imagens . . . . .	33
6.5	Criação da Rede Neural com Keras . . . . .	35
6.6	Treinamento e teste . . . . .	37
6.7	Predição . . . . .	40
<b>7</b>	<b>Verificação e Validação . . . . .</b>	<b>42</b>
<b>8</b>	<b>Análise dos Resultados . . . . .</b>	<b>43</b>
<b>9</b>	<b>Conclusão e Melhorias Futuras . . . . .</b>	<b>45</b>
	<b>Referências . . . . .</b>	<b>46</b>

<b>Apêndices</b>	<b>49</b>
<b>APÊNDICE A Código Arduino . . . . .</b>	<b>50</b>
<b>APÊNDICE B Códigos Raspberry Pi . . . . .</b>	<b>55</b>
B.1 communicationRaspberry.py . . . . .	55
B.2 sendDataTraining.py . . . . .	56
B.3 receiveModel.py . . . . .	59
B.4 predictingFrames.py . . . . .	61
B.5 Classes adicionais . . . . .	65
<b>APÊNDICE C Códigos Servidor . . . . .</b>	<b>66</b>
C.1 communicationServer.py . . . . .	66
C.2 receiveDataTraining.py . . . . .	66
C.3 trainingNeuralNetwork.py . . . . .	69
C.4 sendModel.py . . . . .	71
C.5 tunning.py . . . . .	73

# 1 Introdução

Os robôs são dispositivos pré-programados para realizar atividades de maneira autônoma, compostos por partes eletrônicas e mecânicas. Desde que configurados de maneira correta, eles podem realizar suas tarefas com precisão e agilidade, sendo assim muito úteis para o dia-a-dia do homem. Devido a essas características, os robôs são muito utilizados em processos industriais, linhas de produção, onde geralmente exige-se padrão e repetitividade das ações. Nas linhas de produção de automóveis, por exemplo, utilizam-se braços robóticos para a soldagem das estruturas metálicas, chassis, pintura ([ABREU, 2011](#)). Também existem os robôs responsáveis por manipular essas estruturas, levar de um ponto ao outro no chão de fábrica, realizar movimentos preciso e ágeis poupano tempo se comparado com a mesma tarefa desempenhada por força humana.

Dentro da linha de produção em uma fábrica, todos os processos sãometiculosa-mente definidos, controlados e supervisionados a fim de se reduzir qualquer erro causando desperdício de tempo e material. Devido a isso, robôs atuam de maneira limitada, desem-penhando funções específicas e muito bem conhecidas. Robôs que transportam peças, por exemplo, possuem a trajetória bem definida no chão de fábrica. Todos os seus movimentos devem ser conhecidos e realizados de maneira precisa.

Da mesma forma que humanos recebem e geram estímulos, como a audição, o tato, paladar, olfato, visão, fala, movimentação dos membros, os robôs necessitam de estímulos externos para se comunicarem com o ambiente ao redor. Existem sensores que fazem o pa-pel de receptores de estímulos, como sensores de presença, de luz, infravermelho, umidade, ultrassônico, câmeras e também existem atuadores que são responsáveis por transmitir ao mundo externo ações, como motores, pistões pneumáticos, solenoides ([WARNECKE et al., 2007](#)). Para conseguirem “ver”, os robôs utilizam sensores de luz, infravermelho, por exemplo, e os mais sofisticados sistemas com câmeras. Mas além da câmera, é necessário o processamento dos sinais recebidos e saber como interpretá-los. No ser humano, esse papel é feito pelo cérebro, que recebe os estímulos elétricos dos olhos e forma as imagens que enxergamos, nos permitindo identificar rostos, lugares, objetos.

O presente documento tem como objetivo relatar o desenvolvimento de um robô capaz de seguir uma trajetória de maneira automática utilizando uma câmera e uma Rede Neural Artificial (do inglês: *Artificial Neural Network - ANN*) com processamento de imagens. A trajetória é predeterminada fisicamente por meio de faixas demarcadas, simulando um ambiente de chão de fábrica. O dispositivo é equipado com uma câmera apontada para o chão, com o intuito de capturar em tempo real a marcação da trajetória na posição atual e utilizar essa informação para alimentar a rede neural, que cumpre o pa-

pel de guiar o veículo ao longo do percurso por meio de motores acoplados às rodas. Para poder identificar a trajetória em tempo real, é necessário realizar o processamento quadro por quadro e utilizá-los como entrada para a ANN. Essa tarefa exige muitos recursos computacionais, tanto para a captura e tratamento dos *frames* quanto para o processamento da ANN, não sendo adequado, por exemplo, a utilização de um microcontrolador simples. Por conta disso, foi utilizado um microcomputador Raspberry Pi para capturar as imagens com uma câmera Picamera. Para a elaboração da ANN, foi utilizada a biblioteca Keras para linguagem Python, além da biblioteca OpenCV para o tratamento de imagens. As operações de recebimento de sinais do controle remoto e o controle dos motores, por se tratarem de operações mais simples, são realizadas através de um microcontrolador Arduino. A comunicação entre o Arduino e a Raspberry é feita através de um cabo USB utilizando o protocolo serial UART. O robô conta com dois modos de operação: modo treinamento, onde é possível controlar o robô manualmente com um controle remoto com sensor infravermelho e assim obter os dados de treinamento para a ANN; e o modo previsão, onde depois da ANN treinada, o robô percorre o percurso de maneira automática, captando as imagens e tomando as decisões de controle dos motores.

Nos próximos capítulos serão apresentados, de maneira detalhada, todos os passos da construção do robô, a obtenção das imagens da câmera, o tratamento dos *frames*, a concepção da ANN, como os dados foram treinados e utilizados para classificação e como ocorre controle dos motores. Além disso, também serão apresentados os resultados obtidos com a utilização da ANN, como se chegou na configuração atual da mesma e os parâmetros utilizados para o treinamento. Também será apresentado um exemplo real de aplicação na indústria, discutindo as diferenças entre o método utilizado no exemplo e o apresentado nesse documento.

## 2 Conceitos básicos

### 2.1 Redes Neurais Artificiais

De maneira resumida, "Rede Neural Artificial é um modelo computacional inspirado em como as redes neurais biológicas contidas no cérebro humano processam informação" (KARN, 2016). ANNs são consideradas um sub-conjunto da área de Aprendizagem Profunda (*Deep Learning*), que por sua vez, é um sub-campo da Aprendizagem de Máquina (*Machine Learning*), sendo muito utilizadas para reconhecimento/detecção de padrões, inteligência artificial para jogos, mineração de dados, classificação, etc.

Assim como no cérebro, que existem bilhões de neurônios interconectados trafegando pulsos nervoso por meio das sinapses, o componente básico das redes neurais também são neurônios. Na Figura 1 pode-se observar a representação de um único neurônio, onde cada entrada  $X_i$  tem um peso  $w_i$  associado. A saída  $Y$  do neurônio é definida pela função  $f$ , denominada **função de ativação**, que recebe como entrada a soma dos produtos entre as entradas do neurônio e seus respectivos pesos.

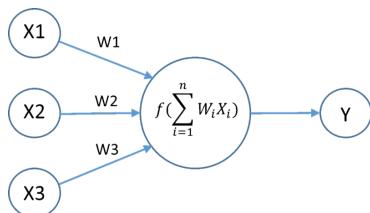
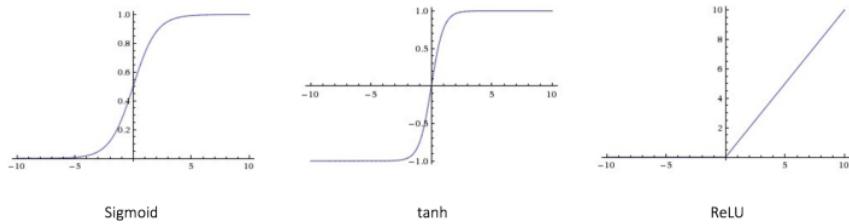
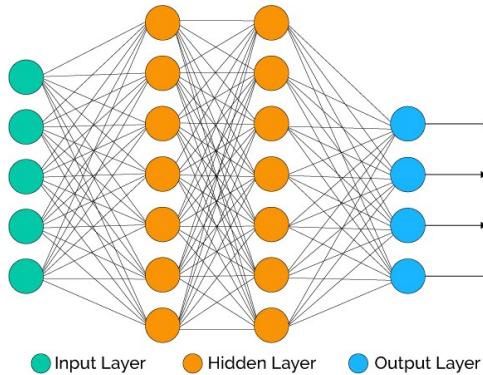


Figura 1: Representação de um neurônio.<sup>1</sup>

A função de ativação tem por objetivo adicionar à ANN a característica de não-linearidade, ou seja, não apenas ativar ou desativar (binário) a saída do neurônio. Existem algumas funções muito utilizadas, como a `sigmoid`, `tanh` e `relu`, e suas curvas podem ser observadas na Figura 2.

Existem diversas arquiteturas de ANN, sendo que uma das mais conhecidas é a Multilayer Perceptron. Nela, os neurônios são arranjados por camadas e é composta por: uma camada de entrada dos dados (*input layer*) responsável por capturar os dados que serão aprendidos do mundo externo; uma camada de saída (*output layer*), contendo as informações de como as tarefas são aprendidas; e entre essas camadas, uma ou mais camadas denominadas camadas ocultas (*hidden layers*), que transformam a entrada em algo utilizável pela camada de saída (Fig. 3) (KRIESEL, 2007; XenonStack, 2016).

<sup>1</sup> <https://medium.com/chingu/neuron-explained-using-simple-algebra-example\-\-b18f5e280845>. Acesso em: 10/11/2017.

Figura 2: Curvas das funções de ativação mais utilizadas.<sup>2</sup>Figura 3: Rede Neural Multilayer Perceptron.<sup>3</sup>

O conjunto de dados utilizado para o aprendizado da ANN é composto, geralmente, pelo conjunto de treinamento, utilizado para ajustar os pesos dos neurônios; e o conjunto de teste, utilizado para mensurar o quão precisa é a rede neural depois de treinada. O aprendizado da ANN se dá por três maneiras: aprendizado supervisionado, não-supervisionado e semi-supervisionado. No **aprendizado supervisionado**, para cada dado inserido como entrada na rede existe uma saída conhecida. Assim, os pesos das conexões entre os neurônios são ajustados para corresponder a saída desejada. A quantidade de dados de treinamento utilizados na rede é chamado de **batch size** e o número de vezes que a ANN utiliza todo o batch para treinar é chamado de **epoch**.

Existem alguns algoritmos para realizar o treinamento da rede, chamados de **otimizadores**. Um dos mais simples e conhecidos é o do gradiente descendente, que é utilizado no aprendizado supervisionado. Seu funcionamento consiste ajustar os pesos de forma a minimizar a diferença entre a saída obtida e a saída desejada. Outro otimizador é o *back propagation*, onde o erro (diferença entre saída desejada e obtida) é propagado pelos neurônios da *output layer* até a *input layer*.

Um dos grandes desafios na utilização de Redes Neurais é o ajuste dos parâmetros de treinamento. Quantas camadas ocultas, quantos neurônios por camada, quais funções de ativação utilizar, quais otimizadores, o tamanho do **batch**, número de **epochs** são questões frequentes no momento de modelagem de uma ANN, pois interferem diretamente

<sup>2</sup> <https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/>. Acesso em: 10/11/2017.

<sup>3</sup> <https://hackernoon.com/overview-of-artificial-neural-networks-and-its-applications\2525c1addff7>. Acesso em: 10/11/2017.

no seu desempenho, isto é, na taxa de acerto de classificação de dados. Diante das inúmeras possibilidades de combinação desses parâmetros, muitas vezes torna-se inviável testar cada configuração de parâmetros e no fim escolher aquela ao qual a Rede Neural teve um desempenho superior. Sendo assim, existem algumas técnicas para escolha dos parâmetros, que serão discutidas ao longo deste artigo.

## 2.2 Raspberry Pi

Raspberry Pi é uma série de dispositivos categorizados como *single-board computers* (computadores de placa única, na tradução do inglês) desenvolvidos pela Raspberry Pi Foundation no Reino Unido e que pode ser utilizado em projetos eletrônicos, desempenhando muitas das funções que um computador convencional faz ([Raspberry Pi Foundation, 2017b](#)). Atualmente existem diferentes modelos e configurações de placas, atendendo as mais diversas demandas de projeto e aplicabilidades. Existem modelos de U\$5 (Pi Zero) até modelos mais completos de U\$35, como o utilizado nesse projeto: **Raspberry Pi 3 Model B** (Fig. 4).



Figura 4: Raspberry Pi 3 Model B.<sup>4</sup>

Esse modelo é a terceira geração lançada no mercado (2016) e conta com um processador Quad-Core ARM Cortex A-53 de 1.2Ghz (64bit) e chipset Broadcom BCM2837, 1GB de memória RAM DDR2, conectividade wireless LAN BCM43438 e bluetooth 4.1 integradas, adaptador WiFi 802.11n, 4 portas USB 2.0, saída de áudio e vídeo RCA e também saída de vídeo full-HDMI, 40 pinos de entrada/saída de propósito geral (GPIO), barramentos para Raspberry Pi camera (CSI) e Raspberry Pi touchscreen display (DSI) ([Raspberry Pi Foundation, 2017a](#)). Medindo apenas  $85 \times 56 \times 17\text{mm}$ , esse modelo de Raspberry Pi também conta com uma GPU Broadcom VideoCore IV compatível com Open GL ES2 2.0, OpenVG acelerado em hardware ([RS-Components, 2017](#)).

<sup>4</sup> <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Acesso em: 26/10/2017.

Como todo computador é necessário instalar um Sistema Operacional (SO) para poder utilizá-lo. Nesse modelo, o SO é instalado em um micro SD card, que é inserido no slot da placa. Atualmente existem diversos SOs disponíveis gratuitamente. O mais conhecido e oficialmente suportado pela Raspberry Foundation é o Raspbian, uma versão da distribuição Linux Debian. Mas também existem outros como Arch Linux ARM, RISC OS Pi, também é possível instalar o Chromium OS, da Google e o Windows 10 IoT Core, da Microsoft. O Raspbian OS pode ser encontrado facilmente para download na página oficial da Raspberry Foundation, juntamente com softwares para auxiliar a instalação do SO no SD Card.

## 2.3 Picamera

A Picamera é um módulo oficial disponibilizado pela Raspberry Foundation. A primeira versão disponibilizada em 2013 conta com uma câmera de lentes fixas de 5MP e é capaz de capturar imagens de até  $2592 \times 1944$  pixels e gravar vídeos de 1080p a 30 FPS, 720p a 60 FPS e  $640 \times 480p$  a 60/90 FPS ([PiSupply, 2013](#)).

A conexão com a Raspberry Pi é feita através de um cabo flat de 15 pinos que conecta no slot CSI da placa, localizado entre a conexão de áudio/video RCA e a saída HDMI (Fig. 5). A conexão Camera Serial Interface (CSI) é uma interface de comunicação que permite transmissão bidirecional de alta velocidade entre a câmera e o processador ([MIPI Alliance, 2017](#)). A Picamera pode ser utilizada através de linha de comando no shell do SO ou por meio da biblioteca em Python `picamera` ([JONES, 2016](#)).



Figura 5: Raspberry Pi 3 Model B com Picamera v1.3 conectada. <sup>5</sup>

<sup>5</sup> <https://rlx.sk/en/camera-module/4672-rpi-camera-c-waveshare-raspberry-pi-camera-module-v13-compatible-with-the-original.html>. Acesso em: 29/10/2017.

## 2.4 Biblioteca Keras + TensorFlow

TensorFlow é uma biblioteca *open-source* criada dentro da Google em 2015 para projetar, construir e treinar modelos de *Deep Learning*. O conceito de funcionamento baseia-se em grafos de fluxo de dados, onde os nós dos grafos representam operações matemáticas e os vértices representam *arrays* de dados multidimensionais (tensors) ([TensorFlow, 2016](#)). A biblioteca possui API para Python, C++, Java, entre outras.

Muitas vezes, por tratar-se de operações mais complexas, a utilização do TensorFlow puro (ou outras bibliotecas de computação numérica, como o Theano) para *Deep Learning* pode ser um pouco trabalhosa. Pensando nisso, a biblioteca Keras foi criada como um *wrapper* de alto nível. Desenvolvida em Python, Keras é uma API capaz de rodar sobre TensorFlow, Theano ou CNTK. Amplamente utilizada para construção de redes neurais, a biblioteca possui uso amigável, reduzindo o número de operações necessárias do usuário para realizar determinada tarefa, além de ser modular e facilmente extensível ([Keras, 2017](#)).

A estrutura principal no Keras, utilizada para organizar as camadas da rede, é o **model**. Existem alguns tipos de models disponíveis, sendo o tipo **Sequential** o mais simples, que nada mais é do que uma pilha de linear de camadas, onde é possível instanciar cada camada da ANN de maneira sequencial.

## 2.5 Biblioteca OpenCV

OpenCV (Open Source Computer Vision Library) é uma biblioteca originalmente desenvolvida na Intel nos anos 2000 e tem por objetivo tornar a visão computacional mais acessível entre usuários e desenvolvedores de diversas áreas, incluindo centenas de algoritmos de visão computacional ([MARENCONI, 2009](#); [OpenCV, 2017](#)). Possui interfaces em linguagens C, C++, Python e Java, sendo suportada pelas plataformas Windows, Linux, Mac OS, iOS e Android.

A biblioteca é organizada de maneira modular, contando com os seguintes módulos disponíveis: funcionalidades essenciais, para definição de estruturas de dados, para serem utilizados em outros módulos; processamento de imagens, agregando filtros lineares e não-lineares, transformações geométricas, conversão de espaço de cores, etc; análise de vídeo, com algoritmos de rastreamento de objetos, *background subtraction*; calibragem de câmera; detecção de objetos, como faces, olhos, pessoas, carros; interface de usuário; interface de captura e codificação de vídeo; algoritmos otimizados para GPU de diferentes módulos do OpenCV.

Por ser uma biblioteca *open-source* e estar sob licença BSD (Berkeley Software Distribution), indicando que pode ser compilada para diversas novas plataformas, há atu-

almente uma versão de compilação de OpenCV para Raspberry Pi ([GASPAR, 2014](#)). O processo de compilar na própria Raspberry pode levar algumas horas. A versão OpenCV-Python é um *wrapper* em Python da implementação original em C/C++. Essa funcionalidade permite que se tenha rapidez em processamento, semelhante ao código original em C/C++ e facilidade no uso por ser em Python ([MORDVINTSEV, 2017](#)).

## 2.6 Arduino

Arduino é uma plataforma de prototipagem eletrônica *open-source* baseada na integração de software e hardware fáceis de usar ([Arduino, 2017](#)). Projetada para ser uma única placa, conta com um microcontrolador Atmel AVR, pinos de entrada/saída e também uma linguagem de programação própria, com sintaxe semelhante às linguagens C/C++. O Arduino conta também com uma ambiente de programação próprio, a Arduino IDE, além de ser possível gravar o código no microcontrolador através da conexão USB, já que o Arduino também conta com um controlador embutido de USB para Serial.

Atualmente existem diversos modelos de Arduino, com diferentes microcontroladores, velocidades de processamento, dimensões, quantidade de memória, etc. Uma das versões mais conhecidas, e também a escolhida para a concepção do projeto, é o **Arduino UNO Rev.3** (Fig. 6), que vem equipado com um microcontrolador Atmega328 de 16MHz, memória ROM de 32kB, 14 pinos de entrada/saída digitais (sendo 6 com saída PWM) e 6 pinos de saída analógica ([Sparkfun, 2015](#)).



Figura 6: Arduino UNO Rev.3.<sup>6</sup>

O microcontrolador Atmega328 presente no Arduino UNO, possui 13 pinos de entrada/saída digitais, operando a nível lógico baixo (0V) e alto (5V). Os pinos 0 e 1, são utilizados para comunicação serial  $R_x$  e  $T_x$ , respectivamente. Os pinos 3, 5, 6, 9, 10 e 11 podem ser utilizados para saída PWM. Além disso, existem 6 pinos de entrada analógica,

<sup>6</sup> <https://learn.sparkfun.com/tutorials/arduino-comparison-guide>. Acesso em: 30/10/2017.

que são ligados a conversores A/D com resolução de 10 bits, convertendo níveis de tensão para valores de 0 a 1023 (Fig. 7).

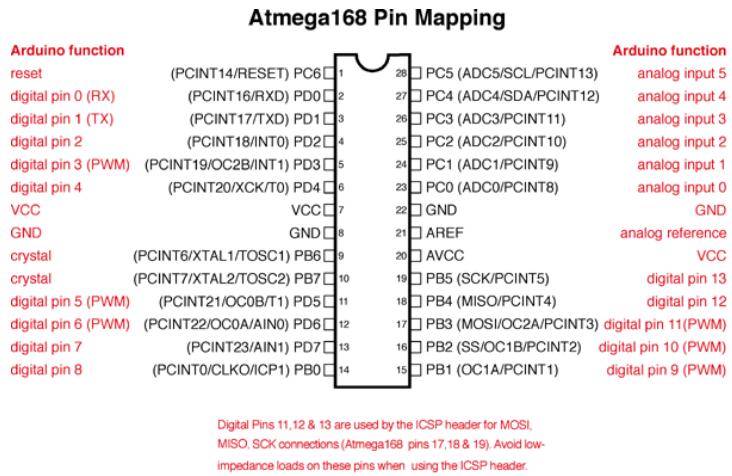


Figura 7: Mapeamento de pinos do microcontrolador Atmega328 contido no Arduino UNO.<sup>7</sup>

## 2.7 Chassis

O chassis consiste em toda a estrutura utilizada para abrigar o sistema lógico, como a Raspberry Pi, a Picamera, o Arduino UNO e também as fontes de energia que alimentam o sistema. O chassis, para esse projeto, é composto por: uma base de acrílico com furações para fixação de componentes; dois motores de corrente contínua (CC) que operam na faixa de 3–6V; acoplado a cada motor, existe uma caixa de redução (1:48) de velocidade e aumento de torque; duas rodas fixadas aos eixos das caixas de redução; uma roda articulada na parte traseira para dar sustentação e permitir a movimentação livre do robô (Fig. 8).

## 2.8 Ponte H

Ponte-H é um circuito eletrônico que permite realizar o controle da polaridade (isto é, a direção) de uma corrente que flui através de uma carga. Devido a sua natureza de construção, o seu funcionamento pode ser realizado através do chaveamento de componentes eletrônicos, utilizando-se do método de PWM. Dessa forma, esse tipo de circuito é muito utilizado para controlar a direção de rotação de motores CC e também o módulo de tensão aplicada, que consequentemente controla a velocidade de rotação.

A Figura 9(a) mostra o diagrama genérico desse circuito controlando uma carga  $M$  por meio de uma fonte de tensão  $V_{in}$ , que é composto por quatro chaves ( $S_1$  a  $S_4$ ). Acionando as chaves  $S_1$  e  $S_4$ , a corrente flui em um sentido (Fig. 9(b)) e acionando as

<sup>7</sup> <https://www.arduino.cc/en/Hacking/PinMapping168>. Acesso em: 31/10/2017.

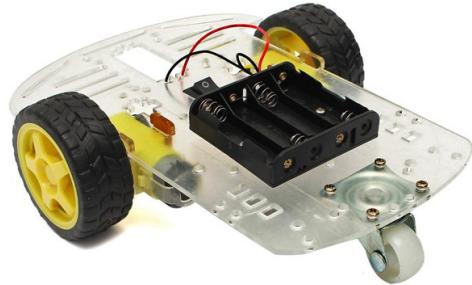
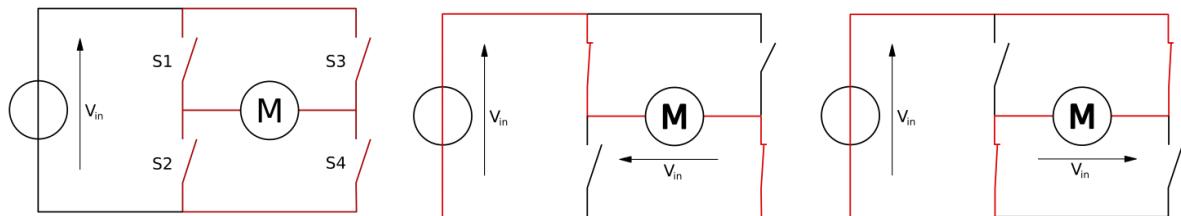


Figura 8: Chassis para robô com duas rodas motorizadas.<sup>8</sup>

chaves  $S_2$  e  $S_3$  a corrente flui no sentido oposto (Fig. 9(c)). Ao operar esse circuito, deve-se tomar o cuidado em não acionar ao mesmo tempo as chaves  $S_1$  e  $S_2$  ou  $S_3$  e  $S_4$ , o que causaria um curto-círcuito na fonte  $V_{in}$ .



(a) Diagrama genérico de uma Ponte H (b) Corrente fluindo no Sentido 1 (c) Corrente fluindo no Sentido 2

Figura 9: Funcionamento da Ponte-H.<sup>9</sup>

<sup>8</sup> <https://www.amazon.in/Makerfire-Arduino-Motor-Robot-Chassis/dp/B01NCL6ECZ>. Acesso em: 03/11/2017.

<sup>9</sup> [https://pt.wikipedia.org/wiki/Ponte\\_H](https://pt.wikipedia.org/wiki/Ponte_H). Acesso em: 05/11/2017.

### 3 Revisão Bibliográfica

Segundo ISO (2012), "robô é um mecanismo atuador programável em dois ou mais eixos com um grau de autonomia, movendo-se dentro de seu ambiente, para realizar tarefas determinadas". Entende-se como parte do robô seu sistema de controle e a respectiva interface. Além disso, os robôs são classificados como industriais ou de serviço. Os robôs industriais são controlados automaticamente, reprogramáveis, com manipulador universal e programados em três ou mais eixos. Já os robôs de serviço são aqueles que executam tarefas úteis para humanos ou equipamentos, exceto aplicações de automação industrial, como por exemplo, inspeção, montagem, empacotamento, manufatura.

Segundo a definição acima, os Veículos Guiado Automaticamente (AGVs) podem ser considerados como uma categoria de robôs industriais. São definidos como veículos móveis programáveis, usados em aplicação industrial para transportar material dentro do chão de fábrica (NETTO, 2010; SOUZA; ROYER, 2013). São programados para seguirem caminhos definidos (filoguiados) no ambiente de operação, como marcações no chão (fitas refletivas, por exemplo), trilhos magnéticos, rádio frequência, lasers, etc. (DZIWIS, 2005; DAS; PASAN, 2016).

Os robôs seguidores de linha existentes no mercado costumam utilizar de um mecanismo de detecção de linha bem simples: existe um diodo emissor de luz (seja infravermelha ou não) apontado para baixo, bem próximo ao chão, que emite luz constante. Próximo a esse, existem sensores que captam a luz refletida. O controle da movimentação do robô é então realizado com base na luz captada quando a luz emitida é refletida nas fitas que demarcam a trajetória, como pode ser visto no trabalho realizado por Pscheidt (2007), que implementa um AGV utilizando microcontrolador PIC e par de sensores ópticos emissor-detector. A trajetória percorrida pelo robô é feita através de uma linha de cor preta colocada acima de uma plataforma de cor branca. O contraste de cores é então utilizado para realizar a leitura da posição atual do AGV e assim controlar os motores de passo acoplados às rodas.

Seguindo a mesma ideia, Miranda et al. (2017) implementou um sistema AGV parecido, porém a unidade de controle utilizada é um microcontrolador ARM, juntamente com sensor de distância para evitar colisões. Nesse trabalho, o sensor de trajetória utilizado é uma barra de sensor óptico de 5 canais e a trajetória é composta por uma faixa branca entre duas faixas pretas, também sendo controlado pela percepção de contraste de cores.

Em contrapartida, Filho (2010) elaborou em um sistema de visão computacional para controlar um robô móvel, ou seja, através da utilização de um computador que mapeia com uma câmera o trajeto a ser feito pelo robô e então o controla remotamente.

Nessa abordagem, o robô segue apenas comandos enviados via rádio frequência e a tarefa de processamento de imagens fica por conta de um computador externo, que utiliza filtros de separação de cores para identificar os obstáculos e determinar a trajetória.

[Araujo et al. \(2006\)](#) também empregou a técnica de visão computacional para controlar um AGV. Aqui, foi utilizado uma câmera e o Kit Mindstorms, da Lego, para a concepção do projeto. A visão computacional é dada pela câmera posicionada acima do cenário, capturando imagens panorâmicas para a detecção de obstáculos. Os objetos são detectados utilizando filtros de imagens e transformando-os em coordenadas cartesianas. Com isso, calcula-se a trajetória que o robô deverá fazer, desviando-se dos objetos até chegar ao seu alvo no cenário.

No trabalho desenvolvido por [Wankhade e Shriwas \(2012\)](#), a técnica empregada foi um pouco diferente das demais apresentadas. O sistema também utiliza visão computacional através de uma câmera, porém a imagem capturada é a visão do trajeto que o AGV tem no momento e não uma vista panorâmica de todo o percurso. O processamento é feito por um laptop, que recebe as imagens da câmera e utiliza uma série de filtros para detectar cor, borda e o centro da "pista" onde o robô está se direcionando. O controle dos motores é realizado por meio de um microcontrolador ATmega8 e a comunicação entre este e o laptop é obtida por comunicação serial (UART).

Ainda na linha de robôs móveis, [Silva \(2016\)](#) desenvolveu um robô móvel que utiliza o sistema de sensores dispostos no ambiente para se localizar e conseguir navegar de forma autônoma, por meio de uma plataforma Android acoplada e uma câmera como sensor. Nesse caso, a trajetória é calculada com base na localização atual do robô e não por meio de uma linha marcada no chão.

## 4 Estudo de caso - AGVs Embraer

No contexto de Veículos Guiados Automaticamente (AGV), essa Seção apresenta e discute a aplicação desses dispositivos na indústria. O objeto de estudo é a utilização de AGVs para realizar tarefas industriais na empresa Embraer, fabricante de aeronaves transnacional brasileira, 3<sup>a</sup> maior empresa fabricante de jatos comerciais no mundo.



Figura 10: AGVs utilizados na Embraer aguardando chamado na Ferramentaria.

Os AGVs estão presentes no setor de Usinagem, na unidade Faria Lima, em São José dos Campos, e são responsáveis por transportar ferramentas de usinagem entre a Ferramentaria e as máquinas operantes. São 4 AGVs operando no transporte de carrinhos com suportes para ferramentas de usinagem. Os AGVs são comandados por um painel central presente na Ferramentaria, onde o operador pode solicitar que um carro vá até determinado ponto buscar ou levar ferramentas. Os carros são guiados por uma faixa branca e preta pintada no chão da fábrica, delimitando todo o trajeto possível. Para captar o trajeto, é utilizado uma barra de 5 sensores ópticos, que captam luz refletida sobre as faixas.

A Ferramentaria funciona como uma *warehouse* para os AGVs, pois quando a missão é cumprida, o robô volta automaticamente para o ponto inicial, aguardando novo chamado. Todo o percurso possui tags RFID para o robô poder se localizar, ou saber que aquele é um ponto de coleta ou entrega de ferramentas. As tags também são utilizadas para informar ao AGV quando se deve parar, devido à algum cruzamento, e obter a localização do mesmo para indicar no painel de controle ao operador. Os carros também contam, por medidas de segurança, com sensores de presença para evitar colisões com pessoas ou máquinas. Além disso, durante seu funcionamento, luzes piscantes e avisos sonoros equipados no carro são acionados para alertar a passagem do mesmo pela fábrica.

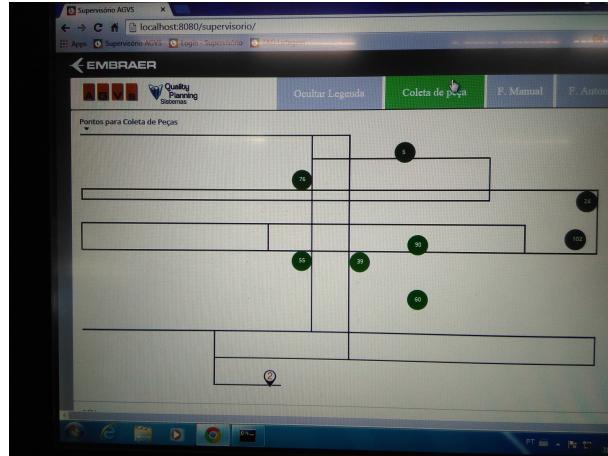


Figura 11: Painel de controle dos AGVs indicando os pontos de coleta durante o trajeto no chão de fábrica.



Figura 12: Carrinho de ferramentas carregado pelos AGVs. O AGV possui dois pinos deslizantes que se encaixam nos suportes do carrinho, que é puxado ao longo do trajeto.

Por ser um chão de fábrica, onde há poeira, em alguns pontos luminosidade baixa, o trajeto lido pelo robô muitas vezes pode ser comprometido. Com o tempo, devido a circulação de pessoas e outros equipamentos, as faixas pintadas no chão perdem sua intensidade, dificultando os sensores a realizarem uma leitura correta. Além disso, existem também pisos com desnível que interferem drasticamente na leitura dos sensores. Para minimizar os problemas de leitura dos AGVs, os caminhos são regularmente limpados, o que gera um custo e ocupação de tempo adicional. Mas, muitas vezes não é suficiente

para garantir o funcionamento dos AGVs, causando um erro de leitura e fazendo com que o carro saia do percurso e se perca, necessitando uma correção manual do operador.



Figura 13: Trecho de trajeto percorrido pelo AGV onde é possível observar desgaste da tinta e desnível do piso.

## 5 Proposta

A proposta para o projeto em questão é desenvolver um dispositivo capaz de seguir trajetórias predeterminadas de maneira automática e ágil através da utilização da visão computacional com processamento de imagens em tempo real e aprendizagem de máquina por redes neurais artificiais, caracterizando-se como um Veículo Guiado Automaticamente.

O robô, construído sobre um Chassi, conta com uma câmera para a captura de imagens em tempo real, que é enviada e processada por uma Raspberry Pi 3. A ANN utilizada é do tipo Multilayer Perceptron, construída com o auxílio da biblioteca Keras e TensorFlow. A criação do conjunto de dados de treinamento é feita com o controle manual do robô sobre o trajeto que está sendo aprendido, ou seja, o aprendizado é supervisionado. O usuário controla os movimentos por meio de um Controle Remoto IR. Os comandos do controle remoto são enviados juntamente com o frame capturado da câmera, correspondendo ao dado de treinamento e sua classificação conhecida. Com a ANN treinada, o robô pode entrar no estado de predição, realizando a trajetória de maneira automática e sem a intervenção do Controle Remoto. O controle dos motores CC e a recepção dos sinais IR do controle remoto são realizadas pelo Arduino, que se comunica com a Raspberry Pi 3 por meio uma comunicação serial UART.

Pelas facilidades de integração entre as bibliotecas e a plataforma Raspberry Pi, o projeto, com exceção da programação do Arduino, é desenvolvido utilizando linguagem Python. Deseja-se obter um sistema que possa navegar pelo circuito de maneira segura e confiável. Para tal, visa-se encontrar uma boa configuração dos parâmetros para a ANN de modo que se obtenha uma boa taxa de acerto dos dados de treinamento, aumentando a confiança do sistema para classificar novos dados.

Por ser uma técnica que utiliza visão computacional em vez de uma barra de sensores ópticos, espera-se que os resultados sejam melhores em relação às técnicas utilizadas nos trabalhos citados na Seção 3, pois, nesse tipo de sistema, as condições do ambiente, especialmente se for um chão de fábrica, podem interferir na leitura dos sensores e causar um erro no controle do equipamento, fazendo-o se desviar da trajetória determinada, como visto na Seção 4. Falta de luminosidade nas faixas refletivas, poeira, sujeira, desnível do piso são problemas que podem ser contornados utilizando o tratamento adequado de imagens.

# 6 Implementação da Proposta

Como mencionado na Seção 5, a proposta do projeto é construir um robô que percorre, de maneira automática, um determinado trajeto, utilizando Visão Computacional e Redes Neurais Artificiais.

O robô, constituído por uma base de fixação, dois motores de corrente contínua com rodas acopladas e uma roda de movimento livre, é equipado com: uma placa Raspberry Pi 3 Model B; uma câmera Picamera apontada para o chão e conectada à Raspberry; uma placa Arduino conectada à Raspberry por meio de um cabo USB; uma placa de circuito impresso controlada pelo Arduino, contendo uma Ponte-H (CI L293D) responsável por realizar o controle de potência dos motores; e fontes de alimentação para a Raspberry, o Arduino e os motores. Além disso, ao Arduino também é acoplado um Receptor de sinais IR, que recebe os comandos de um Controle Remoto convencional. O caminho ao qual o robô percorre consiste em uma faixa preta em cima de uma faixa branca demarcada no chão.

## 6.1 Arquitetura Geral

De maneira geral, o robô seguidor de linha possui dois modos de operação: o modo **Treinamento** e o modo **Predição**, sendo ambos selecionados através do Controle Remoto. Os sinais são interpretados pelo Arduino, executando uma rotina específica para cada comando recebido. O modo Predição só pode ser executado corretamente após o modo Treinamento ser executado, pelo menos uma vez, pois é necessário ter um modelo treinado previamente (Fig. 14).

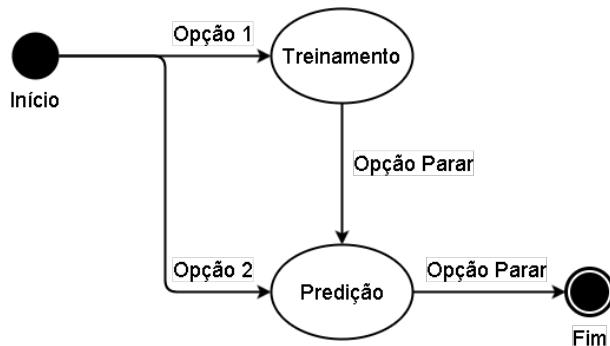


Figura 14: Diagrama de Estados - Modos de Operação da Aplicação.

No modo Treinamento, os movimentos do robô são controlados também via Controle Remoto pelo usuário. O Arduino então envia o comando pressionado pelo usuário para a Raspberry Pi ao mesmo tempo que controla os motores, pelos pinos de I/O presen-

tes no Arduino, que serão explicados posteriormente. A Raspberry Pi recebe o comando através da conexão serial UART e captura o frame da câmera Picamera. O conjunto frame + comando pressionado é então utilizado para compor os dados de treinamento da Rede Neural que são enviados para o Servidor por meio de uma comunicação Socket. O Servidor é responsável por coletar os dados de treinamento, treinar a Rede Neural e então enviar o modelo gerado juntamente com os pesos obtidos dos neurônios para a Raspberry Pi.

Uma vez que a Raspberry Pi tem o modelo da Rede Neural e os pesos dos neurônios, a Rede então é construída e está pronta para classificar novos dados. O modo Predição é executado quando se tem um treinamento prévio da Rede Neural. Nele, a Raspberry captura o frame da Picamera, realiza a classificação, que é convertido em movimentos para o robô (Forward, Backward, Left, Right). O comando a ser realizado é enviado da Raspberry Pi para o Arduino pela conexão serial e o Arduino, por sua vez, controla os motores.

O código `communicationArduino.ino` (Apêndice A), escrito para Arduino, realiza todo o controle dos motores, envio e recebimento de dados via comunicação serial e recepção de sinais IR. O código `sendDataTraining.py` (Apêndice B.2) escrito em Python para Raspberry Pi contém a classe e funções que executam o processo de recebimento dos comandos do Arduino e envio dos frames para o Servidor, enquanto os códigos `receiveModel.py` e `predictingFrames.py` (Apêndices B.3 e B.4) são responsáveis por, respectivamente, receber o modelo treinado pelo Servidor e realizar o processo de classificação dos novos dados. Além disso, o código `communicationRaspberry.py` (Apêndice B.1) é responsável por instanciar as classes citadas e controlar as funções da Raspberry Pi. Do lado do Servidor, os códigos `receiveDataTraining.py`, `trainingNeuralNetwork.py` e `sendModel.py` (Apêndices C.2, C.3 e C.4) possuem as classes e funções para realizar o recebimento dos dados de treinamento da Raspberry Pi, treinar a Rede Neural utilizando os parâmetros pré-determinados e enviar o modelo gerado juntamente com os pesos dos neurônios para a Raspberry Pi. Da mesma forma que na Raspberry Pi, o código `communicationServer.py` (Apêndice C.1) é responsável por instanciar e controlar as classes e funções mencionadas. Todos os códigos encontram-se no Apêndice (página 50) e serão detalhados ao longo dessa Seção.

Na Figura 15 é possível observar um diagrama em blocos da Arquitetura da Aplicação e a comunicação entre cada componente.

## 6.2 Receptor IR

O sensor receptor de IR utilizado no projeto foi o TSOP 4838, da fabricante Vishay, com frequência portadora de 38 kHz. É possível observar na Figura 16 que o sensor possui três pinos: 1 - OUT: saída do sinal demodulado captado pelo sensor, que pode

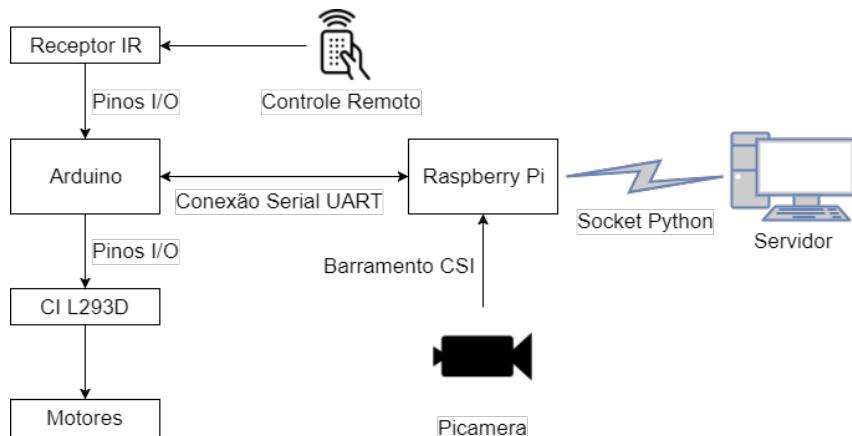
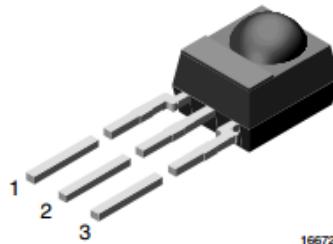


Figura 15: Diagrama em blocos da arquitetura geral da Aplicação.

ser conectado diretamente ao microcontrolador, para decodificação do sinal; 2 - GND: pino terra; e 3 -  $V_s$ : pino de alimentação do sensor, que pode operar entre 2.5V até 5.5V (Vishay Semiconductors, 2016).



#### MECHANICAL DATA

##### Pinning for TSOP44.., TSOP48..:

1 = OUT, 2 = GND, 3 =  $V_s$

##### Pinning for TSOP22.., TSOP24..:

1 = OUT, 2 =  $V_s$ , 3 = GND

Figura 16: Diagrama de pinos do Receptor de IR TSOP 4838.<sup>10</sup>

Os pinos 2 e 3 são conectados, respectivamente, aos pinos GND e 3.3V do Arduino enquanto o pino 1 é conectado ao pino digital 11. Para decodificar os sinais recebidos pelo pino 11, foi necessário a utilização de uma biblioteca adicional no código `communicationArduino.ino`, a `IRremote` (SHIRRIF, 2012).

```
#include <IRremote.h>
```

A utilização da biblioteca é simples, bastando apenas inicializá-la no pino desejado, configurá-la para receber os sinais e ler os valores decodificados. Abaixo encontra-se um trecho de código exemplificando seu uso.

```
1   int RECV_PIN = 11;           // Porta conectada ao pino 1 do TSOP 4838
2
```

<sup>10</sup> <https://www.vishay.com/docs/82459/tsop48.pdf>. Acesso em: 13/11/2017.

```

3     IRrecv irrecv(RECV_PIN);      // Instanciação da classe no pino 11
4     decode_results results;
5
6     if(irrecv.decode(&results)){
7         codeIR = (results.value); // Armazena o sinal decodificado em codeIR
8         irrecv.resume();          // Libera o pino para proxima leitura
9     }

```

Cada tecla do Controle Remoto emite uma onda de frequência diferente, que é demodulada pelo sensor TSOP 4838 e decodificada pela biblioteca `IRremote`. Após a leitura do valor decodificado, tem-se um código em hexadecimal para cada tecla. Assim, é possível saber qual tecla foi pressionada e recebida e com isso executar diferentes ações. Na Tabela 1 encontra-se a relação de códigos hexadecimais para tecla do Controle Remoto utilizado no projeto.

Tabela 1: Sinais decodificados em hexadecimal e suas respectivas teclas no Controle Remoto

Código HEX	Tecla
2FDD827	Forward
2fdf807	Backward
2fd7887	Left
2fd58a7	Right
2fdc837	Stop
2fd807f	Modo Treinamento
2fd40bf	Modo Predição

### 6.3 Controle dos motores - CI L293D

Como mencionado na Seção 2.8, o controle de potência e direção de rotação de um motor é comumente realizado através de um circuito de Ponte-H. Para o controle dos dois motores de corrente contínua do robô, seria necessário uma Ponte-H para cada motor. Sendo assim, foi utilizado o CI L293D, que nada mais é do que um circuito integrado controlador de potência de 4 canais, caracterizando-se como um circuito Ponte-H dupla (Fig. 17).



Figura 17: Exemplo de CI L293D, da STMicroelectronics.<sup>11</sup>

O CI L293D utilizado possui encapsulamento DIP, com 16 pinos (Fig. 18). Os pinos 4, 5, 12 e 13 são pinos terra 0V e o pino 16 (V<sub>CC1</sub>) é utilizado para alimentar o CI com tensão de 5V. O pino 8 (V<sub>CC2</sub>) é utilizado para alimentar os motores, com tensão entre 4.5V e 36V. Como dito anteriormente, o L293D possui controle sobre 4 canais, onde cada motor é conectado à 2 canais. O motor M1 é conectado aos pinos 4 e 6 (1Y e 2Y), sendo controlado pelos pinos 2 e 7 (1A e 2A); enquanto o motor M2 é conectado aos pinos 11 e 14 (3Y e 4Y), sendo controlado pelos pinos 10, 15 (3A e 4A), respectivamente. O pino 1 (1,2EN) habilita/desabilita o controle dos canais 1Y e 2Y e o pino 9 (3,4EN) habilita/desabilita o controle dos canais 3Y e 4Y. Através dos pinos 1 e 9 também é possível realizar o controle de velocidade dos motores, aplicando a técnica de PWM sobre os mesmos.

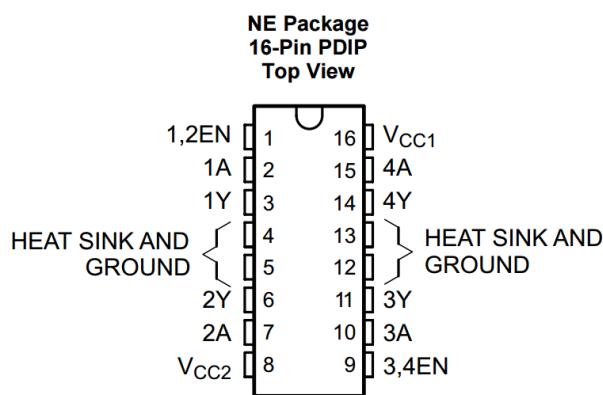


Figura 18: Mapeamento de pinos do L293D. <sup>12</sup>

Na Tabela 2 pode-se observar os comandos acionados nos pinos de controle 1A, 2A, 3A e 4A e os sentidos de rotação ( $\leftarrow$  e  $\rightarrow$ ) dos motores M1 e M2. O valor 1 representa que foi aplicado nível lógico alto (5V) e o valor 0 representa aplicação de nível lógico baixo (0V). Devido a natureza de operação da Ponte-H, nunca se deve aplicar nível lógico alto nos dois controles de um motor ao mesmo tempo, isto é, em 1A e 2A ou 3A e 4A pois pode causar curto-circuito e danificar a fonte de tensão.

Tabela 2: Comandos de Controle do L293D

Motor 1			Motor 2		
1A	2A	M1	3A	4A	M2
0	0	Parado	0	0	Parado
0	1	$\leftarrow$	0	1	$\rightarrow$
1	0	$\rightarrow$	1	0	$\leftarrow$
1	1	Inválido	1	1	Inválido

<sup>11</sup> <https://store.thinkerspace.my/motor-driver/55-1293d-dip.html>. Acesso em: 15/11/2017.

<sup>12</sup> <http://www.ti.com/lit/ds/symlink/l293.pdf>. Acesso em: 15/11/2017.

Para facilitar a conexão dos pinos do L293D com os pinos do Arduino, foi confecionada uma placa de circuito impresso (PCB) para acomodar o CI e estender seus pinos para conexões com *jumpers* para Arduino.

Primeiro, foi desenvolvido o diagrama esquemático contendo o CI e conectores utilizando o software ISIS Proteus (Fig. 19). Nessa disposição, nota-se que o conectores M1\_CTR e M2\_CTR agregam os pinos de controle de cada motor. O conector M1\_CTR possui os pinos 1, 2, 3 que correspondem, respectivamente, aos pinos 1A, 2A e 1,2EN do CI, ao mesmo tempo que o conector M1 possui os pinos 1 e 2 que são conectados aos pinos 1Y e 2Y do L293D. A relação de pinos dos conectores M2\_CTR e M2 com os pinos do CI é análoga.

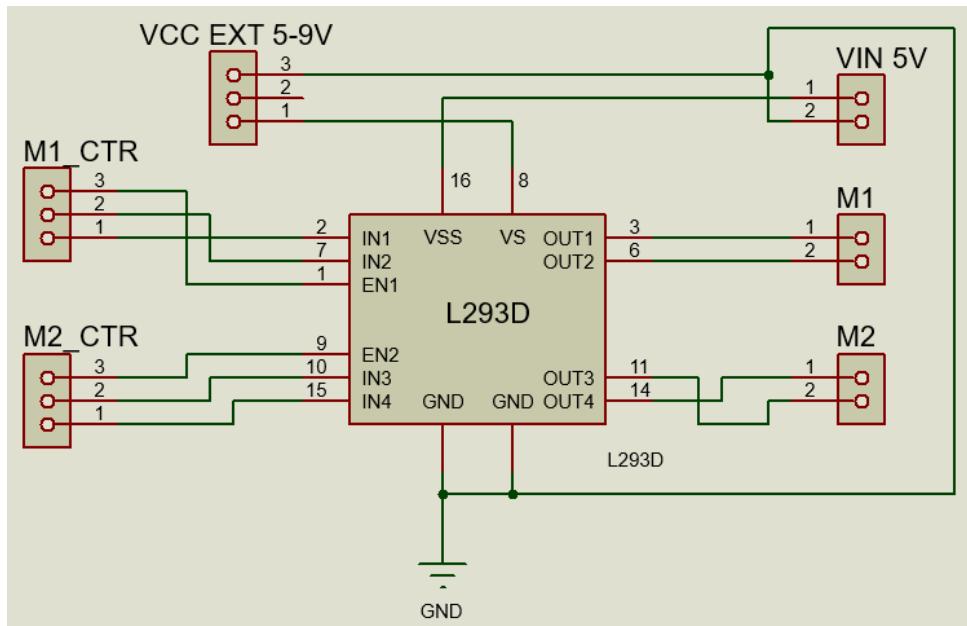


Figura 19: Diagrama esquemático da PCB para L293D elaborado no software ISIS Proteus.

Depois de criado o diagrama esquemático, elabora-se o layout do circuito que será impresso para formar as trilhas de cobre. Esse diagrama, representado pela Figura 20 foi desenvolvido com o software ARES Proteus, impresso em papel fotográfico e realizado transferência térmica para uma placa de fenolite virgem (Fig. 21(a)). Após isso, é realizado a corrosão do cobre com Percloroeto de Ferro, gerando as trilhas do circuito (Fig. 21(b)).

No Arduino os pinos 1, 2 e 3 do conector M1\_CTR estão conectados aos pinos digitais 4, 5 e 6 respectivamente, assim como os pinos 1, 2 e 3 do conector M2\_CTR são conectados aos pinos digitais 8, 9 e 10. A única restrição para a escolha dos pinos é que os pinos 3 dos dois conectores devem ser ligados à portas digitais que possuem PWM, para ser possível realizar o controle de velocidade de rotação.

O controle de direção do robô é feito da seguinte maneira: movimentar para frente significa acionar os dois motores para o sentido A; movimentar para trás significa mover

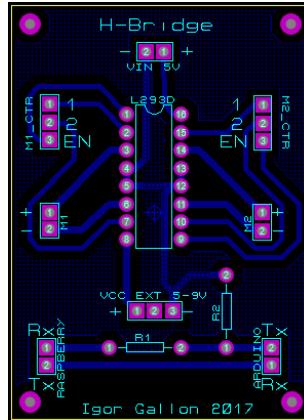
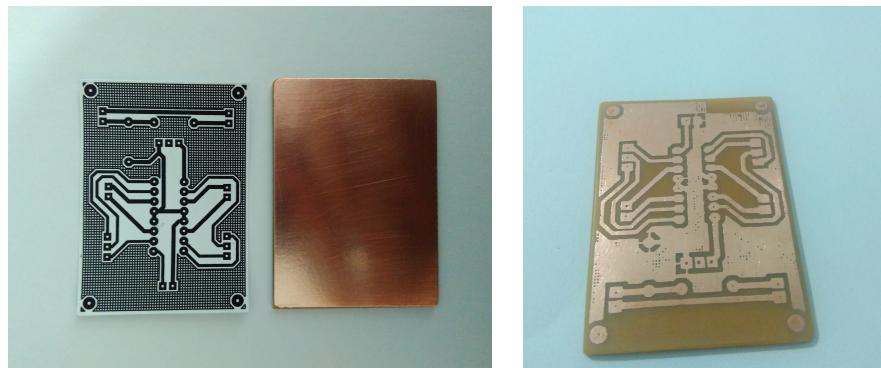


Figura 20: Layout da PCB para L293D elaborado no software ARES Proteus.



(a) Placa de fenolite antes da transferência térmica e corrosão (b) Placa de fenolite depois da transferência térmica e corrosão

Figura 21: Processo de transferência e corrosão da PCB.

os dois motores para o sentido inverso de A; movimentar para a esquerda significa mover o motor direito para o sentido A e o motor esquerdo permanecer parado; e movimentar para a direita significa acionar o motor esquerdo para o sentido A e permanecer o motor direito parado.

Abaixo pode-se observar trechos de código exemplificando o controle do movimento dos motores utilizando a função `digitalWrite()` do Arduino, que configura as portas digitais em nível lógico alto (`HIGH`) ou baixo (`LOW`).

```

1 void forward(){
2     digitalWrite(M1A, HIGH);      // Turn forward Motor 1
3     digitalWrite(M1B, LOW);
4     digitalWrite(M2A, HIGH);      // Turn forward Motor 2
5     digitalWrite(M2B, LOW);
6 }
7 void right(){
8     digitalWrite(M1A, HIGH);      // Turn forward Motor 1
9     digitalWrite(M1B, LOW);
10    digitalWrite(M2A, LOW);       // Turn off Motor 2

```

```
11     digitalWrite(M2B, LOW);  
12 }
```

## 6.4 Composição do conjunto de dados de treinamento

### 6.4.1 Instalação Picamera e OpenCV

Para poder utilizar a Picamera com a Raspberry Pi, é necessário configurá-la previamente. O primeiro passo é conectar o cabo flat da câmera no slot CSI da Raspberry. Com a placa ligada e o SO inicializado, é necessário executar o seguinte comando no Terminal Bash:

```
sudo raspi-config
```

Esse comando inicia a tela gráfica de menus de configuração da Raspberry, onde é possível habilitar a Picamera. Após isso, é necessário instalar a biblioteca Picamera para Python, executando o seguinte comando também no Terminal:

```
sudo pip install picamera
```

A instalação da biblioteca OpenCV na Raspberry Pi é um processo de muitos passos e demorado, em vista da quantidade de bibliotecas e dependências adicionais necessárias para seu funcionamento. Além disso, a biblioteca OpenCV é otimizada para o hardware da Raspberry, sendo compilado no momento de instalação. O processo leva ao todo aproximadamente 1 hora e meia e pode ser visto em detalhes no tutorial elaborado por [Rosebrock \(2016\)](#).

### 6.4.2 Aquisição e tratamento das imagens

Para enviar os dados de treinamento para o Servidor, primeiramente é necessário abrir uma conexão socket TCP/IP com o mesmo. Enquanto isso, do lado do Servidor, é criado um socket esperando por conexão da Raspberry Pi. Essa operação é feita através da biblioteca Socket em Python, tanto na Raspberry Pi quanto no Servidor. Também é necessário inicializar a câmera Picamera, configurando a resolução e a taxa de frames por segundo. No projeto, os frames são capturados com resolução  $32 \times 24$  pixels e a taxa de atualização é de 10 FPS.

Em seguida, a Raspberry Pi espera algum comando enviado do Arduino pela conexão serial. Assim que conexão serial detectar uma nova mensagem, verifica-se se é uma mensagem de parada do controle. Se sim, o procedimento é encerrado e enviado uma mensagem via socket para o Servidor sinalizando a parada de captura de dados e então o socket, de ambos os lados, é fechado. Caso contrário, a Raspberry Pi captura um frame

da Picamera e cria um pacote de dados binários para transmitir pelo socket. Esse pacote é criado utilizando a estrutura `struck` do Python.

A Figura 22 exemplifica a disposição dos dados e os tipos contidos na mensagem enviada pelo socket. **BV** representa o byte de validação da mensagem. Se  $BV = 1$ , significa que a mensagem enviada contém um frame válido e se  $BV = 0$ , significa que a mensagem enviada não contém frame e portanto é uma mensagem de término de envio. **CL** representa a classificação do frame capturado. Essa classificação é obtida através do comando recebido pelo Arduino, sendo convertido conforme a Tabela 3. **FRAME\_LEN** representa tamanho em bytes do frame a ser enviado e **FRAME\_BYTES** é o frame em formato de bytes que será transmitido.

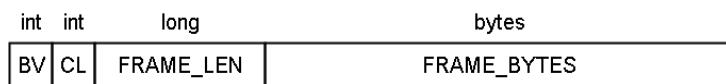


Figura 22: Disposição dos dados na mensagem enviada via socket utilizando `struct`.

Tabela 3: Conversão dos comandos pressionados em classificação de frames.

Controle Remoto	Mensagem Serial	Classificação
Backward	BW	0
Forward	FW	1
Left	LF	2
Right	RH	3

O envio de dados via serial, no Arduino, ocorre pela utilização da biblioteca `Serial`, sendo inicializada a 9600 bits/segundo pela função `Serial.begin(9600)` e os dados são transmitidos pela função `Serial.write(msg)`. Na Raspberry Pi, o Arduino é identificado como um dispositivo e está disponível na porta `/dev/ttyACM0`. A conexão então é feita com a biblioteca `serial`, em Python, inicializando o dispositivo e recebendo dados da seguinte forma:

```

1 # Inicializando dispositivo serial
2 arduino = serial.Serial('/dev/ttyACM0', 9600)
3
4 # Recebendo dados do dispositivo
5 msg = arduino.readline()

```

No Servidor, o processo então ocorre de maneira a restaurar o frame e sua classificação das mensagens recebidas pelo socket. Quando o Servidor detecta a chegada de uma mensagem válida, isto é, com  $BF = 1$ , é extraído do pacote a classificação e o frame em bytes. Com isso, aplica-se uma série de filtros da biblioteca OpenCV, conforme esquematizado na Figura 23. O frame é montado no momento de descompactação como uma matriz de  $32 \times 24 \times 3$ , pois a resolução é de  $32 \times 24$  pixels e cada pixel possui 3 canais (RGB).

Na primeira etapa, a matriz é convertida em um array de 1 dimensão. Aplica-se o filtro de RGB para escala cinza, que converte os pixels de 3 canais para apenas 1. Em seguida aplica-se o filtro **Threshold Binary**, que converte os valores de pixel de tons de cinza para 0 ou 1. O funcionamento desse filtro se dá da seguinte maneira: determina-se uma "faixa de corte", denominada *threshold value*, no intervalo da escala de cinza, todo valor pixel que for abaixo do valor de *threshold*, é convertido em 0 e toda vez que for acima, é convertido em 1. Por fim, o frame é salvo no formato de imagem JPG no diretório /dataTraining do Servidor no formato {classFrame}.{frameID}.jpg, onde **classFrame** é a classificação do frame recebido e **frameID** é um ID único para cada frame (Fig. 24).



Figura 23: Operações gráficas OpenCV aplicadas ao frame antes de serem salvos.



Figura 24: Exemplo de frame após aplicação dos filtros salvo na memória e classificado como "Right". A imagem é salva com nome 3.16.jpg, onde 3 corresponde à classe Right e 16 é o ID do frame no diretório.

As Figuras 25 e 26 possuem os fluxogramas de funcionamento do envio dos frames pela Raspberry Pi e o recebimento dos mesmos pelo Servidor, respectivamente.

## 6.5 Criação da Rede Neural com Keras

Depois de composto o conjunto de dados de treinamento, é necessário criar a Rede Neural. Como explicado na Seção 2.4, a estrutura principal do Keras é chamada **model**.

Existem dois tipos de modelos que podem ser criados no Keras: o sequencial e o funcional. O mais conhecido e também utilizado nesse projeto é o sequencial. Nele, o modelo é construído por camadas sequenciais interligadas. O trecho de código abaixo exemplifica a criação de um modelo utilizando a API **Sequential** presente no Keras. Primeiro, instancia-se o modelo **model** (linha 1) e as camadas são adicionadas utilizando a função **add**, onde todos os neurônios da camada anterior estão conectados com todos os

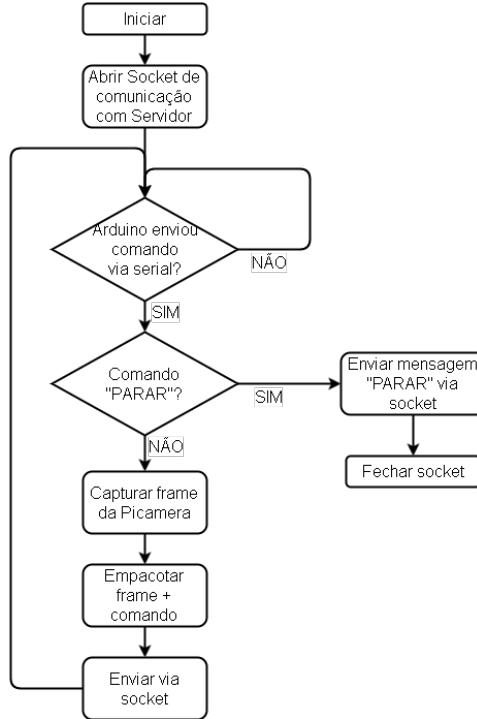


Figura 25: Fluxograma do algoritmo de envio de dados de treinamento. Código `sendDataTraining.py` executado na Raspberry Pi.

neurônios da camada próxima. As camadas são definidas por meio da API Dense. Define-se a primeira camada oculta contendo 1152 neurônios que está interligada a camada de entrada com 2304 neurônios (linha 3) por meio do parâmetro `input_dim=2304`. Ambas as camadas utilizam em seus neurônios a função de ativação ReLU (Fig. 2), através do parâmetro `activation='relu'`, e os valores iniciais dos pesos são definidos de maneira aleatória e uniforme para todos os neurônios por meio do parâmetro `init='uniform'`. Em seguida, define-se a última camada, de saída, com 4 neurônios e função de ativação Softmax (linhas 5 e 6).

```

1 model = Sequential()
2
3 model.add(Dense(1152, input_dim=2304, init='uniform',
4                 activation='relu'))
5 model.add(Dense(4))
6 model.add(Activation("softmax"))

```

A camada de entrada deve possuir um neurônio para cada valor de entrada. Como cada dado de treinamento é representado por uma imagem com resolução  $32 \times 24$  pixels e cada pixel possui 3 valores (RGB), então um dado de entrada possui  $32 * 24 * 3 = 2304$  valores. A camada de saída deve possuir um neurônio para cada possível valor de classificação, que pode ser: Backward, Forward, Left e Right. Os demais parâmetros, como o número de neurônios da camada oculta e as funções de ativação utilizadas foram definido pelo método de Ajuste de Hiperparâmetros, que será definido na Seção 8.

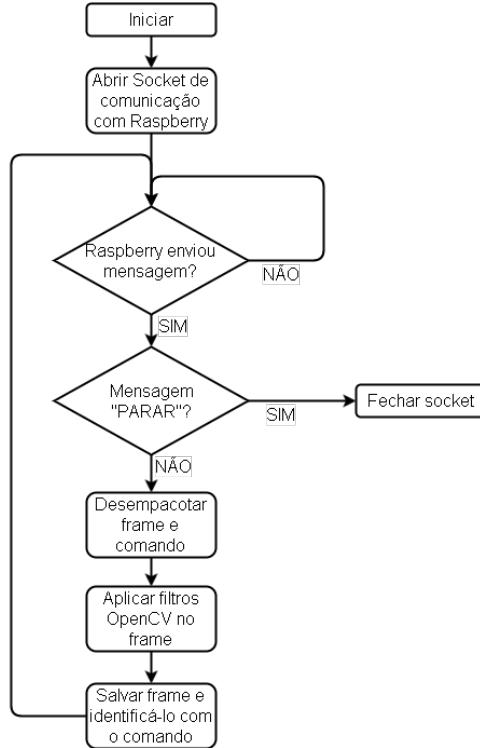


Figura 26: Fluxograma do algoritmo de recebimento de dados de treinamento. Código `receiveDataTraining.py` executado no Servidor.

A Figura 27 contém a representação gráfica do modelo Keras gerado, onde é possível observar as dimensões das camadas de neurônios totalmente conectadas entre si. Na camada de saída, cada neurônio está associado a uma classificação do dado.

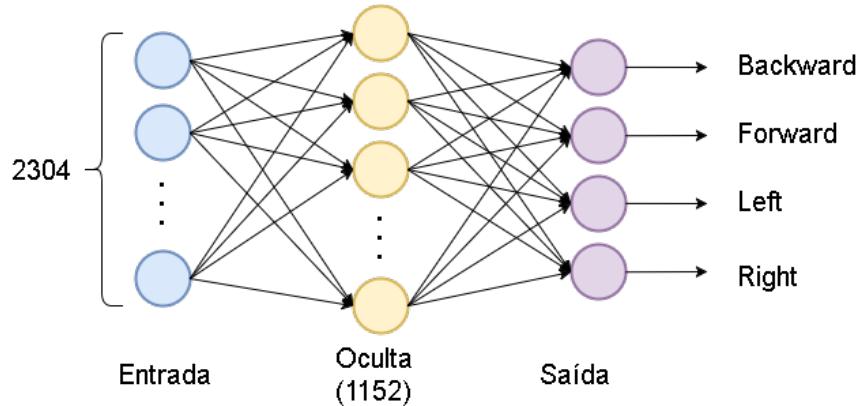


Figura 27: Dimensões do modelo gerado com Keras.

## 6.6 Treinamento e teste

O treinamento é realizado da seguinte maneira: carregam-se as imagens do diretório `/dataTraining`, criando um array de frames (`data`) e um array das suas respectivas classificações (`labels`). As classificações são transformadas em arrays de 4 posições de

valores binários, onde cada posição representa uma possível classificação. Esses array alimentam as saídas dos neurônios da camada de saída na operação de treinamento. A posição que conter valor 1 representa o seu respectivo neurônio ativo. Por exemplo, se a classificação do frame  $F$  for 2, o array conterá  $[0, 0, 1, 0]$ , indicando que o neurônio associado a "Left" deve estar ativo quando a entrada for o frame  $F$ . Na Tabela 4 encontra-se a relação entre a classificação do frame com o array utilizado para realizar o treinamento da Rede Neural.

Tabela 4: Relação entre classificação e valores dos neurônios de saída.

Classificação	Neurônio	Array
0	Backward	$[1, 0, 0, 0]$
1	Forward	$[0, 1, 0, 0]$
2	Left	$[0, 0, 1, 0]$
3	Right	$[0, 0, 0, 1]$

A próxima etapa consiste em dividir o conjunto de dados carregados do diretório em dois sub-conjuntos: o sub-conjunto de dados de treinamento e o sub-conjunto de dados de teste. Essa operação é realizada com a função `train_test_split()`, onde foi definido que, do total de dados, 75% são utilizados para treinamento e 25% para teste.

```
(trainData, testData, trainLabels, testLabels) = train_test_split(data,
    labels, test_size=0.25)
```

Compila-se o modelo definido por meio da função `compile()`. O otimizador escolhido para o projeto foi o Stochastic Gradient Descent (SGD) com uma taxa de aprendizado ( $lr$ ) de 1% (linha 1). Além disso, devido caráter da aplicação, por ser uma Rede Neural para classificação em categorias e não binária, a função objetivo (ou *loss function*) mais adequada é a `categorical_crossentropy` (linha 3).

Para treinar o modelo, utiliza-se a função `fit()` (linha 6), passando os dados de treinamento (tanto os frames, representados por `trainData` quanto seus respectivos arrays de classificação `trainLabels`), o número de `epochs` e o `batch size`, que nesse caso, foram 50 e 80 respectivamente. Esses valores também foram obtidos por meio do método de Ajuste de Hiperparâmetros.

```
1 sgd = SGD(lr=0.01)
2
3 model.compile(loss="categorical_crossentropy", optimizer=sgd,
4     metrics=["accuracy"])
5
6 model.fit(trainData, trainLabels, validation_split=0.33,
7     nb_epoch=50, batch_size=80, verbose=1, shuffle=False)
```

Após o treinamento, é preciso avaliar o modelo utilizando o sub-conjunto de dados de teste. Essa etapa é cumprida por meio da função `evaluate()` que recebe como

parâmetros os frames de teste (`testData`) e suas classificações (`testLabels`) que serão comparadas com os resultados gerados pela rede, obtendo uma avaliação da acurácia do modelo treinado.

```
(loss , accuracy) = model.evaluate(testData , testLabels ,
batch_size=80, verbose=1)
```

Essa operação retorna dois valores, `loss` e `accuracy`, que são, respectivamente, o valor de perda, que deve ser minimizado, e a acurácia, que representa a taxa de acerto das classificações feitas.

Por fim, o modelo é salvo no formato JSON (`model.json`) pela função `to_json()` (linhas 1 e 4) e os pesos dos neurônios treinados são salvos em um arquivo no formato .h5 (`model.h5`) pela função `save_weights()` (linha 6).

```
1 json_model = model.to_json()
2
3 with open( "model.json" , "w" ) as json_file :
4     json_file . write(json_model);
5
6 model.save_weights( "model.h5" )
```

A Figura 28 contém o fluxograma resumindo a operação de treinamento da Rede Neural.

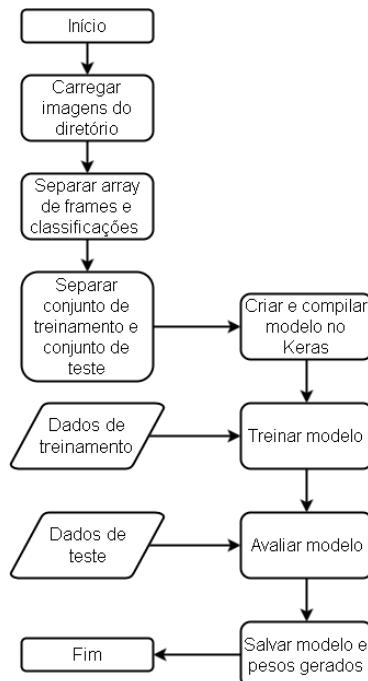


Figura 28: Fluxograma do algoritmo de treinamento da Rede Neural. Código `trainingNeuralNetwork.py` executado no Servidor.

## 6.7 Predição

Sempre quando a Rede Neural é treinada, um novo modelo e conjunto de pesos dos neurônios são gerados pelo Servidor. Para que a Raspberry Pi consiga realizar a predição de novos dados, é necessário então obter o modelo e os pesos do Servidor. Essa tarefa é realizada também por meio de mensagens enviadas via socket.

O processo é bem simples: a Raspberry Pi abre um socket esperando conexão e o Servidor abre um socket para conectar à Raspberry. Uma vez que a conexão foi estabelecida, o Servidor carrega os arquivos `model.json` e `model.h5` em um *buffer*. A mensagem a ser enviada é semelhante ao pacote representado pela Figura 22, que contém na sequência: um bit de validação; o tamanho em bytes do arquivo JSON contendo o modelo; o arquivo em formato de bytes; o tamanho em bytes do arquivo H5 contendo os pesos e o arquivo em formato de bytes. Quando a Raspberry Pi detecta uma mensagem válida no socket, os arquivos são desempacotados e salvos na memória.

Com o modelo e os pesos salvos na memória, a Raspberry Pi está apta para reconstruir a Rede Neural e realizar novas classificações, no modo Predição. A reconstrução é feita através da função `model_from_json()` e em seguida aplica-se a função `compile()` na mesma. Os pesos são carregados por meio da função `load_weights()`.

Primeiro, captura-se um novo frame da Picamera, aplicando os filtros RGB para Gray e *Threshold Binary*, assim como visto na Figura 23. A predição de fato então é realizada pela função `predict()`, que recebe como entrada o frame capturado e retorna a classificação no formato de array de 4 posições, onde cada posição possui a probabilidade de classificação, representando a saída de cada neurônio da camada de saída. Por exemplo, se a saída da predição for um array do tipo [0.02, 0.06, 0.86, 0.06], significa que a probabilidade desse frame ser classificado como "Left" é maior que os demais. Sendo assim, a conversão da classificação é feita pegando o índice do array onde o valor é máximo.

O índice obtido então é convertido conforme a Tabela 3, e a mensagem é enviada para o Arduino via serial utilizando o comando `arduino.write(msg)`. No Arduino, as mensagens são recebidas por meio da função nativa `serialEvent()`, que detecta quando a porta serial possui dados no buffer a serem lidos. Por fim, o Arduino recebe a mensagem da Raspberry Pi e controla os motores de acordo com o comando recebido.

Na Figura 29 é possível observar o fluxograma de funcionamento da operação de predição na Raspberry Pi.

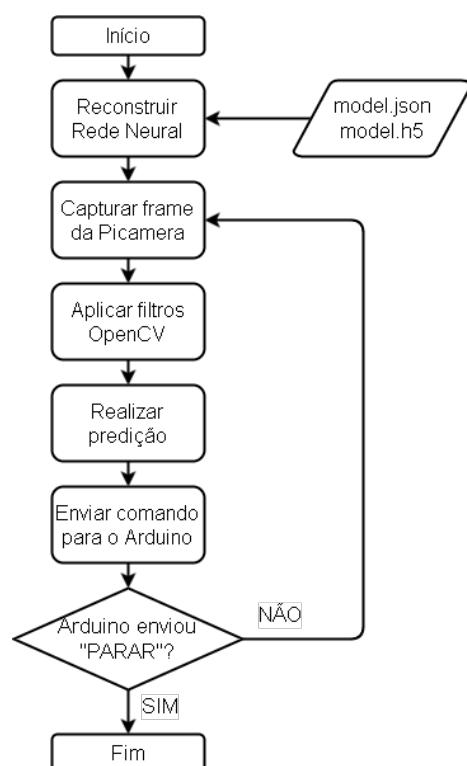


Figura 29: Fluxograma do algoritmo de predição de novos dados. Código `predictingFrames.py` executado na Raspberry Pi.

## 7 Verificação e Validação

As três etapas executadas sobre uma Rede Neural são: treinamento, validação e teste. Para cada uma delas, existe um sub-conjunto de dados que devem ser utilizados. Como visto na Seção 6.6, o treinamento é realizado por meio da função `fit()` utilizando o sub-conjunto de dados de treinamento `trainData` e `trainLabels`. A etapa de validação é realizada por meio do parâmetro `validation_set` na função `fit()`. O teste é realizado utilizando o sub-conjunto de dados de teste  `testData` e `testLabels`, executados pela função `compile()` sobre o modelo Keras.

Sendo assim, do total de dados coletados, os sub-conjuntos são formados da seguinte forma:

- 75% do total de dados coletados são utilizados para **treinamento**;
- 20% dos dados de treinamento são utilizados para **validação**;
- 25% do total de dados coletados são utilizados para **teste**.

Ao final do processo de treinamento, obtém-se os níveis de perda e acurácia da rede treinada. O objetivo então é sempre obter uma elevação da acurácia, ou seja, a taxa de acerto na classificação de dados e consequentemente diminuir a perda (ou erro), tomando cuidado com o **overfitting**, que é quando o modelo gerado se ajusta muito bem aos dados de treinamento porém tem desempenho inferior quando realiza a classificação de novos dados.

Depois que o sistema foi desenvolvido, testado e embarcado na Raspberry Pi e no Arduino, o grande desafio é então melhorar o desempenho da Rede Neural. O intuito é obter bons resultados na classificação de novos dados, simulando uma operação normal de percurso. Entende-se por operação normal onde a recorrência de previsões será muito maior do que de novos treinamento da Rede Neural. Imaginando a utilização do sistema em um chão de fábrica, o sistema só seria calibrado, isto é, realizado o treinamento da Rede para reconhecer o caminho implementado, apenas uma vez enquanto no dia-a-dia o robô operaria apenas identificando o caminho e percorrendo trajetórias.

O desempenho da Rede Neural pode ser modificado através da combinação de parâmetros de configuração do modelo. Para cada combinação, é necessário compilar, treinar, validar e testar o modelo a fim de verificar a acurácia obtida. Foram utilizados alguns conjuntos de parâmetros para se obter, dentre eles, aquele onde a Rede obteve maior taxa de acerto, que serão explicados na Seção 8.

## 8 Análise dos Resultados

Para se obter um melhor desempenho da Rede Neural, alguns parâmetros foram testados: número de neurônios da camada oculta; função de ativação dos neurônios; `batch_size` e `epochs`. As opções combinadas para cada parâmetro foram:

- Número de neurônios da camada oculta: 72, 144, 288, 576, 1152;
- Função de ativação: ReLU, Sigmoid, Softmax;
- `Batch_size`: 10, 40, 80;
- `Epochs`: 10, 50, 100.

É possível observar que nesse conjunto, são ao todo 135 combinações possíveis de parâmetros. Quanto maior é o número de parâmetros a serem testados e as opções para cada um deles, maior é o número de combinações, podendo aumentar drasticamente.

O método de Ajuste de Hiperparâmetros consiste em comutar os parâmetros da Rede Neural, determinados pelo usuário, compilar, treinar, validar e testar, obtendo os valores de perda e acurácia. A partir disso, escolhe-se a combinação de parâmetros onde os resultados obtidos foram os melhores.

A biblioteca em Python `scikit-learn` é utilizada para *Machine Learning* e possui integração com o Keras, sendo possível utilizar dois métodos para Ajuste de Hiperparâmetros: `GridSearch` e `RandomizedSearch`. `GridSearch` consiste na técnica de executar todas as combinações possíveis de parâmetros e, ao final de todo o processo, escolher aquela cuja Rede Neural obteve melhor acurácia. Esse método é o mais preciso, pois varre todo o espaço de possibilidades e garante a melhor combinação, porém a desvantagem está na utilização dos recursos computacionais. Dependendo do tamanho do conjunto de parâmetros e as possibilidades de comutação, é inviável a sua utilização. Já o método de `RandomizedSearch`, como o próprio nome diz, consiste em varrer o espaço de possibilidades de maneira aleatória, limitado a um número de iterações definido pelo usuário. Esse método, de maneira geral, encontra uma boa combinação de parâmetros em um tempo menor e utilizando menos recursos computacionais, porém não garante o melhor resultado.

Para o projeto, por ser um espaço de 135 possibilidades, optou-se pela utilização do método `GridSearch`, utilizando a função `GridSearchCV()`. A operação foi realizada de maneira *offline*, ou seja, depois de obtido o conjunto de dados para treinamento, executou-se uma vez o método de Ajuste de Hiperparâmetros no Servidor, devido à maior

disponibilidade de recursos computacionais. O processo, ao todo, levou cerca de 45 minutos utilizando o espaço de possibilidades mostrado anteriormente, consumindo em média 2.3GB de memória RAM do Servidor, que é equipado com um processador Intel i3 2.10GHz com memória cache de 3MB, e memória RAM total de 4GB.

A melhor combinação encontrada pelo método `GridSearch`, que obteve **74.46%** de acertos de novos dados, foi:

- Número de neurônios da camada oculta: 1152;
- Função de ativação: ReLU;
- `Batch_size`: 80;
- `Epochs`: 50.

A Tabela 5 reúne o custo médio, valor encontrado no mercado, para cada componente utilizado no projeto. Não leva-se em consideração o custo de uma máquina externa como Servidor pois essa não foi necessária ser adquirida para a concepção do projeto. Foi necessário a compra de um cartão microSD para instalar o SO da Raspberry Pi. Fontes de Energia envolvem pilhas recarregáveis para alimentar os motores e uma bateria para alimentar a Raspberry Pi, Picamera e Arduino.

Tabela 5: Custo estimado de cada componente. Valor médio encontrado no mercado.

Componente	Custo estimado
Raspberry Pi 3 Model B	R\$ 160,00
Cartão microSD 16GB	R\$ 30,00
Câmera Picamera	R\$ 40,00
Arduino UNO Rev. 3	R\$ 50,00
CI L293D	R\$ 12,00
Chassis	R\$ 50,00
Fontes de Energia	R\$ 60,00

## 9 Conclusão e Melhorias Futuras

Como resultado do trabalhado realizado até o momento, tem-se um robô seguidor de linha que utiliza Redes Neurais Artificiais para treinar percursos e com isso conseguir se guiar automaticamente, sem intervenção humana. A técnica empregada envolve a utilização de Visão Computacional e Aprendizagem de Máquina embarcadas em Raspberry Pi.

Todo o sistema constitui de uma placa Raspberry Pi 3 para captura de imagens de uma câmera Picamera, uma placa Arduino, que controla motores de corrente contínua. Todos os componentes são de custo acessível, tornando-se uma aplicação viável economicamente, em vista das vantagens de se utilizar a visão computacional se comparado com métodos utilizados atualmente. O emprego de uma câmera juntamente com um tratamento de imagens bem ajustado pode tornar o sistema mais robusto à avarias do percurso, como baixa luminosidade, desgaste das faixas guia, desnível do piso, situações estas que são recorrentes no chão de fábrica, como discutido na Seção 4.

Utilizando a técnica de Ajuste de Hiperparâmetros, chegou-se numa taxa de 74.46% de acertos para novos dados, indicando uma bom desempenho da Rede Neural para esse tipo de aplicação.

Apesar das ferramentas e técnicas apresentadas serem bem empregadas para a situação descrita no início do projeto, é importante ressaltar que há sempre necessidade de melhorar o que já foi desenvolvido e buscar desenvolver novas funcionalidades e técnicas. Uma das melhorias que podem ser realizadas é a utilização de motores mais robustos, com maior precisão e torque. A utilização de técnicas, como PID, para controlar o torque e velocidade dos motores, garantindo consistência das operações.

Em relação à Redes Neurais, é possível realizar estudos de novas técnicas e ajuste de parâmetros a fim de melhorar o desempenho do modelo. Outra melhoria que pode ser realizada é a tentativa de embarcar o processo de treinamento da Rede na Raspberry Pi, tornando assim o robô independente de um dispositivo externo para seu treinamento.

# Referências

- ABREU, P. *Robótica Industrial - Aplicações industriais de robôs*. Universidade do Porto - FEUP, 2011. Disponível em: <[https://web.fe.up.pt/~aml/maic\\_files/aplicacoes.pdf](https://web.fe.up.pt/~aml/maic_files/aplicacoes.pdf)>. Acesso em: 25/10/2017. Citado na página 10.
- ARAUJO, S. A. de et al. Navegação autônoma de robôs: Uma implementação utilizando o kit lego mindstorms. Exacta, p. 123–125, 2006. Disponível em: <<http://periodicos.unesc.net/sulcomp/article/viewFile/995/932>>. Acesso em: 13/11/2017. Citado na página 21.
- Arduino. *Arduino Introduction*. 2017. Disponível em: <<https://www.arduino.cc/en-Guide/Introduction>>. Acesso em: 30/10/2017. Citado na página 17.
- DAS, S. K.; PASAN, M. K. Design and methodology of automated guided vehicle - a review. IOSR Journal of Mechanical and Civil Engineering, 2016. Disponível em: <<https://www.researchgate.net/publication/301261727>>. Acesso em: 13/11/2017. Citado na página 20.
- DZIWIS, D. Automated/self guided vehicles (agv/sgv) and system design considerations. St. Onge Company, 2005. Disponível em: <[http://www.werc.org/assets/1-workflow\\_staging/Publications/430.PDF](http://www.werc.org/assets/1-workflow_staging/Publications/430.PDF)>. Acesso em: 13/11/2017. Citado na página 20.
- FILHO, P. L. de P. *Utilização de um Sistema de Visão Computacional para o Controle de um Robô Móvel*. 9th Brazilian Conference on Dynamics, Control and their Applications, 2010. Disponível em: <<http://www.sbmac.org.br/dincon/trabalhos/PDF/image/68606.pdf>>. Acesso em: "12/11/2017". Citado na página 20.
- GASPAR, D. E. R. Raspberry pi: a smart video monitoring platform. Instituto Superior Técnico, Lisboa, Portugal, 2014. Disponível em: <<https://fenix.tecnico.ulisboa.pt/downloadFile/563345090413381/dissertacao.pdf>>. Acesso em: 10/11/2017. Citado na página 17.
- ISO. *ISO 8373:2012(en) Robots and robotic devices — Vocabulary*. 2012. Disponível em: <<https://www.iso.org/obp/ui/iso:std:iso:8373:ed-2:v1:en>>. Acesso em: 13/11/2017. Citado na página 20.
- JONES, D. *Picamera release-1.13*. 2016. Disponível em: <<https://picamera.readthedocs.io/en/release-1.13/index.html>>. Acesso em: 28/10/2017. Citado na página 15.
- KARN, U. *A Quick Introduction to Neural Networks*. 2016. Disponível em: <<https://ujjwalkarn.me/2016/08/09/quick-intro-neural-network/>>. Acesso em: 10/11/2017. Citado na página 12.
- Keras. *Keras Documentation*. 2017. Disponível em: <<https://keras.io/>>. Acesso em: 11/11/2017. Citado na página 16.

KRIESEL, D. *A Brief Introduction to Neural Networks*. [s.n.], 2007. Disponível em: <[http://www.dkriesel.com/en/science/neural\\_networks](http://www.dkriesel.com/en/science/neural_networks)>. Citado na página 12.

MARENCONI, D. S. M. *Tutorial: Introdução à Visão Computacional usando OpenCV*. RITA-UFRGS, 2009. Disponível em: <[http://seer.ufrgs.br/rita/article/view-rita\\_v16\\_n1\\_p125/7289](http://seer.ufrgs.br/rita/article/view-rita_v16_n1_p125/7289)>. Acesso em: 08/11/2017. Citado na página 16.

MIPI Alliance. *MIPI Camera Serial Interface 2 (MIPI CSI-2)*. MIPI Alliance, 2017. Disponível em: <<https://www.mipi.org/specifications/csi-2>>. Acesso em: 28/10/2017. Citado na página 15.

MIRANDA, D. A. de et al. Desenvolvimento de sistema automatizado agv para guiar veículos acionados por motores elétricos. 2º Congresso Nacional de Inovação e Tecnologia - INOVA 2017, São Bento do Sul, SC, 2017. Disponível em: <<http://www.inova.ceplan.udesc.br/index.php/inova/article/view/28>>. Acesso em: 13/11/2017. Citado na página 20.

MORDVINTSEV, A. K. A. Opencv-python tutorials documentation. v. 1, 2017. Disponível em: <<https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf>>. Acesso em: 09/11/2017. Citado na página 17.

NETTO, M. V. F. *Um Framework baseado em Padrões para a construção de Sistemas Multi-Agentes Auto-Organizáveis*. Pontifícia Universidade Católica do Rio de Janeiro - PUC RIO, 2010. 64-65 p. Disponível em: <[https://www.maxwell.vrac.puc-rio.br/Busca\\_etds.php?strSecao=resultadonrSeq=16434@1](https://www.maxwell.vrac.puc-rio.br/Busca_etds.php?strSecao=resultadonrSeq=16434@1)>. Acesso em: "13/11/2017". Citado na página 20.

OpenCV. *Open Source Computer Vision - Introduction*. 2017. 127 p. Disponível em: <<https://docs.opencv.org/master/d1/dfb/intro.html>>. Acesso em: 08/11/2017. Citado na página 16.

PiSupply. *Raspberry Pi Camera Board v1.3 (5MP, 1080p)*. PiSupply, 2013. Disponível em: <<https://www.pi-supply.com/product/raspberry-pi-camera-board-v1-3-5mp-1080p/>>. Acesso em: 28/10/2017. Citado na página 15.

PSCHEIDT Élio R. *Robô Autônomo - Modelo Chão de Fábrica*. 2007. Disponível em: <<http://www.up.edu.br/blogs/engenharia-da-computacao/wp-content/uploads/sites/6/2015/06/2007.11.pdf>>. Acesso em: 13/11/2017. Citado na página 20.

Raspberry Pi Foundation. *Raspberry Pi 3 Model B Specifications*. Raspberry Pi Foundation, 2017. Disponível em: <<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>>. Acesso em: 26/10/2017. Citado na página 14.

Raspberry Pi Foundation. *Raspberry Pi FAQS - What is a Raspberry Pi?* Raspberry Pi Foundation, 2017. Disponível em: <<https://www.raspberrypi.org/help/faqs/introWhatIs>>. Acesso em: 26/10/2017. Citado na página 14.

ROSEBROCK, A. *Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3*. 2016. Disponível em: <<https://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3>>. Acesso em: 15/09/2017. Citado na página 33.

RS-Components. *Raspberry Pi 3 Model B*. RS-Components, 2017. Disponível em: <<http://docs-europe.electrocomponents.com/webdocs/14ba/0900766b814ba5fd.pdf>>. Acesso em: 26/10/2017. Citado na página 14.

SHIRRIFF, K. *Arduino IR-Remote*. 2012. Disponível em: <<https://github.com/z3t0-Arduino-IRremote>>. Acesso em: 15/11/2017. Citado na página 28.

SILVA, L. G. D. *Navegação Autônoma de Sistemas Robóticos por meio de uma Plataforma Android*. [S.l.]: Escola de Engenharia de São Carlos - EESC USP, 2016. Citado na página 21.

SOUZA, J. de; ROYER, R. Implantação de um sistema agv - veículo guiado automaticamente. um estudo de caso. XXXIII Encontro Nacional de Engenharia de Produção ENEGEP, 2013. Disponível em: <[http://www.abepro.org.br/biblioteca-enegep2013\\_TN\\_STO\\_177\\_010\\_22461.pdf](http://www.abepro.org.br/biblioteca-enegep2013_TN_STO_177_010_22461.pdf)>. Acesso em: 13/11/2017. Citado na página 20.

Sparkfun. *Arduino Comparison Guide*. 2015. Disponível em: <<https://learn.sparkfun.com/tutorials/arduino-comparison-guide>>. Acesso em: 30/10/2017. Citado na página 17.

TensorFlow. *About TensorFlow*. 2016. Disponível em: <<https://www.tensorflow.org/>>. Acesso em: 11/11/2017. Citado na página 16.

Vishay Semiconductors. *IR Receiver Modules for Remote Control Systems. TSOP22.., TSOP24.., TSOP48.., TSOP44..* 2016. Disponível em: <<https://www.vishay.com/docs/82459/tsop48.pdf>>. Acesso em: 14/11/2017. Citado na página 28.

WANKHADE, T.; SHRIWAS, P. Design of lane detecting and followeing autonomous robot. Journal of Computer Engineering (IOSRJCE), v. 2, p. 45–48, 2012. Disponível em: <<https://pdfs.semanticscholar.org/451f/281c4893341fe942f26686f7c6e9e0b29e7a.pdf>>. Acesso em: 13/11/2017. Citado na página 21.

WARNECKE, H. et al. *Manipulator Design, in Handbook of Industrial Robotics, Second Edition*. [S.l.]: John Wiley Sons, Inc., Hoboken, NJ, USA, 2007. Citado na página 10.

XenonStack. *Overview of Artificial Neural Networks and its Applications*. 2016. Disponível em: <<https://hackernoon.com/overview-of-artificial-neural-networks-and-its-applications-2525c1addff7>>. Acesso em: 10/11/2017. Citado na página 12.

# Apêndices

# APÊNDICE A – Código Arduino

```
#include <IRremote.h>

// Key codes for IR communication
unsigned long FORWARD      = 0x2FDD827;
unsigned long BACKWARD     = 0x2FDF807;
unsigned long LEFT          = 0x2FD7887;
unsigned long RIGHT         = 0x2FD58A7;
unsigned long STOP          = 0x2FDC837;
unsigned long PAUSE         = 0x2FD38C7;
unsigned long TRAINING_MODE = 0x2FD807F; // Key 1
unsigned long PREDICT_MODE  = 0x2FD40BF; // Key 2
unsigned long IDLE_MODE     = 0x2FDC03F; // Key 3

// Pins
int M1A        = 4;           // Motor 1 - A
int M1B        = 5;           // Motor 1 - B
int M1EN       = 6;           // Motor 1 - Enable (PWM)
int M2A        = 8;           // Motor 2 - A
int M2B        = 9;           // Motor 2 - B
int M2EN       = 10;          // Motor 2 - Enable (PWM)
int RECV_PIN   = 11;          // IR Receiver

// Messages for Serial communication
char msgTraining [] = "TR\n";
char msgPredicting [] = "PD\n";
char msgForward [] = "FW\n";
char msgBackward [] = "BW\n";
char msgLeft [] = "LF\n";
char msgRight [] = "RH\n";
char msgStop [] = "ST\n";

IRrecv irrecv(RECV_PIN);
decode_results results;

int speedM1 = 115;
int speedM2 = 115;
String serialData = "";           // Data received from Serial communication
boolean onSerialRead = false;    // Determines when stop to receive data
float codeIR;                   // Code got from IR control
```

```

void setup() {
    pinMode(M1A, OUTPUT);
    pinMode(M1B, OUTPUT);
    pinMode(M2A, OUTPUT);
    pinMode(M2B, OUTPUT);
    pinMode(M1EN, OUTPUT);
    pinMode(M2EN, OUTPUT);

    Serial.begin(9600);           // Initialize serial communication
    serialData.reserve(200);

    irrecv.enableIRIn();          // Setting the IR sensor receiver
}

void loop(){
    if(irrecv.decode(&results)){
        codeIR = (results.value);

        if(codeIR == TRAINING_MODE){
            irrecv.resume();    // Receives the next value from IR sensor
            training();
        }
        if(codeIR == PREDICT_MODE){
            irrecv.resume();    // Receives the next value from IR sensor
            predicting();
        }
    }
}

void training(){
    // Inform the Raspberry that the Training mode has begun
    Serial.write(msgTraining);
    enableMotor();                // Enable motors

    while(codeIR != STOP){

        if(irrecv.decode(&results)){
            codeIR = (results.value);
            drive(codeIR);   // Drive the motor according to the given command
            irrecv.resume();
        }
    }
}

void predicting(){
}

```

```
// Inform the Raspberry that the Predicting mode has begun
Serial.write(msgPredicting);

enableMotor();

codeIR = 0;

while(codeIR != STOP){
    serialEvent();

    if(onSerialRead){
        if(serialData == msgForward){
            forward();
        }
        if(serialData == msgLeft){
            left();
        }
        if(serialData == msgRight){
            right();
        }
        if(serialData == msgBackward){
            backward();
        }
    }

    serialData = "";
    onSerialRead = false;
}

// Verify if the STOP button has been pressed
if(irrecv.decode(&results)){
    codeIR = (results.value);
    irrecv.resume();
}

Serial.write(msgStop);
stopMotor();
disableMotor();
}

void drive(float command){

    if(command == FORWARD){
        Serial.write(msgForward);
        forward();
    }
}
```

```
if(command == LEFT){
    Serial.write(msgLeft);
    left();
}
if(command == RIGHT){
    Serial.write(msgRight);
    right();
}
if(command == BACKWARD){
    Serial.write(msgBackward);
    backward();
}
if(command == STOP){
    Serial.write(msgStop);
    stopMotor();
    disableMotor();           // Disable motors
}
if(command == PAUSE){
    stopMotor();
}
}

void enableMotor(){
    analogWrite(M1EN, speedM1);   // Enable Motor 1
    analogWrite(M2EN, speedM2);   // Enable Motor 2
}

void disableMotor(){
    digitalWrite(M1EN, LOW);     // Disanable Motor 1
    digitalWrite(M2EN, LOW);     // Disanable Motor 2
}

void forward(){
    digitalWrite(M1A, HIGH);     // Turn forward Motor 1
    digitalWrite(M1B, LOW);
    digitalWrite(M2A, HIGH);     // Turn forward Motor 2
    digitalWrite(M2B, LOW);
}

void backward(){
    digitalWrite(M1A, LOW);      // Turn backward Motor 1
    digitalWrite(M1B, HIGH);
    digitalWrite(M2A, LOW);      // Turn backward Motor 1
    digitalWrite(M2B, HIGH);
}

void left(){
```

```
digitalWrite(M1A, LOW);      // Turn backward Motor 1
digitalWrite(M1B, LOW);
digitalWrite(M2A, HIGH);     // Turn forward Motor 2
digitalWrite(M2B, LOW);
}

void right(){
    digitalWrite(M1A, HIGH);   // Turn forward Motor 1
    digitalWrite(M1B, LOW);
    digitalWrite(M2A, LOW);    // Turn backward Motor 2
    digitalWrite(M2B, LOW);
}

void stopMotor(){
    digitalWrite(M1A, LOW);    // Turn off Motor 1
    digitalWrite(M1B, LOW);
    digitalWrite(M2A, LOW);    // Turn off Motor 2
    digitalWrite(M2B, LOW);
}

// Trigger event when it detects an input of serial
void serialEvent(){
    while(Serial.available()){
        char inChar = (char)Serial.read();
        if (inChar == '\n'){
            onSerialRead = true;
            serialData += '\n';
        } else{
            serialData += inChar;
        }
    }
}
```

# APÊNDICE B – Códigos Raspberry Pi

## B.1 communicationRaspberry.py

```

import RPi.GPIO as gpio
import serial
import time
import sys
import os
from Messages import Messages
from sendDataTraining import SendDataTraining
from receiveModel import ReceiveModel
from predictingFrames import PredictingFrames

currentState = "stopped";
msg = ""

def training(arduino):

    serverPC = "192.168.0.100"
    #serverPC = "10.0.0.108"

    # Create connection socket with Server PC
    train = SendDataTraining(serverPC, arduino)

    print( "[ Raspberry_CONTROLLER] Sending(frames ... )")
    train.sendFrames()

    try:
        print( "[ Raspberry_CONTROLLER] Waiting for(model) from Server PC... ")
        model = ReceiveModel(' ')
        model.receive()
    except Exception as e:
        print(str(e))
    finally:
        pass

def predicting(arduino):

    print( "[ Raspberry_CONTROLLER] Start(prediction ... )")
    predict = PredictingFrames(arduino)

```

```

predict.run()

# Instantiate Serial communication
print("[Raspberry CONTROLLER] Establishing Serial \
Communication with Arduino ...")

arduino = serial.Serial('/dev/ttyACM0', 9600)
#arduino = serial.Serial('/dev/ttyAMA0', 9600)

while 1:

    try:

        print("[Raspberry CONTROLLER] Waiting for \
Arduino command ...")
        msg = arduino.readline()
        print(msg)

        if msg != "":

            if msg == Messages.msgStop:
                break

            if msg == Messages.msgTraining:
                print("[Raspberry CONTROLLER] Training mode!")
                currentState = "training"
                # training mode
                training(arduino)

            if msg == Messages.msgPredicting:
                print("[Raspberry CONTROLLER] Predicting mode!")
                currentState = "predicting"
                # predicting mode
                predicting(arduino)

    except:
        pass

print("[Raspberry CONTROLLER] Stopped ...")

```

## B.2 sendDataTraining.py

```

# Picamera reference: http://www.pyimagesearch.com/2015/03/30/accessing-the
# -raspberry-pi-camera-with-opencv-and-python/
# Socket reference: https://pymotw.com/2/socket/tcp.html

```

```

# Streaming by socket reference: http://picamera.readthedocs.io/en/release
# -1.9/recipes1.html#capturing-to-a-network-stream
# Struct referece: https://docs.python.org/2/library/struct.html

__author__ = 'Igor Gallon'

from Resolution import Resolution
from Messages import Messages
import picamera
import io
import time
import socket
import struct

class SendDataTraining(object):

    def __init__(self, host, serialPort):
        # PC Server address to connect
        # 192.168.1.103
        self.HOST = host
        self.PORT = 8000
        self.res = Resolution(320, 240); # Frame resolution

        self.frameClass = 1

        # Instantiate Serial comunication
        selfarduino = serialPort

        self.openConnection()

    def openConnection(self):
        # Create a TCP/IP socket
        self.clientSocket = socket.socket(socket.AF_INET, socket.
                                         SOCK_STREAM)
        self.serverAddress = (self.HOST, self.PORT)

        print("[TRAINING] Trying to connect to Server ... ")
        self.clientSocket.connect(self.serverAddress) # Connect to PC
        Server
        print("[TRAINING] Connected!")
        # Make a file-like object out of the connection
        self.connection = self.clientSocket.makefile('wb')

    def closeConnection(self):
        print("[TRAINING] Closing connection ... ")
        self.connection.close()

```

```

        self.clientSocket.close()

def convertToClass(self, message):
    classes = {
        Messages.msgBackward: 0,
        Messages.msgForward: 1,
        Messages.msgLeft: 2,
        Messages.msgRight: 3
    }

    return classes.get(message, -1)

def sendFrames(self):

    msg = ""
    classification = 0

    try:
        # Initializing picamera
        with picamera.PiCamera() as camera:
            # Sets the camera resolution
            camera.resolution = (self.res.width, self.res.height)
            camera framerate = 20           # 20 frames per second

            camera.start_preview()         # Start a preview
            time.sleep(2)    # Wait camera initializing (adjust
                            # luminosity or focus)

        stream = io.BytesIO()

        print("[TRAINING] Start streaming ...")

        while 1:

            try:
                # Check if Arduino sent data
                msg = self.arduino.readline();

                print("[TRAINING] Sending frame with class : {}".format(msg))

                if msg != "":
                    if msg == Messages.msgStop:
                        break

                    # Capture frame from camera

```

```

        camera.capture(stream, 'jpeg')

#Write the byte validation of message (1: valid
   / 0: not else)
self.connection.write(struct.pack('<I', 1))
self.connection.flush()

classification = self.convertToClass(msg)

#Write the command pressioned when training ->
   class of current sent frame
self.connection.write(struct.pack('<I',
   classification))
self.connection.flush()

#Write the length of the capture to the stream
   and flush to
#ensure it actually gets sent
self.connection.write(struct.pack('<L', stream.
   tell()))
self.connection.flush()

#Rewind the stream and send the image data over
   the wire
stream.seek(0)
self.connection.write(stream.read())

#Reset the stream for the next capture
stream.seek(0)
stream.truncate()

except:
    pass

print( "[TRAINING] Stop . . . ")
# Write the byte validation zero signalling the end of
   trasmission
self.connection.write(struct.pack('<I', 0))

finally:
    self.closeConnection()

```

### B.3 receiveModel.py

```

import io
import socket
import struct
import json

```

```

class ReceiveModel(object):

    def __init__(self, host):
        # Listen to all connections
        self.HOST = host
        self.PORT = 8000

        self.openConnection()

    def openConnection(self):
        self.clientSocket = socket.socket(socket.AF_INET, socket.
            SOCK_STREAM)      # Create a TCP/IP socket
        self.serverAddress = (self.HOST, self.PORT)
        self.clientSocket.bind(self.serverAddress)
                            # Bind connection to Server PC

        print( "[RASPBERRY_RECEIVE_MODEL] Waiting for Server connection...")

        # Listen for incoming connections
        self.clientSocket.listen(1)

        print( "[RASPBERRY_RECEIVE_MODEL] Connection established...")
        # Accept a single connection and make a file-like object out of it
        self.connection = self.clientSocket.accept()[0].makefile('rb')

    def closeConnection(self):
        print( "[RASPBERRY_RECEIVE_MODEL] Closing connection")
        self.connection.close()
        self.clientSocket.close()

    def receive(self):

        try:
            while True:
                # Read the verification byte. If the verification is zero,
                quit the loop
                verification = struct.unpack('<I', self.connection.read(
                    struct.calcsize('<I')))[0]

                if verification == 0:
                    break

                # Read length of JSON (NN model)

```

```

        json_length = struct.unpack( '<L', self.connection.read(
            struct.calcsize( '<L' ))) [0]

        # Receive JSON Neural Network model
        json_string = self.connection.read(json_length)
        #json_object = json.loads(json_string)

        # Receive length of model weights file
        weights_length = struct.unpack( '<L', self.connection.read(
            struct.calcsize( '<L' ))) [0]

        # Receive Model Weights file
        model_string = self.connection.read(weights_length)

        stream = io.BytesIO()
        # Save model
        with open("model.json", "w") as json_file:
            stream.write(json_string)
            json_file.write(stream.getvalue())
            stream.seek(0)
            print("[RASPBERRY]RECEIVE_MODEL] Neural Network model saved!")

        with open("model.h5", "w") as weights_file:
            stream.write(model_string)
            weights_file.write(stream.getvalue())
            stream.seek(0)
            print("[RASPBERRY]RECEIVE_MODEL] Weights saved!")

        self.connection.write(struct.pack( '<I', len(modelBytes)))
        self.connection.flush()

    finally:
        self.closeConnection()

```

## B.4 predictingFrames.py

```

# Picamera reference: http://www.pyimagesearch.com/2015/03/30/accessing-the-
# -raspberry-pi-camera-with-opencv-and-python/
# Socket reference: https://pymotw.com/2/socket/tcp.html
# Streaming by socket reference: http://picamera.readthedocs.io/en/release-
# -1.9/recipes1.html#capturing-to-a-network-stream
# Struct referece: https://docs.python.org/2/library/struct.html

__author__ = 'Igor Gallon'

from keras.layers import Dense

```

```

from keras.models import model_from_json
from keras.models import load_model
from keras.optimizers import SGD
from keras.utils import np_utils
from Resolution import Resolution
from Messages import Messages
import numpy as np
import cv2
import picamera
import io
import time
import socket
import struct

class PredictingFrames(object):

    def __init__(self, serialPort):

        # Instantiate Serial comunication
        self.arduino = serialPort
        self.frameRate = 20      # 20 frames per second
        self.sizeFrame = Resolution(320, 240)
        self.sizeData = (32, 24)
        self.thresholdParam = 30
        self.numClasses = 4

        with open("model.json", "r") as json_file:
            print("[PREDICTION] Loading model...")
            model_json = json_file.read()
            self.model = model_from_json(model_json)

        print("[PREDICTION] Compiling model...")
        sgd = SGD(lr=0.01)
        self.model.compile(loss="categorical_crossentropy", optimizer=sgd,
                           metrics=["accuracy"])

        print("[PREDICTION] Loading weights...")
        self.model.load_weights("model.h5")

        print("[PREDICTION] Model and Weights have been loaded successfully
              !")

    def convertToMessage(self, classification):
        classes = {
            0: Messages.msgBackward,
            1: Messages.msgForward,
        }

```

```

    2: Messages.msgLeft,
    3: Messages.msgRight
}

return classes.get(classification, Messages.msgForward)

def run(self):

    msg = ""
    direction = ""
    prob = []
    classification = 0

    print("[PREDICTION] Starting prediction ...")
    try:

        # Initializing picamera
        with picamera.PiCamera() as camera:
            print("[PREDICTION] Initializing picamera ...")
            # Sets the camera resolution
            camera.resolution = (self.sizeFrame.width, self.sizeFrame.height)
            camera framerate = self.frameRate
            camera.start_preview()      # Start a preview
            time.sleep(2)   # Wait camera initializing (adjust
                            # luminosity or focus)

        stream = io.BytesIO()

        while 1:

            if self.arduino.inWaiting() > 0:
                msg = self.arduino.readline()

            if msg == Messages.msgStop:
                break

            try:
                #Capture frame from camera
                camera.capture(stream, format='jpeg')
                image_mat = np.fromstring(stream.getvalue(), dtype=
                    np.uint8)                      # Converting image
                                                # stream array into may numpy format
                frame = cv2.imdecode(image_mat, 1)
                                            #
                # Decoding the image

```

```

frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Apply Gray
filter (convert to 1-channel)
frame = cv2.threshold(frame, self.thresholdParam,
255, cv2.THRESH_BINARY)[1] # Binarizing image
(0 and 1 pixel values)
frame = cv2.resize(frame, self.sizeData)
# Resize
image
frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2RGB).
flatten() # Convert image
to RGB (3-channels) and flat into an 1D-array
array = np.array(frame)/255.0

# Convert to numpy array and normalize pixel
values from [0,255] to [0,1]
array = array[None, :]

# Verticalize array

prob = self.model.predict(array)
#
Get the probabilities array from prediction of
frame
# Get the index of max value in prabilities array
# Backward (0): [max, __, __, __] / Forward (1): [__,
max, __, __] / Left (2): [__, __, max, __] / Right
(3): [__, __, __, max]
classification = np.argmax(prob)

direction = self.convertToMessage(classification)
print("[PREDICTION] " + direction + " " +
Probabilities + " ".format(direction, prob))

# Send direction to Arduino
self.arduino.write(direction)

except Exception as e:
    print(str(e))

#Reset the stream for the next capture
stream.seek(0)
stream.truncate()

except Exception as e:
    print(str(e))
finally:

```

```
print( "[ PREDICTION ] Stopping . . . " )
```

## B.5 Classes adicionais

```
class Messages():

    # Messages for Serial communication
    msgTraining      = "TR\n";
    msgPredicting    = "PD\n";
    msgForward       = "FW\n";
    msgBackward      = "BW\n";
    msgLeft          = "LF\n";
    msgRight         = "RH\n";
    msgStop          = "ST\n";

# Frame resolution class

class Resolution():
    def __init__(self, w, h):
        self.width = w
        self.height = h
```

# APÊNDICE C – Códigos Servidor

## C.1 communicationServer.py

```

from receiveDataTraining import ReceiveDataTraining
from trainingNeuralNetwork import TrainingNeuralNetwork
from sendModel import SendModel
import keyboard

while 1:

    try:
        if keyboard.is_pressed('q'):
            break

        print("[SERVER] Creating connection ...")
        data = ReceiveDataTraining('')

        print("[SERVER] Receiving data training ...")
        data.receive()

        print("[SERVER] Training neural network ...")
        neuralnetwork = TrainingNeuralNetwork()
        model = neuralnetwork.run()
        neuralnetwork.save(model)

        raspberryAddress = '192.168.0.104',
        #raspberryAddress = '10.0.0.109'

        print("[SERVER] Sending Model and Weights to Raspberry in Address"
              "{}".format(raspberryAddress))
        send = SendModel(raspberryAddress)
        send.send()

    except Exception as e:
        print(str(e))
        break

```

## C.2 receiveDataTraining.py

```
# Socket reference: https://pymotw.com/2/socket/tcp.html
```

```

import cv2
import numpy as np
import io
import socket
import struct

class ReceiveDataTraining(object):

    def __init__(self, host):
        # Listen to all connections
        self.HOST = host
        self.PORT = 8000

        # Parameter used in binaryzation
        self.thresholdParam = 30

        self.frameID = 1
        self.verification = -1

        self.openConnection()

    def openConnection(self):
        # Create a TCP/IP socket
        self.serverSocket = socket.socket(socket.AF_INET, socket.
                                         SOCK_STREAM)
        self.clientAddress = (self.HOST, self.PORT)
        self.serverSocket.bind(self.clientAddress) # Bind connection to
                                                    # Raspberry client

        print(" [SERVER_RECEIVE_DATA_TRAINING] Waiting for a Raspberry
              connection ... ")

        # Listen for incoming connections
        self.serverSocket.listen(1)

        print(" [SERVER_RECEIVE_DATA_TRAINING] Connection established ... ")
        # Accept a single connection and make a file-like object out of it
        self.connection = self.serverSocket.accept()[0].makefile('rb')

    def closeConnection(self):
        print(" [SERVER_RECEIVE_DATA_TRAINING] Closing connection ... ")
        self.connection.close()
        self.serverSocket.close()

```

```

def receive(self):

    try:
        while True:
            # Read the verification byte. If the verification is zero,
            quit the loop
            self.verification = struct.unpack('<I', self.connection.
                read(struct.calcsize('<I')))[0]
            if self.verification == 0:
                break

            # Read the classification of the current frame
            # (1: foward / 2: left / 3: right / 4: backward)
            frameClass = struct.unpack('<I', self.connection.read(
                struct.calcsize('<I')))[0]

            print( "[SERVER_RECEIVE_DATA_TRAINING] Classification"
                received :{} .format(frameClass))

            #Read the length of the image as a 32-bit unsigned int
            image_len = struct.unpack('<L', self.connection.read(struct.
                calcsize('<L')))[0]

            #Construct a stream to hold the image data and read the
            image
            #data from the connection
            image_stream = io.BytesIO()
            image_stream.write(self.connection.read(image_len))
            #Rewind the stream
            image_stream.seek(0)
            # Converting image stream array into may numpy format
            image_mat = np.fromstring(image_stream.getvalue(), dtype=np.
                uint8)

            frame = cv2.imdecode(image_mat, 1) # Decoding the image
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # Apply
            Gray filter
            frame = cv2.threshold(frame, self.thresholdParam, 255, cv2.
                THRESH_BINARY)[1] # Binarizing image

            #Save image in path format:
            #data_training/{ class_label }.{ image_num }.jpg
            cv2.imwrite('dataTraining/{ classification }.{ idf }.jpg',
                format(classification=frameClass, idf=self.frameID),
                frame)
            self.frameID += 1

```

```

cv2.imshow( "Streaming from Raspberry Pi" , frame )
cv2.waitKey(1)

finally:

    cv2.destroyWindow( "Streaming from Raspberry Pi" )
    self.closeConnection()

```

### C.3 trainingNeuralNetwork.py

```

# References: http://www.pyimagesearch.com/2016/09/26/a-simple-neural-
# network-with-python-and-keras/
# imutils library reference: https://github.com/jrosebr1/imutils

from sklearn.preprocessing import LabelEncoder
from sklearn.cross_validation import train_test_split
from keras.models import Sequential
from keras.models import model_from_json
from keras.models import load_model
from keras.layers import Activation
from keras.optimizers import SGD
from keras.layers import Dense
from keras.utils import np_utils
from imutils import paths
import numpy as np
import argparse
import cv2
import os

class TrainingNeuralNetwork(object):

    def __init__(self):
        # Size of imported images (10x smaller than original)
        self.size = (32,24)
        self.numClasses = 4 # Foward / Left / Right / Backward
        self.testSize = 0.25 # 25% of data destinated to test set

        self.nNodesInput = 32*24*3
        self.nNodesOutput = self.numClasses
        self.actvFunHidden = "relu"

        # Data and labels
        self.data = []
        self.labels = []

```

```

def run(self):
    # Gets list of images from './dataTraining' path
    print( "[NEURAL_NETWORK] Getting list of images from",
          dataTraining' folder ...")
    imageList = list(paths.list_images("./dataTraining"))

    # Load images into 'data' and extracts the class label into
    'labels'
    for (i, img) in enumerate(imageList):
        image = cv2.imread(img)
                    # Reads the image
        imageRawVector = cv2.resize(image, self.size).
                        flatten()           # Creates a vector of raw
                        pixel intensities (length = w*h*3) - RGB
        self.data.append(imageRawVector)
                    # Appends image as vector format

        label = img.split(os.path.sep)[-1].split(".") [0]
                    # Extracts the class label in
                    the format: 'data_training/{class_label}.{
                    image_num}.jpg'
        self.labels.append(int(label))
                    # Appends label as integer

    print( "[NEURAL_NETWORK] Processed {} images of {}".
          format(i, len(imageList)))

    self.data = np.array(self.data) / 255.0
                    # Converts the pixel values from [0,
                    255.0] to [0, 1] interval
    self.labels = np_utils.to_categorical(self.labels, self.
                                         numClasses)      # Backward (0): [1,0,0,0] / Foward (1):
                                         [0,1,0,0] / Left (2): [0,0,1,0] / Right (3): [0,0,0,1]

    print( "[NEURAL_NETWORK] Partitioning data in training (75%)"
          [/ testing split (25%) ... ])
    (trainData, testData, trainLabels, testLabels) =
        train_test_split(self.data, self.labels, test_size=self.
                         testSize)

    np.random.seed(7)

    # Modeling the network
    model = Sequential()
    model.add(Dense(576, input_dim=self.nNodesInput, init=
                  "uniform", activation=self.actvFunHidden))
    model.add(Dense(32, init="uniform", activation=self.

```

```

        actvFunHidden))
model.add(Dense( self.nNodesOutput))
model.add(Activation("softmax"))

# Train the model using SGD
print( "[NEURAL_NETWORK] □ Compiling □ model . . . ")
sgd = SGD(lr=0.01)
model.compile(loss="categorical_crossentropy", optimizer=
    sgd, metrics=["accuracy"])

print( "[NEURAL_NETWORK] □ Training □ model . . . ")
model.fit(trainData, trainLabels, validation_set=0.2,
    nb_epoch=50, batch_size=128, verbose=1, shuffle=False)

# Show the accuracy on the testing set
print( "[NEURAL_NETWORK] □ Evaluating □ on □ testing □ set . . . ")
(loss, accuracy) = model.evaluate(testData, testLabels,
    batch_size=128, verbose=1)
print( "[NEURAL_NETWORK] □ Loss = {:.4f} , □ Accuracy : □ {:.2f}%".format(
    loss, accuracy * 100))

self.data = []
self.labels = []

return model

def save(self, model):
    # Save Keras model in file
    json_model = model.to_json()
    with open("model.json", "w") as json_file:
        json_file.write(json_model);

    # Save weights
    model.save_weights("model.h5")

print( "[NEURAL_NETWORK] □ Model □ and □ weights □ saved □ to □ disk ! ")

```

## C.4 sendModel.py

```

import socket
import struct
import json

class SendModel(object):

    def __init__(self, host):
        self.HOST = host

```

```

        self.PORT = 8000

        self.openConnection()

def openConnection(self):
    # Create a TCP/IP socket
    self.serverSocket = socket.socket(socket.AF_INET, socket.
        SOCK_STREAM)
    self.clientAddress = (self.HOST, self.PORT)

    print ("[SERVER_SEND_MODEL] Trying to connect to Raspberry... ")
    self.serverSocket.connect(self.clientAddress) # Connect to
        Raspberry client

    # Make a file-like object out of the connection
    self.connection = self.serverSocket.makefile('wb')

def closeConnection(self):
    print ("[SERVER_SEND_MODEL] Closing connection... ")
    self.connection.close()
    self.serverSocket.close()

def send(self):

    endTransmission = 0

    # Write the byte validation of message (1: valid / 0: not else)
    self.connection.write(struct.pack('<I', 1))
    self.connection.flush()

    with open('model.json', 'r') as json_file:
        # Load the model from saved file - serialize json object
        json_data = json.load(json_file)
        jsonString = json.dumps(json_data)
        jsonBytes = jsonString.encode('utf-8')

        # Send the length of JSON string
        self.connection.write(struct.pack('<L', len(jsonBytes)))
        self.connection.flush()
        # Send model
        self.connection.write(jsonBytes)
        self.connection.flush()

    print ("[SERVER_SEND_MODEL] Neural Network Model sent... ")

```

```

        with open( 'model.h5' , 'rb' ) as weights:
            modelBytes = weights.read()

            self . connection . write( struct . pack( '<L' , len( modelBytes ) ) )
            self . connection . flush()

            self . connection . write( modelBytes )
            self . connection . flush()

            print( " [SERVER_SEND_MODEL] □ Weights sent ... " )

# Write the byte validation zero signalling the end of trasmission
            self . connection . write( struct . pack( '<I' , 0 ) )
            self . connection . flush()

            self . closeConnection()

```

## C.5 tunning.py

```

# References: http://www.pyimagesearch.com/2016/09/26/a-simple-neural-
# network-with-python-and-keras/
# imutils library reference: https://github.com/jrosebr1/imutils

from sklearn.preprocessing import LabelEncoder
from sklearn.cross_validation import train_test_split
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Activation
from keras.optimizers import SGD
from keras.layers import Dense
from keras.utils import np_utils
from imutils import paths
import numpy as np
import time
import cv2
import os

def create_model(nodes1=72, nodes2=36, activation='relu'):
    # Modeling the network
    model = Sequential()
    model.add(Dense(nodes1, input_dim=2304, init="uniform", activation=
activation))
    model.add(Dense(nodes2, init="uniform", activation=activation))
    model.add(Dense(4))

```

```

model.add(Activation('softmax'))
sgd = SGD(lr=0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd, metrics=[

    "accuracy"])

return model

if __name__ == '__main__':
    size = (32,24) # Size of imported images (10x smaller than original)
    testSize = 0.25 # 25% of data destineted to test set
    numClasses = 4

    # Hyperparameter space
    batch_size = [10, 40, 80, 128]
    epochs = [10, 50, 100]
    activationHidden = ['relu', 'sigmoid', 'softmax']
    #activationOutput = ['relu', 'sigmoid', 'softmax']
    #optimizer = ['sgd', 'adam', 'rmsprop', 'adagrad', 'adadelta', 'adamax',
    #             'nadam', 'tfoptimizer']
    optimizers = ['sgd', 'adam', 'adamax']
    nodes1 = [1152, 576, 288, 144, 72]
    nodes2 = [576, 288, 144, 72, 36]
    nodes3 = [288, 144, 72, 36, 18]

    param_grid = dict(batch_size=batch_size, epochs=epochs, activation=
        activationHidden, nodes1=nodes1, nodes2=nodes2)

    # Data and labels
    data = []
    labels = []

    print("Loading the input data ...")
    # Gets list of images from './dataTraining' path
    imageList = list(paths.list_images("./dataTraining"))

    # Load images into 'data' and extracts the class label into 'labels'
    for (i, img) in enumerate(imageList):
        image = cv2.imread(img) # Reads the image
        # Creates a vector of raw pixel intensities (length = w*h*3) - RGB
        imageRawVector = cv2.resize(image, size).flatten()
        data.append(imageRawVector) # Appends image as vector format
        # Extracts the class label in the format: 'data_training/{

            class_label}.{image_num}.jpg'
        label = img.split(os.path.sep)[-1].split('.')[0]
        labels.append(int(label)) # Appends label as integer
    # Converts the pixel values from [0, 255.0] to [0, 1] interval
    data = np.array(data) / 255.0

```

```
# Backward (0): [1,0,0,0] / Foward (1): [0,1,0,0] / Left (2): [0,0,1,0]
# / Right (3): [0,0,0,1]
labels = np_utils.to_categorical(labels, numClasses)

print("Spliting data training ...")
(trainData, testData, trainLabels, testLabels) = train_test_split(data,
    labels, test_size=testSize)

seed = 7
np.random.seed(seed)

print("Defining the grid search parameters ...")
model = KerasClassifier(build_fn=create_model, verbose=0)
print("Tuning hyperparameters via grid search (exhaustive method) -"
      "parallel mode")
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=4)
print("Fitting data ...")

try:
    start = time.time()
    grid_result = grid.fit(trainData, trainLabels)
    print("Grid search took {:.2f} seconds".format(time.time() - start))
except Exception as e:
    print(str(e))

print("Best : %f using %s" % (grid_result.best_score_, grid_result.
    best_params_))
means = grid_result.cv_results_[ 'mean_test_score' ]
stds = grid_result.cv_results_[ 'std_test_score' ]
params = grid_result.cv_results_[ 'params' ]
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```