UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Bruno Manica

MINI CARRO AUTÔNOMO COM DEEP LEARNING

Florianópolis

2019

Bruno Manica

MINI CARRO AUTÔNOMO COM DEEP LEARNING

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação.

Florianópolis, 30 de junho 2019.

Prof. Dr. José Francisco Danilo De Guadalupe Correa Fletes
Coordenador do Curso

Banca Examinadora:

Prof.ª Dra. Juliana Eyng
Orientadora

Prof.ª Dra. Jerusa Marchi

Prof. Dr. Elder Rizzon Santos

Este trabalho é dedicado aos meus
amados pais,
Alberto e Elenise.

AGRADECIMENTOS

Se os tijolos não estiverem benfeitos, o castelo cai.

E para construir um grande castelo, preciso de um monte de tijolos. Felizmente, conheço um monte de tijoleiros.

Os meus agradecimentos e estima vão para todos aqueles bons amigos que me emprestaram seus ouvidos e sua sabedoria, para que pudesse colocar cada tijolo no lugar certo.

Não há palavras suficientes para Augusto Zwirtes, Alexandre Behling, Daniela Preto e Henry Rodrigues, que estiveram presentes nos dias bons e ruins, em cada bendita adversidade que a vida colocou em minha frente. Basta dizer que não teria escrito este trabalho sem eles.

À minha prima Camila Puschnerat, por ser um exemplo e inspiração em minha família.

Às professoras que tão amavelmente me emprestaram seus conhecimentos, Juliana Eyng e Jerusa Marchi, para que meus tijolos pudessem ser bons e sólidos. Sua compreensão e sábios conselhos ajudaram nos momentos mais difíceis.

Finalmente, mas não por último, todo meu amor e gratidão a Jéssica Jennifer, por caminhar cada passo ao meu lado.

"Seja a mudança que você quer ver no mundo."

Mahatma Gandhi

RESUMO

Veículos estão se desenvolvendo em plataformas móveis interconectadas. A razão para que isso esteja acontecendo reside na determinação política e econômica que se direciona para o meio ambiente ecosustentável, a redução da superlotação de carros em grandes cidades, assim como os avanços na informação e comunicação estão sendo introduzidos nos veículos modernos. A detecção de auto-estradas é uma das principais tarefas no sistema de percepção de veículos autônomos. As abordagens baseadas em inteligência artificial são populares para esta tarefa devido a enorme quantidade de dados que temos disponível atualmente, por conta do amplo uso de dispositivos e serviços conectados em veículos. Neste projeto foi utilizada uma técnica em inteligência artificial chamada Redes Neurais Convolucionais, com o intuito de mapear os pixels brutos de uma câmera frontal diretamente para os comandos de direção em um mini carro que simula características de pilotagem semelhantes a veículos de produção.
Com dados mínimos de treinamento de humanos, o objetivo deste projeto é criar um sistema que aprende a navegar sobre uma auto-estrada em miniatura, simulando o comportamento de um carro autônomo comercial.

Palavras-chave: Inteligência Artificial. Carro Autônomo. Raspberry Pi. Aprendizado Profundo.

ABSTRACT

Vehicles are developing on mobile interconnected platforms. The reason why it is happening resides on political and economic determination that directs itself to a sustainable environment, as well as a way to ease the overcrowding of cities, the same way advances on information and communications are being introduced on modern passenger vehicles.
Lane detection is one of the main duties on the perception system of autonomous vehicles. The approaches based on artificial intelligence are popular to perform this task due to the large amount of data we have currently, because of the broad use of devices and services connected to the vehicles.
It was used on this project and approach on artificial intelligence called convolutional neural networks, with the intent to map the raw pixels from a front facing camera directly to steering commands in a mini car that simulates the characteristics of driving similar to production vehicles. With minimum human training data, the goal of this project is to create a system that learns to navigate on a miniature lane, simulating the behaviour of and commercial autonomous vehicle.

Keywords: Autonomous Car. Artificial Intelligence. Raspberry Pi. Deep Learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Os sistemas de transporte terrestre tem evoluído de simples sistemas eletromecânicos para complexos sistemas controlados por computador. Veículos foram inicialmente usados para lazer e trabalho, mas eles se tornaram parte de nossas vidas integralmente como um meio de transporte conveniente.

Nos últimos anos a sociedade mudou sua percepção a respeito de sistemas de transporte. Veículos eram referenciados como um meio de conveniência e status social, mas hoje os sistemas de transporte são uma fonte de preocupação com o grande número de acidentes, limitações ecológicas, o alto preço de combustível, entre outros. Governos, indústrias e a sociedade em geral estão se direcionando para os meios de transporte sustentáveis, para endereçar os problemas referidos (BEIKER, 2012).

Uma das principais missões de um veículo autônomo é executar um percurso de um ponto inicial até um ponto final com segurança e exatidão (WEI, 2015). Atualmente, os veículos autônomos estão restritos a operar em ambientes específicos, embora veículos possam operar autonomamente sob condições ideais, como condições meteorológicas favoráveis, há muitos problemas técnicos antes que possam operar em todas condições (LITMAN, 2017), sendo necessário a constante avaliação e aprimoramento de percepção de ambiente através de sensores. Segundo (PAYTON, 1986), diferentes estratégias podem ser necessárias nessas situações para fazer o uso especializado de sensores, e realizar tarefas que são especializadas para as condições do ambiente.

Hoje, os problemas de localização, mapeamento, percepção, controle de veículo, trajetória e planejamento de decisões de alto nível associadas com o desenvolvimento de veículos autônomos, permanecem abertos para desafios que ainda precisam ser resolvidos por sistemas incorporados em plataformas em produção (FRIDMAN et al., 2017). Neste contexto, este projeto tem como objetivo, analisar um subconjunto de problemas e situações encontradas por esses veículos autônomos, simulando o comportamento de um carro autônomo utilizando métodos computacionais.

1.1 OBJETIVOS

Neste trabalho, será apresentado um veículo autônomo em miniatura e como o mesmo alcança a capacidades de navegação com inteligência artificial. O veículo utilizará Redes Neurais Convolucionais e definirá necessidades funcionais para atingir a capacidade de navegação autônoma.

1.1.1 Objetivo Geral

O objetivo geral desde trabalho é implementar um sistema de navegação autônomo embarcado em um mini carro usando hardware de baixo custo utilizando Redes Neurais.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Compreender o estado da arte em relação a Redes Neurais aplicadas a veículos autônomos e robótica;
- Avaliar, pesquisar e adaptar métodos da literatura para simular com objetos do mundo real um carro autônomo em miniatura;
- Alimentar um conjunto de dados relacionados a autoestradas do mundo real, para que sejam aplicadas a Rede Neural do carro;
- Realizar testes sobre o método selecionado através de experimentos que mensurem a capacidade do carro em trafegar uma autoestrada simulada, utilizando medidas como sua precisão e poder de adaptação à pista;
- Redigir a monografia descrevendo o trabalho e resultados obtidos.

1.2 METODOLOGIA

Para atingir o objetivo de criar um carro autônomo em minuatura, foi necessário desenvolver o trabalho em quatro fases:

- Revisão da literatura;

- Modelagem do sistema e implementação;

- Montagem dos componentes de hardware;

- Criação de um banco de imagens capturadas a partir do modelo físico.

1.2.1 Revisão da Literatura

A revisão da literatura teve como base artigos científicos e trabalhos acadêmicos que estudam a aplicação de Redes Neurais em veículos e sistemas robóticos. A pesquisa teve como foco a modelagem do sistema neural e suas respectivas metodologias.

1.2.2 Modelagem do sistema e implementação

O passo precedente à implementação foi a modelagem geral do sistema. Foram necessárias análises, a partir da literatura revisada, de quais ferramentas e componentes seriam necessários para atingir o objetivo de criar um mini carro autônomo. A biblioteca de Redes Neurais Tensorflow versão 1.12 foi escolhida devido ao grande número de recursos disponíveis para aprendizado, como sua abrangência de aplicações em diversos trabalhos e artigos. A compatibilidade entre componentes eletrônicos e a vanguarda de software é vital para o funcionamento previsível do sistema, por tal motivo a linguagem de programação Python versão 3.7 foi utilizada para a implementação de software, pois possui uma ampla variedade de bibliotecas para componentes de hardware.

1.2.3 Montagem dos componentes de hardware

Foram necessários diferentes testes com diversos componentes para a escolha final do hardware, para que o mini carro tivesse o funcionamento que comporta sua funcionalidade, que é ter características semelhantes a um veículo comercial, tal como motores, rodas com esterço e alimentação energética do sistema. O capítulo de Ferramentas e Meios explicita detalhadamente os componentes utilizados.

1.2.4 Criação de um banco de imagens a partir do modelo físico

A realização deste trabalho depende da captura de imagens para serem utilizadas como dados de treinamento. As imagens foram obtidas ao longo dos testes realizados na montagem do hardware e em cenários arranjados especificamente para simular uma auto estrada por meio do controle do carro manualmente, com o intuito de simular um motorista conduzindo um veículo.

1.3 ESTRUTURA DO TRABALHO

Este trabalho é dividido em seis capítulos principais, sendo eles:

- Fundamentação teórica: Fornece um embasamento de conceitos que foram utilizados e mencionados ao longo da leitura;
- Trabalhos correlatos: Análise de pesquisas que atendem o objetivo deste trabalho;
- Desenvolvimento: Descrição detalhada de todas as ferramentas necessárias para a confecção do mini carro, e apresenta a modelagem e implementação do sistema;
- Experimentos: Descreve os diferentes métodos de experimentação da pilotagem do mini carro autonomamente;
- Conclusão: Apresenta a conclusão de pesquisa e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo será apresentado o embasamento teórico que será aplicado no desenvolvimento do projeto, para proporcionar um melhor entendimento sobre os experimentos na área de Redes Neurais.

2.1 APRENDIZADO PROFUNDO

Aprendizado profundo(*Deep Learning*, em inglês), é uma técnica de aprendizado de máquina que emprega redes neurais profundas. A rede neural profunda é uma rede neural multicamada que contém duas ou mais camadas ocultas. Embora isso possa ser simples, esta é a verdadeira essência do *Deep Learning*. A figura 1 ilustra o conceito de *Deep Learning* e sua relação com o *Machine Learning*.

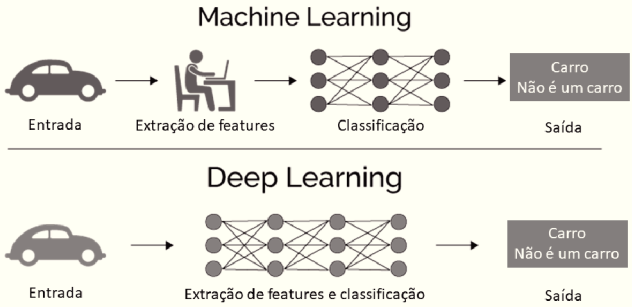


Figura 1 Diferença entre Machine Learning e Deep Learning.
Fonte: (GENç, 2019)

A rede neural profunda está no lugar do produto final do aprendizado de máquina e a regra de aprendizado se torna o algoritmo que gera o modelo(a rede neural profunda) a partir dos dados de treinamento.

2.1.1 Rede Neural Convolucional

Convolutional Neural Networks, ou CNNs, são uma classe biologicamente inspirada de modelos de aprendizagem profunda, que explicitamente aproveita a estrutura espacial das imagens através de conec-

tividade restrita entre camadas (filtros locais), compartilhamento de parâmetros (convoluções) e neurônios especiais de construção de invariância local (LECUN et al., 1990).

Na detecção de imagens no mundo real há grandes variações visuais, tais como as que se referem a qualidade da captura e iluminação, por esse motivo exigem um modelo discriminativo avançado para diferenciar com precisão os recursos da imagem.

Consequentemente, modelos eficazes para este tipo de problema tendiam a ser computacionalmente proibitivos (VEDALDI; LENC, 2015), porém com o advento das CNNs, estes problemas foram contornados pela sua eficiência computacional, onde alcançam excelente desempenho e conquistou a maioria dos campos da visão computacional (SIMONYAN; VEDALDI; ZISSERMAN, 2013).

2.1.2 Arquitetura da CNN

Uma rede neural convolucional é uma sequência de camadas, e cada camada transforma um volume de ativações em outro através de uma função diferenciável. Usamos três tipos principais de camadas para construir as arquiteturas: Camada Convolucional, Camada de Pooling ou Retificação, e Camada Completamente Conectada (KARPATHY, 2019).

2.1.2.1 Camada Convolucional

A camada convolucional é o núcleo do bloco de construção de uma rede convolucional que faz a maior parte do trabalho computacional pesado (KARPATHY, 2019). Os dados usados com uma rede convolucional geralmente consistem em vários canais, cada canal sendo a observação de uma quantidade diferente em algum ponto no espaço ou no tempo, como uma imagem, e toda imagem pode ser considerada como uma matriz de valores de pixel.

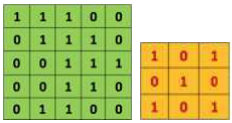


Figura 2 – Matrizes exemplo.
Fonte: (KARN, 2016)

Considere uma imagem, representada na figura 2, de tamanho 5 x 5 cujos valores de pixel são apenas 0 e 1, e outra matriz 3 x 3. A convolução passo a passo da imagem 5 x 5 sobre a matriz 3 x 3 pode ser computada como mostrada nas figura 5. A matriz representada na cor alaranjada representa o *kernel*(ou filtro). A matriz do filtro desliza sobre a matriz da imagem original por 1 pixel (esta operação é chamada de *stride*), para cada posição da matriz do kernel, é computada a multiplicação entre as matrizes, e a saída da multiplicação da matriz é somado.

O resultado da soma forma um único elemento da matriz de saída representada na cor rosada. A matriz resultante é chamada de Mapa de Recursos(*feature map*, em inglês).

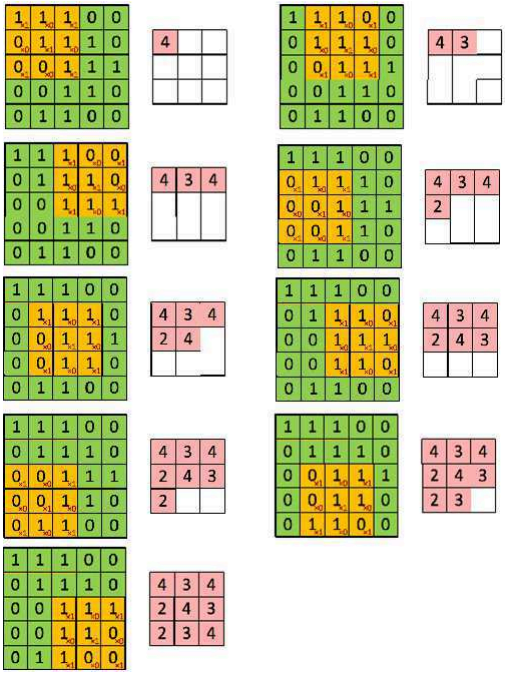


Figura 3 – Passos de convolução sobre matrizes.
Fonte: (KARN, 2016)

Em suma, a camada de convolução filtra a imagem de entrada e produz o *feature map*. Os recursos da imagem são extraídos na camada de convolução, determinado pelo *kernel*. Portanto, os recursos que a

camada de convolução extrai varia dependendo de qual filtro convolução é usado.

O *feature map* que o filtro de convolução cria é processado através da função de ativação antes da camada ceder a saída. A função de ativação da camada de convolução é idêntico ao de uma rede neural comum.

2.1.2.2 Camada de Retificação

Para melhor entendimento das mecanismos das funções de ativação, devemos analisar a entidade mais simples da rede neural artificial, o nêuron. Em um suma, um neurônio artificial calcula a "soma ponderada" de sua entrada, acrescenta um viés e então decide se deve ser "disparado" ou não. Considere o neurônio da figura 26, agora representado matematicamente na equação 2.1:

$$y = \sum(peso \times entrada) + vies \tag{2.1}$$

O valor de y pode ser qualquer valor variando de infinito negativo a infinito positivo. O neurônio realmente não conhece os limites do valor. O padrão de disparo de um neurônio pode ser definido fazendo uma analogia com o cérebro humano, e as funções de ativação são definidas para esse propósito, para verificar o valor y produzido por um neurônio e decidir se as conexões externas devem considerar esse neurônio como disparado ou não.

A função de ativação ReLU é definida pela seguinte equação 2.2:

$$\phi(x) = max(0, x) \tag{2.2}$$

Onde tem uma saída x, se x é positivo, senão a saída do neurônio é zero. O gráfico da 2.2 é representada na figura 4. ReLU é de natureza não linear, e combinações de ReLu também são não lineares (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

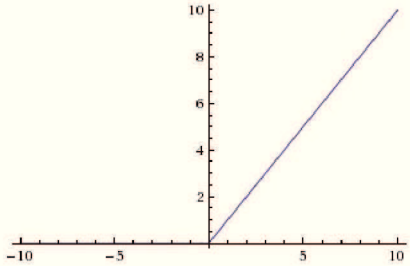


Figura 4 Gráfico da função ReLU.

2.1.2.3 Camadas Totalmente Conectadas

Neurônios em uma camada totalmente conectada têm conexões completas com todas as ativações na camada anterior, como visto no capítulo A.1.2. Suas ativações podem, portanto, ser computadas com uma multiplicação de matrizes seguida por um deslocamento de tendência.

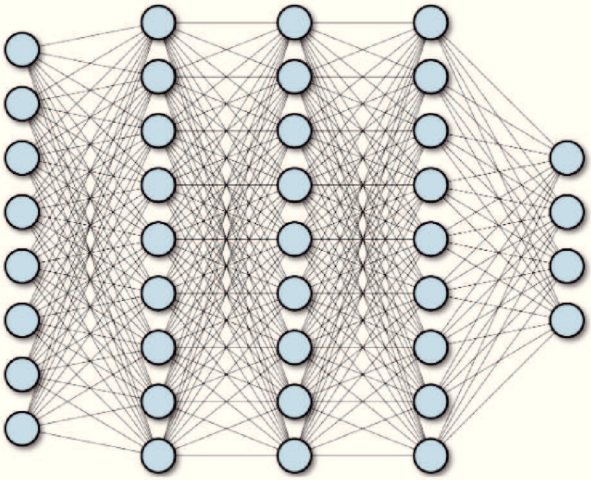


Figura 5 Camadas totalmente conectadas.
Fonte: (ZADEH, 2019)

A única diferença entre as camadas totalmente conectadas, ilus-

trada na figura 5, e camadas convolucionais é que os neurônios na camada convolucional estão conectados apenas a uma região local na entrada, e que muitos dos neurônios compartilham parâmetros. No entanto, os neurônios em ambas as camadas ainda computam produtos escalares, de modo que sua forma funcional é idêntica.

2.1.2.4 Dropout

O termo *dropout* se refere a ignorar randomicamente unidades (neurônios) durante a fase de treinamento(SRIVASTAVA et al., 2014). O termo ignorar tem como conotação não considerar os neurônios durante uma passada específica do treinamento. Para cada estágio de treinamento, nodos são ignorados randomicamente com a probabilidade p, de tal forma que terá uma rede reduzida resultante.

A função *dropout* é necessária para prevenir o *overfitting*, isto é, o modelo foi treinado com muitos dados semelhantes e com isso não abstrai resultados generalizados. Dessa forma, o modelo é treinado de tal maneira que não aprende características interdependentes.

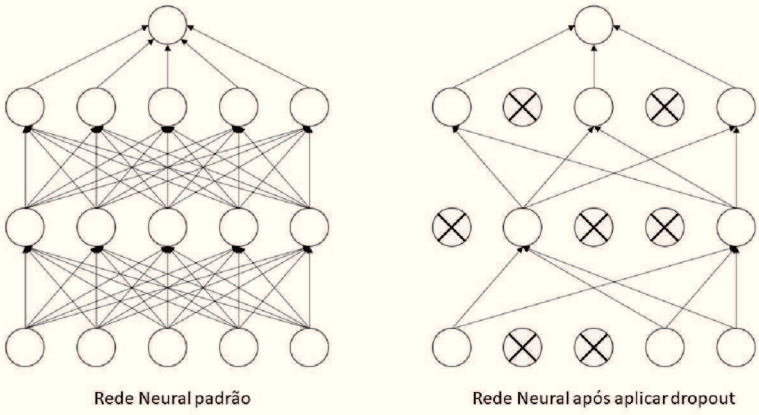


Figura 6 Dropout aplicado a uma rede neural.
Fonte: (SRIVASTAVA et al., 2014)

2.2 CLASSIFICAÇÃO E REGRESSÃO

Os dois tipos mais comuns de aplicação de aprendizado supervisionado são classificação e regressão. A classificação é a aplicação mais predominante em aprendizado de máquina. O problema de classificação se concentra em encontrar literalmente classes às quais os dados pertencem. Em contraste, a regressão não determina uma classe, ao invés disso estima um valor contínuo, como no caso deste trabalho, um ângulo.

No artigo de (HEYLEN et al., 2018), que é relacionado a pilotagem autônoma com redes neurais profundas, foi utilizado o Erro Quadrático Médio para avaliar a função perda, isto é, a divergência entre os valores preditos e os valores de ângulos reais. O autor baseou-se sobre a métrica do Erro Quadrático Médio(equação 2.3), uma vez que permite tomar a magnitude do erro em conta e atribuir uma maior perda para erros maiores do que outras métricas, como o Erro Absoluto.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_{predito} - y_{real}\right)^2 \tag{2.3}$$

Isso é desejável, pois isso pode levar a um melhor comportamento de condução, assumimos que é mais fácil para o sistema se recuperar de muitos pequenos erros do que alguns grandes erros.

O artigo de (FISCHER; DOSOVITSKIY; BROX, 2015) tem como objetivo estimar a orientação de imagens com redes neurais convolucionais, para obtenção do ângulo necessário para retificar imagens de duas dimensões. A figura 7 ilustra o resultados dos experimentos realizados, comparando duas redes neurais treinadas. A regressão é feita através da previsão do cosseno e o seno do ângulo. O ângulo real é então obtido pelo arco tangente. A regressão direta do ângulo como um escalar iria quebrar a continuidade em um ponto do ciclo.

Figura 7 – Resultados obtidos com regressão de pose de imagens.
Fonte: FISCHER, Philipp et al., 2017

O capítulo a seguir irá apresentar trabalhos científicos relacionados à Redes Neurais e suas aplicações em robótica móvel e veículos autônomos.

3 TRABALHOS RELACIONADOS

Este capítulo tem o intuito de fornecer ao leitor resumos de trabalhos realizados no mesmo escopo deste trabalho, ou seja, Aprendizado de Máquina aplicado a robótica, veículos autônomos e detecção de imagens.

O trabalho de (PEREIRA; OLIVEIRA, 2016) utiliza parte do escopo deste trabalho, que é a detecção e análise de imagens, obtidas através do hardware Raspberry Pi. Em suma, o objetivo do trabalho é detectar a presença de pessoas em imagens usando visão computacional. Este projeto utiliza metodologia semelhante que será utilizada neste trabalho de conclusão de curso, em que seu objetivo é realizar o processamento das imagens obtidas pelo chip Raspberry Pi em tempo de execução. Os resultados do trabalho de PEREIRA foram bastante satisfatórios no quesito de detecção de imagens, com resultados de detecção de imagens próximo a 100% em imagens isoladas.

Em (DUGULEANA; MOGAN, 2016) discute o uso de Redes Neurais para o desvio de obstáculos em robôs móveis, utilizando a ferramenta Matlab. O estudo propõe uma solução para o percorrimento da trajetória de um robô livre de colisões em entidades estáticas e estacionárias. O robô do estudo obtém imagens a partir de uma câmera instalada em seu chassi, e a comunicação entre o robô e o Matlab é muito similar a qual será aplicada neste trabalho de conclusão de curso, porém fazendo o uso de um hardware desenvolvido pela empresa chamada Mobile Robots. O meio de comunicação utilizado é Wifi, e o planejamento da trajetória é proposto através de Redes Neurais em Matlab, o qual emite parâmetros para que o robô seja direcionado o sentido correto. Os resultados, conforme descritos no artigo, dependem muito do hardware utilizado para simulação.

O estudo de (MEDINA-SANTIAGO et al., 2014) utiliza uma abordagem semelhante ao artigo apresentado anteriormente, porém utilizando um sensor ultrassônico ao invés da captura de imagens. O hardware utilizado é muito semelhante ao Raspberry Pi e que é muito utilizado em entidades educacionais, o Arduino. A Rede Neural utilizada nesse artigo é a Multi-Layer Perceptron [1], pois este tipo de rede neural tem melhores características para resolver problemas de classificação. Foi concluído que a rede neural utilizada é uma excelente ferramenta para robôs móveis e evasão de obstáculos, o qual ambos se classificam como

[1] Multi-Layer Perceptron é uma classe de Redes Neurais que consiste em três camadas(oculta, entrada e saída).

informações imprecisas.

Há trabalhos que fazem a implementação e análise de algoritmos dentro do escopo de navegação autônoma em robótica, como no artigo de (PANDEY; PARHI, 2016), que propõe uma rede neural feedforward com retro propagação em Matlab que tem como saída um ângulo, o qual direciona o robô. A medida que a navegação baseada em redes neurais controla a direção do robô, e o robô encontra um obstáculo, a rede neural processa a imagem e os dados dos sensores, e retorna um ângulo, o qual fará o robô se mover e desviar do obstáculo encontrado em seu caminho. Os resultados do experimento demonstraram a eficácia de um robô controlado pela rede neural, o qual se move de um ponto a outro em um ambiente com um alvo especificado.

O trabalho de conclusão de curso de (OMRANE; MASMOUDI; MASMOUDI, 2018) utiliza o mesmo hardware, o Raspberry Pi 3 para o controle físico do carro, e uma câmera para obter imagens para navegação e a navegação controlada pelo Matlab utilizando a biblioteca simulink. O robô obtém as imagens em escala de cinza, as envia para a rede neural artificial. O artigo apresenta um modelo cinemático do carro de controle remoto. O resultado teve uma precisão de 96% ao operar o robô, e alcançou os requisitos de navegação autônoma.

As abordagens atuais para o reconhecimento de objetos fazem uso essencial dos métodos de aprendizado de máquina. Para melhorar seu desempenho, pode-se coletar conjuntos de dados maiores, aprender modelos mais poderosos e usar melhores técnicas, como o deep learning.

O controle autônomo do veículo terrestre tem vastos benefícios potenciais comerciais e militares. A maior dificuldade é lidar com ambientes desconhecidos, interferência em sensores e autoaprendizagem para otimizar o desempenho durante a condução. Avanços no aprendizado de máquina para reconhecimento de padrões, problemas de controle e aprendizado ponto-a-ponto prometem melhorias nessas áreas.

O trabalho de (LECUN et al., 1990), teve o objetivo de explorar os benefícios do aprendizado de máquina em um veículo miniatura. Foi construído um veículo de teste, o qual consistiu em um caminhão controlado por rádio comercialmente disponível, que foi equipado com duas câmeras de vídeo e outros sensores. Foi escolhida uma rede neural convolucional, cuja arquitetura é inspirada pelo sistema nervoso humano de baixo nível para a visão.

Este projeto teve como base no trabalho de (BOJARSKI et al., 2016), que trabalharam juntamente com a empresa de componentes de computação gráfica, NVIDIA, para implementar um sistema ponto a ponto de para carros autônomos.

3.1 ARTIGO NVIDIA

Este trabalho de conclusão de curso tem como fundamentação principal o trabalho de (BOJARSKI et al., 2016) e (CHEN; HUANG, 2017). Ambos artigos fazem o uso da abordagem de ponta a ponta, ilustrado na figura 8(*end to end*, em inglês) utilizando redes neurais convolucionais para obter o ângulo de direção adequado para manter carro na pista. O modelo de rede neural convolucional (CNN) pega quadros de imagem bruta como entrada e produz a ângulos de direção.

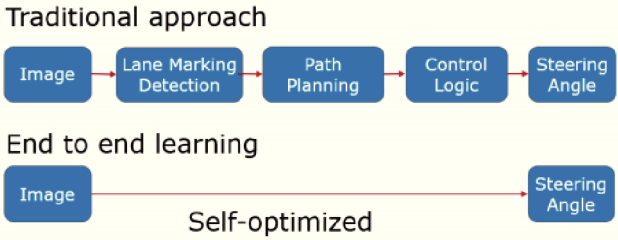


Figura 8 Abordagem ponta a ponta.
Fonte: CHEN, Zhilu; HUANG, Xinming, 2017)

Em (CHEN; HUANG, 2017), o modelo é treinado e avaliado usando o conjunto de dados comma.ai, que contém imagens da vista frontal e os dados do ângulo de direção capturados ao dirigir na estrada. Ao contrário da abordagem tradicional que manualmente decompõe o problema de condução autônoma em componentes como detecção de faixa, planejamento de caminho e direção controle, o modelo end-to-end pode dirigir diretamente o veículo a partir dos dados da câmera frontal. A figura 9 ilustra a precisão do experimento através de uma simulação, comparando os ângulos de esterço do volante do ser humano e os ângulos preditos pela Rede Neural.

O modelo implementado no artigo consiste em uma CNN de três camadas convolucionais e duas camadas totalmente conectadas. A camada de entrada é uma imagem em formato RGB e camada de saída é o ângulo de direção previsto para a imagem de entrada. A primeira camada convolucional usa um núcleo 9 x 9 e um passo de 4 x 4. As seguintes duas camadas convolucionais usam núcleo 5 x 5 e um passo 2 x 2.

As camadas convolucionais são principalmente para extração de recursos, e às camadas totalmente conectadas são principalmente para

a previsão do ângulo de direção. As camadas de dropout são usadas para evitar overfitting. E não há camadas de agrupamento porque os mapas de recursos são pequenos.

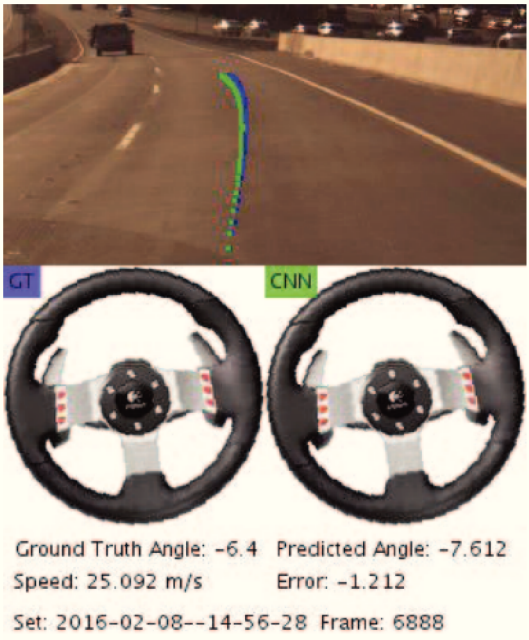


Figura 9 Simulação dos ângulos obtidos.
Fonte: (CHEN, Zhilu; HUANG, Xinming, 2017

Como métrica de avaliação, foram computados as diferenças entre o ângulo real e o ângulo previsto, porém esta métrica é questionável segundo o artigo. O motorista humano pode não manter o veículo no centro da pista o tempo todo. Enquanto o veículo permanecer na pista, os ângulos previstos estão condizentes, portanto não tem que ser exatamente o mesmo que o motorista humano. Em segundo lugar, tanto o movimento do veículo quanto o controle de direção são contínuos, assim, a avaliação quadro a quadro não é apropriada.

Na perspectiva prática em veículos autônomos, (BOJARSKI et al., 2016) utilizou o hardware de processamento NVIDIA DRIVE^{TM} PX 2, o qual é uma plataforma de computação de código aberto de inteligência artificial para carros que permite que fabricantes de carros acelerem a produção de veículos automatizados e autônomos. O hardware foi

montado sobre a plataforma de um carro comercial comum modelo Ford Focus 2013.

A figura 10 mostra um diagrama de blocos simplificado do sistema de coleta para dados de treinamento. Três câmeras são montadas atrás do pára-brisa do carro de aquisição de dados. Os comandos de direção aplicados pelo motorista humano é obtido barramento CAN[2] do veículo. A arquitetura da rede neural será discutida na seção de implementação, visto que foi utilizada a arquitetura deste artigo para a construção do mini carro autônomo. Como avaliação de desempenho autônomo, foi estimada as porcentagem do tempo que a rede neural pode dirigir o carro. A métrica é determinada pela contagem de intervenções humanas simuladas. Essas intervenções ocorrem quando o veículo se afasta da linha central em mais de um metro.

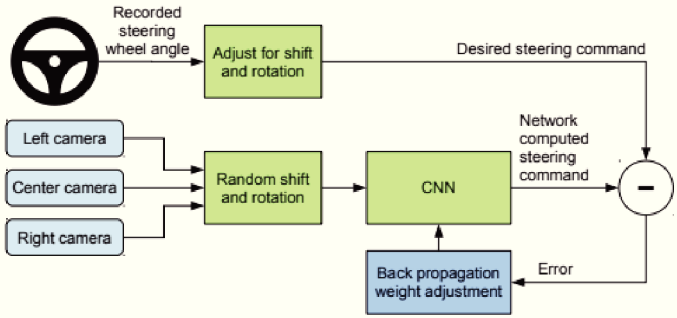


Figura 10 Modelo de implementação do artigo da NVIDIA
Fonte: (BOJARSKI, Mariusz et al. 2016)

[2]O CAN é definido pela International Standardization Organization (ISO) como um barramento serial de comunicação desenvolvido originalmente para aplicações automotivas. Dele é possível obter dados da central eletrônica do veículo.

4 DESENVOLVIMENTO

Para desenvolver o mini carro, além da implementação da Rede Neural, foi necessário implementar os controles dos atuadores mecânicos do hardware, a criação de um sistema de controle manual do carro e a construção utilizando uma série componentes eletrônicos, os quais estão descritos a seguir:

- Um computador Desktop com processador Intel(R) core(TM) i5-2500 @ 3.3Ghz, com placa de vídeo dedicada ATI Radeon 6990 com 4GB de memória;
- Um Raspberry Pi 3 modelo B;
- Sistema Operacional Linux Ubuntu 18.04 instalado no Desktop;
- Sistema Operacional Raspbian(Unix) instalado no Raspberry Pi;
- Módulos de hardware para Raspberry Pi, como câmera, placa de circuito integrado, motores elétricos e baterias;
- Python 3.7 instalados em ambos dispositivos;
- Bibliotecas específicas para Python 3.7.

4.1 FERRAMENTAS

4.1.1 Raspberry Pi

Este capítulo provém informações sobre o Raspberry Pi, como informações gerais e componentes básicos. O Raspberry Pi é um computador do tamanho de um cartão de crédito fabricado pela Fundação Raspberry Pi. A idéia de produzir o Raspberry Pi se iniciou em 2006 com a idealização de que a geração de jovens está deixando de obter conhecimento sobre a operação de computadores.

Um grupo de acadêmicos da Universidade de Cambridge decidiu desenvolver um computador muito pequeno, o qual todos poderiam comprar e criar um ambiente de aprendizado em programação. O projeto Raspberry Pi se tornou promissor com a aparição de processadores móveis de baixo custo com muitos recursos avançados, possibilitando o desenvolvimento do mesmo, o qual foi criado pela fundação Raspberry

Pi, em que seu primeiro produto foi lançado em 2012 (HALFACREE; UPTON, 2012).

O modelo de Raspberry Pi que será utilizado neste trabalho é chamado de Raspberry Pi 3 modelo B, que possui um processador quad-core de 900MHz ARM Cortex-A7 CPU, 1 GB de memória RAM. O diagrama que contém seus componentes básicos é mostrado na figura 11. Os detalhes dos componentes são listados abaixo:

- Conector de energia: Usado para alimentar a placa na porta de 5V micro USB. A necessidade de consumo de energia varia entre 700mA e 1000mA , dependendo dos dispositivos periféricos conectados.

- High-Definition Multimedia Interface (HDMI): Conexão para um display de alta definição (HD) .

- Interface de Camera(CSI): Utilizado para o módulo da camera especificamente projetada para o Raspberry Pi.

- Conector de Áudio: Saída de áudio através da entrada 3.5mm, o qual também suporta saída analógica de vídeo.

- Porta Ethernet: É um conector que provê a velocidade de 10/100 Mbit/s através de um cabo RJ45(LAN).

- Porta USB : Suporta qualquer dispositivo USB tal como teclados, mouses e WebCam.

- GPIO : É o conjunto de pinos universais de entrada e saída(I/O) de propósitos gerais, como conectar placas de expansão ou dispositivos para controlar a CPU.

Um dos recursos principais que necessitam de atenção especial são os pinos de GPIO, que são pinos genéricos e reutilizáveis, os quais podem ser configurados com um valor lógico 0 ou 1. Cada pino age como entrada ou saída dependendo do propósito do usuário. Pinos da GPIO são configurados individualmente, porém configurar um grupo de 8 pinos GPIO resulta em uma porta GPIO, permitindo executar as mesmas operações de pinos individualmente configurados. O diagrama dos pinos da GPIO é mostrado na figura 12.

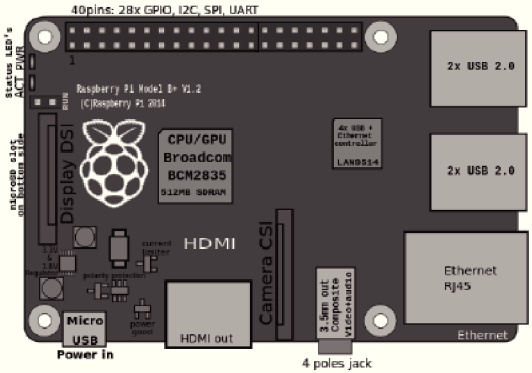


Figura 11 Componentes do chip Raspberry Pi.
Fonte: (BOGDANOV, 2017)

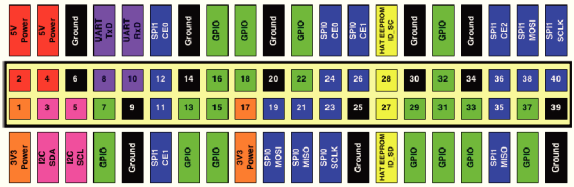


Figura 12 Diagrama da GPIO.
Fonte: O Autor.

4.1.2 Sistema Operacional

O Raspberry Pi é executado em derivados especiais do sistema operacional Linux. Estes derivados possibilitam seleção do SO melhor atendendo às necessidades especializadas do chip. Raspbian é o sistema operacional desenvolvido para o Raspberry Pi. É um sistema operacional livre baseado no Debian GNU / Linux e otimizado para o hardware Raspberry Pi (a arquitetura do processador ARM). Raspbian vem com mais de 35.000 pacotes, ou software pré-compilado em um formato agradável para fácil instalação em um Raspberry Pi. A compilação inicial foi concluída em junho de 2012, mas a distribuição continua ativa, desenvolvida com ênfase na melhoria da estabilidade e desempenho de tantos pacotes Debian quanto possível.

Item	Quantidade
Bateria 18650 3,3V 3500mA	2
Regulador de tensão	1
Motor DC 6V	2
Servo Motor 3V	1
Raspberry Pi 3 modelo B	1
Câmera 5MP para Raspberry Pi	1
Ponte H L298N	1

Tabela 1 – Tabela de componentes eletrônicos

4.1.3 Componentes Eletrônicos

Para a montagem do carro utilizado neste trabalho, foram necessários diversos componentes de hardware e ferramentas prototipação. Os componentes utilizados foram:

O carro foi montado sobre um chassi fabricado em material MDF e os componentes eletrônicos foram alojados dentro e sobre o chassi. O Raspberry Pi é alimentado por uma bateria do tipo 18650, a qual tem a capacidade de fornecer corrente suficiente para o chip, através de um regulador de tensão para preservar o chip caso haja algum pico de tensão, enquanto a segunda bateria energiza a Ponte H, o qual distribui a corrente para ambos motores DC.

4.1.4 Montagem dos Componentes

Para construção do carro autônomo, foi necessário um chassi específico de comportasse os equipamentos eletrônicos. O chassi é fabricado em material sintético e possui encaixes para variados tipos de motores DC do mercado, assim como suporte para encaixe de servo motor para controle de esterço do eixo dianteiro.

É no eixo dianteiro que se concentra ambos motores DC, os quais tracionam do carro, como também o servo motor. Os componentes foram alojados de tal forma que facilitasse a retirada de componentes defeituosos, como também para a distribuição homogênea de peso na superfície do carro, o qual se tornou um fator importante para o tracionamento do carro em superfícies escorregadias.

A alimentação do mini carro é feita por duas bateria de lítio tipo 18650. Estas baterias são componentes de carros autônomos comerciais

atuais, devida sua alta capacidade de gerar corrente. O Raspberry Pi é alimentado por uma bateria, que possui um carregador de bateria específico, o qual um dispositivo eletrônico desenvolvido especialmente para atuar em conjunto com baterias modelo 18650. Seu diferencial é a possibilidade de ser utilizado como um regulador de tensão com amplas possibilidades, pois fornece tensão regulada de 3V a 5V DC.

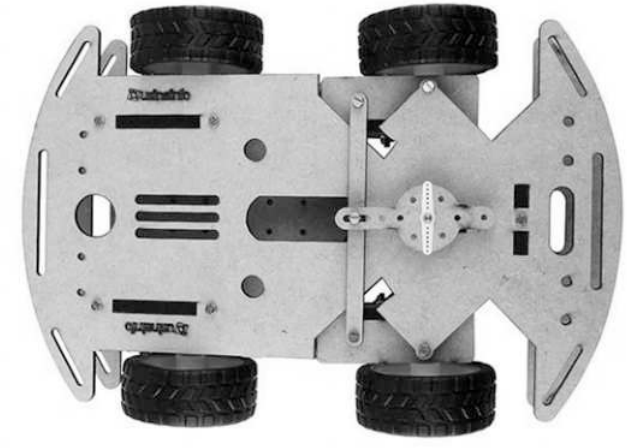


Figura 13 – Chassis do mini carro.
Fonte: https://www.usinainfo.com.br/ Fonte: O Autor.

Por conseguinte, os motores são alimentados através de um circuito eletrônico, chamado Ponte H L298. O L298 é um circuito monolítico integrado que suporta alta tensão, alta corrente, projetado para aceitar níveis lógicos TTL padrão e unidades de cargas indutivas, como relés, solenoides, DC e motores de passo.

Este dispositivo é alimentado pela outra bateria 18650, e transmite a corrente para os motores de acordo com os sinais lógicos enviados para a Ponte H. A câmera é de simples e eficiente utilização, sendo própria para o dispositivo Raspberry Pi com conexão para slot de fita plana. Trata-se de uma câmera digital com resolução de 5 megapixels, possuindo a capacidade de tirar fotos ou fazer filmagens em alta definição quando instalada.

4.1.5 Esquemático Elétrico

Para realizar a montagem do mini carro, é necessário o entendimento dos circuitos eletrônicos que compõe o projeto. Todos os componentes eletrônicos foram montados dentro de suas devidas especificações, para garantir a durabilidade dos componentes robóticos ao longo da fase de testes e experimentação. A figura 14 representa o esquemático elétrico do projeto.

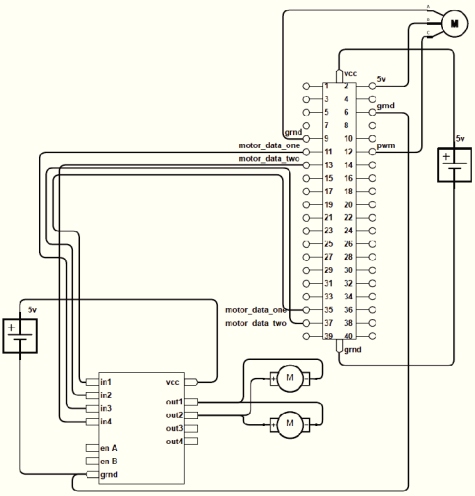


Figura 14 Esquemático elétrico.
Fonte: O Autor

4.2 MODELAGEM DO SISTEMA

O sistema do mini carro autônomo é baseado no modelo cliente-servidor, onde o chip Raspberry Pi tem como função ser o cliente, e o Computador Desktop o servidor. Resumidamente, o sistema tem as seguinte funcionalidades:

- Capturar imagens através de uma câmera, localizada na parte frontal do carro, ligada ao Raspberry Pi;

- Realizar a conexão via rede sem fio Wi-Fi entre o cliente e o servidor via TCP/IP;

- Pré processamento da imagem no cliente, transformando-a em escala de cinza;
- Envio da imagem obtida do cliente para o servidor;
- Processamento da imagem, obtendo a região de interesse, para então alimentar a Rede Neural;
- A Rede Neural retorna um ângulo de esterço;
- Podem ser inseridos comandos ao carro através de um joystick conectado ao servidor;
- É retornado ao cliente os comandos para o controle do carro;
- O Raspberry Pi processa o sinal de retorno, e envia os sinais de comando para os motores do carro.

A figura 15 demonstra graficamente a modelagem do sistema, como as caixas de cor roxa o cliente, e cor esverdeada o servidor.

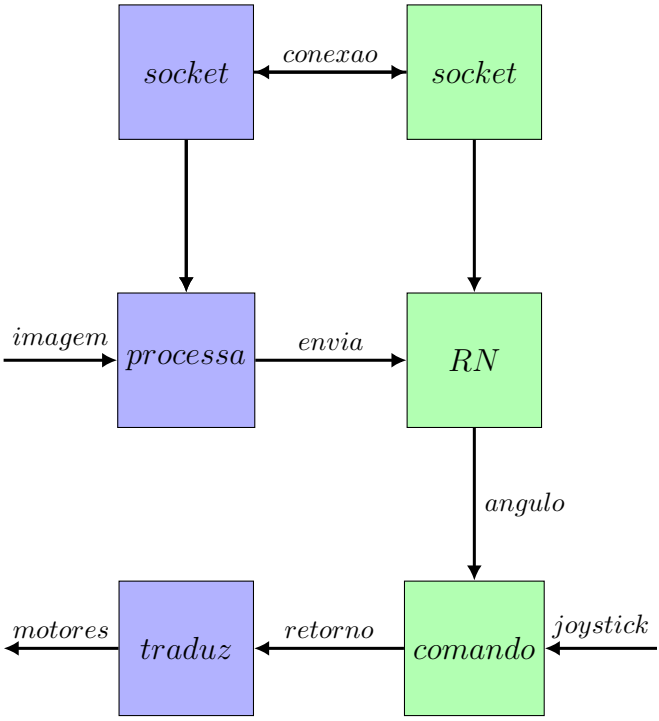


Figura 15 – Modelagem do sistema.

4.3 BIBLIOTECAS

Com o intuito de facilitar a implementação, comunicação e compatibilidade entre os componentes de software, foram utilizadas bibliotecas que foram utilizadas em trabalhos e artigos científicos, e que possuem sólida documentação para a linguagem de programação Python.

4.3.1 Tensorflow

O Tensorflow é uma estrutura computacional para construir modelos de aprendizado de máquina(ABADI et al., 2015). A biblioteca fornece uma variedade de kits de ferramentas diferentes que permitem construir modelos de Redes Neurais no nível de abstração desejado. Pode se usar APIs de nível inferior para criar modelos definindo uma série de operações matemáticas. Como alternativa, pode usar APIs de nível superior para especificar arquiteturas predefinidas, como regressores lineares ou redes neurais.

4.3.2 OpenCV

O OpenCV (Open Source Computer Vision Library) é uma biblioteca de software de visão computacional e aprendizado de máquina em código aberto. Foi construído para fornecer uma infraestrutura comum para aplicativos de visão computacional e para acelerar o uso da percepção da máquina nos produtos comerciais (BRADSKI, 2000).

A biblioteca tem mais de 2500 algoritmos otimizados, o que inclui um conjunto abrangente de algoritmos de visão computacional e de aprendizado de máquina clássicos e de última geração. Esses algoritmos podem ser usados para detectar e reconhecer rostos, identificar objetos, classificar ações humanas em vídeos, rastrear movimentos de câmera, rastrear objetos em movimento, extrair modelos 3D de objetos, produzir nuvens de pontos 3D a partir de câmeras estéreo, juntar imagens para produzir alta resolução imagem de uma cena inteira, encontrar imagens semelhantes de um banco de dados de imagem, remover olhos vermelhos de imagens tiradas usando flash, etc.

4.3.3 Pigpio

PigPio é uma biblioteca para o chip Raspberry Pi, que permite controle da GPIO. O motivo pelo uso desta biblioteca, ao invés da biblioteca tradicional WiringPi(biblioteca de GPIO mais comumente usada), é o fato de ter a interface de modulação de pulso PWM implementada com alta precisão. Isto possibilita alta sensibilidade na aplicação no esterço do servo motor, o qual dita a direção do carro.

4.3.4 PyGame

É uma biblioteca multiplataforma projetada para facilitar a gravação de software multimídia, como jogos, na linguagem de programação Python. Possui interface com o joystick do video game Xbox, que foi utilizado neste trabalho.

4.4 IMPLEMENTAÇÃO

A implementação é dividida em três módulos: conexão, controle e rede neural. Os três módulos foram implementados separadamente, contudo há interdependência dos módulos para o funcionamento mini carro. Os capítulos seguintes especificam os detalhes da implementação por meio de descrição de funcionalidades, diagramas e algoritmos.

4.4.1 Conexão

Para realizar a comunicação entre o chip Raspberry Pi e o computador Desktop através de rede TCP, foi utilizada a interface de *socket*, de tal modo que a comunicação entre cliente e servidor fosse estabelecida. A programação em socket é iniciada importando a biblioteca 'socket', e criando um socket simples.

No lado do cliente, foi criada uma instância de *socket* e passado dois parâmetros. O primeiro parâmetro é AF_INET e o segundo é SOCK_STREAM. AF_INET o qual refere a endereços da classe ipv4. O parâmetro SOCK_STREAM significa que é orientado ao protocolo TCP.

4.4.1.1 Cliente

O servidor interage com o *endpoint*(cliente), que no caso é o chip Raspberry Pi. Primeiramente é necessário criar um objeto *socket*, e conectá-lo ao servidor na porta 8001, por conseguinte os dados são recebidos do servidor, e a conexão é encerrada.

Algorithm 1 Criação de um socket no cliente.

Require: *socket*
1: $s \leftarrow socket.socket(AF\_INET, SOCK\_STREAM)$
2: $port \leftarrow 8001$
3: $s.connect(192.168.0.20, port)$

Por fim, o cliente e o servidor estabelecem a conexão entre si, de modo que as imagens captadas pelo Raspberry Pi são transformadas em byte array e então enviadas através da conexão estabelecida entre os dispositivos. Então a imagem é recebida no servidor, decodificada para formato de imagem e processada pela Rede Neural. A resposta do servidor corresponde a um comando de retorno, que é enviado do servidor para o cliente como resposta da conexão. O comando retornado pelo servidor, e recebido pelo endpoint é um comando interpretado pelo cliente como um comando nos atuadores em hardware.

4.4.1.2 Servidor

No lado do servidor, o qual se refere ao PC Desktop, é necessário utilizar o método da biblioteca socket chamado bind(), o qual liga a um IP e porta específicos para que o servidor possa escutar as requisições naquele IP e porta. O servidor possui o método listen(), o qual coloca o servidor no modo de escuta. Isso permite que o servidor escute as conexões recebidas. O servidor também possui o método *accept()* e *close()*, os quais iniciam e fecham conexões com o cliente.

4.4.2 Controle

Os capítulos seguintes explicam em detalhes a implementação do controle do carro, em ambos cliente(Raspberry Pi) e servidor(Desktop), ou seja, como é criado o sinal de controle, enviado para o cliente e traduzido em sinais de comando para os atuadores mecânicos.

Algorithm 2 Criação de socket no servidor

Require: *socket*
1: $s \leftarrow socket.socket(AF\_INET, SOCK\_STREAM)$
2: $port \leftarrow 8001$
3: $(s.bind(port)$
4: $while(True)$:
5: $\quad s.accept()$

4.4.2.1 Servidor

Para que o mini carro pudesse ser controlado e treinado, foi necessário implementar o controle manual, no qual o usuário utiliza um controle de video game Xbox para acelerar, frear e mudar de direção. O sinal de controle do carro é a mensagem de retorno do servidor, após o cliente enviar a imagem(frame) capturada da câmera localizada na parte frontal do carro, e enviar através do socket TPC/IP discutido na seção 4.4.1.1 . É utilizada a biblioteca da linguagem Python chamada PyGame, que disponibiliza uma interface em uma linguagem de programação de alto nível os sinais enviados do joystick. Segue abaixo na figura 16 o diagrama do controle e suas respectivas ações:

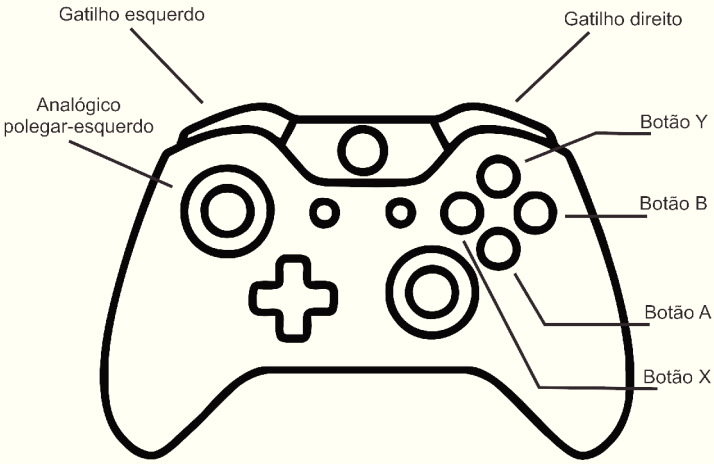


Figura 16 Controles do carro.
Fonte: O Autor.

- Analógico do polegar esquerdo, para mudar a direção do carro;

- Gatilho direito, para acelerar os motores;
- Gatilho direito, para frear os motores;
- Y ativa o controle manual;
- B encerra a conexão com o servidor;
- A inicia a captura de imagens;
- X ativa o controle autônomo.

Usando a biblioteca PyGame, o sinal obtido do botão polegar esquerdo varia do valor '-1' a '1' do tipo inteiro, e os sinais dos botões gatilho variam de '0' a '1' do tipo inteiro. Para traduzir o sinal do joystick em algo compreensível para o desenvolvedor, o sinal do controle foi transformado na escala de 0 a 255, para facilitar a transmissão para o cliente dos bytes em valor do tipo inteiro. No botão polegar esquerdo, foi definido a seguinte parametrização de valores, para construção do sinal de retorno:

Tabela 2 – Sinal do joystick, botão polegar esquerdo.

Sinal do joystick	Sinal transformado	Ação correspondente
entre 0 e 255	'0' + ' 0 a 255´	Carro vira para direita, com sensibilidade que varia de 0 a 255.
entre 0 e -255	'1' + ' 0 a 255´	Carro vira para esquerda, com sensibilidade que varia de 0 a 255.

Nos botões de gatilho, seguem as parametrizações, de acordo com o sinal recebido pelo joystick:

Tabela 3 – Sinal do joystick, gatilho direito.

Sinal do joystick	Sinal transformado	Ação correspondente
0 ou 1	'0' + '0' ou '0' + '1'	Carro frea ou Carro acelera para frente

Tabela 4 – Sinal do joystick, gatilho esquerdo.

Sinal do joystick	Sinal transformado	Ação correspondente
0 ou 1	‘1’ + ‘0’ ou ‘1’ + ‘1’	Carro frea ou Carro acelera em modo reverso

Segue o exemplo , se o for recebido do joystick os seguintes sinais:

Tabela 5 – Exemplo de transformação de sinal do joystick.

Botão acionado	Sinal recebido	Sinal transformado
Botão polegar esquerdo	-125	’1 032’
Gatilho direito	1	’0 1’

A mensagem de retorno será montada da seguinte forma:

$$retorno = [direcao][angulo][direcaodemovimento][velocidade] \quad (4.1)$$

Por conseguinte, a mensagem de retorno será:

$$retorno = 103201 \quad (4.2)$$

O sinal de retorno é enviado pela rede ao cliente neste padrão em formato de String, onde no cliente é transformado para comandos mecânicos, que será descrito no capítulo seguinte.

4.4.2.2 Cliente

A decodificação da mensagem recebida do servidor é bastante simples, fazendo caminho inverso da montagem da mensagem mostrada acima, porém transformando o sinal em pulsos de clock, para transmissão do sinal da GPIO do Raspberry Pi para os motores. A mensagem de retorno recebida do servidor é decodificada em seu formato tipo String original, onde é fragmentada para tradução em sinal de pulsos de clock.

O primeiro passo da tradução é a obtenção do ângulo, o qual será convertido em frequência de pulsos de clock, enviado ao controle servo motor. São extraídos os 4 primeiros caracteres da String, onde o primeiro caractere representa a direção(esquerda ou direita) representado pela equação 4.3, e o três caracteres seguintes representam o

ângulo do movimento, representados na equação 4.4.

$$direcao = [caracteres\ \ da\ \ posicao\ \ 0\ \ da\ \ mensagem] \tag{4.3}$$

$$angulo = [caracteres\ \ da\ \ posicao\ \ 1\ \ a\ \ 3\ \ da\ \ mensagem] \tag{4.4}$$

A String de mensagem é transformada para o formato de inteiro, com o intuito de realizar cálculos para transformação dos valores em sinais lógicos. O caractere de posição 1 da String mensagem é multiplicado por 100, o terceiro é multiplicado por 10 e o quarto caractere permanece o mesmo valor. A representação desse método está na equação 4.5, o qual foi criado empiricamente com a finalidade de calibrar a intensidade do sinal caso o comando recebido do servidor exija que o carro esterce com mais intensidade.

$$angulo = [caractere\ 1 \times 100]\ [caractere\ 2 \times 10]\ [caractere\ 1 \times 1] \tag{4.5}$$

O valor ângulo tem seu sinal alterado pelo primeiro caractere(posição 0 no vetor de comando), que dita a direção de esterço. Caso o primeiro caractere da String mensagem seja '0'(zero), o ângulo será multiplicado por (-1), senão permanece o ângulo original. e por fim, a mensagem ângulo resultante será a mensagem em pulsos de clock, que será enviado ao servo motor através da GPIO. A implementação da tradução de ângulo está no algoritmo 3. Tomando o exemplo da seção 4.4.2.1 como sinal recebido, temos:

$$retorno = 103201 \tag{4.6}$$

$$direcao = 1 \tag{4.7}$$

$$angulo = 100 \times 0 + 10 \times 3 + 2 = 32 \tag{4.8}$$

A tradução do sinal de comando de esterço para pulso de clock é feito de acordo com a frequência de pulsos, o qual tem a medida em *Hertz*. O ângulo sem esterço(zero graus) equivale a 1600Hz, e seus limites são de acordo com o limite físico do eixo do carro, que é aproximandamente 35 graus. A figura 17 exemplifica a transformação do sinal para o movimento de direção:

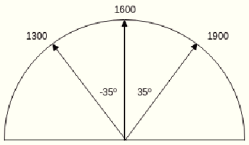


Figura 17 Transformação do sinal para pulso de clock.

O segundo passo envolve a tradução dos caracteres da quinta e sexta posição da mensagem recebida. O caractere da mensagem na posição 4(posição 3 no vetor comando) dita o sentido da direção(frente ou reverso), e o outro caractere é utilizado para acionar o acelerador ou o reverso do carro. Como o Raspberry Pi possui somente um pino do tipo PWM(pino de pulso variável, o qual é usado para o controle do servo motor), a tradução se resume em enviar o sinal a partir de dois pinos de sinal fixo, que enviam sinal alto ou baixo(0 ou 1).

Algorithm 3 Decodificação da mensagem de comando pelo cliente para controle do servo motor.

Require: *pigpio*

1: $angulo \leftarrow 100 \times int(dir[1]) + 10 \times int(dir[2]) + int(dir[3])$
2: $if\ int(dir[0]) == 0:$
3: $\quad angulo* = -1$
4: $angulo_{real} \leftarrow angulo \times 1600$ ▷ O angulo é transformado em pulsos de clock
5: $set\_servo\_pulsewidth(18, angulo_{real})$ ▷ Envio de sinal para o servo motor

Se o caractere do índice 4 da String mensagem for 0, e o carácter do índice 5 for 1, duas variáveis terão seus valores atribuídos como segue:

Algorithm 4 Decodificação da mensagem de comando pelo cliente para controle do motor DC.

Require: *pigpio*

1: $if\ int(dir[4]) == 0\ and\ int(dir[5]) > 0:$
2: $\quad write(MOTOR_1, 0)$ ▷ Atribui sinal alto e baixo para o pino da GPIO
3: $\quad write(MOTOR_2, 1)$

4.4.2.3 Rotina principal

Para controlar todas funcionalidades do servidor, a rotina principal, representada no algoritmo 5, possui as configurações de rede entre o cliente e servidor e parâmetros do carro. Também estão contidas as sub-rotinas de carregamento de imagens via transmissão pela rede, obtenção de dados de entrada do joystick e avaliação de imagens pela rede neural.

A primeira etapa da rotina principal é iniciar uma thread da sub

rotina ação de joystick. Nela é montada a mensagem de comando com os valores de esterço e direção, de acordo com os botões acionados do joystick, o qual é um laço infinito que é executado ao longo de toda execução da rotina principal do servidor para obtenção de qualquer ação do usuário.

Algorithm 5 Pseudo-código da rotina principal.

Require: $socket, pygame, opencv, tensorflow$
1: **Init**
2: $enviar\_comandos \leftarrow 1$
3: $imagem\_contador \leftarrow 0$
4: $comando\_conexao, video\_conexao \leftarrow configura\_conexao()$
5: $controle \leftarrow Thread(acao\_joystick)$
6: $while\ enviar\_comandos\ ==\ 1$
7: $imagem \leftarrow carregar\_imagem(video\_conexao)$
8: $if\ joystick\_input(0) == 1$ ▷ se botão A é pressionado
9: $salvar\_imagem(imagem\_contador.concatena(comandos))$ ▷ salvar imagem
10: $if\ joystick\_input(2) == 1$ ▷ se botão X é pressionado
11: $modo\_autonomo \leftarrow 1$
12: $if\ joystick\_input(3) == 1$ ▷ se botão Y é pressionado
13: $modo\_autonomo \leftarrow 0$
14: $if\ modo\_autonomo == 1$
15: $controle \leftarrow predicao\_RN(imagem)$ ▷ predicao da RN
16: $comando\_conexao.envia(controle.codificar())$
17: $if\ joystick\_input(1) == 1$ ▷ se botão B é pressionado
18: $modo\_autonomo \leftarrow 1$
19: $comando \leftarrow 000000$
20: $comando\_conexao.envia(controle.codificar())$
21: $exit()$
22: $imagem\_contador\ +=\ 1$
23: **End**

O segundo passo é iniciar o laço principal da função main, o qual primeiramente irá obter a imagem através do socket de conexão. O socket é passado como parâmetro para a sub-rotina de carregar imagens, que se encarrega de decodificar os dados recebidos da conexão e formatá-los corretamente para o formato de imagem.

Ao longo do laço principal, há a verificação dos sinais emitidos

ao pressionar os botões do joystick. Caso algum dos botões forem pressionados, o laço principal irá entrar nas condicionais de acordo com as funções atribuídas da figura 16. Se o botão que ativa o modo autônomo for pressionado, é chamada a sub-rotina de predição da rede neural, o qual recebe a imagem como parâmetro e a normaliza. O grau de esterço é predizido pela rede neural segundo seu modelo, e então convertido para uma mensagem de comando.

Por fim, após a mensagem de comando for obtida, seja pela predição da rede neural ou por comando do usuário, ela é codificada e enviada através do socket de comando, e assim o cliente recebe seu respectivo comando e atua nos controles físicos.

4.4.3 Rede Neural

Uma rede neural convolucional é usada para gerar ângulos de direção, baseado no *streaming* de vídeo alimentado pela frente do carro. Para que isso funcione, primeiro a rede precisa ser treinada. O usuário deve dirigir manualmente o carro e "mostrar"a rede neural como o carro deveria se comportar ao redor da pista. Isso inclui imagens do que o carro "vê", bem como a entrada que o usuário está dando nessas condições.

Os dados obtidos são imagens capturadas e salvas a partir do streaming de vídeo. Cada *frame* salvo possui um ângulo associado na nomenclatura da imagem, relacionando o que o carro vê, com o ângulo de esterço inserido pela pessoa que controla o carro.

Uma vez que a CNN tenha recebido esses dados, ela é treinada em um modelo de como o carro deve ser conduzido ao redor da pista. Se a pista for muito alterada ou as condições de iluminação se tornarem drasticamente diferentes de quando a CNN foi originalmente treinada, pode ser necessário treinar novamente para as novas condições. Na implementação atual do projeto, apenas os ângulos de direção são gerados.

Uma vez obtido o modelo treinado utilizando aprendizado supervisionado, o carro pode ser executado no modo autônomo. Quando configurado para o modo autônomo, o servidor pega a imagem de entrada transmitida do Raspberry Pi e processa-a usando a rede neural convolucional, tendo como saída um ângulo de direção apropriado.

A pista deve ter um bom contraste entre as áreas dirigíveis. O cenário de teste foi construído sobre uma superfície branca fosca, e o traçado foi feito com papel e fita adesiva, ambos de cor preto fosco,

com o intuito de evitar qualquer reflexo de luz que possa invalidar o treino da rede neural.

4.4.3.1 Arquitetura da Rede Neural Convolucional

A arquitetura da CNN foi baseada no artigo de (BOJARSKI et al., 2016). São treinados os pesos da rede para minimizar o erro quadrático médio entre a saída do comando de direção pela rede. A arquitetura de rede é mostrada na Figura 18. A rede consiste em 9 camadas, incluindo uma camada de normalização, 5 camadas convolucionais e 3 camadas totalmente conectadas.

A primeira camada da rede realiza a normalização da imagem. O normalizador é hard-coded e não é ajustado no processo de aprendizagem. As camadas convolucionais foram projetadas para realizar a extração de características e foram escolhidas empiricamente através de uma série de experiências que variavam as configurações das camadas.

Foram usadas convoluções em passos nas três primeiras camadas convolucionais com uma passada de 2×2 e um núcleo de 5×5 e uma convolução passada com um tamanho de kernel 3×3 nas duas últimas camadas convolucionais. Também há cinco camadas convolucionais com três camadas totalmente conectadas, levando a um valor de controle de saída valor é o raio de giro. O diagrama da arquitetura é apresentado na figura 18:

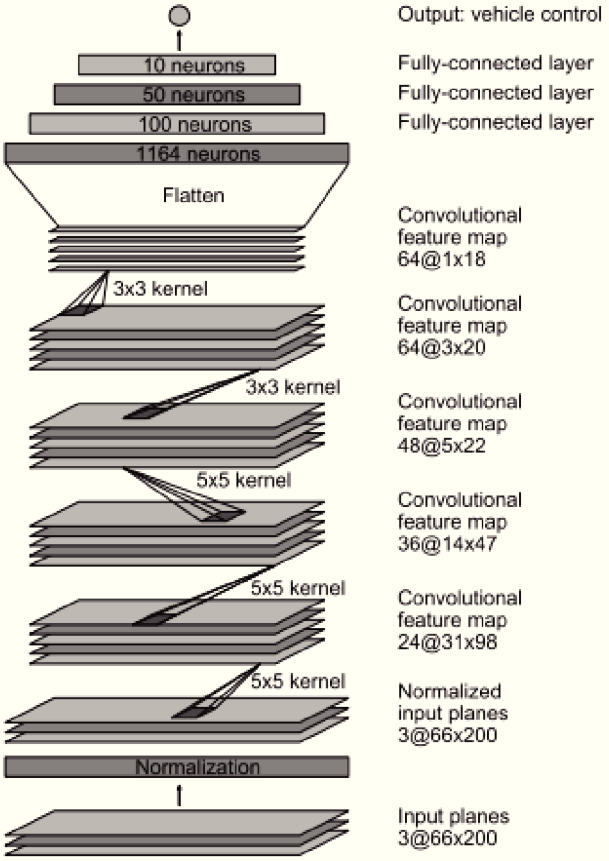


Figura 18 Arquitetura da rede neural.
Fonte: (BOJARSKI, Mariusz et al. 2016).

4.4.3.2 Implementação da CNN

Neste capítulo, irá ser descrito em detalhes o que cada parte do modelo implementado faz. Os comentários serão feitos linha-por-linha ou em blocos de código.

No aprendizado de máquina, a função que estamos tentando resolver é tipicamente representada como $Wx + b = y$, onde dado x (a lista de imagens de entrada) e y (a lista de instruções de direção cor-

respondentes), e queremos encontrar o melhor combinação de W e b para fazer o equilíbrio da equação.

Algorithm 6 Inicialização de variáveis peso e viés.

Require: $tensorflow$

```
peso ← truncated_normal(shape, stddev=0.1)
vies ← constant(0.1, shape)
```

W e b não são números únicos, mas sim coleções de coeficientes. Essas coleções são multidimensionais e o tamanho dessas coleções correspondem ao número de nós na rede de aprendizado de máquina. Portanto, no código acima, o objeto *peso* representa W e o objeto *vies* representa b, no sentido generalizado.

Esses objetos W e b são iniciados com uma função chamada normal truncada. Isso significa que quando uma coleção de *W's* e *b's* são criados inicialmente, os valores dos coeficientes individuais devem ser atribuídos aleatoriamente com base na distribuição normal (ou seja, uma curva em forma de sino) com um desvio padrão de 0,1. O desvio padrão dita o quão aleatórios queremos que os coeficientes iniciais sejam.

Algorithm 7 Inicialização de variáveis imagem e angulo.

Require: $tensorflow$

```
x ← placeholder(float32, shape = [None, 66, 200, 3]))
y ← placeholder(float32, shape = [None, 1])
```

Neste trecho são definidos espaços reservados para as variáveis x e y. Esses espaços reservados definem as dimensões das variáveis (essas variáveis representam uma coleção de valores, não apenas um único número). A variável x é configurada para receber uma imagem de certas dimensões, e y é configurado para esperar um único número como uma saída (ou seja, o ângulo de direção).

Algorithm 8 Camada convolucional.

Require: $tensorflow$

```
conv ← relu(conv2d(x, peso) + vies)
```

Acima se encontra a primeira camada convolucional. Em *conv* é descrito o objeto intermediário que aplica a função de convolução às

entradas x e *peso*, somado a *vies* à saída da convolução. Em seguida, são processados por meio da função de ativação ReLU.

A convolução 2D(conv2D) é calculada de forma semelhante a convolução de 1 dimensão: se desliza seu *kernel* sobre a entrada, calcula as multiplicações de elementos e as soma. Porém ao invés do *kernel* ser um vetor, neste caso são matrizes. Essa função está manipulando a imagem para realçar recursos distintos que são mais propícios ao treinamento do modelo.

Há quatro camadas convolucionais, que funcionam exatamente da mesma maneira que a primeira camada, mas em vez de usar x como uma entrada, elas usam a saída da camada anterior. Em uma Rede Neural Convolucional, no topo da pilha do modelo, existem camadas convolucionais que aprendem quais operações de convolução, ou seja, manipulações de imagem que destacam os melhores recursos para aprender.

Algorithm 9 Camadas convolucionais.

Require: $tensorflow$

$peso_1 \leftarrow truncated\_normal([5, 5, 3, 24])$
$vies_1 \leftarrow constant(shape = [24])$
$conv_1 \leftarrow relu(conv2d(x, peso_1, 2) + vies_1)$

$peso_2 \leftarrow truncated\_normal([5, 5, 24, 36])$
$vies_2 \leftarrow constant(shape = [36])$
$conv_2 \leftarrow relu(conv2d(conv_1, peso_2, 2) + vies_2)$

$peso_3 \leftarrow truncated\_normal([5, 5, 36, 48])$
$vies_3 \leftarrow constant(shape = [48])$
$conv_3 \leftarrow relu(conv2d(conv_2, peso_3, 2) + vies_3)$

$peso_4 \leftarrow truncated\_normal([3, 3, 48, 64])$
$vies_4 \leftarrow constant(shape = [64])$
$conv_4 \leftarrow relu(conv2d(conv_3, peso_4, 2) + vies_4)$

$peso_5 \leftarrow truncated\_normal([3, 3, 64, 64])$
$vies_5 \leftarrow constant(shape = [64])$
$conv_5 \leftarrow relu(conv2d(conv_4, peso_5, 2) + vies_5)$

No próximo passo, há as camadas totalmente conectadas que tentam aprender como produzir a saída correta com base nos recursos realçados pelas camadas convolucionais. Em sua execução há a redução de resolução e retificadores que aceleram o processo de aprendizagem.

Essencialmente, as camadas convolucionais estão fornecendo um espaço de características significativo, de baixa dimensão e invariante,

e a camada totalmente conectada está aprendendo uma função possivelmente linear nesse espaço. A saída das camadas convolucionais representa recursos de alto nível nos dados. Por fim, após os passos totalmente conectados a camada de saída da retorna um único valor contínuo, o qual é o ângulo de esterço.

Algorithm 10 Camadas totalmente conectadas.

1: $conv_5 \leftarrow tf.reshape(h_conv5, [-1, 1152])$

2: $peso_1fc \leftarrow truncated\_normal([1152, 1164])$
3: $vies_1fc \leftarrow constant(shape = [1164])$
4: $fc_1 \leftarrow relu(tf.matmul(conv_5, peso_1fc) + vies_1fc)$
5: $fc_1drop \leftarrow dropout(fc_1, prob)$

6: $peso_2fc \leftarrow truncated\_normal([1164, 100]])$
7: $vies_2fc \leftarrow constant(shape = [50])$
8: $fc_2 \leftarrow relu(tf.matmul(fc_1drop, peso_2fc) + vies_2fc)$
9: $fc_2drop \leftarrow dropout(fc_2, prob)$

10: $peso_3fc \leftarrow truncated\_normal([100, 50])$
11: $vies_3fc \leftarrow constant(shape = [50])$
12: $fc_3 \leftarrow relu(tf.matmul(fc_2drop, peso_3fc) + vies_3fc)$
13: $fc_3drop \leftarrow dropout(fc_3, prob)$

14: $peso_4fc \leftarrow truncated\_normal([50, 10])$
15: $vies_4fc \leftarrow constant(shape = [50])$
16: $fc_4 \leftarrow relu(tf.matmul(fc_3drop, peso_4fc) + vies_4fc)$
17: $fc_4drop \leftarrow dropout(fc_4, prob)$

18: $peso_5fc \leftarrow truncated\_normal([10, 1])$
19: $vies_5fc \leftarrow constant(shape = [1])$
20: $y \leftarrow tf.matmul(fc_4drop, peso_5fc) + vies_5fc$

No item 4 no artigo da NVIDIA, que especifica o sistema de aprendizado profundo que está sendo usado, é descrito que as camadas convolucionais foram projetadas para executar a extração de recursos e escolhidas empiricamente por meio de uma série de experimentos que variavam as configurações das camadas (BOJARSKI et al., 2016).

4.4.3.3 Algoritmo de Otimização e Função Perda

No contexto de Redes Neurais, otimização é o processo de encontrar o conjunto de parâmetros peso que minimiza a função de perda. O objetivo da otimização é encontrar a melhor solução global de modelos,

possivelmente não lineares, na presença de vários ótimos locais. Este trabalho só irá focar no algoritmo otimizador Adam(*Adam Optimizer, em inglês*).

Algorithm 11 Otimização com Erro Quadrático Médio e otimizador Adam.

Require: $tensorflow$

1: $subtracao \leftarrow subtract(model.y_{pred}, model.y)$ ▷ valor predizido menos valor real
2: $quadrados \leftarrow square(subtacao)$
3: $perda \leftarrow reduce\_mean(quadrados)$
4: $treinamento \leftarrow AdamOptimizer.minimize(perda)$

Adam é um método para otimização estocástica eficiente que requer apenas gradientes de primeira ordem com pouca necessidade de memória. O método calcula as taxas individuais de aprendizagem adaptativa para diferentes parâmetros de estimativas de primeiro e segundo momentos dos gradientes; o nome Adam é derivado da estimativa de momento adaptativo (KINGMA; BA, 2014).

Em suma, o algoritmo de otimização Adam é um otimizador com parâmetros de aprendizados adaptativos. Em (BELLO et al., 2017), é mostrada a comparação entre o Algoritmo Descendente Estocástico, Adam, RMSProp, Momentum. O estudo revela que esses quatro otimizadores superam os outros algoritmos otimizadores por uma grande margem.

4.4.4 Região de Interesse

Com o intuito de alimentar a rede neural somente com imagens que possuam características relevantes, foi utilizada a técnica de extração da região de interesse, a qual neste caso são somentes imagens da pista o qual o mini carro navega. Devido a posição da câmera se localizar próxima a superfície, foi atribuída empiricamente como a região de interesse a metade inferior da imagem capturada. Por conseguinte, as imagens salvas durante a captura de imagem e no passo que antecede a chamada da função de predição da rede neural, é utilizada a funcionalidade da biblioteca OpenCV para extração da região desejada.

5 EXPERIMENTOS

Antes que o carro possa ser executado em modo autônomo, ele precisa ser treinado manualmente. A obtenção dos dados de treinamento ocorre da seguinte forma:

- O vídeo é capturado pelo Raspberry Pi usando o módulo da câmera que é montado na parte superior do carro;
- O vídeo é transmitido através da rede sem fio para o servidor Desktop;
- Um joystick de Xbox conectado ao servidor é usado para controlar manualmente o carro;
- Cada quadro de vídeo recebido é armazenado no servidor, junto com a entrada do usuário correspondente (entrada de direção e entrada de aceleração).

5.1 BANCO DE IMAGENS

O usuário deve dirigir manualmente o carro ao redor do circuito de treinamento (figura 19) várias vezes para coletar imagens suficientes e cobrir as diferentes condições que o carro pode encontrar. Os dados de treinamento foram coletados controlando manualmente o carro em um circuito com ampla variedade de curvas de diferentes ângulos, o qual foi obtido uma amostra de aproximadamente 15000 imagens em escala de cinza para o treinamento da Rede Neural.

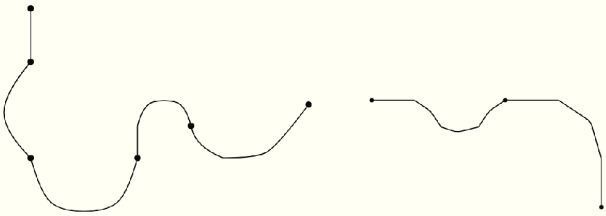


Figura 19 Diagrama de circuitos de treinamento.

Para treinar uma rede neural é necessário selecionar os quadros a serem usados. Os dados coletados são rotulados com o número do

quadro capturado e atividade do motorista (permanecendo em uma pista, alternando faixas, voltas e assim por diante). Para treinar uma CNN para seguir a pista, são selecionados apenas os dados em que o motorista estava em circunstâncias de pilotagem(dentro da pista, entre as faixas delimitadoras da estrada). Então, as imagens capturadas foram selecionadas em uma taxa de descarte de 3:5, ou seja, a cada 5 imagens processadas, 3 são descartadas.

Figura 20 Imagem capturada utilizada para treinamento.

Uma maior taxa de amostragem resultaria em incluir imagens que são altamente semelhantes, as quais não fornecem muita informação útil. Para remover a tendenciosidade de dirigir em linha reta, os dados de treinamento incluem uma proporção maior de quadros que representam curvas da estrada.

Para análise da qualidade das imagens capturadas, foram extraídas amostras de diferentes momentos da captura no circuito de testes e usando-as como de estímulo das camadas convolucionais da rede neural. A figura 21 exemplifica este passo do experimento.

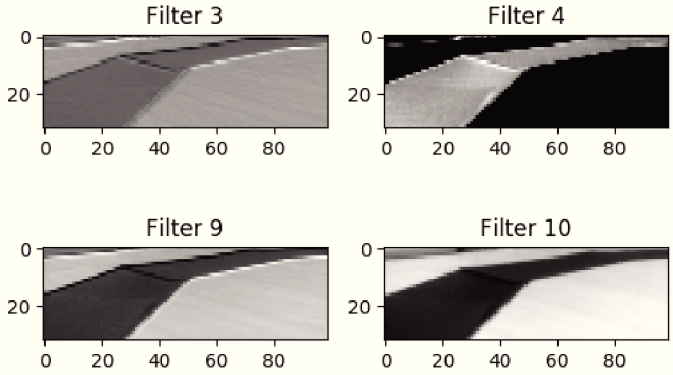


Figura 21 Exemplo de ativação da primeira camada convolucional.

5.1.1 Treinamento da CNN

Após a coleta de dados, procede a etapa treinamento da Rede Neural Convolucional. Nesta etapa foram utilizadas 10000 imagens selecionadas aleatoriamente do banco de imagens para treinamento, e 2000 imagens são usadas para avaliação do erro associado ao modelo treinado.

5.2 MÉTRICAS DO SISTEMA

Com o intuito de analisar a implementação do cliente e servidor, foram realizados índices de referências para obter o impacto do tempo de resposta do trânsito de dados entre dispositivos em diferentes circunstâncias, como por exemplo o tempo entre o envio de uma imagem até o comando para ativar os sinais dos motores.

Por conseguinte, as avaliações dos tempos de tramitação de dados resumem-se nas seguintes metodologias:

- Avaliação entre envio da imagem do cliente para o servidor, até sinal de retorno atuar nos sinais dos motores;
- Tempo de execução do fluxo inteiro no servidor, desde o recebimento da imagem até o envio do sinal de retorno, com e sem avaliação da imagem pela rede neural;
- Tempo de avaliação da imagem pela Rede Neural;
- Taxa de quadros por segundos recebidos.

Os resultados obtidos regem os seguintes parâmetros:

- Tempo é mensurado em milissegundos;
- A captura de dados de tempo é realizada por dez minutos contínuos, para cada avaliação;
- Cada avaliação tem iteração de duas vezes;
- A distância entre cliente, servidor e modem de rede sem fio é um raio de dois metros;
- A latência de rede é menor que 5 milissegundos com o uso exclusivo do cliente e servidor;

- O tempo médio é calculado pela média aritmética dos valores obtidos.

Segue na tabela abaixo os resultados obtidos:

Tabela 6 – Resultados de tempo de resposta do sistema.

Avaliação	Tempo(ms)
Envio da imagem e sinal de retorno	104.90
Tempo de execução do servidor sem RN	33.98
Tempo de execução do servidor com RN	40.64
Tempo da Rede Neural	5.74
Quadros por segundo	21.30

A partir dos dados obtidos, a média do processamento de todo fluxo de dados é de 104.90 milissegundos. Utilizando a distribuição normal com nível de significância de 5%, o processamento do sinal tem 95% de chances do tempo estar entre 92 e 117 milissegundos, segundo o intervalo de confiança.

5.3 EXPERIMENTOS PRÁTICOS

A fim de obter resultados do comportamento da condução autônoma do mini carro, foi utilizada a seguinte metodologia:

- Montagem de um circuito de testes;
- Criação de diferentes versões de Redes Neurais Convolucionais utilizando a arquitetura proposta em 4.4.3.1, e aplicação em testes práticos;
- Avaliação de desempenho autônomo;
- Avaliação de trajetória;

Cada etapa da metodologia aplicada será relatada em detalhes nos capítulos a seguir.

5.3.1 Circuito de Testes

Para garantir uma maior abrangência de casos da avaliação da Rede Neural, foi montado em um espaço físico um circuito de testes. A pista criada teria que possuir diferentes ângulos de entrada de curva em ambas direções, trechos em linha reta para avaliar a capacidade do mini carro se manter no centro da pista e diferentes larguras de faixas. É necessário também que o circuito de teste se encontre em um ambiente que possibilite diferentes variações de luminosidade. A figura 22 representa a pista utilizada para os experimentos de condução autônoma.

Figura 22 – Circuito de teste.

5.3.2 Modelos de Rede Neural

Foram criados diversos modelos de redes neurais, dos quais foram selecionados os que tiveram melhor desempenho na pista de testes. A palavra modelo, no contexto do experimento, se refere às versões da Rede Neural, em que as diferenças dos modelos estão nos parâmetros de

treinamento, especificamente o número de lotes(número de amostras de treinamento) e nas épocas(um *forward* e *backward pass* da amostra de treinamento). A arquitetura da Rede Neural utilizada no experimento é a mesma citada no capítulo 4.4.3.1. A tabela 7 apresenta os resultados obtidos, em função da precisão resultante:

Tabela 7 Modelos de Redes Neurais Criados.

Modelo	Épocas	Lotes	Erro(%)
1	50	128	4.19
2	30	256	5.50
3	30	512	5.87
4	50	64	2.28

Todos modelos obtidos foram testados na prática com a finalidade de avaliar a diferença do comportamento do carro com modelos que possuem variação de precisão. Os gráficos da função erro dos modelos com melhores desempenho são mostrados abaixo:

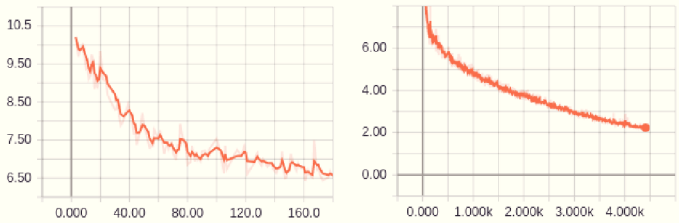


Figura 23 Função erro do modelo 1 e modelo 4.

5.3.3 Simulação

O passo antecedente aos testes práticos foi a simulação dos modelos de rede neural criados, com o objetivo de analisar o possível comportamento do carro associando os ângulos de esterço preditos e reais(inserido pelo usuário durante a captura de imagens). O resultado da simulação, mostrado na figura 24, utilizou um conjunto de 3600 imagens que não foram utilizadas durante o treinamento.

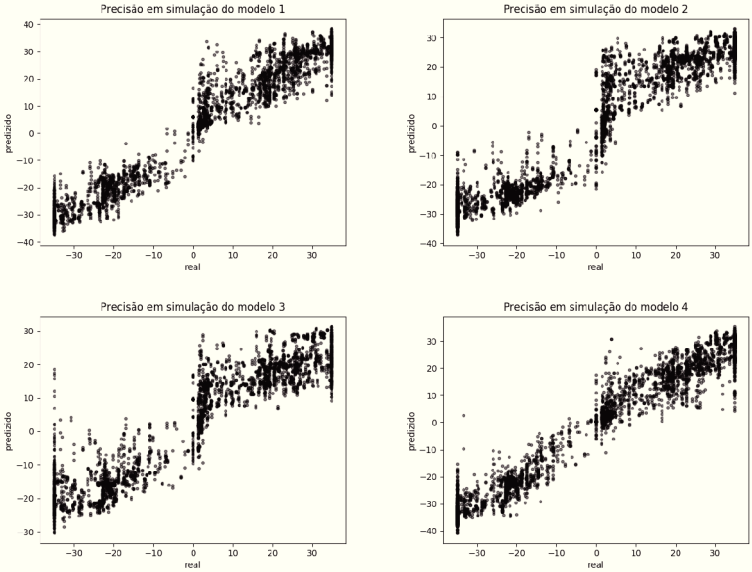


Figura 24 Comparação dos ângulos reais e preditos.

Nos gráficos é possível notar a maior esparcidade dos ângulos no modelo de menor precisão, o qual possívelmente terá maiores dificuldades de manter a trajetória na pista de testes real. A tabela 8 mostra a média dos ângulos(em graus) e o desvio padrão de cada modelo.

Tabela 8 Média e desvio padrão dos modelos.

	Modelo 1	Modelo 2	Modelo 3	Modelo 4
Média	4.24	5.82	6.43	4.22
Desvio padrão	0.56	1.35	1.42	0.23

5.3.4 Avaliação de Capacidade Autônoma

Segundo (ZANCHIN et al., 2017), a definição de um carro autônomo é um veículo com a capacidade de sentir o ambiente em que está inserido e navegar através deste ambiente sem a ajuda ou intervenção de um ser humano. Com o intuito de avaliar a capacidade autônoma do mini carro, foram atribuídas os seguintes requisitos ao qual o carro deve comportar:

- Capacidade de se manter na trajetória;
- Capacidade de fazer curvas;
- Capacidade de correção de trajetória;

Foram analisados também o número de vezes que o carro colidiu com a borda da pista(erros), e se houve perda de controle e saiu do circuito. O mini carro teve como parâmetro de teste realizar o circuito duas vezes consecutivas no sentido horário, e duas vezes consecutivas no sentido anti-horário. O teste foi repetido em todas situações com luminosidade alta e baixa.

A influência da luminosidade refere-se ao comportamento do carro em ambas situações, e sua avaliação é em função dos erros. Segue abaixo a tabela que descreve os valores atribuídos a intensidade luminosa, onde n é o número de erros em condições de alta luminosidade:

Tabela 9 – Parâmetros de influência luminosa.

Influência	Baixa	Média	Alta
Erros	n + 2	n + 5	n + 8

Os resultados dos experimentos práticos com o carro no circuito de testes seguem na tabela 10:

Tabela 10 – Desempenho do mini carro, sendo conduzido autonomamente.

Modelo	Erros (alta/baixa) luminosidade	Corrigiu trajetória	Saiu do circuito	Influência luminosa
1	10 alta / 3 baixa	Sim	Sim	Alta
2	6 baixa / –	Não	Sim	Alta
3	– / –	Não	Sim	Alta
4	5 alta / 1 baixa	Sim	Não	Média

Após a realização dos testes, o modelo 4 teve seu desempenho autônomo conferido. O mini carro foi controlado pela Rede Neural autonomamente no circuito de testes até a carga de suas bateria esgotarem. Em nenhum momento o mini carro saiu da pista, e realizou as correções de trajetórias para se manter no centro da pista.

5.3.5 Avaliação de Trajetória

Foram selecionados dois modelos de redes neurais apresentados no capítulo anterior para este experimento. Com o intuito de avaliar o impacto do erro do modelo de rede neural no mini carro autônomo e seu respectivo comportamento, um novo circuito de teste foi montado. O novo circuito contém somente três curvas em direções opostas. A avaliação da capacidade de adaptação da trajetória foi realizada seguindo os seguintes parâmetros:

- Ambos modelos são conduzidos em modo autônomo, partindo do mesmo ponto inicial;
- O circuito de teste de trajetória está sob condições de alta luminosidade;
- São capturadas e gravadas as imagens da câmera e o ângulo de esterço do carro ao longo de cada iteração do experimento;
- O trajeto percorrido pelo carro é obtido através de marcadores adaptados ao chassi do carro.

Após a captura de imagens, foram obtidos os seguintes resultados:

Tabela 11 – Média e desvio padrão dos ângulos obtidos.

	Modelo 1	Modelo 4
Média	9.23	8.20
Desvio padrão	0.78	0.34

Através das imagens capturadas e seus respectivos ângulos, além da trajetória marcada na superfície, foi possível reconstruir o caminho percorrido pelo carro. O diagrama abaixo(figura 25) compara o comportamento da correção de trajetória de ambos modelos.

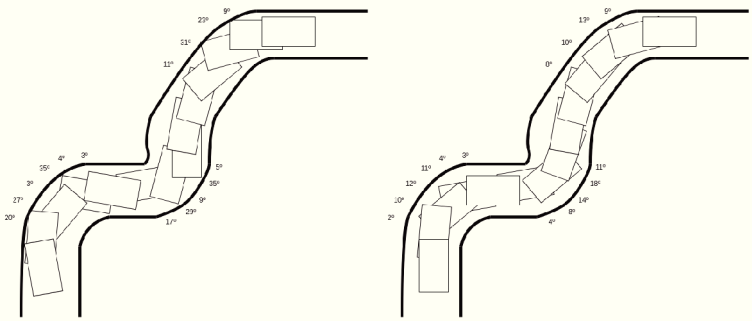


Figura 25 Trajetória do modelo 1 e modelo 4, respectivamente.

Em cada trecho de curva no trajeto, foram marcados os ângulos atuados pela rede neural. Devido ao fato do projeto consistir em um objeto cinemático, qualquer ângulo de esterço mal predizido acarretará em maiores correções de trajetória para se manter na pista. É possível observar que o modelo 1, que possui menor precisão neste experimento, é capaz de manter a trajetória realizando uma série de correções bruscas, ao contrário do modelo 4, que se mantém na pista executando movimentos sutis.

6 CONCLUSÃO

O presente trabalho buscou analisar o funcionamento de redes neurais convolucionais na prática, sobre um tópico bastante comentado e controverso atualmente, a condução autônoma de um veículo.

Em relação aos trabalhos relacionados, com o passar dos anos houve uma imensa evolução em relação que diz respeito a precisão das redes neurais convolucionais para dirigibilidade autônoma, alcançando um comportamento de condução muito similar a humana.

Ao longo da implementação do trabalho, foi evidente o quão efetivo foi o uso de redes neurais convolucionais com a ferramenta Tensorflow para atingir o objetivo do trabalho. Levando em consideração o quão prático foi este projeto, foi possível relatar a magnitude do impacto dos parâmetros de treinamento e modelos com diferentes precisões na condução autônoma, assim como as variações de ambiente podem afetar drasticamente a precisão da predição da rede neural.

Por fim, a partir da arquitetura de rede neural implementada, foi possível obter excelentes resultados utilizando um hardware de baixo custo para simular um veículo real, confirmando assim os resultados obtidos do artigo da NVIDIA.

6.1 TRABALHOS FUTUROS

O maior desafio deste projeto foi criar um sistema que tivesse capacidade de processamento satisfatório para conduzir o mini carro autonomamente. Neste contexto, podem ser aplicadas outras modelagens de implementação diferente do proposto por este trabalho. O artigo de (BECHTEL et al., 2018) oferece uma abordagem diferente, utilizando o processamento do próprio chip Raspberry Pi para o processamento de imagens e predição da rede neural.

Outra sugestão de implementação é a detecção de objetos. O sistema implementado não possui este tipo de funcionalidade para condição de parada autônoma. Por conseguinte, é possível utilizar a mesma modelagem cliente-servidor, concentrando todo processamento no servidor, e implementar detecção de objetos. Pode-se tomar como base o artigo de (PRABHAKAR et al., 2017), o qual utiliza redes neurais convolucionais para detecção de objetos na estrada, assim como uma ferramenta de visão computacional, como a biblioteca OpenCV.

Além de visão computacional, veículos autônomos fazem o uso

de sensores ultrassônicos. Desse modo, é possível combinar o uso de ambos recursos de visão computacional e detecção de objetos móveis por meio de sensores, como mencionado no artigo de (CHO et al., 2014), que utiliza esta metodologia em ambientes urbanos.

REFERĘNCIAS

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. <https://www.tensorflow.org/>.

BECHTEL, M. G. et al. Deeppicar: A low-cost deep neural network-based autonomous car. In: IEEE. *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. [S.l.], 2018. p. 11–21.

BEIKER, S. A. Legal aspects of autonomous driving. *Santa Clara L. Rev.*, HeinOnline, v. 52, p. 1145, 2012.

BELLO, I. et al. Neural optimizer search with reinforcement learning. In: JMLR. ORG. *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. [S.l.], 2017. p. 459–468.

BOGDANOV, V. *IOT platforms overview*. 2017. <https://iot.intersog.com/blog/iot-platforms-overview-arduino-raspberry-pi-intel-galileo-and-others/>. Acessado em 2019-05-30.

BOJARSKI, M. et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

CHEN, Z.; HUANG, X. End-to-end learning for lane keeping of self-driving cars. In: IEEE. *2017 IEEE Intelligent Vehicles Symposium (IV)*. [S.l.], 2017. p. 1856–1860.

CHO, H. et al. A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In: IEEE. *2014 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.], 2014. p. 1836–1843.

DUGULEANA, M.; MOGAN, G. Neural networks based reinforcement learning for mobile robots obstacle avoidance. *Expert Systems with Applications*, Elsevier, v. 62, p. 104–115, 2016.

FISCHER, P.; DOSOVITSKIY, A.; BROX, T. Image orientation estimation with convolutional networks. In: SPRINGER. *German Conference on Pattern Recognition*. [S.l.], 2015. p. 368–378.

FRIDMAN, L. et al. Mit autonomous vehicle technology study: Large-scale deep learning based analysis of driver behavior and interaction with automation. *arXiv preprint arXiv:1711.06976*, 2017.

GENç Özgür. *Notes on Artificial Intelligence, Machine Learning and Deep Learning for curious people*. 2019. <https://towardsdatascience.com/notes-on-artificial-intelligence-ai-machine-learning-ml-and-deep-learning-dl-for-56e51a2071c2>. Acessado em 2019-05-30.

HALFACREE, G.; UPTON, E. *Raspberry Pi User Guide*. 1st. ed. [S.l.]: Wiley Publishing, 2012. ISBN 111846446X, 9781118464465.

HEYLEN, J. et al. From pixels to actions: Learning to drive a car with deep neural networks. In: IEEE. *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. [S.l.], 2018. p. 606–615.

KARN, U. *An Intuitive Explanation of Convolutional Neural Networks*. 2016. <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>. Acessado em 2019-05-29.

KARPATHY, A. *Stanford.edu Convolutional Neural Networks for Visual Recognition*. 2019. <http://cs231n.github.io/convolutional-networks/>. Acessado em 2019-05-12.

KIM, P. Matlab deep learning. In: *With Machine Learning, Neural Networks and Artificial Intelligence*. [S.l.]: Springer, 2017.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2012. p. 1097–1105.

LECUN, Y. et al. Handwritten digit recognition with a back-propagation network. In: *Advances in neural information processing systems*. [S.l.: s.n.], 1990. p. 396–404.

LITMAN, T. *Autonomous vehicle implementation predictions*. [S.l.]: Victoria Transport Policy Institute Victoria, Canada, 2017.

MEDINA-SANTIAGO, A. et al. Neural control system in obstacle avoidance in mobile robots using ultrasonic sensors. *Journal of applied research and technology*, Elsevier, v. 12, n. 1, p. 104–110, 2014.

NIELSEN, M. A. *Neural networks and deep learning*. [S.l.]: Determination press San Francisco, CA, USA:, 2015.

NILSSON, N. J. *Principles of artificial intelligence*. [S.l.]: Morgan Kaufmann, 2014.

OMRANE, H.; MASMOUDI, M. S.; MASMOUDI, M. Neural controller of autonomous driving mobile robot by an embedded camera. In: IEEE. *2018 4th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*. [S.l.], 2018. p. 1–5.

PANDEY, A.; PARHI, D. R. New algorithm for behaviour-based mobile robot navigation in cluttered environment using neural network architecture. *World Journal of Engineering*, Emerald Group Publishing Limited, v. 13, n. 2, p. 129–141, 2016.

PAYTON, D. An architecture for reflexive autonomous vehicle control. In: IEEE. *Proceedings. 1986 IEEE International Conference on Robotics and Automation*. [S.l.], 1986. v. 3, p. 1838–1845.

PEREIRA, C. A.; OLIVEIRA, S. M. M. d. *Detecção de pessoas em imagens, implementando técnicas de visão computacional em um Raspberry Pi*. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2016.

PRABHAKAR, G. et al. Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving. In: IEEE. *2017 IEEE Region 10 Symposium (TENSYMP)*. [S.l.], 2017. p. 1–6.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. *Learning internal representations by error propagation*. [S.l.], 1985.

SIMONYAN, K.; VEDALDI, A.; ZISSERMAN, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

VEDALDI, A.; LENC, K. Matconvnet: Convolutional neural networks for matlab. In: ACM. *Proceedings of the 23rd ACM international conference on Multimedia*. [S.l.], 2015. p. 689–692.

WEI, D. C. M. *Método de desvio de obstáculos aplicado em veículo autônomo.* Tese (Doutorado) — Universidade de São Paulo, 2015.

ZADEH, B. R. R. B. *TensorFlow for Deep Learning.* 2019. <https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html>. Acessado em 2019-05-30.

ZANCHIN, B. C. et al. On the instrumentation and classification of autonomous cars. In: IEEE. *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC).* [S.l.], 2017. p. 2631–2636.

APÊNDICE A – Aprendizado de Máquina

Segundo (NILSSON, 2014), o aprendizado de máquina (*machine learning*, em inglês) se refere às mudanças nos sistemas que executam tarefas associadas a inteligência artificial. Tais tarefas envolvem reconhecimento, diagnósticos, planejamento, controle de robôs, previsão, etc. Isso se reflete na técnica que analisa os dados e encontra um modelo por si só e é chamado de *learning*(Aprendizado). O processo se assemelha a um treinamento com dados (documentos, imagens) para resolver o problema de encontrar um modelo. Por conseguinte, os dados que o *Machine Learning* usa para processar um modelo são chamados de "dados de treinamento", um termo bastante usado neste trabalho.

As técnicas de Machine Learning podem ser classificadas em três tipos, dependendo do método de treinamento.

•Aprendizado supervisionado

•Aprendizado não supervisionado

•Aprendizado por reforço

O Aprendizado supervisionado é muito similar ao processo pelo qual humanos aprendem coisas. Considere que os humanos podem obter novos conhecimentos a medida que exercitam (resolvem) problemas. Por exemplo, a partir de um problema encontrado, o ser humano aplica o conhecimento que possui para resolver o problema, e compara com a solução. Caso a solução não esteja correta, o conhecimento é modificado, e estes passos são repetidos até que se aprenda a solução do problema. Quando aplicamos essa analogia ao processo de Machine Learning, o problema e a solução correspondem aos dados de treinamento, e o conhecimento corresponde ao modelo.

Da mesma forma, o aprendizado não supervisionado contém somente entrada de dados, sem solução para os dados correspondentes.

A.1 REDES NEURAIS

Os modelos do Machine Learning podem ser implementados de várias formas. Redes Neurais é uma delas. Na abordagem convencional de programação, dizemos ao computador o que fazer, quebrando grandes problemas em muitas pequenas tarefas precisamente definidas que o computador pode executar facilmente(NIELSEN, 2015, p. 23).

Por outro lado, em uma rede neural não informamos ao computador como resolver nosso problema. Ao invés disso, ele aprende a partir de dados observacionais, descobrindo sua própria solução para o problema em questão.

A.1.1 Nodos de uma Rede Neural

Sempre quando aprendemos algo, nosso cérebro guarda conhecimento, em contrapartida o computador usa memória para guardar a informação. Embora ambos guardem informações, seus mecanismos são completamente diferentes. O computador guarda informações em locais específicos da memória, enquanto o cérebro não possui capacidade de armazenamento, ele só transmite sinais de um neurônio para outro. O cérebro é uma rede gigantesca de neurônios, e a associação de neurônios forma uma informação específica (KIM, 2017, p. 20).

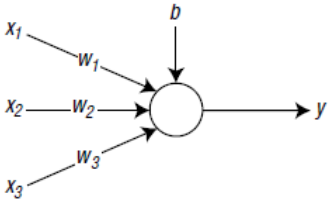


Figura 26 – Nodo de uma rede neural
Fonte: Adaptado de (NIELSEN, 2015)

A Rede Neural imita o mecanismo do cérebro. Como o cérebro é composto de conexões de inúmeros neurônios, a rede neural é construída com conexões de nodos, os quais são elementos que correspondem aos neurônios do cérebro. As Redes Neurais imitam a associação de neurônios, usando um valor "peso".

Na figura 26, o círculo e a flecha da figura denotam o fluxo do sinal, respectivamente $\boldsymbol{x_1}$, $\boldsymbol{x_2}$ e $\boldsymbol{x_3}$ são as entradas, $\boldsymbol{w_1}$, $\boldsymbol{w_2}$ e $\boldsymbol{w_3}$ são os pesos associados aos sinais correspondentes. Por fim, b é a tendência(ou viés) o qual é outro fator associado ao armazenamento de informação. O sinal de entrada é multiplicado pelo peso antes de alcançar o nodo, e uma vez que os sinais já pesados sejam coletados pelo nodo, esses valores são adicionados. A soma dos pesos deste exemplo é calculado na equação A.1:

$$\boldsymbol{y = (w_1 x_1) + (w_2 x_2) + (w_3 x_3) + b} \tag{A.1}$$

Essa equação indica que o sinal com um peso maior tem maior efeito. Por exemplo, se o peso $\boldsymbol{w_1}$ tem o valor igual a 1, e $\boldsymbol{w_2}$ tem o valor atribuído 5, o sinal $\boldsymbol{x_2}$ possui 5 vezes mais efeito que $\boldsymbol{x_1}$. Quando

$\boldsymbol{w_1}$ é zero, $\boldsymbol{x_1}$ não é transmitido ao neurônio.

A.1.2 Camadas de uma Rede Neural

Da mesma forma que o cérebro é uma gigantesca rede de neurônios, uma rede neural é uma rede de nodos. É possível criar uma variedade de redes neurais, dependendo de como os nodos são conectados. A rede neural mais comumente usada utiliza uma estrutura de nodos em camadas (KIM, 2017), como mostrado na figura 27.

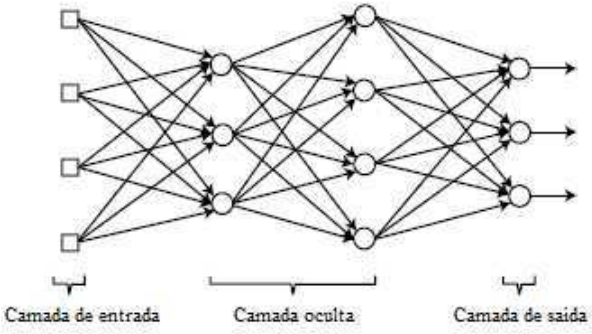


Figura 27 – Camadas de uma rede neural
Fonte: Adaptado de (NIELSEN, 2015)

O grupo de nodos de quadrados da figura 27 são chamados de nodos de entrada, e agem como uma passagem que transmite os sinais para os nodos seguintes, portanto não calculam a função de peso e ativação. Os nodos intermediários são chamados de nodos ocultos, e são nomeados de tal forma devido ao fato de não serem acessíveis do lado de fora da rede neural.

Quando nodos ocultos são adicionados a uma rede neural, é produzida uma rede neural multicamadas. Por conseguinte, uma rede neural multicamadas consiste em uma camada de entrada, múltiplas camadas ocultas, e uma camada de saída.

Em uma rede neural com camadas, o sinal entra na camada de entrada, é processado pelas camadas ocultas, e sai pela camada de saída. Durante esse processo, os nodos de uma camada recebem o sinal simultaneamente e enviam o sinal processado para a próxima camada ao mesmo tempo.

Segue abaixo o exemplo adaptado de (KIM, 2017), considerando o diagrama de uma rede neural com uma única camada oculta, representado pela figura 28.

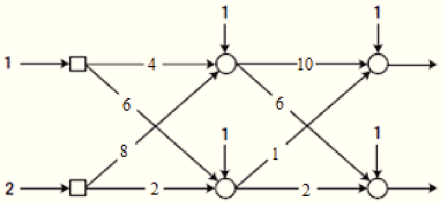


Figura 28 Rede neural com uma única camada oculta.
Fonte: Adaptado de (KIM, 2017)

Por conveniência, será utilizada a função de ativação linear apresentada na figura 29, a qual permitirá que os nodos transmitam a soma dos pesos.

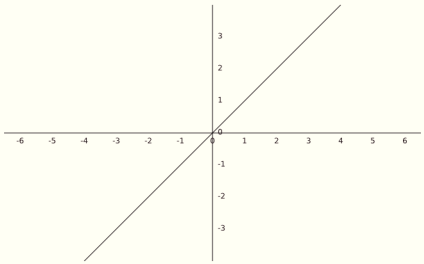


Figura 29 A função de ativação é linear.

Para calcular a saída da camada oculta, representada na figura 30, não há necessidade de calcular os nodos de entrada, já que só transmitem o sinal:

O primeiro nodo da camada oculta calcula sua saída como:

Soma:

$$y = (4 \times 1) + (8 \times 2) + 1 = 2 \tag{A.2}$$

Saída:

$$\phi(y) = y = 21 \tag{A.3}$$

Da mesma forma, o segundo nodo da camada oculta calcula a saída:

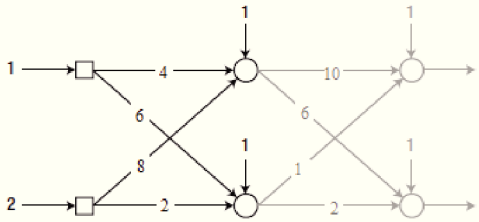


Figura 30 Exemplificação da sáida da camada oculta.
Fonte: Adaptado de (KIM, 2017)

Soma:

$$y = (2 * 8) + (2 * 2) + 1 = 21 \tag{A.4}$$

Saída:

$$\phi(y) = v = 21 \tag{A.5}$$

Então, o cálculo da soma dos pesos são combinados em uma matriz:

$$v = \begin{pmatrix} 4 \times 1 + 8 \times 1 + 1 \\ 2 \times 8 + 2 \times 2 + 1 \end{pmatrix} = \begin{pmatrix} 4 \times 8 \\ 6 \times 2 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 21 \\ 21 \end{pmatrix} \tag{A.6}$$

Os pesos do primeiro nodo da camada oculta estão na primeira linha da matriz resultante, e os pesos do segundo nodo estão na segunda linha. Esse resultado pode ser generalizado na seguinte equação:

$$y = Wx + b \tag{A.7}$$

Onde x é o vetor sinal de entrada e b o vetor de tendência. Na figura 31, podemos determinar as saídas da camada de saída, cujo procedimento é idêntico a camada previamente apresentada.

Soma dos pesos:

$$v = \begin{pmatrix} 10 & 1 \\ 6 & 2 \end{pmatrix} \begin{pmatrix} 21 \\ 21 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 232 \\ 68 \end{pmatrix} \tag{A.8}$$

Saída:

$$saida = \phi(y) = y = \begin{pmatrix} 232 \\ 68 \end{pmatrix} \tag{A.9}$$

Por fim, este projeto trabalhará com sub-redes que respondem

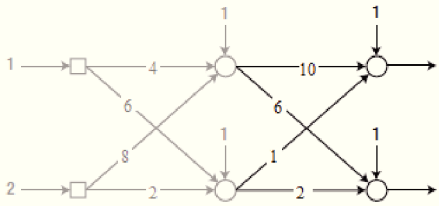


Figura 31 Saída da camada oculta.
Fonte: Adaptado de (KIM, 2017)

a perguntas que podem facilmente ser respondidas ao nível de pixels únicos. Essas perguntas podem, por exemplo, ser sobre a presença ou ausência de formas muito simples em pontos específicos da imagem, os quais podem ser respondidas por neurônios individuais conectados aos pixels brutos da imagem.

O resultado final é uma rede que quebra uma pergunta muito complicada, isto é, se uma imagem mostra um rosto ou não, em perguntas muito simples respondíveis a nível de pixels individuais.

Isto é feito através de redes neurais que possuem muitas camadas, com as primeiras camadas respondendo de forma muito simples a perguntas específicas sobre a imagem de entrada, e camadas posteriores construindo uma hierarquia de conceitos mais complexos e abstratos.

Redes com este tipo de estrutura de muitas camadas - duas ou mais camadas ocultas - são chamadas de redes neurais profundas (NIELSEN, 2015).

A.1.3 *Backpropagation*

O algoritmo *backpropagation*, ou retropropagação, foi originalmente introduzido em meados de 1970, mas sua importância não foi totalmente apreciada até um famoso artigo de 1986 de David Rumelhart, Geoffrey Hinton, e Ronald Williams. Esse artigo descreve várias redes neurais em que a retropropagação funciona com muito mais eficiência do que as abordagens anteriores à aprendizagem, tornando possível a utilização de redes neurais para resolver problemas que antes eram insolúveis. Hoje, retropropagação é um dos principais o algoritmos de aprendizagem em redes neurais (NIELSEN, 2015).

O natureza geral da método de treinamento com retropropagação é alcançar o balanço entre a capacidade de responder corretamente a

entrada de dados que são utilizados para treinamento, e a habilidade de predizer a entrada de um dados que seja similar, mas não idêntico, ao que é utilizado no treinamento.

O treinamento de uma rede neural que possui retropropagação envolve dois estágios: o *forward pass*, que se refere ao processo de cálculo dos valores das camadas de saída dos dados de entrada, percorrendo todos os neurônios da primeira à última camada, e o *backward pass*, que é o processo de cálculo do gradiente da função perda usando o algoritmo gradiente descendente ou similar(RUMELHART; HINTON; WILLIAMS, 1985). A computação é feita da última camada para a primeira camada.

A.1.4 Aprendizado Supervisionado em uma Rede Neural

Como mencionado brevemente no capítulo A deste trabalho, há três modos de treinar uma rede neural, porém para o desenvolvimento deste projeto somente o aprendizado supervisionado será usado. De maneira geral, o aprendizado supervisionado precede da seguinte forma:

1. Os pesos são inicializados;
2. É obtida a entrada dos dados de treinamento, que é uma tupla de valores entrada, saída correta, o qual é inserida na rede neural. A partir da saída da rede, é obtido o resultado e calculado o erro em comparação com a saída correta;
3. Os pesos são ajustados para corrigir o erro da função perda;
4. Enquanto houver dados de treinamento, repetir passos 2 e 3.

O conceito de aprendizado supervisionado é ilustrado na figura 32, para entender claramente os passos descritos.

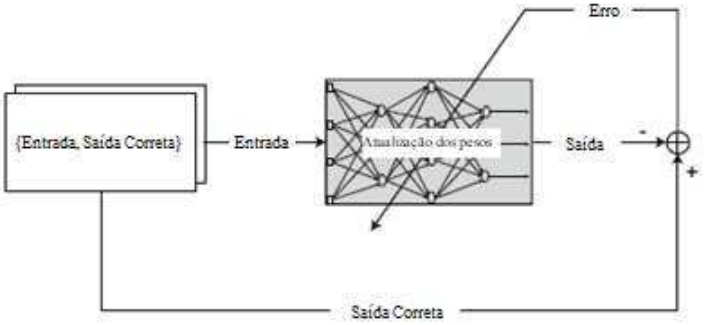


Figura 32 – Passos de aprendizado.
Fonte: Adaptado de (KIM, 2017)

APÊNDICE B – Fontes

```
modo_autonomo = 0
pausar_captura = 1
velocidade_autonoma = 1
porta_controle = 8000
porta_video = 8001
mensagem = "000000"
control_message = "000000"
enviar_comandos = 1
controle_manual = 1
def acao_joystick():

    left_thumb_x = 0
    left_trigger = 2
    right_trigger = 5
    while enviar_comandos == 1:
        pygame.event.pump()
        turn_angle = gamepad.get_axis(left_thumb_x)
        throttle = gamepad.get_axis(right_trigger)
        reverse_throtle = gamepad.get_axis(left_trigger)
    mensagem = "0000000"
def carregar_imagem(video_conn):

    tamanho_imagem = 0
    stream_image = 0
    while tamanho_imagem != 0:
        image_bytes = struct.unpack('<L',
        video_conn.read(struct.calcsize('<L')))[0]
        if not image_bytes:
            print("Erro")
        imagem_stream = io.BytesIO()
        imagem_stream.write(video_conn.read(image_bytes))
        imagem_stream.seek(0)
        stream_image = Image.open(imagem_stream)
        stream_image = array(stream_image)
        tamanho_imagem = stream_image.size
    stream_image = stream_image[:, :, ::-1]

    return stream_image

def config_conexao():
    server_socket = socket.socket()
```

```
    server_socket.setsockopt(socket.SOL_SOCKET,
    socket.SO_REUSEADDR, 1)
    server_socket.bind((host, porta_controle))
    server_socket.listen(0)
    command_client, addr = server_socket.accept()
    video_socket = socket.socket()
    video_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    video_socket.bind((host, porta_controle + 1))
    video_socket.listen(0)
    video_client = video_socket.accept()[0].makefile('rb')
    print("Video client connected!")
    return command_client, video_client

def predicao_rn(full_image):
    global smoothed_angle
    global sess
    global saver
    feed_image = scipy.misc.imresize(full_image[-150:],
                                                [66, 200]) / 255.0
    degrees = model.y.eval(
        session=sess,
        feed_dict={model.x: [feed_image], model.keep_prob: 1.0})[0]
        [0] * 180.0 / scipy.pi
    if degrees < 0:
        direction = "1"
    else:
        direction = "0"
    str_angle = str(int(abs(degrees*255/180)))
    str_angle = str_angle.zfill(3)
    ret_command = direction + str_angle + "011"
    return ret_command

def main():
    comando_conn, video_conn = config_conexao()
    Thread(target=acao_joystick).start()
    while enviar_comandos == 1:
        try:
            image = carregar_imagem(video_conn)
            if gamepad.get_button(0) == 1:
                pausar_captura = 1 - pausar_captura
                if pausar_captura == 1:
```

```
                comando_conn.send(controle)
            if pausar_captura == 0:
                height, width, color = image.shape
                roi = image[int(height / 2):height, :]
                cv2.imwrite('./captured_images/frame
                {:>010}_command-{}.jpg'.format(image_n, mensagem),
                image_n += 1
            if gamepad.get_button(2) == 1:
                modo_autonomo = 1 - modo_autonomo
                sleep(0.2)
            if gamepad.get_button(3) == 1:
                modo_autonomo = 0
                controle_manual = 1
                sleep(0.2)
            if modo_autonomo == 1:
                controle_manual = 0
                controle = predicao_rn(image)
            else:
                controle = mensagem
            comando_conn.send(controle.encode())
            if gamepad.get_button(1) == 1:
                comando_conn.send("000000".encode())
                cv2.destroyAllWindows()
                enviar_comandos = 0

        except:
            video_conn.close()
            sys.exit()
    video_conn.close()
```

APÊNDICE C – Artigo

Mini Carro Autônomo com Deep Learning

Bruno Manica

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brasil

bruno.manica@grad.ufsc.br

Abstract. *Vehicles are developing on mobile interconnected platforms. The reason why it is happening resides on political and economic determination that directs itself to a sustainable environment. Lane detection is one of the main duties on the perception system of autonomous vehicles. The approaches based on artificial intelligence are popular to perform this task due to the large amount of data we have currently, because of the broad use of devices and services connected to the vehicles. It was used on this project and approach on artificial intelligence called convolutional neural networks, with the intent to map the raw pixels from a front facing camera directly to steering commands in a mini car that simulates the characteristics of driving similar to production vehicles. With minimum human training data, the goal of this project is to create a system that learns to navigate on a miniature lane, simulating the behaviour of and commercial autonomous vehicle.*

Resumo. *Veículos estão se desenvolvendo em plataformas móveis interconectadas. A razão para que isso esteja acontecendo reside na determinação política e econômica que se direciona para o meio ambiente eco-sustentável As abordagens baseadas em inteligência artificial são populares para esta tarefa devido a enorme quantidade de dados que temos disponível atualmente, por conta do amplo uso de dispositivos e serviços conectados em veículos.Neste projeto foi utilizada uma técnica em inteligência artificial chamada Redes Neurais Convolucionais, com o intuito de mapear os pixels brutos de uma câmera frontal diretamente para os comandos de direção em um mini carro que simula características de pilotagem semelhantes a veículos de produção.*

1. Introdução

Nos últimos anos a sociedade mudou sua percepção a respeito de sistemas de transporte. Veículos eram referenciados como um meio de conveniência e status social, mas hoje os sistemas de transporte são uma fonte de preocupação com o grande número de acidentes, limitações ecológicas, o alto preço de combustível, entre outros. Governos, indústrias e a sociedade em geral estão se direcionando para os meios de transporte sustentáveis, para endereçar os problemas referidos [Beiker 2012].

Uma das principais missões de um veículo autônomo é executar um percurso de um ponto inicial até um ponto final com segurança e exatidão [Wei 2015]. Atualmente, os veículos autônomos estão restritos a operar em ambientes específicos, embora veículos possam operar autonomamente sob condições ideais, como condições meteorológicas favoráveis, há muitos problemas técnicos antes que possam operar em todas condições

[Litman 2017], sendo necessário a constante avaliação e aprimoramento de percepção de ambiente através de sensores. Segundo [Payton 1986], diferentes estratégias podem ser necessárias nessas situações para fazer o uso especializado de sensores, e realizar tarefas que são especializadas para as condições do ambiente.

2. Fundamentação Teórica

Neste capítulo será apresentado a fundamentação teórica que foi aplicada no desenvolvimento do projeto, para proporcionar um melhor entendimento sobre os experimentos na área de Redes Neurais.

Aprendizado profundo(*Deep Learning*, em inglês), é uma técnica de aprendizado de máquina que emprega redes neurais profundas. A rede neural profunda é uma rede neural multicamada que contém duas ou mais camadas ocultas. Embora isso possa ser simples, esta é a verdadeira essência do *Deep Learning*.

Convolutional Neural Networks, ou CNNs, são uma classe biologicamente inspirada de modelos de aprendizagem profunda, que explicitamente aproveita a estrutura espacial das imagens através de conectividade restrita entre camadas (filtros locais), compartilhamento de parâmetros (convoluções) e neurônios especiais de construção de invariância local [LeCun et al. 1990].

2.1. Classificação

Os dois tipos mais comuns de aplicação de aprendizado supervisionado são classificação e regressão. A classificação é a aplicação mais predominante em aprendizado de máquina. O problema de classificação se concentra em encontrar literalmente classes às quais os dados pertencem. Em contraste, a regressão não determina uma classe, ao invés disso estima um valor contínuo, como no caso deste trabalho, um ângulo.

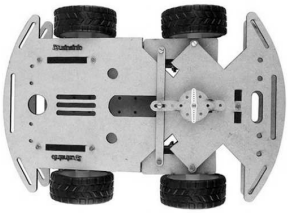
No artigo de [Heylen et al. 2018], que é relacionado a pilotagem autônoma com redes neurais profundas, foi utilizado o Erro Quadrático Médio para avaliar a função perda, isto é, a divergência entre os valores preditos e os valores de ângulos reais. O autor baseou-se sobre a métrica do Erro Quadrático Médio, uma vez que permite tomar a magnitude do erro em conta e atribuir uma maior perda para erros maiores do que outras métricas, como o Erro Absoluto.

3. Modelo Proposto

Este artigo tem como fundamentação principal o trabalho de [Bojarski et al. 2016] e [Chen and Huang 2017]. Ambos artigos fazem o uso da abordagem de ponta a ponta, utilizando redes neurais convolucionais para obter o ângulo de direção adequado para manter carro na pista. O modelo de rede neural convolucional (CNN) pega quadros de imagem bruta como entrada e produz a ângulos de direção.

O modelo implementado no artigo consiste em uma CNN de três camadas convolucionais e duas camadas totalmente conectadas. A camada de entrada é uma imagem em formato RGB e camada de saída é o ângulo de direção previsto para a imagem de entrada. A primeira camada convolucional usa um núcleo 9 x 9 e um passo de 4 x 4. As seguintes duas camadas convolucionais usam núcleo 5 x 5 e um passo 2 x 2.

As camadas convolucionais são principalmente para extração de recursos, e às camadas totalmente conectadas são principalmente para a previsão do ângulo de direção. As



camadas de dropout são usadas para evitar overfitting. E não há camadas de agrupamento porque os mapas de recursos são pequenos.

4. Ferramentas

Para desenvolver o mini carro, além da implementação da Rede Neural, foi necessário implementar os controles dos atuadores mecânicos do hardware, a criação de um sistema de controle manual do carro e a construção utilizando uma série componentes eletrônicos, os quais estão descritos a seguir:

- Um computador Desktop com processador Intel(R) core(TM) i5-2500 @ 3.3Ghz, com placa de vídeo dedicada ATI Radeon 6990 com 4GB de memória;
- Um Raspberry Pi 3 modelo B;
- Sistema Operacional Linux Ubuntu 18.04 instalado no Desktop;
- Sistema Operacional Raspbian(Unix) instalado no Raspberry Pi;
- Módulos de hardware para Raspberry Pi, como câmera, placa de circuito integrado, motores elétricos e baterias;
- Python 3.7 instalados em ambos dispositivos;
- Bibliotecas específicas para Python 3.7.

4.1. Componentes Eletrônicos

Para construção do carro autônomo, foi necessário um chassi específico de comportasse os equipamentos eletrônicos. O chassi é fabricado em material sintético e possui encaixes para variados tipos de motores DC do mercado, assim como suporte para encaixe de servo motor para controle de esterço do eixo dianteiro.

É no eixo dianteiro que se concentra ambos motores DC, os quais tracionam do carro, como também o servo motor. Os componentes foram alojados de tal forma que facilitasse a retirada de componentes defeituosos, como também para a distribuição homogênea de peso na superfície do carro, o qual se tornou um fator importante para o tracionamento do carro em superfícies escorregadias. A alimentação do mini carro é feita por duas bateria de lítio tipo 18650. Estas baterias são componentes de carros autônomos comerciais atuais, devida sua alta capacidade de gerar corrente. O Raspberry Pi é alimentado por uma bateria, que possui um carregador de bateria específico, o qual um dispositivo eletrônico desenvolvido especialmente para atuar em conjunto com baterias modelo 18650. Seu diferencial é a possibilidade de ser utilizado como um regulador de tensão com amplas possibilidades, pois fornece tensão regulada de 3V a 5V DC.

Por conseguinte, os motores são alimentados através de um circuito eletrônico, chamado Ponte H L298. O L298 é um circuito monolítico integrado que suporta alta

tensão, alta corrente, projetado para aceitar níveis lógicos TTL padrão e unidades de cargas indutivas, como relés, solenoides, DC e motores de passo. Este dispositivo é alimentado pela outra bateria 18650, e transmite a corrente para os motores de acordo com os sinais lógicos enviados para a Ponte H. A câmera é de simples e eficiente utilização, sendo própria para o dispositivo Raspberry Pi com conexão para slot de fita plana. Trata-se de uma câmera digital com resolução de 5 megapixels, possuindo a capacidade de tirar fotos ou fazer filmagens em alta definição quando instalada.

5. Modelagem do Sistema

O sistema do mini carro autônomo é baseado no modelo cliente-servidor, onde o chip Raspberry Pi tem como função ser o cliente, e o Computador Desktop o servidor. Resumidamente, o sistema tem as seguinte funcionalidades:

- Capturar imagens através de uma câmera, localizada na parte frontal do carro, ligada ao Raspberry Pi;
- Realizar a conexão via rede sem fio Wi-Fi entre o cliente e o servidor via TCP/IP;
- Pré processamento da imagem no cliente, transformando-a em escala de cinza;
- Envio da imagem obtida do cliente para o servidor;
- Processamento da imagem, obtendo a região de interesse, para então alimentar a Rede Neural;
- A Rede Neural retorna um ângulo de esterço;
- Podem ser inseridos comandos ao carro através de um joystick conectado ao servidor;
- É retornado ao cliente os comandos para o controle do carro;
- O Raspberry Pi processa o sinal de retorno, e envia os sinais de comando para os motores do carro.

A figura 1 demonstra graficamente a modelagem do sistema, como as caixas de cor roxa o cliente, e cor esverdeada o servidor.

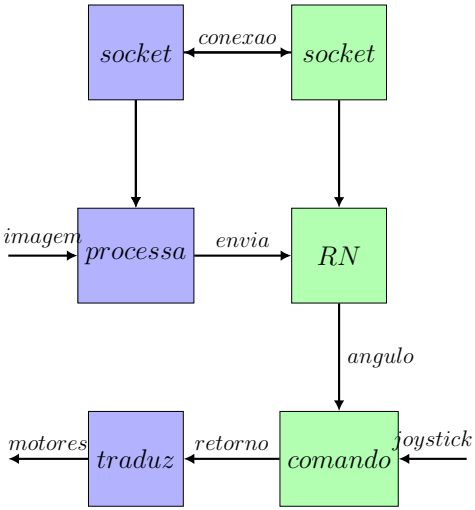


Figure 1. Modelagem do sistema.

6. Experimentos

Antes que o carro possa ser executado em modo autônomo, ele precisa ser treinado manualmente. A obtenção dos dados de treinamento ocorre da seguinte forma:

- O vídeo é capturado pelo Raspberry Pi usando o módulo da câmera que é montado na parte superior do carro;
- O vídeo é transmitido através da rede sem fio para o servidor Desktop;
- Um joystick de Xbox conectado ao servidor é usado para controlar manualmente o carro;
- Cada quadro de vídeo recebido é armazenado no servidor, junto com a entrada do usuário correspondente (entrada de direção e entrada de aceleração).

O usuário deve dirigir manualmente o carro ao redor do circuito de treinamento várias vezes para coletar imagens suficientes e cobrir as diferentes condições que o carro pode encontrar. Os dados de treinamento foram coletados controlando manualmente o carro em um circuito com ampla variedade de curvas de diferentes ângulos, o qual foi obtido uma amostra de aproximadamente 15000 imagens em escala de cinza para o treinamento da Rede Neural.

Para treinar uma rede neural é necessário selecionar os quadros a serem usados. Os dados coletados são rotulados com o número do quadro capturado e atividade do motorista (permanecendo em uma pista, alternando faixas, voltas e assim por diante). Para treinar uma CNN para seguir a pista, são selecionados apenas os dados em que o motorista estava em circunstâncias de pilotagem(dentro da pista, entre as faixas delimitadoras da estrada). Então, as imagens capturadas foram selecionadas em uma taxa de descarte de 3:5, ou seja, a cada 5 imagens processadas, 3 são descartadas.

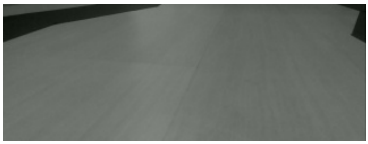


Figure 2. Exemplo de imagem capturada.

Após a coleta de dados, procede a etapa treinamento da Rede Neural Convolucional. Nesta etapa foram utilizadas 10000 imagens selecionadas aleatoriamente do banco de imagens para treinamento, e 2000 imagens são usadas para avaliação do erro associado ao modelo treinado.

7. Resultados

Foram criados diversos modelos de redes neurais, dos quais foram selecionados os que tiveram melhor desempenho na pista de testes. A palavra modelo, no contexto do experimento, se refere às versões da Rede Neural, em que as diferenças dos modelos estão nos parâmetros de treinamento, especificamente o número de lotes(número de amostras de treinamento) e nas épocas(um *forward* e *backward pass* da amostra de treinamento). A tabela 1 apresenta os resultados obtidos, em função da precisão resultante:

Table 1. Modelos de Redes Neurais Criados.

Modelo	Épocas	Lotes	Erro(%)
1	50	128	4.19
2	30	256	5.50
3	30	512	5.87
4	50	64	2.28

O passo antecedente aos testes práticos foi a simulação dos modelos de rede neural criados, com o objetivo de analisar o possível comportamento do carro associando os ângulos de esterço preditos e reais(inserido pelo usuário durante a captura de imagens). O resultado da simulação, mostrado na figura **??**, utilizou um conjunto de 3600 imagens que não foram utilizadas durante o treinamento.

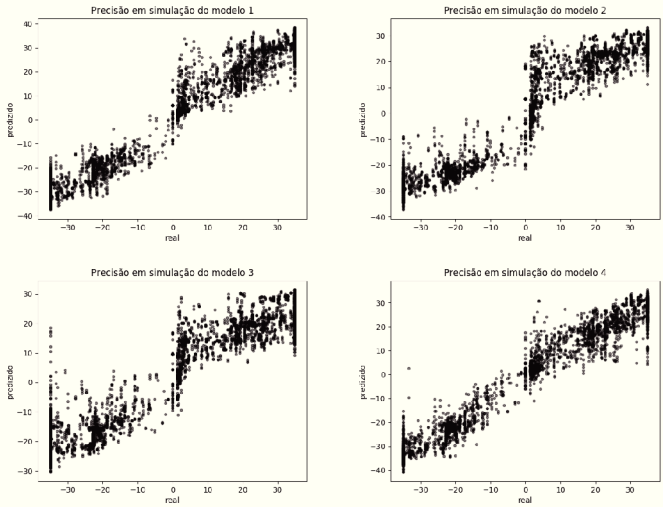


Figure 3. Gráfico da simulação.

Nos gráficos é possível notar a maior esparcidade dos ângulos no modelo de menor precisão, o qual possívelmente terá maiores dificuldades de manter a trajetória na pista de testes real. A tabela 2 mostra a média dos ângulos(em graus) e o desvio padrão de cada modelo.

Table 2. Média e desvio padrão dos modelos.

	Modelo 1	Modelo 2	Modelo 3	Modelo 4
Média	4.24	5.82	6.43	4.22
Desvio padrão	0.56	1.35	1.42	0.23

7.1. Avaliação de Capacidade Autônoma

Segundo [Zanchin et al. 2017], a definição de um carro autônomo é um veículo com a capacidade de sentir o ambiente em que está inserido e navegar através deste ambiente sem a ajuda ou intervenção de um ser humano. Com o intuito de avaliar a capacidade autônoma do mini carro, foram atribuídas os seguintes requisitos ao qual o carro deve comportar:

- Capacidade de se manter na trajetória;
- Capacidade de fazer curvas;
- Capacidade de correção de trajetória;

Foram analisados também o número de vezes que o carro colidiu com a borda da pista(erros), e se houve perda de controle e saiu do circuito. O mini carro teve como parâmetro de teste realizar o circuito duas vezes consecutivas no sentido horário, e duas

vezes consecutivas no sentido anti-horário. O teste foi repetido em todas situações com luminosidade alta e baixa.

A influência da luminosidade refere-se ao comportamento do carro em ambas situações, e sua avaliação é em função dos erros. Segue abaixo a tabela que descreve os valores atribuídos a intensidade luminosa, onde n é o número de erros em condições de alta luminosidade:

Table 3. Parâmetros de influência luminosa.

Influência	Baixa	Média	Alta
Erros	n + 2	n + 5	n + 8

Os resultados dos experimentos práticos com o carro no circuito de testes seguem na tabela 4:

Table 4. Desempenho do mini carro, sendo conduzido autonomamente.

Modelo	Erros (alta/baixa) luminosidade	Corrigiu trajetória	Saiu do circuito	Influência luminosa
1	10 alta / 3 baixa	Sim	Sim	Alta
2	6 baixa / –	Não	Sim	Alta
3	– / –	Não	Sim	Alta
4	5 alta / 1 baixa	Sim	Não	Média

Após a realização dos testes, o modelo 4 teve seu desempenho autônomo conferido. O mini carro foi controlado pela Rede Neural autonomamente no circuito de testes até a carga de suas bateria esgotarem. Em nenhum momento o mini carro saiu da pista, e realizou as correções de trajetórias para se manter no centro da pista.

7.2. Avaliação de Trajetória

Foram selecionados dois modelos de redes neurais apresentados no capítulo anterior para este experimento. Com o intuito de avaliar o impacto do erro do modelo de rede neural no mini carro autônomo e seu respectivo comportamento, um novo circuito de teste foi montado. O novo circuito contém somente três curvas em direções opostas. A avaliação da capacidade de adaptação da trajetória foi realizada seguindo os seguintes parâmetros:

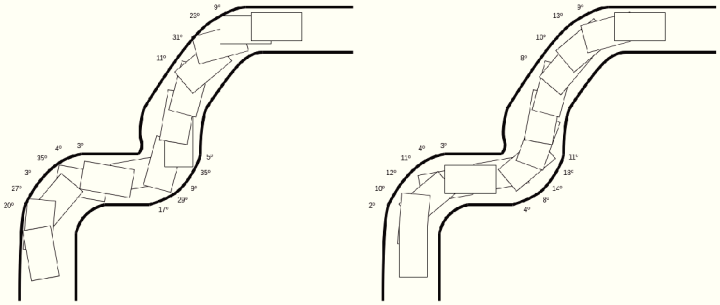
- Ambos modelos são conduzidos em modo autônomo, partindo do mesmo ponto inicial;
- O circuito de teste de trajetória está sob condições de alta luminosidade;
- São capturadas e gravadas as imagens da câmera e o ângulo de esterço do carro ao longo de cada iteração do experimento;
- O trajeto percorrido pelo carro é obtido através de marcadores adaptados ao chassi do carro.

Após a captura de imagens, foram obtidos os seguintes resultados:

Table 5. Média e desvio padrão dos ângulos obtidos.

	Modelo 1	Modelo 4
Média	9.23	8.20
Desvio padrão	0.78	0.34

Através das imagens capturadas e seus respectivos ângulos, além da trajetória marcada na superfície, foi possível reconstruir o caminho percorrido pelo carro. O diagrama abaixo(figura 7.2) compara o comportamento da correção de trajetória de ambos modelos.



Em cada trecho de curva no trajeto, foram marcados os ângulos atuados pela rede neural. Devido ao fato do projeto consistir em um objeto cinemático, qualquer ângulo de esterço mal predizido acarretará em maiores correções de trajetória para se manter na pista. É possível observar que o modelo 1, que possui menor precisão neste experimento, é capaz de manter a trajetória realizando uma série de correções bruscas, ao contrário do modelo 4, que se mantém na pista executando movimentos sutis.

8. Conclusão

O presente trabalho buscou analisar o funcionamento de redes neurais convolucionais na prática, sobre um tópico bastante comentado e controverso atualmente, a condução autônoma de um veículo.

Ao longo da implementação do trabalho, foi evidente o quão efetivo foi o uso de redes neurais convolucionais com a ferramenta Tensorflow para atingir o objetivo do trabalho. Levando em consideração o quão prático foi este projeto, foi possível relatar a magnitude do impacto dos parâmetros de treinamento e modelos com diferentes precisões na condução autônoma, assim como as variações de ambiente podem afetar drasticamente a precisão da predição da rede neural.

Por fim, a partir da arquitetura de rede neural implementada, foi possível obter excelentes resultados utilizando um hardware de baixo custo para simular um veículo real.

References

Beiker, S. A. (2012). Legal aspects of autonomous driving. *Santa Clara L. Rev.*, 52:1145.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Chen, Z. and Huang, X. (2017). End-to-end learning for lane keeping of self-driving cars. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1856–1860. IEEE.

Heylen, J., Iven, S., De Brabandere, B., Oramas, J., Van Gool, L., and Tuytelaars, T. (2018). From pixels to actions: Learning to drive a car with deep neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 606–615. IEEE.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

Litman, T. (2017). *Autonomous vehicle implementation predictions*. Victoria Transport Policy Institute Victoria, Canada.

Payton, D. (1986). An architecture for reflexive autonomous vehicle control. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1838–1845. IEEE.

Wei, D. C. M. (2015). *Método de desvio de obstáculos aplicado em veículo autônomo*. PhD thesis, Universidade de São Paulo.

Zanchin, B. C., Adamshuk, R., Santos, M. M., and Collazos, K. S. (2017). On the instrumentation and classification of autonomous cars. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2631–2636. IEEE.