Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Desenvolvimento de um sistema de contagem de passageiros em tempo real

Autor: Gustavo Nogueira Rodrigues
Orientador: Prof. Dr. Renato Coral Sampaio

Brasília, DF
2023

Gustavo Nogueira Rodrigues

Desenvolvimento de um sistema de contagem de passageiros em tempo real

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Renato Coral Sampaio

Brasília, DF

2023

Gustavo Nogueira Rodrigues
Desenvolvimento de um sistema de contagem de passageiros em tempo real/ Gustavo Nogueira Rodrigues. – Brasília, DF, 2023-
79 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Renato Coral Sampaio

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. Contagem de passageiros. 2. Sistemas embarcados. I. Prof. Dr. Renato Coral Sampaio. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Desenvolvimento de um sistema de contagem de passageiros em tempo real

CDU 02:141:005.6

Gustavo Nogueira Rodrigues

Desenvolvimento de um sistema de contagem de passageiros em tempo real

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 20 de dezembro de 2023 – Data da aprovação do trabalho:

Prof. Dr. Renato Coral Sampaio
Orientador

Prof. Dr. Alex Reis
Convidado 1

Prof Dr. Tiago Alves da Fonseca
Convidado 2

Brasília, DF
2023

Resumo

A contagem de passageiros consiste em fornecer dados para aprimorar o planejamento da oferta de transporte, bem como auxiliar os passageiros, a partir das informações de lotação dos veículos, a escolher quais itinerários melhor se aplicam para realizar a sua viagem. Diante disso, o propósito deste trabalho é desenvolver um sistema de contagem de passageiros em tempo real que permita realizar a coleta dessas informações de lotação ao longo do percurso dos veículos de transporte público coletivo de passageiros. Para tanto, a solução é composta por um dispositivo embarcado de contagem de passageiros dentro dos veículos e um servidor na nuvem. Neste trabalho, o dispositivo utilizado foi uma *Raspberry Pi 4* ligada a uma câmera a partir da interface *Camera Serial Interface (CSI)*. A solução desenvolvida emprega algoritmos de visão computacional e foi dividida em cinco etapas, sendo elas: captura dos frames da câmera; detecção de passageiros; rastreamento de passageiros; contabilização de passageiros; envio das contabilizações para o servidor na nuvem. Para a detecção de passageiros, foi treinado um modelo *Single Shot Detector (SSD)* que obteve um mAP@0,50:0,95 de 65,79%. Para o rastreamento e a contabilização de passageiros, um conjunto de quatro versões de rastreamento de objetos foram desenvolvidas. Essas versões desenvolvidas consistem em combinar as técnicas de rastreamento *Correlation Tracker*, *Simple Online and Realtime Tracking (SORT)* e *Centroid Tracker*. E para avaliar o desempenho do algoritmo, foi construído um *dataset* de teste composto por vídeos em que foram rotulados o número de entradas e saídas de pessoas de um veículo. O teste foi conduzido na *Raspberry Pi 4*, e a configuração do algoritmo que utiliza o método de rastreamento SORT destacou-se, alcançando valores de precisão e *recall* superiores a 96%, com uma taxa de processamento variando entre 28 e 49 quadros por segundo (FPS).

Palavras-chave: contagem de passageiros. sistemas embarcados. transporte público coletivo. visão computacional. *Raspberry Pi 4*.

Abstract

Passenger counting involves providing data to enhance transportation planning and assisting passengers, based on vehicle occupancy information, in selecting the most suitable itineraries for their journey. Therefore, the purpose of this work is to develop a real-time passenger counting system that allows the collection of occupancy information along the route of public transportation vehicles. To achieve this, the solution consists of an embedded passenger counting device within the vehicles and a cloud server. In this work, the device used was a Raspberry Pi 4 connected to a camera through the Camera Serial Interface (CSI). The developed solution employs computer vision algorithms and is divided into five stages: capturing camera frames, passenger detection, passenger tracking, passenger counting, and sending counts to the cloud server. For passenger detection, a Single Shot Detector (SSD) model was trained, achieving a mAP@0.50:0.95 of 65.79%. For passenger tracking and counting, a set of four object tracking versions were developed, combining Correlation Tracker, Simple Online and Realtime Tracking (SORT), and Centroid Tracker techniques. To evaluate the algorithm's performance, a test dataset was created with labeled entries and exits of people from a vehicle. The test was conducted on the Raspberry Pi 4, and the configuration using the SORT tracking method stood out, achieving precision and recall values exceeding 96%, with a processing rate ranging between 28 and 49 frames per second (FPS).

Key-words: passenger count. embedded systems. public transportation. computer vision. Raspberry Pi 4.

Lista de ilustrações

Figura 1 – Arquitetura da LeNet-5 23
Figura 2 – Convolução de uma imagem com um *kernel* detector de arestas. 24
Figura 3 – Aplicação de *downsampling* com filtros 2x2 e *stride* 2. 25
Figura 4 – Gráfico da função de ativação ReLU. 26
Figura 5 – Indexação linear para vetorizar imagem 27
Figura 6 – Arquitetura da rede SSD 27
Figura 7 – Ilustração das caixas de *ground truth* e das *default boxes* 28
Figura 8 – Gráfico da Gaussiana 2-D 30
Figura 9 – Abordagem de rastreamento baseado em filtros de correlação. 32
Figura 10 – Funcionamento do filtro de Kalman 35
Figura 11 – Exemplo de aplicação do problema de designação 36
Figura 12 – Caminhos alternantes e caminhos de aumento em grafos 38
Figura 13 – Diagrama de representação do funcionamento do SORT 39
Figura 14 – Ilustração do rastreamento de objetos com o SORT. 40
Figura 15 – Associação de objetos entre frames consecutivos. 42
Figura 16 – Contagem de pessoas com auxílio de linhas virtuais 43
Figura 17 – Representação da arquitetura da solução. 45
Figura 18 – Raspberry Pi 4 modelo B. 45
Figura 19 – Módulo Raspberry Pi Cam v1.3. 46
Figura 20 – Frames retirados das filmagens cedidas pela Viação Breda. 49
Figura 21 – Frames retirados do *dataset* PCDS. 50
Figura 22 – Frames retirados do *dataset* do laboratório MIVIA. 51
Figura 23 – Imagens retiradas do *dataset Human Detection* do Kaggle. 51
Figura 24 – *Software* de rotulagem de imagens *LabelImg*. 52
Figura 25 – Ilustração do cálculo a métrica de Intersecção sobre União. 55
Figura 26 – Exemplo numérico do cálculo da *PR Curve*. 56
Figura 27 – Diagrama de atividades do algoritmo de contagem de passageiros. . . . 61
Figura 28 – Montagem do protótipo para testes em tempo real. 64
Figura 29 – Escada utilizada para realização dos testes em tempo real. 65
Figura 30 – Gráfico com desempenho dos modelos de detecção desenvolvidos. . . . 67
Figura 31 – FPS Médio do algoritmo de contagem em diferentes configurações. . . . 68
Figura 32 – Precisão do algoritmo de contagem em diferentes configurações. 69
Figura 33 – *Recall* do algoritmo de contagem em diferentes configurações. 70

Lista de tabelas

Tabela 1 – Composição dos *datasets* construídos. 53
Tabela 2 – Especificação da máquina de treinamento do modelo. 58
Tabela 3 – *Dataset* de teste do algoritmo de contagem de passageiros. 62
Tabela 4 – Dados da tabela de eventos de contagem de passageiros. 63
Tabela 5 – Parâmetros da requisição de consulta dos dados de contagem. 64
Tabela 6 – Resposta da requisição de consulta dos dados de contagem. 64
Tabela 7 – FPS Médio do algoritmo de contagem em diferentes configurações. . . 68
Tabela 8 – Precisão do algoritmo de contagem em diferentes configurações. 69
Tabela 9 – *Recall* do algoritmo de contagem em diferentes configurações. 70

Lista de abreviaturas e siglas

AP	Average Precision
API	Application Programming Interfaces
AWS	Amazon Web Services
CFTV	Circuito Fechado de TV
CNI	Confederação Nacional da Indústria
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CSI	Camera Serial Interface
DFT	Transformada Discreta de Fourier
FFT	Fast Fourier Transform
FPN	Feature Pyramid Network
FPS	Frames Per Second
GCP	Google Cloud Platform
GPS	Global Positioning System
GPU	Graphics Processing Unit
HTTPS	Hyper Text Transfer Protocol Secure
IFFT	Inverse Fast Fourier Transform
IoT	Internet of Things
IoU	Intersection over Union
LAPJV	Jonker-Volgenant for Linear Assignment Problem
mAP	Mean Average Precision
MCU	Microcontroller Unit

MNIST	Modified National Institute of Standards and Technology
MOSSE	Minimum Output Sum of Square Error
MOT	Multiple Object Tracking
MQTT	Message Queuing Telemetry Transport
NBIoT	Narrowband Internet of Things
NMS	Non-Maximum Suppression
ReLU	Rectified Linear Unit
RGB	Red, Green, Blue
RPN	Region Proposal Network
SEMOB-DF	Secretaria de Transporte e Mobilidade do Distrito Federal
SoC	System on Chip
SORT	Simple Online and Realtime Tracking
SSD	Single Shot Detector
VGG	Visual Geometry Group
Wi-fi	Wireless Fidelity
YOLO	You Only Look Once

Sumário

1 INTRODUÇÃO 19
1.1 Objetivos 20
1.1.1 Objetivo Geral 20
1.1.2 Objetivos Específicos 20
1.2 Metodologia 21

2 FUNDAMENTAÇÃO TEÓRICA 23
2.1 Rede Neural Convolucional (CNN) 23
2.1.1 Convolução em Imagens 24
2.1.2 Camada de Convolução 24
2.1.3 Camada de Downsampling 25
2.1.4 Camada de Ativação 25
2.1.5 Camada Fully Connected 26
2.2 Detecção de Objetos: Single Shot Detector (SSD) 26
2.2.1 Arquitetura 27
2.2.2 Treinamento 28
2.3 Rastreamento de Objetos: Correlation Tracker 29
2.3.1 Treinamento do Filtro de Correlação 29
2.3.2 O Algoritmo de Rastreamento 31
2.4 Rastreamento de Objetos: SORT Tracker 33
2.4.1 Filtro de Kalman 33
2.4.1.1 O algoritmo de filtro discreto de Kalman 33
2.4.2 Problema de Designação 35
2.4.3 Algoritmos para o Problema de Designação 37
2.4.3.1 Emparelhamento em Grafos Bipartidos 37
2.4.3.2 Caminhos alternantes e Caminhos de aumento 37
2.4.3.3 Algoritmo Húngaro 37
2.4.3.4 Algoritmo LAPJV 38
2.4.4 O Algoritmo de Rastreamento 39
2.5 Rastreamento de Objetos: Centroid Tracker 41
2.6 Contabilizando Pessoas 43

3 ESPECIFICAÇÃO DO SISTEMA 45
3.1 Materiais 45
3.1.1 Raspberry Pi 4B 45
3.1.2 Módulo de GPS 46

3.1.3 Módulo de Câmera . . . 46
3.1.4 Access Point Wireless . . . 46
3.2 Software Embarcado . . . 47
3.3 Servidor na Nuvem . . . 47
3.3.1 AWS IoT Core . . . 47
3.3.2 Amazon DynamoDB . . . 47
3.3.3 AWS Lambda . . . 47
3.3.4 Amazon API Gateway . . . 48

4 DESENVOLVIMENTO . . . 49
4.1 Seleção de Datasets . . . 49
4.1.1 Contato com a Onboard Mobility e a Viação Breda . . . 49
4.1.2 Contato com a SEMOB-DF . . . 49
4.1.3 People Counting Dataset (PCDS) . . . 50
4.1.4 Dataset Público do Laboratório MIVIA . . . 51
4.1.5 Human Detection Dataset . . . 51
4.2 Rotulagem do Dataset . . . 52
4.3 Modelo de Detecção de Objetos . . . 53
4.3.1 Teste do Modelo . . . 53
4.3.1.1 Matriz de Confusão . . . 53
4.3.1.2 Precisão e Recall . . . 54
4.3.1.3 Intersection over Union (IoU) . . . 54
4.3.1.4 Average Precision (AP) . . . 54
4.3.1.5 Mean Average Precision (mAP) . . . 56
4.3.2 Arquitetura da Rede . . . 56
4.3.3 Treinamento do Modelo . . . 58
4.4 Rastreadores de Objetos . . . 58
4.4.1 Correlation Tracker . . . 58
4.4.2 Sort Tracker . . . 59
4.4.3 Centroid Tracker . . . 59
4.5 Captura de Frames . . . 59
4.6 Contagem de Passageiros . . . 59
4.6.1 Implementação do Algoritmo . . . 59
4.6.2 Teste do Algoritmo . . . 61
4.7 Servidor na Nuvem . . . 62
4.7.1 Envio dos Dados . . . 63
4.7.2 Armazenamento dos Dados . . . 63
4.7.3 Consulta dos Dados . . . 63
4.8 Teste em Tempo Real . . . 64

5 RESULTADOS 67
5.1 Modelo de Detecção de Objetos SSD 67
5.2 Algoritmo de Contagem de Passageiros 67
5.2.1 FPS Médio 68
5.2.2 Precisão 69
5.2.3 Recall 70
5.2.4 Análise das Métricas 71
5.3 Demonstrações 71

6 CONCLUSÃO 73

7 TRABALHOS FUTUROS 75
7.1 Submódulo de GPS 75
7.2 Abordagem com apenas um Processador por Veículo 75

REFERÊNCIAS 77

1 Introdução

Segundo o levantamento sobre transporte público feito em 2015 pela Confederação Nacional da Indústria (CNI), um em cada quatro brasileiros se desloca de ônibus para as atividades do cotidiano, sendo assim um dos principais meios de transporte dos brasileiros (CNI, 2015).

O levantamento aponta também que o brasileiro está mais insatisfeito com a qualidade do transporte público. De acordo com a CNI, o percentual de brasileiros que avalia o transporte público como ótimo ou bom caiu de 39% em 2011 para 24%, assim representando uma redução de 15 pontos percentuais em quatro anos. Já o percentual que avalia o transporte público como ruim ou péssimo passou de 28% em 2011 para 36% em 2014, indicando um acréscimo de 8 pontos percentuais (CNI, 2015).

Além disso, a pesquisa ainda apresenta os principais motivos da baixa adesão do transporte público para os brasileiros que utilizam esse tipo de transporte com menor frequência, raramente ou nunca. Para a maioria desses brasileiros (26%), os motivos são problemas de capilaridade e frequência, ou seja, o transporte público não permite o usuário chegar a todos os lugares e possui poucas linhas e veículos disponíveis para lidar com a demanda. Já a lentidão e atrasos frequentes representam um total de 24% das avaliações. O custo do transporte público é um limitador considerado para 10% dos brasileiros, e outros 8% alegam que o transporte público é desconfortável (está sempre lotado, é sujo, cheira mal, etc) (CNI, 2015).

Um dos principais fatores que contribuem para essa crise do transporte público no Brasil é o atual modelo de sustentação financeira das empresas de transporte. Nesse modelo, o financiamento da operação das empresas é feito majoritariamente pelas tarifas pagas pelos passageiros e, portanto, nem sempre é o suficiente para realizar certas melhorias no sistema de transporte, como o aumento da frota de veículos, por exemplo (CARVALHO et al., 2013).

Diante desse cenário, especialistas da área de transportes elencaram possíveis soluções que podem contribuir para a resolução da atual crise do transporte público no Brasil. Para LINDAU, ALBUQUERQUE e CORRÊA (2021), uma das ações cabíveis é renovar o transporte público por meio de medidas que adequem a oferta de mobilidade à demanda de viagens. De maneira mais precisa, os especialistas afirmam que:

> Renovar passa pela coleta e abertura de dados para entender as mudanças nos padrões de deslocamento das pessoas e pelo aperfeiçoamento de instrumentos como as Pesquisas Origem e Destino (OD). Os dados permitem às cidades planejar tanto a oferta de linhas de ônibus quanto a implantação de infraestrutura para os transportes coletivo e ativo. Poucas são as cidades brasileiras que vêm realizando periodicamente esses levantamentos, em geral através de longos questionários. (LINDAU; ALBUQUERQUE; CORRÊA, 2021)

Posto isto, é possível constatar que a utilização de mecanismos que facilitem a coleta de dados relacionados à demanda de viagens pode aprimorar o planejamento da oferta de transporte, bem como possibilitar outras melhorias. E atualmente, uma das soluções que vêm sendo estudadas para simplificar essa coleta de dados é a contagem de passageiros por meio de filmagens.

Atualmente, os veículos do transporte público contam com um dispositivo denominado como validador, popularmente chamado de catraca, que são utilizados para efetuar a cobrança de tarifa dos passageiros. O validador é normalmente instalado logo após a porta de embarque do veículo, sendo assim, cada passageiro pagante que embarcou no veículo pode ser facilmente contabilizado pelo dispositivo. Dessa forma, é possível contabilizar o total de embarques em cada viagem apenas com o validador. Contudo, o validador não seria capaz de contabilizar, em tempo real, o número de embarques e desembarques por ponto de parada. Portanto, fazer uso apenas do validador não é suficiente para coleta de dados relativos à demanda de viagens.

A empresa Onboard Mobility vem desenvolvendo uma solução que utiliza algoritmos de visão computacional para realizar a contagem de passageiros de forma automática a partir de filmagens do sistema de Circuito Fechado de TV (CFTV) dos ônibus. Segundo BELINAZI (2020), a Onboard Mobility conseguiu obter resultados promissores em testes realizados na Viação Piracicabana, em Brasília, e Consórcio Ótimo e Transfácil, em Belo Horizonte.

Então, tendo-se em vista o contexto apresentado, o objeto de estudo deste trabalho consiste em desenvolver um *software* embarcado que realiza a contagem de passageiros em tempo real.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo deste trabalho é desenvolver um *software* embarcado em tempo real que realiza a contagem de passageiros por meio de algoritmos de visão computacional.

1.1.2 Objetivos Específicos

- Embarcar um sistema operacional Linux em uma plataforma *SoC Raspberry Pi*.
- Implementar o algoritmo de visão computacional que realiza contagem de passageiros a partir da detecção de embarque e desembarque de usuários.
- Implementar o submódulo de captura de *stream* de vídeo da câmera.
- Implementar o submódulo de leitura e armazenamento dos dados de GPS.
- Implementar o submódulo de envio de eventos de embarque e desembarque para a aplicação na nuvem.
- Implementar aplicação na nuvem que seja capaz de receber e fornecer as informações de embarque e desembarque de passageiros dos veículos.
- Construir protótipo de *hardware* para realização dos testes em tempo real.
- Construir *dataset* de teste para avaliar o desempenho do algoritmo de contagem de passageiros.
- Testar o algoritmo de contagem de passageiros tanto no *dataset* construído quanto em tempo real. O teste deve calcular métricas que permitem avaliar o desempenho do algoritmo desenvolvido.

1.2 Metodologia

A metodologia definida para este trabalho é baseada na composição dos componentes mais básicos de maneira que seja possível compor a solução final de contagem de passageiros, em outras palavras, será aplicado metodologia *bottom-up*. Para tanto, o desenvolvimento deste trabalho foi dividido nas seguintes etapas:

1. Revisão bibliográfica necessária para o trabalho.
2. Definição e descrição dos componentes de *software* e de *hardware* da solução.
3. Desenvolvimento dos componentes da solução, contendo sistema embarcado de contagem de passageiros e o servidor na nuvem.
4. Integração e testes dos componentes da solução.
5. Ajustes finais, refinações e melhorias na solução.

2 Fundamentação Teórica

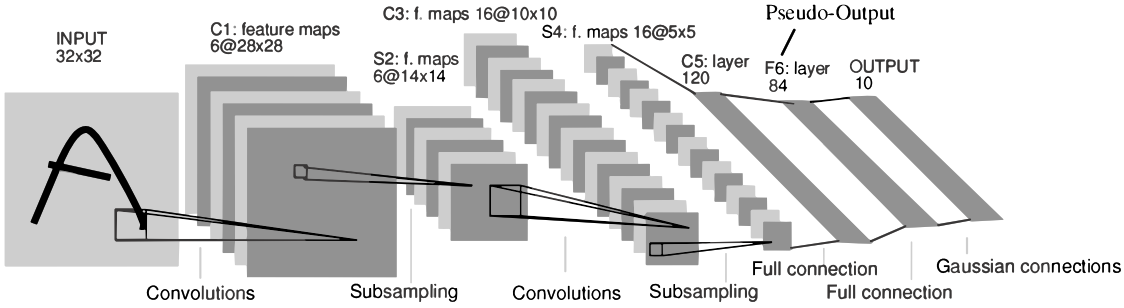
Tecnologias diferentes podem e são usadas em dispositivos de contagem de pessoas, como sensores infravermelho, imagens térmicas, visão computacional e *WiFi counting* (BOX; OPPENLANDER, 1976). Entretanto, o presente trabalho tem como objeto de estudo apenas as abordagens que utilizam visão computacional.

As seções a seguir apresentam os fundamentos da abordagem que será utilizada neste trabalho como solução da contagem de passageiros em tempo real. De forma sucinta, esta abordagem combina algoritmos de detecção e rastreamento de objetos a partir de imagens.

2.1 Rede Neural Convolucional (CNN)

Em 1998 a solução desenvolvida por Lecun et al. (1998) para reconhecer caracteres manuscritos deu origem a uma nova forma de realizar o reconhecimento e processamento de imagens. Esta solução, denominada LeNet-5 e representada na Figura 1, é uma CNN composta por sete camadas no total, sendo três camadas convolucionais, duas camadas de downsampling (subamostragem) e duas camadas fully connected (totalmente conectadas). Após ser treinada e testada com a base de dados MNIST (*Modified National Institute of Standards and Technology*), a LeNet-5 obteve uma taxa de erro de 0,95%. Os resultados obtidos na pesquisa despertaram o interesse de estudiosos e, atualmente, apesar da arquitetura das redes neurais de melhor desempenho não ser a mesma da LeNet-5, a rede serviu como ponto de partida para diversas arquiteturas de redes neurais convolucionais.

Figura 1 – Arquitetura da LeNet-5 contendo todos os seus elementos básicos.



Fonte: Lecun et al. (1998).

As CNNs são uma classe de modelos de aprendizado profundo que foram projetadas especificamente para trabalhar com imagens e outros tipos de dados que possuem uma estrutura espacial (LECUN; BENGIO; HINTON, 2015). Diferente dos métodos tradicionais de extração de características, as CNNs não precisam extrair características

manualmente (LI et al., 2022). Em vez disso, as CNNs utilizam as camadas convolucionais e de *downsampling* para realizar a extração de características dos dados, enquanto as camadas *fully connected*, a partir das características extraídas, são utilizadas para realizar as operações finais de classificação (ROSEBROCK, 2017).

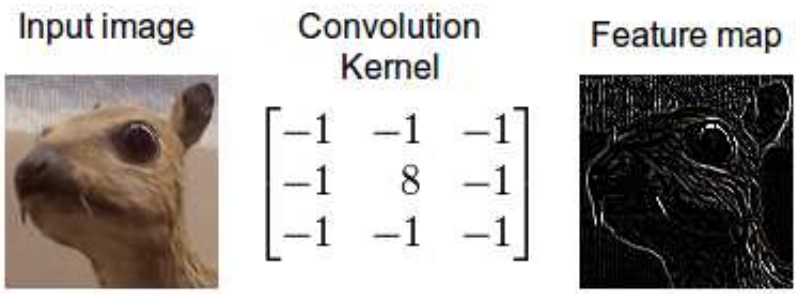
2.1.1 Convolução em Imagens

A operação de convolução envolve aplicar um filtro, também conhecido como kernel, sobre uma imagem para extrair características ou realizar operações de modificação. Matematicamente, a operação de convolução é definida pela Equação 2.1, onde *g(x,y)* é a imagem resultante após a aplicação do *kernel* ω na imagem de entrada *f(x,y)*.

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x-dx, y-dy) \tag{2.1}$$

O processo de convolução funciona da seguinte maneira: o *kernel* é deslizado sobre a imagem, um pixel de cada vez, e é aplicado a uma pequena região da imagem. Em seguida, os valores dos pixels na região são multiplicados pelos valores correspondentes no *kernel* e são somados. O resultado da soma é armazenado em um novo pixel na imagem de saída. Este processo é repetido para cada pixel na imagem, gerando uma nova imagem que pode ter características diferentes da imagem original (GONZALEZ; WOODS, 2008). O resultado desse processo é ilustrado na Figura 2, onde foi aplicado uma convolução na imagem de entrada com um *kernel* detector de arestas.

Figura 2 – Convolução de uma imagem com um *kernel* detector de arestas.[1]



2.1.2 Camada de Convolução

A camada convolucional em CNN é responsável por capturar as características importantes dos dados de entrada. É chamada de convolucional porque realiza uma operação

[1] Disponível em: <https://developer.nvidia.com/discover/convolution>. Acesso em: 01 de fevereiro de 2023.

matemática chamada convolução apresentada na Seção 2.1.1, onde um filtro é aplicado sobre as entradas para aprender características específicas da imagem, assim gerando mapas de características (também chamados de *feature maps*).

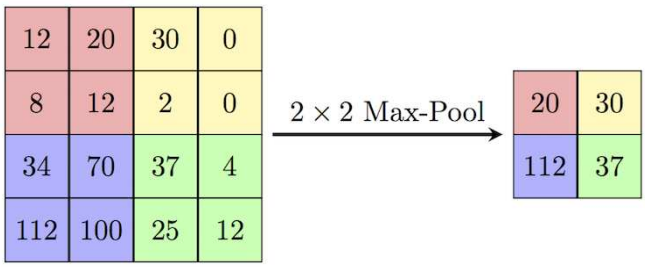
A camada convolucional aplica múltiplos filtros sobre a entrada, cada um aprendendo uma característica diferente, como linhas, bordas, curvas, entre outras. As saídas dos filtros são então combinadas para formar uma nova representação da entrada, com destaque para as características mais importantes. Esta nova representação é passada para as próximas camadas da CNN, onde são aprendidas características cada vez mais complexas (LECUN et al., 1998).

2.1.3 Camada de Downsampling

A camada de *downsampling*, também denominada como camada de *pooling*, tem como objetivo reduzir a dimensão dos mapas de características produzidos por camadas de convolução. Isso é útil porque permite que a rede mantenha informações importantes e descarte informações redundantes e/ou irrelevantes. Além disso, a subamostragem também ajuda a lidar com a translação e a rotação da imagem de entrada (LECUN et al., 1998).

Existem diversos tipos diferentes de *downsampling*, tais como *max pooling*, *average pooling* e *sum pooling*. O *max pooling*, ilustrado na Figura 3, é o método mais comum e consiste em selecionar o valor máximo de um conjunto de valores adjacentes na entrada.

Figura 3 – Aplicação de *downsampling* com filtros 2x2 e *stride* 2. [2]



2.1.4 Camada de Ativação

No contexto das CNNs, a camada de ativação é responsável por aplicar uma função de ativação não linear aos mapas de características gerados na camada anterior, assim possibilitando que a rede aprenda relações não lineares entre as entradas e as saídas.

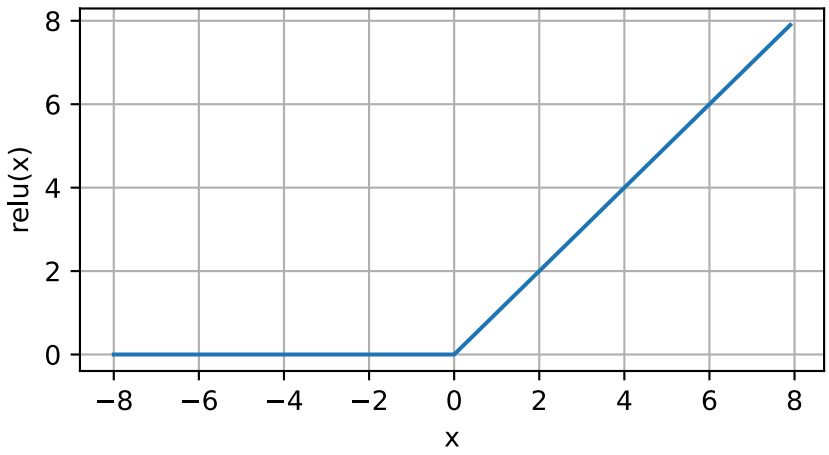
Algumas das funções de ativação mais comuns usadas em CNNs incluem a função ReLU (*Rectified Linear Unit*), a função sigmóide e a função tangente hiperbólica. A função não linear ReLU, expressa pela Equação 2.2 e representada pelo gráfico da Figura 4, é a

[2] Disponível em: <https://computersciencewiki.org/index.php/File:MaxpoolSample2.png>. Acesso em: 01 de fevereiro de 2023.

mais utilizada atualmente ao projetar redes neurais (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

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

Figura 4 – Gráfico da função de ativação ReLU.



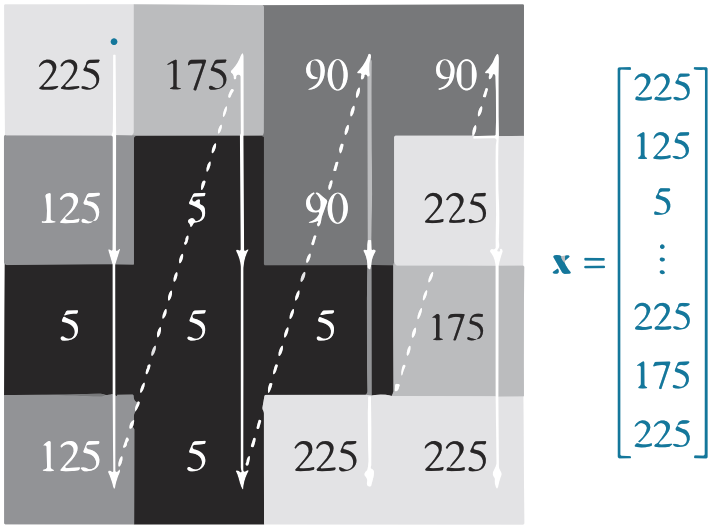
Diferente das funções sigmoidais, a função ReLU é capaz de lidar melhor com problemas causados pelo *Vanish Gradient* (Desaparecimento de Gradiente). Desta forma, ela possibilita o treinamento várias vezes mais rápido da rede neural e sem uma penalidade significativa para a precisão da generalização (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

2.1.5 Camada Fully Connected

Localizada no final da CNN, essa camada é essencialmente uma Rede Neural Perceptron Multicamadas tradicional, também conhecida como *Multilayer Perceptron* (MLP). A camada *fully connected* é responsável por realizar a classificação dos *features maps* produzidos pelas camadas anteriores da rede. Porém, os *features maps* são matrizes 2-D, enquanto as entradas para uma rede *fully connected* são vetores. Sendo assim, é necessário vetorizar os *features maps* 2-D antes de apresentá-los a camada *fully connected*. Para tanto, é utilizada a indexação linear (processo ilustrado na Figura 5), onde cada matriz 2-D é convertida em um vetor, então todos os vetores resultantes são verticalmente concatenados para formar um único vetor (GONZALEZ; WOODS, 2008).

Esse vetor resultante se propaga pela rede neural. Após propagar pela rede, como o número de saídas na rede *fully connected* é igual ao número de classes de padrão a serem classificadas, a saída com o valor mais alto determina a classe da entrada (GONZALEZ; WOODS, 2008).

Figura 5 – Aplicação de indexação linear para vetorizar uma imagem em tons de cinza.



Fonte: Gonzalez e Woods (2008).

2.2 Detecção de Objetos: Single Shot Detector (SSD)

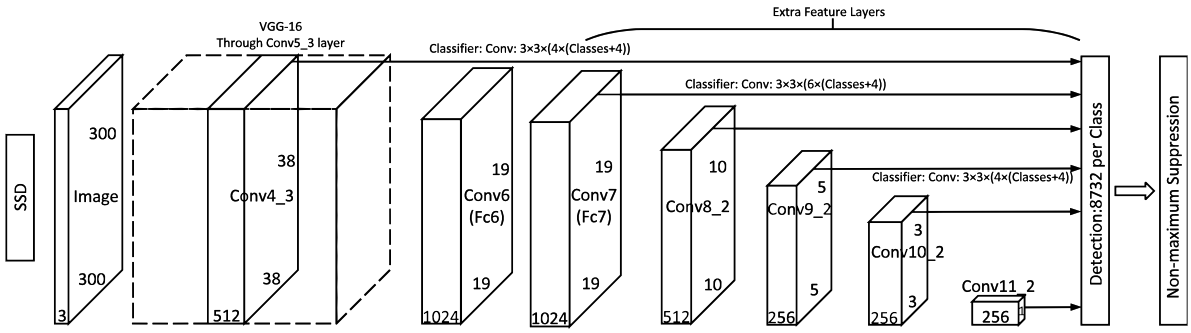
Proposto por Liu et al. (2016), o SSD (*Single Shot Detector*) é um algoritmo de detecção de objetos que utiliza uma rede neural para realizar a detecção de objetos em uma imagem. Assim como o detector YOLO (*You Only Look Once*), o SSD utiliza apenas um disparo (passagem pela rede neural) para detectar vários objetos presentes em uma imagem, entretanto, o SSD é mais rápido que o YOLO e tão preciso quanto técnicas mais lentas que utilizam Redes de Propostas Regionais (*Region Proposal Network*, RPN), como o *Faster R-CNN*. Devido a esses fatores, Liu et al. (2016) afirma que uma das vantagens do SSD é que ele é projetado para ser rápido e eficiente em termos de recursos computacionais, o que o torna adequado para aplicações em tempo real e dispositivos de embarcados.

2.2.1 Arquitetura

A arquitetura proposta para o SSD, representada na Figura 6, utiliza uma rede neural profunda, como a rede VGG (*Visual Geometry Group*), para extrair características de uma imagem. Em seguida, essas características são passadas para várias camadas de convolução de detecção (sendo cada camada responsável por detectar objetos em uma escala específica) que predizem a localização e a pontuação de confiança dos objetos na imagem. Por fim, o SSD utiliza uma técnica chamada *Non-Maximum Suppression (NMS)* para remover sobreposições e selecionar as detecções mais precisas (LIU et al., 2016).

Na arquitetura de uma rede SSD, as camadas profundas cobrem campos receptivos maiores e produzem representações mais abstratas e isso é útil para detectar objetos

Figura 6 – Arquitetura da rede SSD.



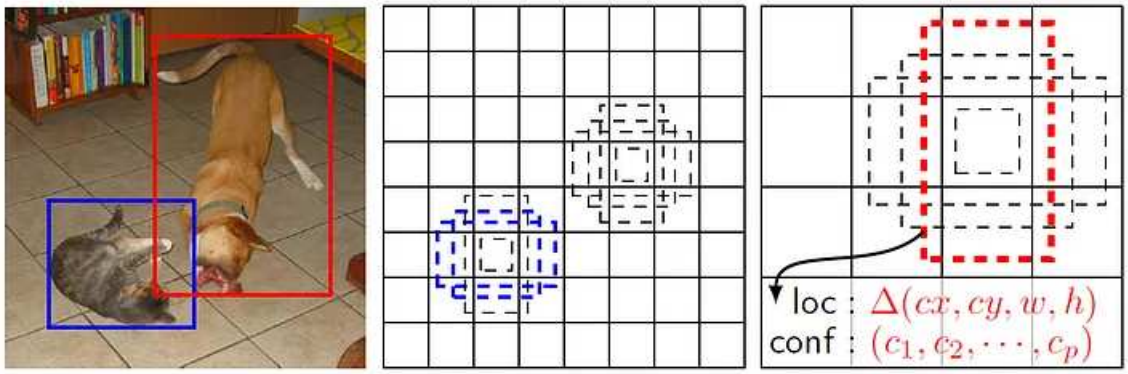
Fonte: Liu et al. (2016).

maiores. Enquanto as camadas iniciais de convolução cobrem campos receptivos menores e são úteis na detecção de objetos menores existentes na imagem.

2.2.2 Treinamento

Para realizar o treinamento, o SDD precisa apenas de uma imagem de entrada e caixas delimitadoras chamadas de *ground truth boxes* (*GT boxes*) para cada objeto na imagem (LIU et al., 2016). As *GT boxes*, apresentadas na Figura 7 (a), são caixas que delimitam as regiões dos objetos na imagem e são utilizadas como referência para avaliar o desempenho do SSD.

Figura 7 – (a) Representação de uma imagem de entrada com uma caixa *ground truth*. (b) *Feature map* 8×8 com *default boxes*. (c) *Feature map* 4×4 com *default boxes*.



Fonte: Liu et al. (2016).

Após algumas convoluções para extrair características, uma camada de *feature maps* de tamanho $m \times n$ (totalizando o número de locais/células) e p canais é obtida, como as grades 8×8 e 4×4 na Figura 7 (b) e (c). Em seguida, uma convolução 3×3 é aplicada nesta camada de *features*. Depois de aplicar a convolução, k caixas delimitadoras, denominadas como *default boxes* (ou caixas padrão), são geradas para cada local (LIU et al., 2016). Conforme ilustrado na Figura 7 (b) e (c), as k caixas delimitadoras

possuem tamanhos e proporções diferentes, assim permitindo a detecção de objetos de diferentes proporções e tamanhos, como pessoas e veículos.

Para cada caixa padrão, é predito um *offset* em relação ao centro da caixa e a pontuação de confiança para todas as classes de objetos (Figura 7 c). Assim, durante o treinamento, as caixas padrão são comparadas com as caixas delimitadoras *ground truth* com o propósito de encontrar as caixas padrão que mais se sobrepõem às caixas *ground truth*.

Por fim, a função de *loss* do modelo SSD é composta pela soma ponderada entre o *loss* de localização e o *loss* de confiança, conforme expresso na Equação:

$$L = \frac{1}{N}(L_{\text{conf}} + \alpha L_{\text{loc}}) \tag{2.3}$$

Onde N representa o número de caixas padrão comparadas e α o peso entre as duas funções de *loss*, o L_{loc} (*Smooth L1*) é o *loss* de confiança e L_{conf} (*Softmax*) o *loss* de localização.

2.3 Rastreamento de Objetos: Correlation Tracker

Uma vez que existem diversas variantes e implementações de rastreadores baseados em filtros de correlação na literatura, esta seção tem como foco apenas apresentar a estrutura geral desse tipo de rastreador de objetos.

A partir da entrada inicial composta por um *frame* de vídeo e a caixa delimitadora do objeto a ser rastreado, o rastreador tem como objetivo ajustar a caixa delimitadora à medida que o objeto alvo se desloca, mesmo que ele altere sua aparência ou tamanho, ou esteja parcialmente obstruído por outros objetos.

Em linhas gerais, rastreadores baseados em filtros de correlação funcionam da seguinte forma: primeiramente, o filtro de correlação passa por um treinamento com um recorte da imagem proveniente da posição da caixa delimitadora do objeto alvo no primeiro *frame* do vídeo. Posteriormente, em cada quadro subsequente, a detecção é feita a partir de um novo recorte feito na posição predita anteriormente. Geralmente, é realiza-se a extração de *features* (feições) dos dados brutos de entrada, bem como aplica-se uma função de janelamento usando uma janela de cosseno, ou uma janela Gaussiana, ao recorte para enfatizar a região central e eliminar as descontinuidades nas bordas da imagem (BARCELLOS, 2021).

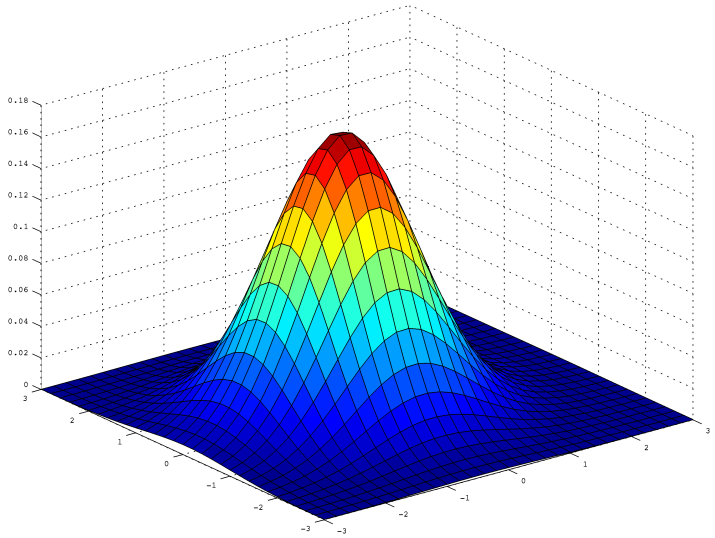
2.3.1 Treinamento do Filtro de Correlação

O primeiro método proposto para treinar filtros de correlação foi o filtro adaptativo *Minimum Output Sum of Square Error* (MOSSE) (BOLME et al., 2010).

Para treinar o filtro MOSSE, considera-se uma ou mais tuplas (x_i, y_i) de dados de treinamento. Neste contexto, x_i representa imagens em escala de cinza contendo visualizações do alvo a ser rastreado, e y_i as saídas desejadas. Embora um filtro simples possa ser obtido apenas com a amostra x e a saída desejada correspondente y, é necessário incluir mais amostras para aprimorar a robustez do filtro de correlação.

A resposta de saída desejada y pode ter diversas formas, mas, no caso do filtro MOSSE e em geral, é gerada a partir do *ground truth* com uma distribuição em forma de Gaussiana 2-D (Figura 8), cujo pico está no centro.

Figura 8 – Gráfico da Gaussiana 2-D.



Para correlacionar de maneira precisa as amostras de entrada com as saídas desejadas, o filtro MOSSE busca um filtro h que minimize a soma do erro quadrático entre as saídas de correlação obtidas e as saídas de correlação desejadas (BARCELLOS, 2021). Sendo assim, treinar o filtro equivale a resolver o seguinte problema de otimização:

$$\min_{h} \sum_{i} \|h \otimes x_i - y_i\|^2 \tag{2.4}$$

onde i indexa cada imagem de treino e $\otimes$ representa a operação de convolução da imagem x_i com o filtro h.

A partir do Teorema da Convolução, é possível reescrever a convolução da função f com a função g como a multiplicação elemento a elemento das transformadas de Fourier

de cada função, conforme as Equações a 2.5 e 2.6 a seguir:

$$\mathcal{F}\{f \otimes g\} = \mathcal{F}\{f\} \odot \mathcal{F}\{g\} \tag{2.5}$$

$$f \otimes g \ = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \odot \mathcal{F}\{g\}\} \tag{2.6}$$

onde $\mathcal{F}$ representa o operador de Transformada de Fourier, $\mathcal{F}^{-1}$ o operador de Transformada Inversa de Fourier, $\odot$ representa o operador de multiplicação elemento a elemento e $\otimes$ representa o operador de convolução.

Do ponto de vista computacional, a complexidade da convolução para uma imagem de tamanho $n \times n$ é $O(n^4)$, ao passo que a multiplicação elemento a elemento utilizando a Transformada Rápida de Fourier (*Fast Fourier Transform* - FFT) requer apenas $O(n^2 \log n)$ (CHEN; HONG; TAO, 2015). Portanto, tratar o problema no domínio da frequência aliado ao uso da FFT oferece uma aceleração considerável ao método.

Dito isso, ao reescrever o problema de otimização da Equação 2.4 no domínio da frequência, temos:

$$\min_{H^*} \sum_i \|H^* \otimes X_i - Y_i\|^2 \tag{2.7}$$

assim, a solução para o filtro H^*, na qual a explicação aprofundada pode ser encontrada no trabalho de Bolme et al. (2010), é dada por:

$$H^* = \frac{\sum_i Y_i \odot X_i^*}{\sum_i X_i \odot X_i^*} \tag{2.8}$$

onde letras maiúsculas denotam as Transformadas Discretas de Fourier (DFTs) dos sinais correspondentes; os operadores $*$ e $\odot$ denotam conjugado complexo e multiplicação elemento a elemento, respectivamente.

Para realizar a detecção de um objeto em uma imagem X (transformada para domínio da frequência), basta aplicar o filtro H^*, treinado para detectar o objeto alvo, a imagem de entrada X, assim gerando uma resposta s denominada de mapa de pontuação ou mapa de resposta:

$$s = \mathcal{F}^{-1}\{H^* \odot X\} \tag{2.9}$$

A Transformada Inversa de Fourier nesta resposta da Equação 2.9 produzirá uma pontuação que idealmente é uma função Gaussiana com seu pico centrado no local do objeto alvo (ROBINSON, 2021).

2.3.2 O Algoritmo de Rastreamento

Inicialmente, o filtro de correlação é treinado a partir do primeiro *frame* e saída desejada obtida por meio do *ground truth* do objeto alvo.

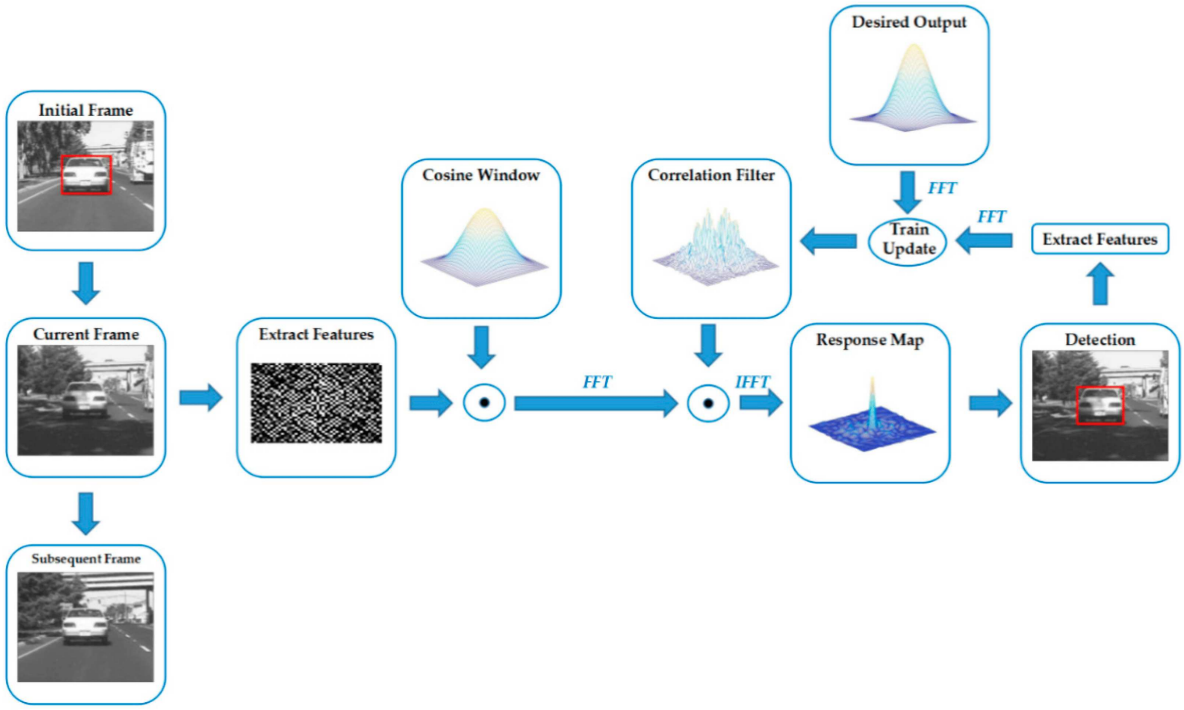
Em seguida, *features* (feições) são extraídas do *frame* atual e multiplicadas por uma função de janelamento, como a janela de cosseno, para destacar a região central e eliminar descontinuidades nas bordas da imagem. Essas *features* são então transformadas para o domínio da frequência por meio da FFT. Um mapa de resposta é gerado multiplicando o filtro de correlação pelas *features* extraídas. Na sequência, esse mapa de resposta é transformado para o domínio do tempo aplicando a IFFT (BARCELLOS, 2021).

Por fim, a posição correspondente ao valor máximo no mapa de resposta é considerada como a posição central do objeto alvo no *frame* atual. E então novas features são extraídas do resultado detectado para treinar e atualizar o filtro de correlação.

Vale destacar que a atualização do filtro de correlação é uma etapa fundamental, uma vez que, à medida que o rastreamento progride, a aparência tanto do alvo quanto do fundo inevitavelmente se modificará. Portanto, se o filtro de correlação não for ajustado conforme necessário, haverá uma probabilidade crescente de falha ao longo do tempo (ROBINSON, 2021).

As etapas descritas do método de rastreamento baseado em filtros de correlação são ilustradas na Figura 9.

Figura 9 – Visão geral da abordagem de rastreamento baseado em filtros de correlação.



Fonte: Yang et al. (2019).

2.4 Rastreamento de Objetos: SORT Tracker

O SORT (*Simple Online and Realtime Tracking*) é uma abordagem de rastreamento de múltiplos objetos (*Multiple Object Tracking - MOT*) voltado para aplicações em tempo real. Proposto por Bewley et al. (2016), o SORT utiliza a combinação de técnicas conhecidas para lidar com o rastreamento de objetos em uma sequência de frames.

Antes de abordar o algoritmo de rastreamento SORT, serão apresentados o filtro de Kalman e o problema de designação, mais conhecido como *Assignment Problem*, uma vez que estes dois conceitos são a base para o funcionamento do SORT.

2.4.1 Filtro de Kalman

Proposto em 1960 por Rudolph Emil Kalman em seu artigo *"A New Approach to Linear Filtering and Prediction Problems"*, o filtro de Kalman consiste em uma abordagem recursiva de filtragem linear de dados discretos. Em outras palavras, o filtro de Kalman é um algoritmo que utiliza uma série de medições coletadas ao longo do tempo, incluindo ruído e outras imprecisões, e gera estimativas que tendem a se aproximar dos valores reais das grandezas medidas, bem como tendem a ser mais precisas que as abordagens baseadas em apenas uma medição. Para tanto, o filtro de Kalman aplica um conjunto de equações matemáticas que fornecem uma solução computacional (recursiva) eficiente do método dos mínimos quadrados, que busca encontrar o melhor ajuste para um conjunto de dados (WELCH; BISHOP, 2001).

Por ter a capacidade de ser processado em tempo real, rápido, eficiente e forte anti-interferência, o filtro Kalman tem sido amplamente aplicado nas áreas de cálculo de órbita, rastreamento de alvos e navegação. Além disso, também desempenha um papel importante nas áreas de navegação integrada e posicionamento dinâmico, fusão de dados de sensores, microeconomia e, especialmente, na área de processamento digital de imagens, onde tem sido aplicado no reconhecimento de padrões, segmentação de imagens e detecção de bordas de imagens (LI et al., 2015).

2.4.1.1 O algoritmo de filtro discreto de Kalman

Segundo Welch e Bishop (2001), o filtro de Kalman estima um processo usando uma forma de controle de *feedback*, ou seja, o filtro estima o estado do processo em algum momento e então obtém *feedback* na forma de medições ruidosas. Sendo assim, as equações para o filtro de Kalman se enquadram em dois grupos: equações de atualização de tempo (equações preditoras) e equações de atualização de medição (equações corretoras). As equações de atualização de tempo são responsáveis por projetar adiante, no tempo, o estado atual e as estimativas de covariância de erros (incertezas de medição) para obter as estimativas a priori para a próxima etapa de tempo. Já as equações de atualização de

medição são responsáveis pelo *feedback*, ou seja, servem para incorporar uma nova medição na estimativa a priori para conseguir obter uma estimativa a posteriori melhorada.

As Equações 2.10 e 2.11 a seguir, referentes à atualização de tempo, projetam as estimativas de estado $\hat{x}_k^-$ e de covariância P_k^- adiante no tempo, isto é, do tempo $k-1$ para o tempo k.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \tag{2.10}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{2.11}$$

Nas Equações 2.10 e 2.11 acima, A representa a matriz de transição de estado; B representa a matriz que relaciona a entrada de controle opcional u_k ao estado x; Q matriz que representa a covariância de ruído do processo. Destaca-se que as matrizes A, B e Q dependem das características modelo do sistema.

Para a atualização de medidas, temos a Equação 2.12 que expressa a atualização do ganho de Kalman K_k, a Equação 2.13 que expressa a atualização do estado $\hat{x}_k$ e, por último, a Equação 2.14 que denota a atualização da covariância do erro P_k.

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{2.12}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{2.13}$$

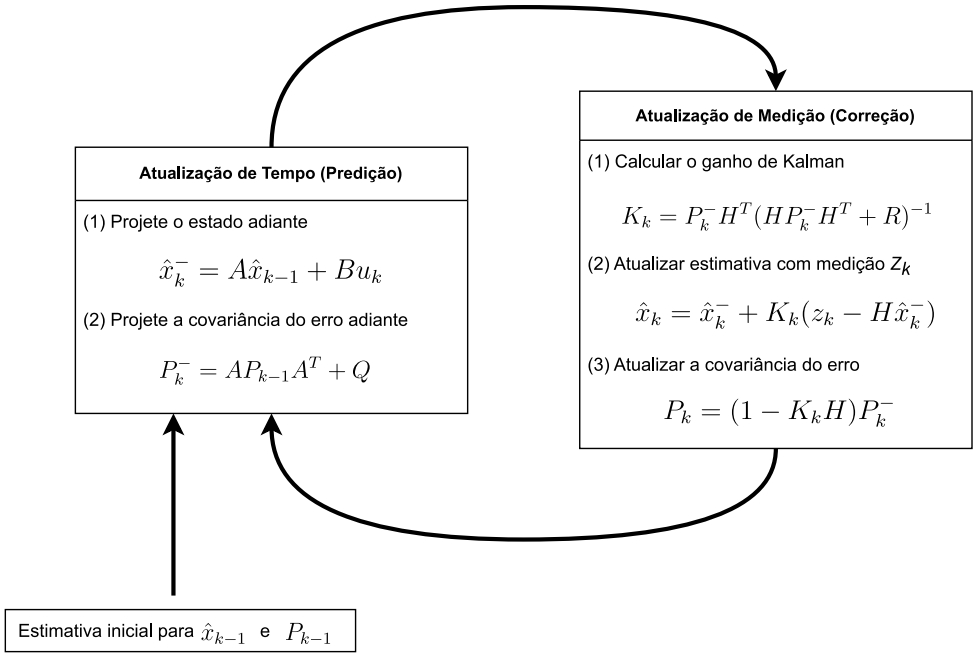
$$P_k = (1 - K_k H)P_k^- \tag{2.14}$$

Nas Equações 2.12, 2.13 e 2.14 acima, H representa a matriz que relaciona o estado à medição z_k; R representa a matriz de covariância do erro de medição. Ambas matrizes, H e R, também dependem do modelo do sistema.

Em suma, após cada par de atualização de tempo e medição, o processo é repetido com as estimativas anteriores a posteriori usadas para projetar as novas estimativas a priori. Para Welch e Bishop (2001), esta natureza recursiva é uma das características que torna as implementações do filtro de Kalman mais viáveis e práticas, uma vez que não é necessário utilizar todos os dados coletados para cada estimativa. Em vez disso, o filtro de Kalman condiciona recursivamente a estimativa atual em todas as medições anteriores.

Para facilitar a compreensão do algoritmo do filtro de Kalman, Welch e Bishop (2001) propõe uma abstração do funcionamento do filtro baseada nos algoritmos do tipo preditor-corretor. Essa abstração é ilustrada no diagrama da Figura 10.

Figura 10 – Funcionamento do filtro de Kalman.



Adaptado de Welch e Bishop (2001).

2.4.2 Problema de Designação

O problema de designação (*Assignment Problem*) é um dos problemas mais conhecidos da programação linear e otimização combinatória. O problema consiste em, dada uma matriz de custos C de tamanho $n \times n$, combinar cada linha com uma coluna diferente de forma que a soma dos custos correspondentes da matriz seja minimizada. Em outros termos, queremos selecionar n elementos de C de modo que haja exatamente um elemento em cada linha e um em cada coluna e a soma dos custos correspondentes seja mínima possível (BURKARD; DELL'AMICO; MARTELLO, 2012).

De forma alternativa, o problema de designação pode ser modelado como um problema da teoria dos grafos. Dado um grafo bipartido $G = (U, V, E)$, podemos representar o conjunto de vértices U como as linhas da matriz C, o conjunto de vértices V como as colunas da matriz C e o conjunto de arestas E como as associações entre vértices do conjunto U e V de custo c_{ij} relativo à aresta $[i, j](i, j = 1, 2, ..., n)$ (BURKARD; DELL'AMICO; MARTELLO, 2012). Dessa forma, o problema é então determinar um emparelhamento perfeito de custo mínimo(ou máximo) em G. Na teoria dos grafos esse problema é chamado de emparelhamento máximo em grafos bipartidos (*Bipartite Maximum-Weight Matching Problem*) e tem como objetivo encontrar um subconjunto de arestas tal que cada vértice pertença a exatamente uma aresta e a soma dos custos dessas arestas seja o mínimo.

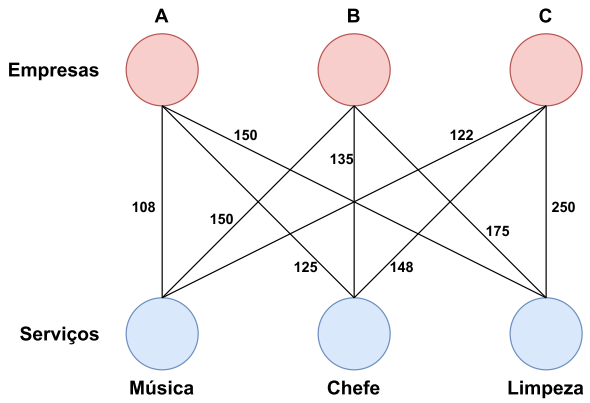
Para exemplificar o problema de designação, considere um cenário no qual uma pessoa pretende fazer uma festa e deseja contratar um músico para se apresentar, um

chefe de cozinha para preparar a comida e um serviço de limpeza para ajudar na limpeza depois da festa. Considere também que há três empresas que oferecem esses três tipos de serviços, mas cada empresa pode fornecer apenas um serviço por vez. Nesse cenário, ilustrado na Figura 11, o problema de designação traduz-se na escolha de cada serviço de forma a minimizar o custo da festa.

Figura 11 – (a) Representação matricial do problema. (b) Representação por grafo bipartido.

Empresa / Custo	Custo para Músico	Custo para Chef	Custo para Limpeza
Empresa A	$108	$125	$150
Empresa B	$150	$135	$175
Empresa C	$122	$148	$250

(a)



(b)

Adaptado de Moore, Landman e Khim (2023).

Uma forma de tentar resolver esse problema é adotar uma abordagem gananciosa simples, onde escolhe-se, dentre os serviços que ainda não foram escolhidos, o serviço de menor custo de uma empresa que ainda não foi utilizada. Sendo assim, seguindo em ordem alfabética dos nomes das empresas, é contratado o serviço de música da "Empresa A", o chefe da "Empresa B" e o serviço de limpeza da "Empresa C", deste modo, totalizando um custo de $493.

No entanto, ao adotar uma abordagem de busca completa, na qual todas as combinações possíveis são listadas e a combinação com o menor custo final é selecionada, obtém-se uma solução diferente da encontrada com abordagem gananciosa. Nesse caso, temos um total de 6(3!) combinações possíveis de escolha. E ao listar as 6 combinações, verifica-se que a melhor solução é contratar o serviço de música da "Empresa C", o chefe da "Empresa B" e o serviço de limpeza da "Empresa A", assim, totalizando um custo de $407.

Dito isso, é possível constatar que a abordagem baseada em busca completa sempre é capaz de produzir a melhor solução global para o problema de designação, porém ao custo de complexidade de tempo $O(n!)$. Já a abordagem gananciosa possui o custo de tempo $O(n^2)$, mas não é capaz de produzir a melhor solução global para o problema.

Tendo isso em vista, algoritmos com melhor complexidade de tempo foram propostos para resolver o problema de designação de forma ótima. Dentre eles, destacam-se

o algoritmo Húngaro e o algoritmo LAPJV (*Jonker-Volgenant for Linear Assignment Problem*).

2.4.3 Algoritmos para o Problema de Designação

Uma vez que o problema de designação pode ser facilmente abstraído como problema de emparelhamento em grafos bipartidos, os algoritmos serão apresentados com o auxílio de conceitos e propriedades provenientes da teoria dos grafos.

2.4.3.1 Emparelhamento em Grafos Bipartidos

Um emparelhamento (*matching*) em um grafo G é um conjunto de arestas M tal que não existem duas arestas adjacentes neste conjunto, ou seja, todo vértice de G incide em no máximo um elemento de M.

Dito isso, um emparelhamento M em um grafo G é dito máximo se ele contém o maior número possível de arestas sem que a propriedade de não-adjacência entre as arestas seja quebrada (Figura 12 c). Mais formalmente, um emparelhamento M é máximo se não existe um emparelhamento M' tal que $|M'| > |M|$.

Já em um grafo ponderado, um emparelhamento de peso máximo é um emparelhamento cuja soma dos pesos das arestas é o maior possível. O contrário vale para o emparelhamento de peso mínimo.

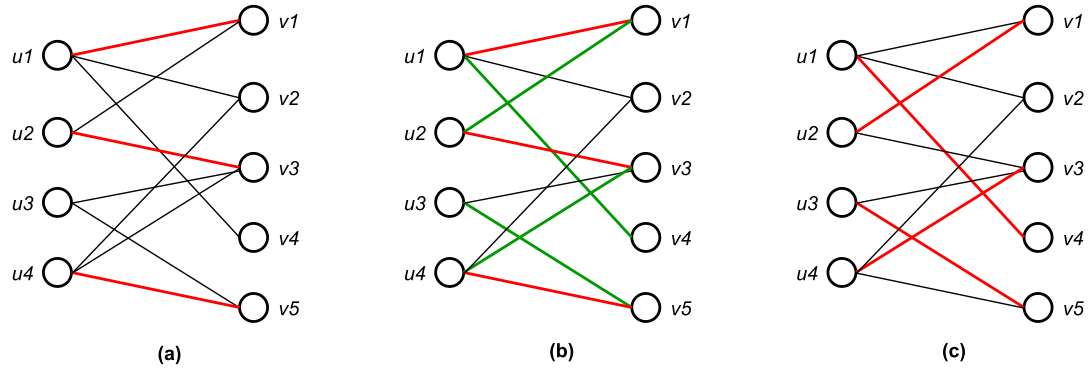
2.4.3.2 Caminhos alternantes e Caminhos de aumento

Os algoritmos de emparelhamento de grafos normalmente utilizam propriedades específicas para identificar regiões subótimas em um emparelhamento, onde melhorias podem ser feitas para atingir o objetivo desejado. Duas propriedades importantes são chamadas de caminhos de aumento e caminhos alternantes, que são usadas para determinar rapidamente se um emparelhamento M é máximo ou mínimo, ou se o emparelhamento pode ser melhorado ainda mais (MOORE; LANDMAN; KHIM, 2023).

Um caminho alternante em um emparelhamento M é um caminho no qual as arestas alternam entre as que estão no conjunto M e as que não estão no conjunto M. E um caminho alternante, cujos vértices de início e fim são livres/não-saturados (vértices sem incidência das arestas de M), é denominado como caminho de aumento de M.

Quando existe um caminho de aumento A em M, temos a oportunidade de incrementar o tamanho do emparelhamento em uma unidade, substituindo as arestas desse caminho que não pertencem a M pelas que pertencem, conforme ilustrado na Figura 12 (b) e (c). Além disso, é importante observar que a ausência de caminhos de aumento é uma característica fundamental do emparelhamento máximo.

Figura 12 – (a) Emparelhamento inicial. (b) Caminho de aumento partindo de $u3$ e chegando em $v4$. (c) Emparelhamento máximo obtido pelo caminho de aumento (b).



2.4.3.3 Algoritmo Húngaro

Proposto originalmente por Kuhn (1955) e aperfeiçoado por Munkres (1957), o algoritmo Húngaro, também conhecido como algoritmo de Kuhn-Munkres, resolve o problema de designação em complexidade de tempo $O(n^3)$.

Segundo Moore, Landman e Khim (2023), o algoritmo opera encontrando caminhos de aumento para alcançar um emparelhamento máximo. Em termos mais formais, o algoritmo busca construir um emparelhamento maior a partir de um emparelhamento atual, M, procurando caminhos de aumento. Cada vez que um caminho de aumento é encontrado, o número de emparelhamentos (ou custo total) é incrementado. A principal ideia por trás desse algoritmo é expandir M pelo caminho de aumento mais curto, garantindo que nenhuma restrição seja violada.

O algoritmo começa com um emparelhamento inicial, que pode ser aleatório ou vazio. Em seguida, utiliza uma busca em largura (*Breadth First Search - BFS*) para construir uma árvore, procurando por caminhos de aumento. Quando um caminho de aumento é encontrado, o emparelhamento é atualizado, adicionando uma nova aresta a ele. O algoritmo prossegue repetindo esse processo, em busca de novos caminhos de aumento. Se a busca não conseguir encontrar mais caminhos de aumento, o algoritmo encerra sua execução, pois o emparelhamento atingiu o tamanho máximo possível (MOORE; LANDMAN; KHIM, 2023).

2.4.3.4 Algoritmo LAPJV

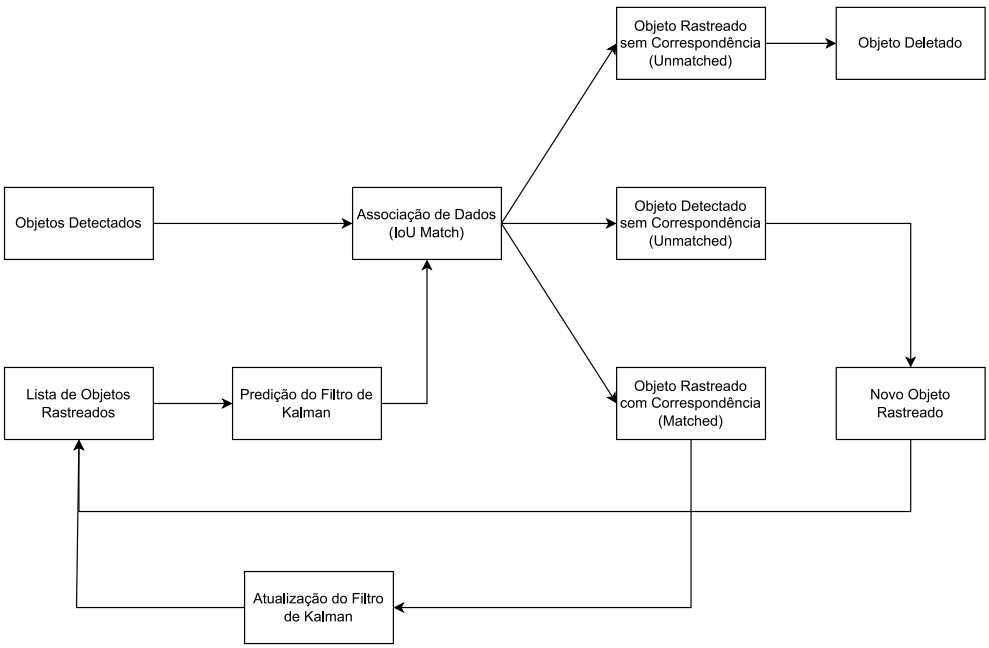
Publicado por Jonker e Volgenant (1987), o algoritmo LAPJV (*Jonker-Volgenant for Linear Assignment Problem*) possui uma notável eficiência na resolução do problema de designação, seja para problemas densos ou esparsos, superando em velocidade os principais algoritmos da literatura, incluindo o algoritmo Húngaro, embora ambos compartilhem a mesma complexidade temporal $O(n^3)$.

Para tanto, o LAPJV implementa a busca do caminho de aumento mais curto de forma mais eficiente que a adotada no algoritmo Húngaro, utilizando uma implementação especial do algoritmo do caminho mais curto de Dijkstra. Além disso, o LAPJV conta também com rotinas de inicialização e pré-processamento que contribuem para a desempenho do algoritmo.

2.4.4 O Algoritmo de Rastreamento

O rastreador de objetos SORT é composto por quatro componentes principais, sendo eles: detecção; modelo de estimativa; associação de dados; criação e exclusão de identidades de rastreamento. O processo de interação desses componentes é ilustrado pela Figura 13 abaixo.

Figura 13 – Diagrama de representação do funcionamento do SORT.



Adaptado de Liu e Juang (2021).

A detecção de objetos é o primeiro passo do rastreador e consiste em detectar as posições dos objetos de interesse em um *frame*. A detecção provê uma caixa delimitadora para cada objeto detectado, assim, possibilitando o rastreamento do objeto durante todo o seu ciclo de vida. Dentre os detectores de objetos utilizados, podemos destacar o YOLO(*You Only Look Once*), SSD(*Single Shot Detector*) e *Faster R-CNN*.

Na etapa do modelo de estimativa, as detecções do *frame* atual são propagadas para o próximo *frame*. Para tanto, é empregado o filtro de Kalman com um modelo linear de velocidade constante entre os *frames*. Ademais, o estado de cada objeto rastreado é modelado com as informações de posição, tamanho e velocidade da caixa delimitadora que circunscreve o objeto. Quando uma detecção é associada a um objeto já rastreado,

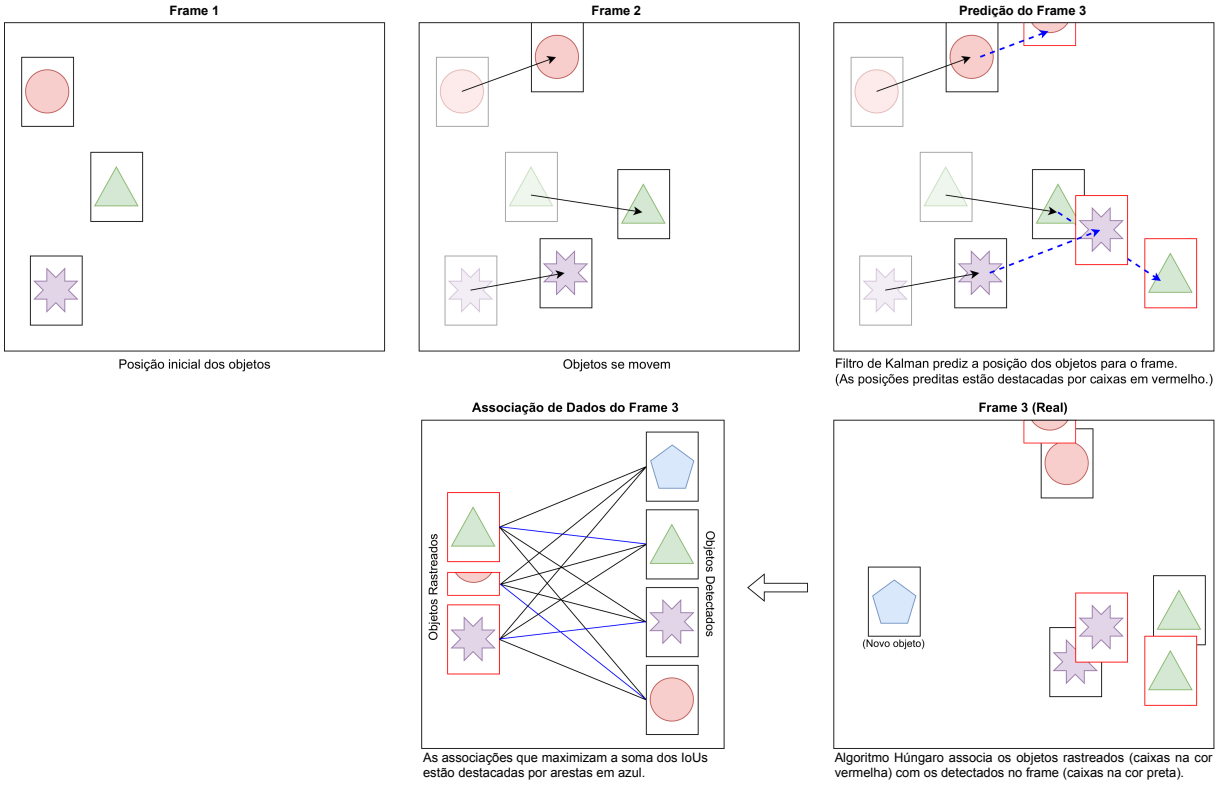
a caixa delimitadora é usada para atualizar o estado do objeto. Se não houver detecção, então o estado é simplesmente o previsto sem atualização de estado (correção).

Então, a partir das caixas delimitadoras dos objetos rastreados e detectados é realizada a associação de dados. Essa etapa consiste em calcular uma matriz de custos a partir das distâncias de Interseção sobre União (*Intersection over Union - IoU*) entre cada detecção e todas as caixas delimitadoras previstas dos objetos rastreados existentes. Dessa forma, as associações são feitas de modo a maximizar a soma dos IoUs entre as caixas delimitadoras dos objetos rastreados e detectados. Tal tarefa é resolvida de forma otimizada aplicando um dos algoritmos apresentados na Seção 2.4.3, como o algoritmo Húngaro.

Por fim, quando objetos entram e saem da imagem, identidades únicas são criadas ou excluídas de acordo com a ação ocorrida. Para tanto, é utilizado o módulo de criação e exclusão de identidades de rastreamento. Mais especificamente, quando um objeto detectado não é associado a nenhum outro objeto já rastreado, uma nova identidade de rastreamento é definida para esse objeto. Já quando um objeto rastreado for associado a uma detecção na qual o IoU for menor que um valor limite denominado IoU_{min}, essa associação será rejeitada, assim indicando que o objeto rastreado não foi detectado no *frame* em questão. E caso um objeto rastreado não seja detectado nos próximos T_{lost} *frames*, a identidade do objeto será excluída e o objeto não será mais rastreado.

O funcionamento do SORT pode ser melhor visualizado na Figura 14, na qual é apresentado o comportamento do algoritmo ao rastrear um conjunto de objetos em uma sequência de *frames*.

Figura 14 – Ilustração do rastreamento de objetos com o SORT.



2.5 Rastreamento de Objetos: Centroid Tracker

Centroid Tracker (Rastreador Centróide) é uma abordagem de rastreamento de objetos que se baseia no centróide dos objetos detectados em uma *stream* de vídeo (ROSEBROCK, 2018). O funcionamento básico do *Centroid Tracker* é detalhado na enumeração de passos a seguir e pode ser visualizado na Figura 15.

1. A detecção de objetos: a primeira etapa é detectar objetos em cada *frame* (quadro) do vídeo. Isso pode ser feito usando técnicas de detecção de objetos como SSD, YOLO, Faster R-CNN, entre outros.

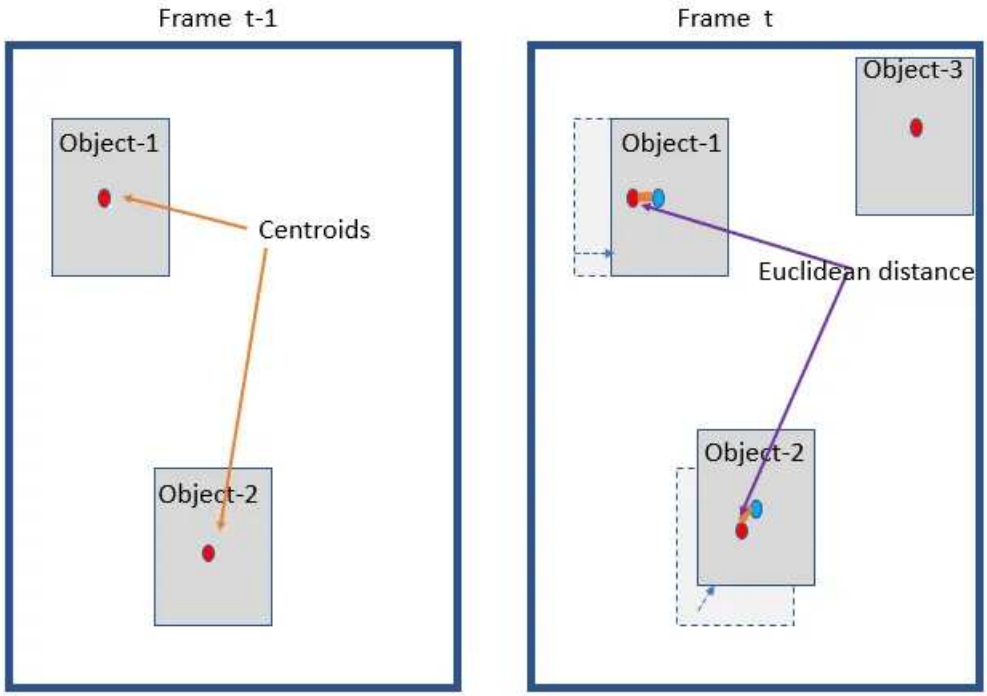
2. Cálculo do centróide: para cada objeto detectado, é calculado o centro geométrico (ou centróide) da forma do objeto.

3. Associação de objetos: na etapa seguinte, o *Centroid Tracker* associa os objetos detectados em cada novo *frame* com os objetos rastreados no *frame* anterior (Figura 15). Isso é feito comparando as distâncias euclidianas entre os centróides dos objetos detectados no novo *frame* e os centróides dos objetos rastreados no *frame* anterior. Então, a partir de um distância *threshold* (distância limite), é possível determinar se houve uma correspondência entre objetos do *frame* novo e os objetos do *frame* anterior.

4. Atualização do rastreamento: se um objeto é identificado como o mesmo objeto rastreado no *frame* anterior, suas coordenadas são atualizadas com as coordenadas do centróide do objeto detectado no novo *frame*. Se um objeto não é associado a nenhum objeto rastreado no *frame* anterior, ele é considerado um novo objeto e é adicionado ao conjunto de objetos rastreados. Por fim, se não há correspondência entre os objetos detectados no *frame* anterior e os objetos rastreados no novo *frame*, o objeto é considerado fora de alcance e é removido da lista de objetos rastreados.

[3] Disponível em: <https://medium.com/aiguys/a-centroid-based-object-tracking-implementation-455021c2c997>. Acesso em: 01 de fevereiro de 2023.

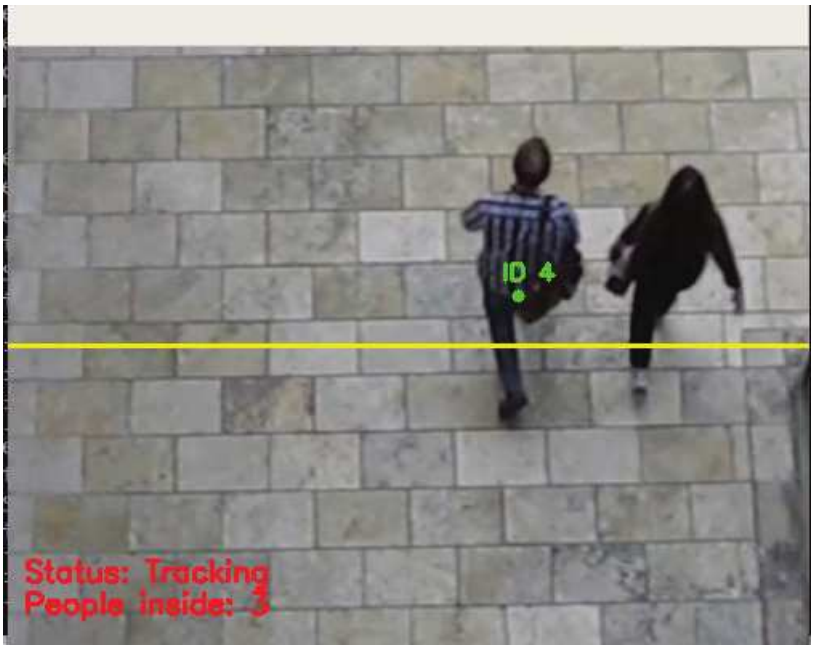
Figura 15 – Associação de objetos entre frames consecutivos. [3]



2.6 Contabilizando Pessoas

Uma abordagem possível para realizar a contagem de pessoas em um ambiente, após a detecção e o rastreamento, é proposta por Amin et al. (2021) e consiste em utilizar o auxílio de linhas virtuais como barreiras que separam a parte externa e interna do ambiente (Figura 16).

Figura 16 – Representação da contagem de pessoas com auxílio de linhas virtuais.



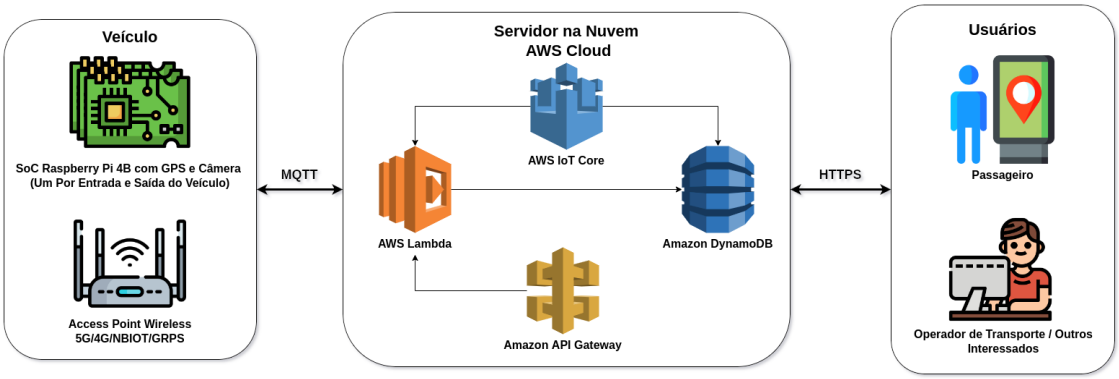
Fonte: Amin et al. (2021).

Dessa forma, se uma pessoa entrar no ambiente e cruzar a barreira, a contagem aumentará e se uma pessoa sair do ambiente e cruzar a barreira, a contagem diminuirá (AMIN et al., 2021).

3 Especificação do Sistema

A arquitetura geral do sistema é representada na Figura 17, onde cada componente do sistema será descrito nas seções deste capítulo.

Figura 17 – Representação da arquitetura da solução.



3.1 Materiais

3.1.1 Raspberry Pi 4B

O SoC (*System on Chip*) utilizado neste trabalho é a *Raspberry Pi 4* modelo B (Figura 18) na versão de 8 GB. O modelo possui um processador *quad-core* Cortex-A72 (ARM v8) 64-bit. Cada dispositivo é responsável por realizar a contagem de passageiros em uma porta (entrada e/ou saída) de um veículo. Inicialmente, o sistema operacional utilizado pela Raspberry Pi será o Ubuntu 22.04.1.

Figura 18 – Raspberry Pi 4 modelo B. [1]

[1] Disponível em: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>. Acesso em: 01 de fevereiro de 2023.

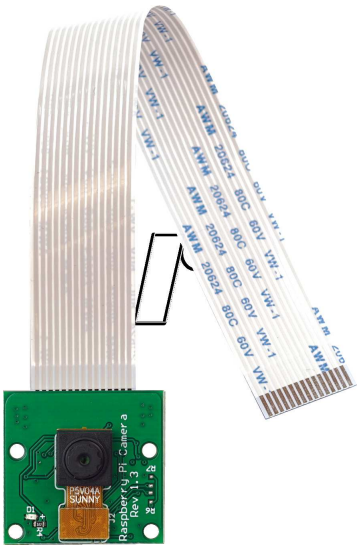
3.1.2 Módulo de GPS

O módulo de GPS (*Global Positioning System*), conectado em cada *Raspberry Pi*, será utilizado para combinar os dados gerados pelos eventos de contagem com o local onde aconteceram os eventos.

3.1.3 Módulo de Câmera

O módulo de câmera será utilizado para coletar as imagens que serão processadas na *Raspberry Pi*, dessa forma, fornecendo as entradas para o algoritmo de contagem de passageiros. O módulo escolhido para este trabalho é o *RPI Cam v1.3* (Figura 19), o qual possui uma interface de comunicação CSI (*Camera Serial Interface*). Além disso, o módulo é capaz de fornecer as seguintes taxas de *frame* por segundo: 30 *fps* para vídeos em 1080p; 60 *fps* para vídeos em 720p; 60/90 *fps* para vídeos em 480p.

Figura 19 – Módulo Raspberry Pi Cam v1.3. [2]



3.1.4 Access Point Wireless

Cada veículo deverá contar com um *Access Point Wireless* (Ponto de Acesso Sem Fio), assim possibilitando o acesso à internet as *Raspberry Pi* instaladas nas entradas e/ou saídas de um veículo. Para que isso seja possível, os *Access Points* devem possuir *Wi-fi* (*Wireless Fidelity*) e um modem de acesso às redes de telefonia celular, como 5G, 4G e NBIoT (*Narrowband Internet of Things*).

[2] Disponível em: <https://www.reichelt.com/de/en/raspberry-pi-camera-5mp-v1-3-rpi-cam-5mp-p314570.html>. Acesso em: 01 de junho de 2023.

3.2 Software Embarcado

A partir da vista superior de uma porta de um veículo, o *software* embarcado na *Raspberry Pi* deverá realizar a contabilização das pessoas que embarcam e desembarcam pela porta. Então, por meio da rede *Wifi* fornecida pelo *Access Point*, a *Raspberry Pi* transmite os dados de embarque e desembarque de passageiros do veículo para o servidor central. Cada *Raspberry Pi* é responsável pela contabilização parcial de passageiros no veículo. Assim, ficando a cargo do servidor na nuvem combinar as contabilizações parciais de forma a gerar a contabilização final de passageiros no veículo.

O sistema embarcado será implementado utilizando a linguagem de programação Python e fará uso das bibliotecas *OpenCV* e *Tensorflow* para auxiliar no desenvolvimento do algoritmo de contagem de passageiros.

3.3 Servidor na Nuvem

A partir das contabilizações parciais de passageiros recebidas, o servidor central deverá realizar as contabilizações finais de passageiros nos veículos, bem como armazenar e disponibilizar essas contabilizações. Para facilitar a implantação do servidor, será utilizado as soluções de nuvem da Amazon, chamadas de *Amazon Web Services (AWS)*.

3.3.1 AWS IoT Core

A AWS IoT fornece serviços de nuvem que conectam dispositivos de IoT (*Internet of Things*) aos serviços de nuvem da AWS.

Para conectar dispositivos de contagem de passageiros em campo, será utilizado o protocolo de mensageria MQTT (*Message Queuing Telemetry Transport*), suportado pela *AWS IoT Core*. Então, os dados publicados pelos dispositivos são enviados para um no banco de dados *DynamoDB*.

3.3.2 Amazon DynamoDB

O *Amazon DynamoDB* é um banco de dados de chave-valor *NoSQL*, *serverless* (sem servidor) e totalmente gerenciado.

Dito isso, o banco de dados *DynamoDB* será utilizado para armazenar os dados de contagem de passageiros recebidos, bem como disponibilizá-los para consultas posteriores.

3.3.3 AWS Lambda

AWS Lambda é um serviço de computação serverless que permite a execução de código sem a necessidade de provisionamento ou gerenciamento de servidores.

No contexto deste trabalho, a *AWS Lambda* será utilizada para realizar as contabilizações finais de passageiros nos veículos, assim como executar códigos que realizam pré-processamento das requisições e consultas no banco de dados *DynamoDB* solicitadas pelo *API Gateway*.

3.3.4 Amazon API Gateway

O *Amazon API Gateway* é uma plataforma de serviços da AWS que permite criar, publicar e gerenciar APIs (*Application Programming Interfaces*) de maneira simples e segura. Além disso, o *Amazon API Gateway* permite a integração com outros serviços AWS, como *Lambda* e *DynamoDB*, assim proporcionando soluções completas de API sem ter que gerenciar servidores.

O *API Gateway* da Amazon será utilizado como interface entre as requisições HTTPS (*Hyper Text Transfer Protocol Secure*) dos usuários e a função Lambda que atende essas requisições por meio de consultas no banco de dados *DynamoDB*.

4 Desenvolvimento

4.1 Seleção de Datasets

4.1.1 Contato com a Onboard Mobility e a Viação Breda

Por meio do contato com a empresa de soluções de mobilidade urbana *Onboard Mobility*, na qual realizei estágio no ano de 2020, foi possível obter acesso a filmagens cedidas pela Viação Breda. As filmagens são da operação de Peruíbe (município do Estado de São Paulo) e capturaram campos de visão superior das portas de embarque e desembarque dos veículos de transporte de passageiros (Figura 20).

Figura 20 – Frames retirados das filmagens cedidas pela Viação Breda.

Fonte: Viação Breda.

4.1.2 Contato com a SEMOB-DF

Para obter acesso a filmagens contemplando os interiores dos veículos de transporte público do Distrito Federal, foi realizada uma reunião com funcionários de diferentes setores da SEMOB-DF (Secretaria de Transporte e Mobilidade do Distrito Federal) para apresentar a solução de contagem de passageiros proposta. Segundo os funcionários a solução é bastante pertinente. Porém, para que fosse possível conceder o acesso às filmagens, os funcionários solicitaram uma nova apresentação que demonstrasse uma primeira versão da solução funcionando.

Além disso, durante a reunião, os funcionários da SEMOB-DF fizeram algumas ponderações importantes que devem ser consideradas durante o desenvolvimento da solução. As principais ponderações feitas são as seguintes:

- Soluções que realizam apenas estimativas de quantidade de passageiros no veículo devem ser evitadas, uma vez que não é possível obter dados que permitam entender a demanda de transporte de forma precisa. Esse tipo de solução é característica de abordagens que utilizam apenas uma câmera em ambientes onde pode haver oclusões.
- Existem situações onde pode haver um falso embarque de passageiros e isso não deve afetar a contagem. Uma situação típica de falso embarque acontece quando um passageiro embarca no veículo apenas para tirar uma dúvida com o motorista e depois desembarca.

Após a descoberta do *dataset* PCDS, apresentado na Seção 4.1.3, optou-se por utilizar apenas os *datasets* que já foram obtidos. Ainda sim, a reunião realizada com a SEMOB-DF foi de muita importância para o desenvolvimento deste trabalho, tendo em vista as considerações ponderadas durante a reunião.

4.1.3 People Counting Dataset (PCDS)

O *People Counting Dataset (PCDS)* é um *dataset* público [1] criado por Sun et al. (2019) que consiste em um conjunto de filmagens gravadas sobre as portas de entrada e saída do ônibus pela câmera *Kinect V1* (Figura 21). Este *dataset* contém 5.464 pares de vídeos, sendo que cada par é composto por um vídeo de profundidade e um vídeo *RGB(Red, Green, Blue)*. Além disso, o *dataset* conta com vídeos no qual há multidão embarcando e desembarcando, bem como vídeos onde não há multidão. O *dataset* conta também com vídeos com luminosidade solar alta e luminosidade solar baixa. De acordo com Sun et al. (2019), o *dataset* foi gravado nas cidades Xi'An, XiNing e YinChuan, na China, no qual cerca de 20.908 pessoas passam pelo local das gravações.

Figura 21 – Frames retirados do *dataset* PCDS.

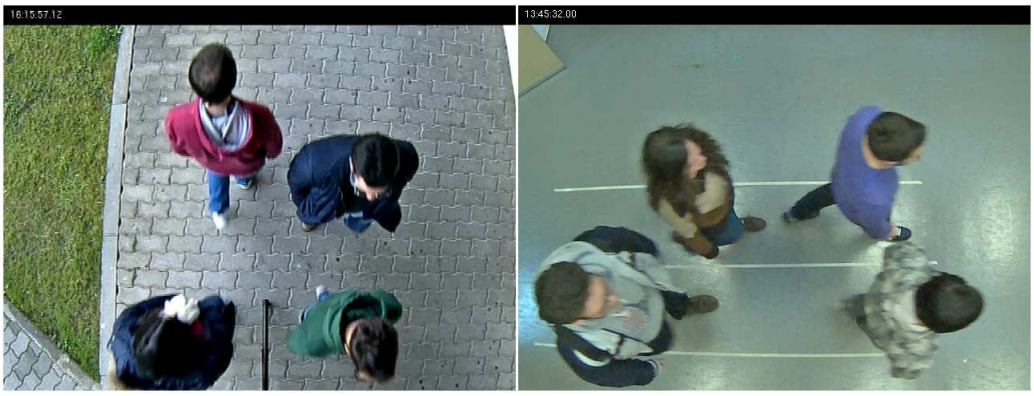
Fonte: Sun et al. (2019).

[1] Disponível em: <https://github.com/shijieS/people-counting-dataset>. Acesso em: 19 de junho de 2023.

4.1.4 Dataset Público do Laboratório MIVIA

O laboratório de pesquisa na área de reconhecimento de padrões e visão computacional da Universidade de Salerno, chamado MIVIA, criou um *dataset* público [2] constituído de filmagens que capturaram campos de visão superior em diferentes ambientes (Figura 22).

Figura 22 – Frames retirados do *dataset* do laboratório MIVIA.



Fonte: Laboratório MIVIA.

4.1.5 Human Detection Dataset

O *Human Detection Dataset* é um *dataset* público [3] do Kaggle e foi criado por Verner (2022)(Figura 23). Segundo o autor, o *dataset* é composto por: imagens de CFTV do Youtube; *dataset* de imagens internas(*indoor*) abertas; imagens de CFTV do próprio autor.

Figura 23 – Imagens retiradas do *dataset Human Detection* do Kaggle.

Fonte: Verner (2022).

[2] Disponível em: <https://mivia.unisa.it/people-detection-dataset/>. Acesso em: 28 de janeiro de 2023.
[3] Disponível em: <https://www.kaggle.com/datasets/constantinwerner/human-detection-dataset>. Acesso em: 26 de maio de 2023.

4.2 Rotulagem do Dataset

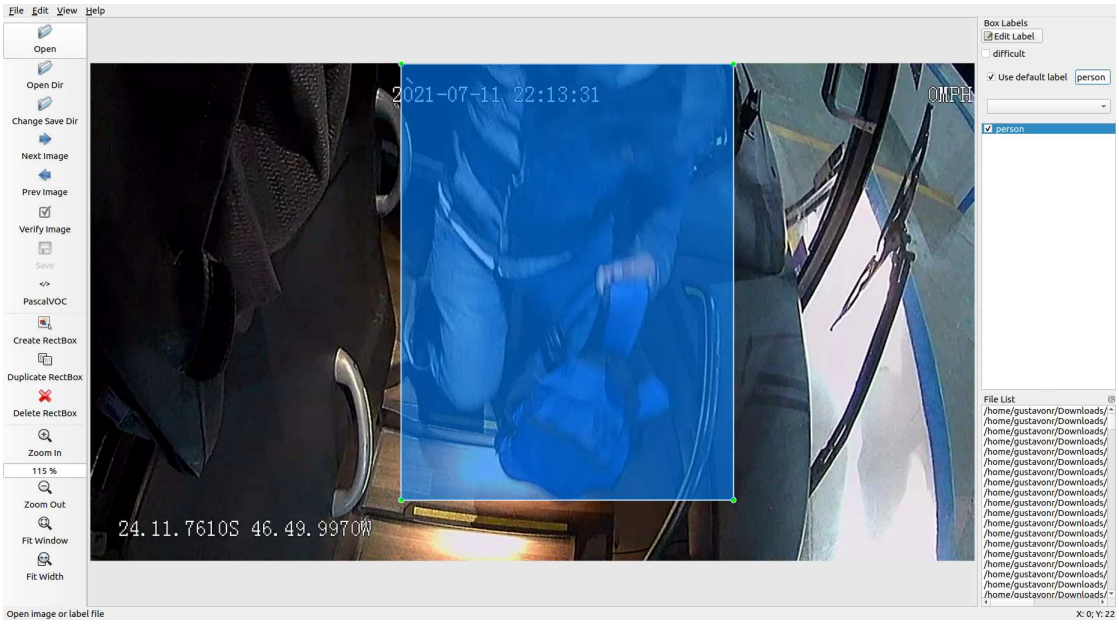
A partir dos *datasets* apresentados na Seção 4.1, foi elaborado um novo *dataset* para o treinamento e validação do modelo de detecção de objetos SSD. A construção do *dataset* consistiu nas seguintes etapas:

1. *Download* das filmagens/imagens dos *datasets*.

2. Seleção de imagens a partir das filmagens baixadas. Esta etapa foi feita com auxílio do *software VLC Media Player* [4].

3. Demarcação das caixas delimitadoras entorno das pessoas presentes nas imagens selecionadas. Para tanto, foi utilizado o *software LabelImg* [5] (Figura 24).

Figura 24 – *Software* de rotulagem de imagens *LabelImg*.



O processo de seleção das imagens foi guiado com base nas seguintes considerações: evitar seleção de imagens parecidas ou idênticas; selecionar imagens com e sem pessoas para cada *dataset*; selecionar imagens com diferentes tipos de fundo, luminosidade e contraste.

Quanto ao processo de demarcação das caixas delimitadoras, as considerações tomadas foram: inserir caixas delimitadoras para todas as pessoas presentes em cada imagem, ou seja, não apenas para a pessoa que estiver em predominância na imagem; contornar com caixas delimitadoras toda área visível de cada pessoa identificada.

[4] VLC Media Player: <https://www.videolan.org/vlc/index.html>.
[5] Software LabelImg: <https://github.com/heartexlabs/labelImg>.

Duas versões do *dataset* foram construídas, sendo que a primeira versão ainda não contava com o *dataset* PCDS. Além disso, cada versão do *dataset* gerou o seu respectivo modelo de detecção de objetos SSD, os quais serão apresentados nas seções seguintes. A Tabela 1 apresenta a composição das versões, indicando o nome do *dataset* de origem e a quantidade de imagens selecionadas.

Tabela 1 – Composição dos *datasets* construídos.

Versão	**Kaggle**	**MIVIA**	**Viação Breda**	**PCDS**	**Total**
v1	45	194	177	-	416
v2	45	218	291	307	861

4.3 Modelo de Detecção de Objetos

4.3.1 Teste do Modelo

O teste do modelo tem como objetivo avaliar a qualidade do modelo treinado, mais especificamente, avaliar quantitativamente a precisão do modelo em um conjunto de dados que não foi utilizado no treinamento. Para tanto, será calculada uma métrica comumente adotada para avaliar modelos de detecção de objetos, chamada de mAP (*Mean Average Precision*). Antes de apresentar o mAP, serão apresentadas outras métricas que devem ser calculadas previamente.

4.3.1.1 Matriz de Confusão

A matriz de confusão é uma tabela que é normalmente utilizada para avaliar o desempenho de um modelo de classificação em aprendizado de máquina. Ela mostra o desempenho do modelo comparando as predições feitas pelo modelo com as classes reais (*ground truth*) dos dados. Vale lembrar que, em modelos de detecção, o *ground truth* é composto por rotulagens e as suas respectivas imagens de origem. As rotulagens por sua vez são definidas pela caixa delimitadora e classe do objeto alvo.

Dito isso, a matriz de confusão é formada por quatro combinações distintas entre valores preditos e valores reais, sendo elas:

- *True Positive (TP)*: quando uma rotulagem que está no *ground truth* corresponde corretamente a uma detecção feita pelo modelo.
- *True Negative (TN)*: quando o modelo não detecta objeto que também não está no *ground truth*.

- *False Positive (FP)*: quando um objeto detectado pelo modelo não corresponde a nenhuma rotulagem que está no *ground truth*. Em outras palavras, uma predição incorreta do modelo.
- *False Negative (FN)*: quando uma rotulagem que está no *ground truth* não corresponde a nenhuma detecção feita pelo modelo. Em outras palavras, uma predição ausente do modelo.

Em modelos de detecção, o fundo da imagem (*background*) é tratado como uma classe negativa e também é comumente incluída na matriz de confusão.

4.3.1.2 Precisão e Recall

A precisão do modelo mede a capacidade do modelo em realizar predições corretamente, ou seja, sem confundir as classes. O cálculo da precisão é realizado a partir da quantidade de *true positives (TP)* e *false positives (FP)*, conforme a Equação 4.1 abaixo.

$$P = \frac{TP}{TP + FP} \tag{4.1}$$

O *recall* do modelo, ou sensibilidade, mede a capacidade do modelo em identificar os casos positivos (*ground truth*). O cálculo do *recall* é realizado a partir da quantidade de *true positives (TP)* e *false negative (FN)*, conforme a Equação 4.2 abaixo.

$$R = \frac{TP}{TP + FN} \tag{4.2}$$

4.3.1.3 Intersection over Union (IoU)

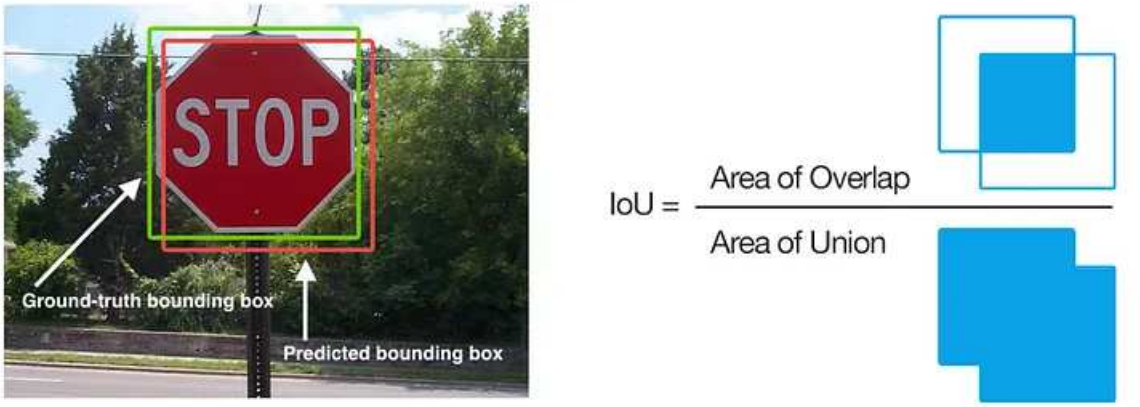
Os modelos de detecção realizam as predições em termos de caixas delimitadoras e classes de interesse. Para verificar se a classe predita está correta, basta comparar com a classe do *ground truth*. Já para verificar se a caixa delimitadora predita corresponde com a caixa do *ground truth* é calculada a métrica de Intersecção sobre União (*Intersection over Union - IoU*) entre ambas as caixas. A métrica IoU permite medir o quanto a caixa delimitadora prevista se ajusta à caixa delimitadora do *ground truth*. O cálculo do IoU consiste em dividir a área sobreposta das caixas sobre a área da união das caixas, conforme ilustrado na Figura 25.

4.3.1.4 Average Precision (AP)

A *Average Precision (AP)*, também chamada de curva AP, é a área sob a curva *precision-recall* (*PR Curve*). Destaca-se que a AP é calculada por classe, ou seja, apenas

[6] Disponível em: <https://medium.com/@jalajagr/mean-average-precision-map-explained-in-object-detection-fb61adf67e
Acesso em: 18 de agosto de 2023.

Figura 25 – Ilustração do cálculo a métrica de Intersecção sobre União.[6]



uma classe de detecção é observada por vez. E para calculá-la, deve-se primeiramente selecionar um valor limiar de IoU, como 0.5, por exemplo. Em seguida, o modelo deve ser testado em conjunto de teste e as detecções preditas devem ser ordenadas em ordem decrescente de confiança. E então as detecções preditas são contabilizadas da seguinte forma:

- Quando várias predições detectam o mesmo objeto, a predição com a maior IoU é classificada como TP, enquanto as demais predições são consideradas FP.
- Se o objeto estiver presente, mas a predição apresentar um IoU abaixo do limiar estabelecido com o *ground truth*, essa predição é também considerada como FP.
- Se nenhuma predição detectar ou classificar corretamente o objeto, a predição será considerada como FN.
- Por fim, se a detecção predita for de objeto que não está no *ground truth*, a predição será considerada como FP.

Feito isso, a *PR Curve* pode ser calculada. Para tanto, o cálculo da precisão e do *recall* é feito para cada detecção predita de forma cumulativa. É desejável que a *PR Curve* diminua monotonicamente, uma vez que sempre existe *trade-off* entre precisão e *recall*, ou seja, aumentar um diminuirá o outro. Porém, nem sempre isso acontece, então para garantir que a curva diminua monotonicamente, deve ser calculada a precisão interpolada $Pinterp$ para cada valor de *recall* R, tomando a precisão máxima para qualquer $R' \geq R$, conforme a Equação 4.3. Dito isso, a AP pode ser obtida por meio da integração numérica da *PR Curve*, conforme apresentado na Equação 4.4.

$$Pinterp(r) = \max_{r':r' \geq r} (r') \tag{4.3}$$

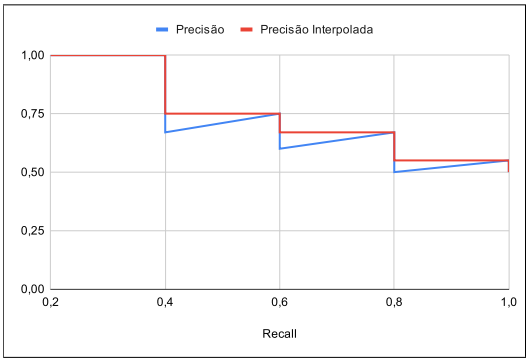
$$AP = \sum_{r=0}^{1} Pinterp(r_{n+1}) \cdot (r_{n+1} - r_n) \tag{4.4}$$

A Figura 26 ilustra o cálculo da *PR Curve* em uma amostra de dez predições, no qual cinco foram consideradas como TP e cinco foram consideradas como FP. É possível observar que a curva PR em azul não diminui monotonicamente, então, para garantir que a curva seja monótona, é calculada a curva interpolada (representada em vermelho).

Figura 26 – (a) Tabela com o cálculo das precisões e *recalls* acumuladas. (b) Gráfico resultante da *PR Curve*.

Confiança	Match (TP / FP)	TP Acumulativo	FP Acumulativo	Precisão	Recall	Precisão Interpolada
0,95	TP	1	0	1 / (1 + 0) = 1	1/5 = 0,20	1
0,92	TP	2	0	2 / (2 + 0) = 1	2/5 = 0,40	1
0,86	FP	2	1	2 / (2 + 1) = 0,67	2/5 = 0,40	0,75
0,81	TP	3	1	3 / (3 + 1) = 0,75	3/5 = 0,60	0,75
0,73	FP	3	2	3 / (3 + 2) = 0,60	3/5 = 0,60	0,67
0,68	TP	4	2	4 / (4 + 2) = 0,67	4/5 = 0,80	0,67
0,64	FP	4	3	4 / (4 + 3) = 0,57	4/5 = 0,80	0,57
0,57	FP	4	4	4 / (4 + 4) = 0,50	4/5 = 0,80	0,55
0,51	TP	5	4	5 / (5 + 4) = 0,55	5/5 = 1	0,55
0,48	FP	5	5	5 / (5 + 5) = 0,50	5/5 = 1	0,5

(a)



(b)

4.3.1.5 Mean Average Precision (mAP)

O cálculo do *Mean Average Precision* (mAP) na detecção de objetos consiste em calcular a média da AP para todas as n classes de detecção, conforme a Equação 4.5.

$$mAP = \frac{1}{n} \sum_{k=1}^{n} AP_k \tag{4.5}$$

Uma forma geralmente utilizada para relatar o mAP é a métrica COCO para mAP@0,50:0,95. Basicamente, isso significa que o mAP é calculado em vários limites de IoU entre 0,50 e 0,95 e, em seguida, a pontuação final de mAP é calculada como a média dos mAPs encontrados para cada limite de IoU.

4.3.2 Arquitetura da Rede

A arquitetura empregada para treinar o modelo de detecção é denominada de *SSD MobileNet V2 FPNLite* 320 × 320. Essa rede opera com resolução de entrada 320 × 320 e consiste na combinação das seguintes arquiteturas: *MobileNet V2* para extração de

features; *Feature Pyramid Network Lite (FPNLite)* para lidar com a extração de *features* em múltiplas escalas; SSD para a detecção de objetos.

A arquitetura *MobileNet V2* é uma rede neural convolucional (CNN) leve, especificamente desenvolvida para computação eficiente em dispositivos móveis e embarcados. Essa arquitetura emprega convoluções separáveis em profundidade e utiliza conexões residuais para diminuir a quantidade de parâmetros, assim melhorando o desempenho da rede. O *MobileNet V2* é empregado para a extração de *features*, sendo responsável por identificar os elementos-chave na imagem de entrada que são posteriormente utilizados na detecção de objetos (SANDLER et al., 2019).

Uma das limitações do SSD é lidar com a detecção de objetos pequenos, uma vez que o mesmo realiza a detecção em vários *feature maps*, mas evita a utilização das camadas inferiores (de alta resolução) para essa tarefa devido ao alto custo computacional associado. Ou seja, o SSD concentra-se apenas nas camadas superiores para a detecção, resultando em um desempenho inferior para objetos pequenos.

A FPNLite é uma versão leve da arquitetura *Feature Pyramid Network (FPN)*, normalmente utilizada como extrator de *features* que recebe uma imagem de escala única, de tamanho arbitrário, como entrada e produz *feature maps* de tamanho proporcional em vários níveis, por meio de uma abordagem totalmente convolucional. Essa abordagem visa proporcionar informações contextuais adicionais para a tarefa de detecção de objetos, aprimorando a precisão por meio da incorporação de informações de múltiplas escalas (LIN et al., 2017).

A FPN é composta por um caminho ascendente e descendente. O caminho ascendente é a rede convolucional convencional para extração de *features*, onde à medida que as camadas são percorridas ascendentemente, a resolução espacial diminui e o valor semântico aumenta, assim estruturas de alto nível (formas mais complexas) podem ser detectadas. Já o caminho descendente permite construir camadas de resolução mais alta a partir de uma camada semântica rica, dessa forma, contribuindo para uma detecção de objetos mais precisa e eficaz, especialmente para objetos pequenos.

A rede *SSD MobileNet V2 FPNLite* 320×320 possui 2,7 milhões de parâmetros treináveis distribuídos em 314 camadas. O desempenho da rede no *dataset* COCO17 [7] foi de 22,2% de mAP e 22 milissegundos de tempo de inferência [8].

Por fim, para treinar o modelo de detecção de passageiros, foi realizado o *fine-tuning* a partir do modelo pré-treinado no *dataset* COCO17.

[7] Dataset COCO17: <https://cocodataset.org>

[8] Rede SSD MobileNet V2 FPNLite: <https://resources.wolframcloud.com/NeuralNetRepository/resources/SSD-Feature-Pyramid-Nets-Trained-on-MS-COCO-Data/>

4.3.3 Treinamento do Modelo

O treinamento do modelo foi realizado nos ambientes do Google Colab e Google Cloud Platform (GCP). Juntamente, foi utilizado a biblioteca TensorFlow Lite para a implementação do treinamento.

De forma sucinta, o Google Colab é um ambiente de *notebooks Jupyter*, onde é possível combinar trechos de código Python executável e hipertextos a partir da sintaxe Markdown.

Enquanto o GCP é provedor de recursos de computação em nuvem que oferece serviços relacionados à computação, armazenamento, inteligência artificial, *analytics*, etc.

Já TensorFlow Lite é uma biblioteca que possui um conjunto de ferramentas que permite treinar modelos de aprendizado de máquina, bem como executar modelos em dispositivos móveis, embarcados e IoT (*Internet of Things*). Ele suporta plataformas como Linux embarcado, Android, iOS e MCU (*Microcontroller Unit*).

Dito isso, uma máquina virtual do GCP (Tabela 2) foi instanciada e conectada ao *notebook* do Google Colab desenvolvido para o treinamento do modelo SSD [9].

Tabela 2 – Especificação da máquina do Google Cloud Platform utilizada no treinamento.

Tipo de Máquina do GCP	n1-highmem-2 [10]
CPU	2 vCPUs(*Virtual Central Processing Units*)
GPU	NVIDIA Tesla T4 (16 GB)
Memória	13 GB
Armazenamento	SSD(*Solid State Drive*) 200 GB

4.4 Rastreadores de Objetos

4.4.1 Correlation Tracker

O *Correlation Tracker* utilizado na etapa de rastreamento por filtro de correlação está disponível na biblioteca Dlib[11]. A biblioteca Dlib é desenvolvida em C++ e possui uma interface (*shared library*) que permite utilizar a biblioteca em Python.

A Dlib implementa o método proposto por Danelljan et al. (2014) que estende os filtros de correlação padrão para *features* multidimensionais. Mais especificamente, o método consiste no rastreamento conjunto de escala e translação do objeto alvo, baseando-se no treinamento de um filtro de correlação *scale-space* tridimensional.

[9] Repositório do notebook de treinamento do modelo SDD: <https://github.com/TCC-GustavoNr/TCC-SSD-MobileNet-Training>
[10] Máquina da GCP: <https://cloud.google.com/compute/docs/general-purpose-machines>
[11] Biblioteca Dlib: <http://dlib.net/imaging.html>

4.4.2 Sort Tracker

A versão do SORT utilizada foi proposta por Bewley et al. (2016) e conta com uma implementação desenvolvida em Python disponibilizada pelo autor[12]. A solução utiliza a implementação do filtro de Kalman da biblioteca FilterPy[13] e o algoritmo LAPJV, revisado na Seção 2.4.3.4, para associar os objetos entre os *frames*.

4.4.3 Centroid Tracker

A implementação desenvolvida foi baseada na solução proposta e implementada por Rosebrock (2018) na linguagem Python. Por ser um rastreador simples, a solução utilizada é completamente análoga ao algoritmo descrito na fundamentação teórica (Seção 2.5).

4.5 Captura de Frames

A captura de *frames* foi desenvolvida utilizando a biblioteca OpenCV e possui dois modos de operação, sendo eles: captura da *stream* de vídeo a partir de arquivo; captura da *stream* de vídeo a partir de uma câmera conectada.

A implementação da captura a partir de uma câmera disponibiliza os *frames* lidos de forma assíncrona. Para tanto, uma *thread* foi criada para ler os *frames* e armazená-los em uma fila. Dessa forma, não é necessário aguardar o tempo de processamento de um *frame* para ler o *frame* seguinte.

4.6 Contagem de Passageiros

A solução completa do algoritmo de contagem de passageiros pode ser consultada no seguinte repositório do GitHub:

<https://github.com/TCC-GustavoNr/TCC-Passenger-Counting-Algorithm>.

4.6.1 Implementação do Algoritmo

Após a construção dos módulos de captura de *frames*, detecção e rastreamento de objetos, iniciou-se a implementação do algoritmo de contagem de passageiros.

Durante o desenvolvimento do algoritmo, foi possível notar que não seria necessário executar o modelo de detecção de passageiros em todos os 30 *frames* que podem ser capturados pela *stream* de vídeo em um intervalo de um segundo. Sendo assim, uma

[12] Implementação do autor do SORT: <https://github.com/abewley/sort/tree/master>
[13] FilterPy: <https://filterpy.readthedocs.io/en/latest/>

vez que a tarefa de detecção é a mais custosa, optou-se por uma abordagem que limita o número de *frames* em que são realizadas as detecções de objetos. Para tanto, um parâmetro denominado como *skip frames* foi incorporado ao algoritmo. Com esse parâmetro, é possível definir a quantidade de *frames* que devem ser ignorados antes de realizar uma nova detecção. Além disso, utilizou-se *Correlation Tracker*, técnica de rastreamento de objetos que permite detectar e rastrear um objeto específico em uma sequência de *frames*, reduzindo o número de *frames* em que é necessário aplicar o modelo de detecção.

Para a tarefa de rastreamento de passageiros, as seguintes versões dos algoritmos de rastreamento foram desenvolvidas:

- *StandardCentroidTracker*: conta apenas com a implementação padrão do *Centroid Tracker*. Nos *frames* onde não há detecção pelo modelo SSD, nada é feito e os passageiros rastreados continuam na mesma posição da última detecção.
- *StandardSortTracker*: conta apenas com a implementação padrão do *SORT Tracker*. Nos *frames* onde não há detecção pelo modelo SSD, são realizadas estimativas das posições dos passageiros rastreados no momento. Tais estimativas são feitas pelo filtro de Kalman presente na implemenetação do SORT.
- *CorrelationCentroidTracker*: conta com o *Centroid Tracker* aliado ao *Correlation Tracker*. Nos *frames* onde há a detecção de passageiros, as detecções preditas são utilizadas para treinar os filtros de correlação do *Correlation Tracker*. Já nos *frames* onde não há detecção, o *Correlation Tracker* é aplicado para predizer as posições atualizadas dos passageiros que estão sendo rastreados no momento. Feito isso, o *Centroid Tracker* é atualizado e os objetos recém detectados são associados aos objetos já rastreados.
- *CorrelationSortTracker*: conta com o *SORT Tracker* aliado ao *Correlation Tracker*. Aplica o *Correlation Tracker* da mesma forma que o *CorrelationCentroidTracker*. E após aplicar o *Correlation Tracker*, o SORT é atualizado, gerando estimativas das posições dos objetos e associando os objetos recém detectados aos objetos já rastreados.

Isso posto, o algoritmo de contagem de passageiros desenvolvido neste trabalho foi dividido nas seguintes etapas iterativas:

1. Captura do *frame* atual a ser processado.
2. Caso o *frame* recuperado esteja em um intervalo em que deve haver a detecção, o modelo de detecção de passageiros é executado para predizer as posições atualizadas dos passageiros rastreados no momento, bem como detectar novos passageiros ainda não detectados. Caso contrário, apenas segue para a etapa seguinte.

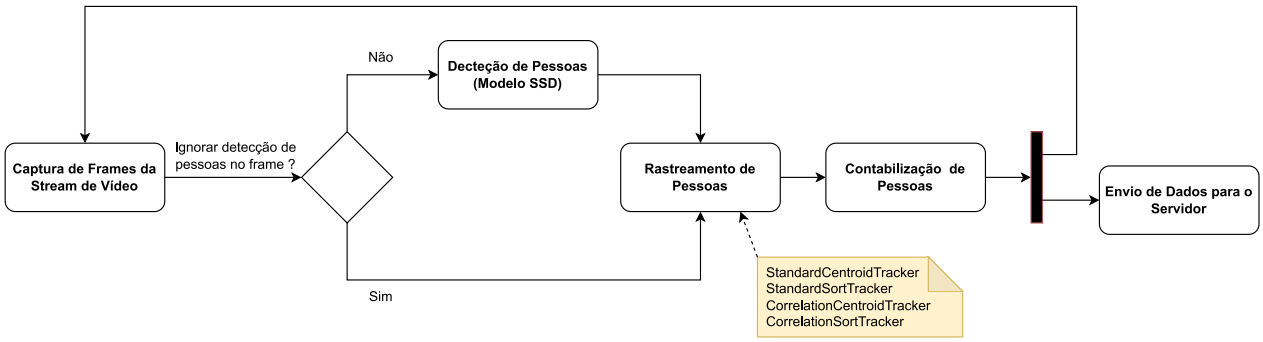
3. Logo depois, o método de rastreamento selecionado executa uma rotina para atualizar a lista de passageiros rastreados. Qualquer um dos quatro métodos descritos anteriormente podem ser utilizados nessa etapa.

4. Com base na lista atualizada de passageiros rastreados, é executada a rotina de contabilização de embarques e desembarques de passageiros. A rotina foi baseada na solução descrita na Seção 2.6.

5. Em seguida, os dados de contagem são adicionados em uma fila, para que, então, sejam enviados para o servidor na nuvem. Por fim, o algoritmo retorna para a primeira etapa de forma paralela.

Em síntese, o algoritmo de contagem de passageiros desenvolvido é representado pelo diagrama de atividades da Figura 27.

Figura 27 – Diagrama de atividades do algoritmo de contagem de passageiros.



4.6.2 Teste do Algoritmo

O propósito do teste é avaliar o desempenho do algoritmo de contagem de passageiros em diferentes situações. Para tanto, foi criado um *dataset* de teste (Tabela 3) de aproximadamente 50 minutos de duração a partir de diferentes vídeos recuperados dos *datasets* apresentados na Seção 4.1. Após selecionar os vídeos, o dataset foi construído da seguinte forma: para cada vídeo selecionado, foi anotado em um arquivo a quantidade de pessoas que embarcam e desembarcam do veículo durante o vídeo. Vale destacar que foram selecionados apenas vídeos não utilizados para treinar o modelo de detecção de passageiros.

O teste consiste em submeter o *dataset* de teste em diferentes configurações do algoritmo de contagem de passageiros. Mais especificamente, o algoritmo será testado com diferentes valores de *skip frames* para cada um dos quatro métodos de rastreamento desenvolvidos.

Tabela 3 – Composição do *dataset* de teste do algoritmo de contagem de passageiros.

Dataset	**Multidão**	**Luminosidade Solar**	**FPS**	**Duração Total (Segundos)**	**Total Entrando**	**Total Saindo**
PCDS	Sim	Alta	25	235	42	43
PCDS	Sim	Baixa	25	297	59	67
PCDS	Não	Alta	25	188	19	21
PCDS	Não	Baixa	25	173	22	28
Breda	Não	Variada	30	1653	41	36
MIVIA	Sim	Alta	30	439	97	98
Total	-	-	-	2985	280	293

As principais métricas que serão extraídas durante o teste são: precisão, *recall* e FPS. Para as métricas de precisão e *recall*, os valores de TP, FP e FN serão avaliados a partir dos seguintes critérios:

- *True Positive (TP)*: quando um embarque ou desembarque é contabilizado corretamente pelo algoritmo.
- *False Positive (FP)*: quando um embarque ou desembarque que não ocorreu é contabilizado pelo algoritmo.
- *False Negative (FN)*: quando um embarque ou desembarque não é contabilizado pelo algoritmo.

4.7 Servidor na Nuvem

Conforme especificado na Seção 3, a aplicação na nuvem desenvolvida neste trabalho tem por objetivo disponibilizar os dados de contagem de passageiros obtidos pela *Raspberry Pi*. Para tanto, a aplicação na nuvem foi estruturada e desenvolvida utilizando o *Serverless Framework*[14].

O *Serverless Framework* permite desenvolver aplicativos no *AWS Lambda* e em outras plataformas de nuvem, que se dimensionam automaticamente e cobram apenas durante a execução, sendo assim capaz de reduzir o custo total de operação e implementação de aplicativos. Para tal, o *Serverless Framework* conta com uma ferramenta de linha de comando e uma sintaxe YAML[15] que simplificam o processo de implantação de código e da infraestrutura de nuvem necessária para diversos casos de uso de aplicativos *serverless*.

A solução completa da aplicação *serverless* pode ser consultada no seguinte repositório do GitHub: <https://github.com/TCC-GustavoNr/TCC-AWS-Cloud-Server>.

[14] Serverless Framework: <https://github.com/serverless/serverless>
[15] YAML: <https://yaml.org/spec/1.2.2/>

4.7.1 Envio dos Dados

Para a comunicação entre os dispositivos de contagem e o servidor na nuvem, foi adotada uma abordagem baseada em eventos. Os eventos são disparados pelo dispositivo de contagem cada vez que uma sequência de embarques e/ou desembarques são detectadas. Os dados dos eventos disparados são enviados pelos dispositivos por meio do protocolo MQTT para o *broker* da *AWS IoT*. A aplicação *serverless*, por sua vez, armazena e disponibiliza os dados dos eventos recebidos.

4.7.2 Armazenamento dos Dados

Para armazenar os dados dos eventos recebidos, um banco de dados *Amazon DynamoDB* foi criado contendo apenas uma tabela de eventos, denominada como *CountEvent*. Os dados recebidos e armazenados na tabela *CountEvent* são apresentados e descritos na Tabela 4.

Tabela 4 – Dados da tabela de eventos de contagem de passageiros.

Atributo	**Descrição**
eventId	Identificador único do evento.
vehicleId	Identificador único do veículo em que o dispositivo está instalado.
deviceId	Identificador único do dispositivo.
startedAt	Data e hora que o evento começou.
endedAt	Data e hora que o evento terminou.
countEntering	Quantidade de pessoas que entraram no veículo durante o período do evento.
countExiting	Quantidade de pessoas que saíram do veículo durante o período do evento.
latitudePos	Latitude da posição que ocorreu o evento.
longitudePos	Longitude da posição que ocorreu o evento.

4.7.3 Consulta dos Dados

Para disponibilizar os dados dos eventos recebidos, uma API HTTPS foi desenvolvida utilizando NodeJS. A API pode ser acessada a partir da seguinte requisição HTTPS:

```
GET {baseUrl}/countEvent?vehicleId={}&deviceId={}&startDate={}&endDate={}
```

Os parâmetros de consulta da requisição são utilizados para fins de contabilização total de entradas e saídas, conforme descrito na Tabela 5.

Caso nenhum parâmetro enviado na requisição seja inválido, a aplicação deverá responder a requisição retornando os dados descritos na Tabela 6.

Tabela 5 – Parâmetros da requisição de consulta dos dados de contagem.

Parâmetro	**Obrigatório**	**Descrição**
vehicleId	Sim	Utilizado para agrupar apenas os eventos de contagem do veículo especificado.
deviceId	Não	Utilizado para agrupar apenas os eventos de contagem do dispositivo especificado.
startDate	Sim	Define o momento a ser utilizado para delimitar o início do agrupamento das contagens.
endDate	Não	Define o momento a ser utilizado para delimitar o fim do agrupamento das contagens. Caso o parâmetro não seja especificado, será utilizado a data e hora atual.

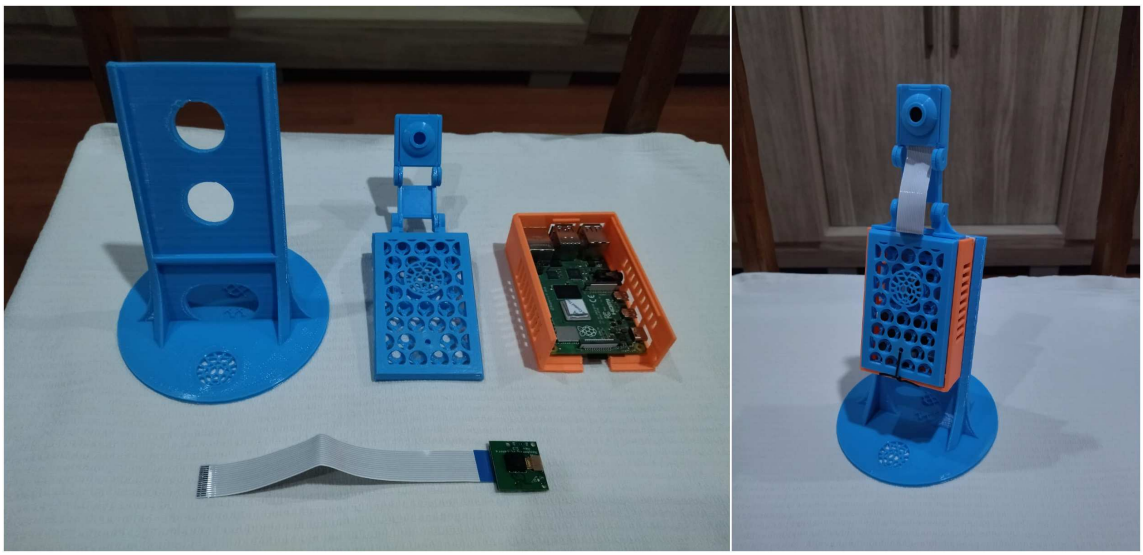
Tabela 6 – Resposta da requisição de consulta dos dados de contagem.

Atributo	**Descrição**
events	Lista com todos os eventos que aconteram no veículo durante o intervalo de tempo especificado na requisição.
totalEntering	Quantidade total de pessoas que entraram no veículo durante o intervalo de tempo especificado na requisição.
totalExiting	Quantidade total de pessoas que saíram do veículo durante o intervalo de tempo especificado na requisição.

4.8 Teste em Tempo Real

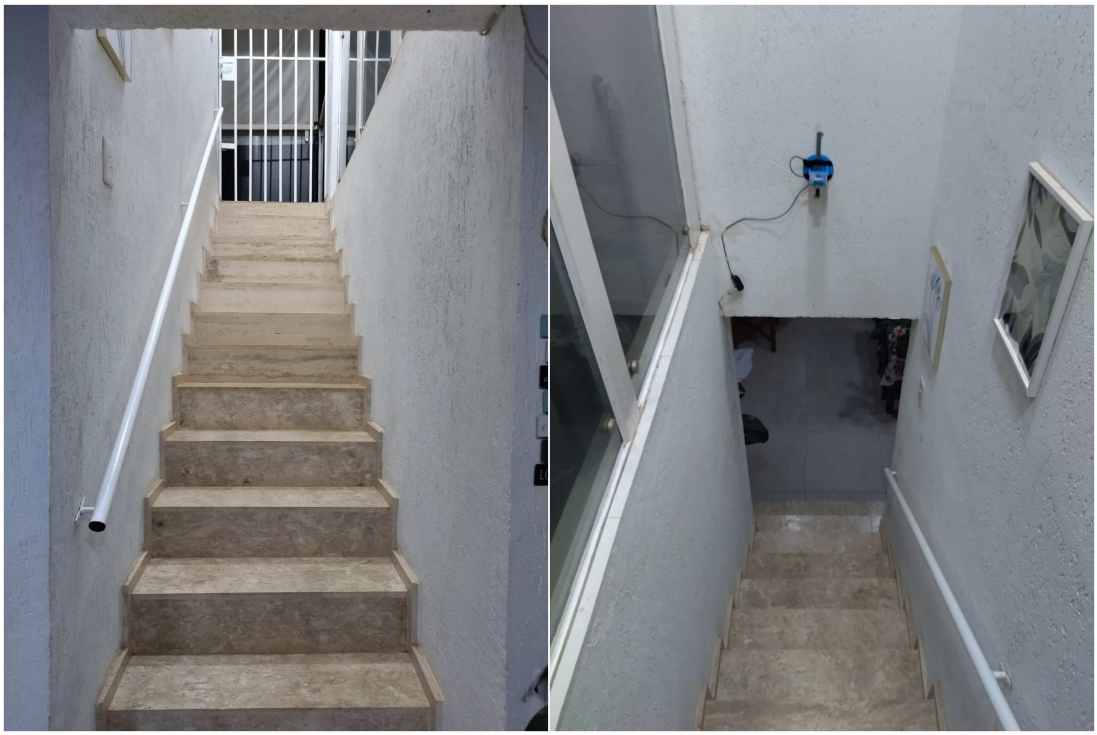
Para realizar o teste da solução em tempo real, foi realizada a impressão 3D de uma *case* para *Raspberry Pi 4* com suporte para câmera. O modelo da *case* utilizada está disponível gratuitamente em: <https://www.thingiverse.com/thing:3732714>. A Figura 28 apresenta o resultado final do protótipo com a *Raspberry* e a câmera montadas na *case* impressa.

Figura 28 – Montagem do protótipo para testes em tempo real.



Para tornar o ambiente de testes o mais parecido possível com um cenário real, o protótipo desenvolvido foi fixado sobre uma escada da minha casa. A escada e a posição fixada podem ser vistas na Figura 29.

Figura 29 – Escada utilizada para realização dos testes em tempo real.



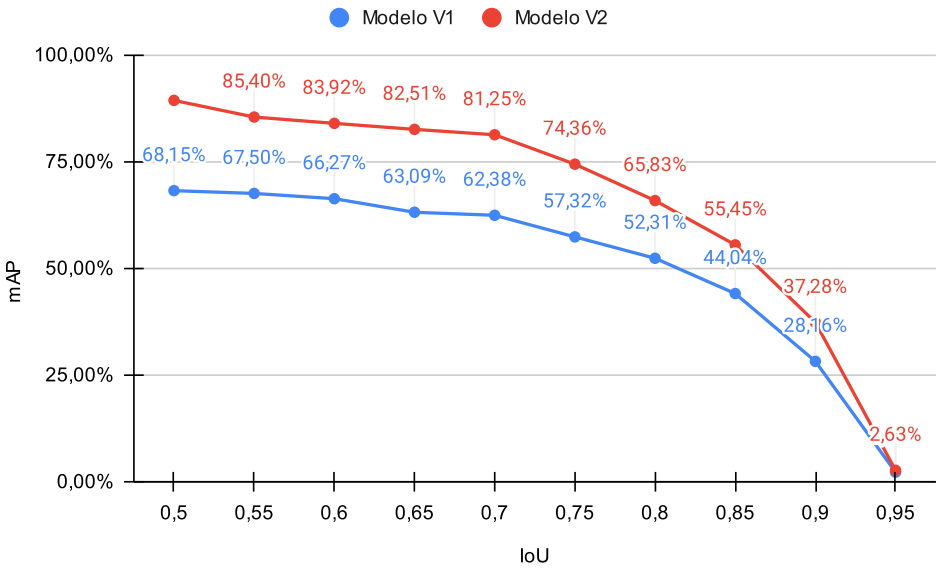
5 Resultados

5.1 Modelo de Detecção de Objetos SSD

Os métodos apresentados na Seção 4.3 foram aplicados para construir duas versões do modelo de detecção de passageiros, sendo que cada versão do modelo foi treinado com a sua versão correspondente do *dataset*, conforme apresentado na Tabela 1.

A primeira versão do modelo, treinada com 40 mil épocas, obteve um mAP@0,50:0,95 de 51,14%. Já a segunda versão do modelo, treinado com 60 mil épocas, obteve um mAP@0,50:0,95 de 65,79%. A Figura 30 apresenta um gráfico com os valores de mAP obtidos para diferentes limites de IoU. Percebe-se um ganho considerável de mAP na segunda versão do modelo em relação à primeira versão. Dessa forma, a segunda versão foi selecionada para ser o detector do algoritmo de contagem de passageiros.

Figura 30 – Gráfico com desempenho dos modelos de detecção desenvolvidos.



5.2 Algoritmo de Contagem de Passageiros

Conforme detalhado na Seção 4.6.2, um *dataset* de 50 minutos de vídeo foi construído para testar o algoritmo de contagem de passageiros em diferentes configurações. O teste foi realizado em uma *Raspberry Pi 4*, como especificado na Seção 3.1.1, e demorou cerca de 32 horas para ser concluído. As principais métricas coletadas serão apresentadas a seguir, mas o relatório completo pode ser acessado em: <https://github.com/TCC-GustavoNr/TCC-Passenger-Counting-Algorithm/tree/main/tests>.

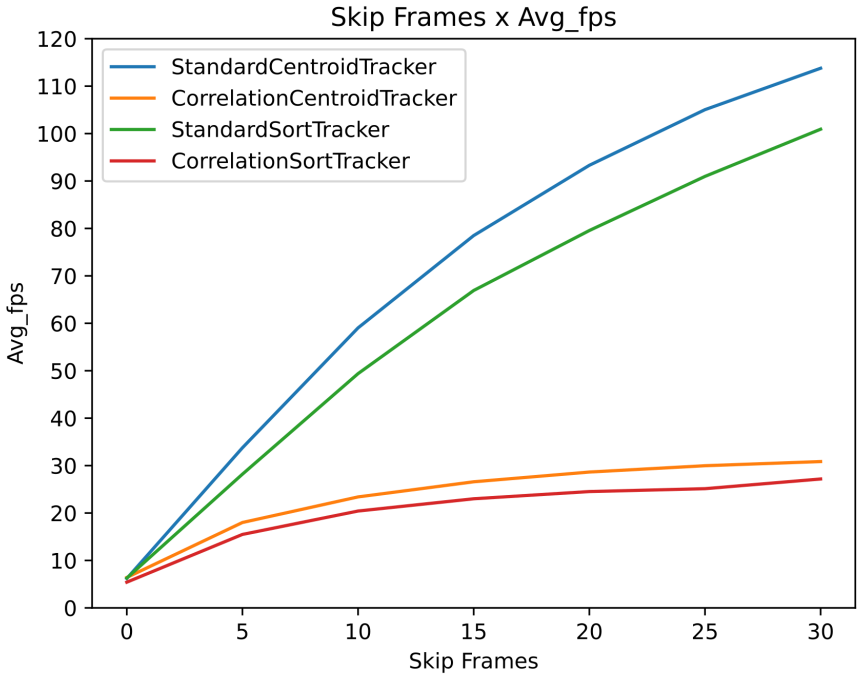
5.2.1 FPS Médio

Para cada configuração do algoritmo de contagem, foi extraída a taxa média de FPS, calculada como a média de *frames* processados por segundo para os vídeos do *dataset* de teste. A Tabela 7 e o gráfico da Figura 31 apresentam os resultados obtidos para a análise do desempenho em termos de FPS. Independente do método de rastreamento, percebe-se que à medida que a quantidade de *skip frames* aumenta, a taxa de FPS também aumenta. Percebe-se também que o custo de realizar a detecção em todos os *frames*, ou seja aplicar o algoritmo com zero *skip frames*, torna a abordagem inviável de ser processada em tempo real.

Tabela 7 – FPS Médio do algoritmo de contagem em diferentes configurações.

Rastreador	**Skip Frames**						
	0	**5**	**10**	**15**	**20**	**25**	**30**
Standard CentroidTracker	6 FPS	34 FPS	59 FPS	79 FPS	93 FPS	105 FPS	114 FPS
Correlation CentroidTracker	6 FPS	18 FPS	23 FPS	27 FPS	29 FPS	30 FPS	31 FPS
Standard SortTracker	6 FPS	28 FPS	49 FPS	67 FPS	80 FPS	91 FPS	101 FPS
Correlation SortTracker	5 FPS	15 FPS	20 FPS	23 FPS	24 FPS	25 FPS	27 FPS

Figura 31 – FPS Médio do algoritmo de contagem em diferentes configurações.



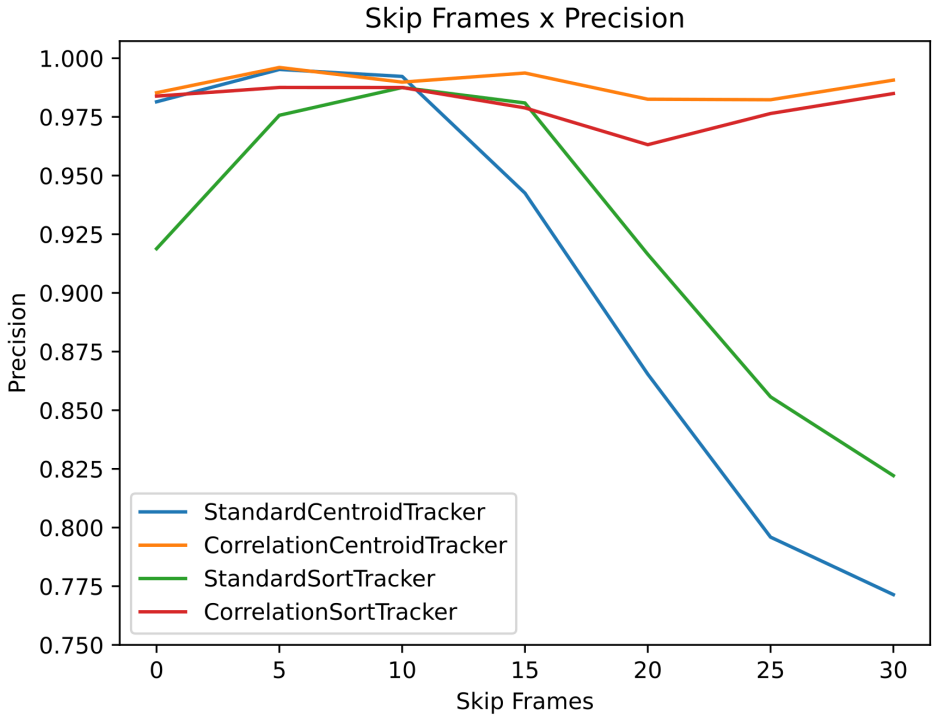
5.2.2 Precisão

No contexto deste trabalho, a análise da precisão permite inferir a capacidade do algoritmo em predizer contabilizações de embarque e desembarque de forma correta. A Tabela 8 e o seu respectivo gráfico ilustrado na Figura 32 apresentam as precisões obtidas no *dataset* de teste para as diferentes configurações do algoritmo. Percebe-se que a precisão se mantém acima de 77% para todas as configurações, assim é possível concluir que as contabilizações feitas pelo algoritmo tendem a ser corretas.

Tabela 8 – Precisão do algoritmo de contagem em diferentes configurações.

Rastreador	**Skip Frames**						
	0	**5**	**10**	**15**	**20**	**25**	**30**
Standard CentroidTracker	98.14%	99.52%	99.22%	94.25%	86.54%	79.59%	77.14%
Correlation CentroidTracker	98.53%	99.61%	98.98%	99.37%	98.25%	98.23%	99.07%
Standard SortTracker	91.88%	97.57%	98.75%	98.09%	91.64%	85.57%	82.21%
Correlation SortTracker	98.39%	98.75%	98.75%	97.88%	96.32%	97.64%	98.5%

Figura 32 – Precisão do algoritmo de contagem em diferentes configurações.



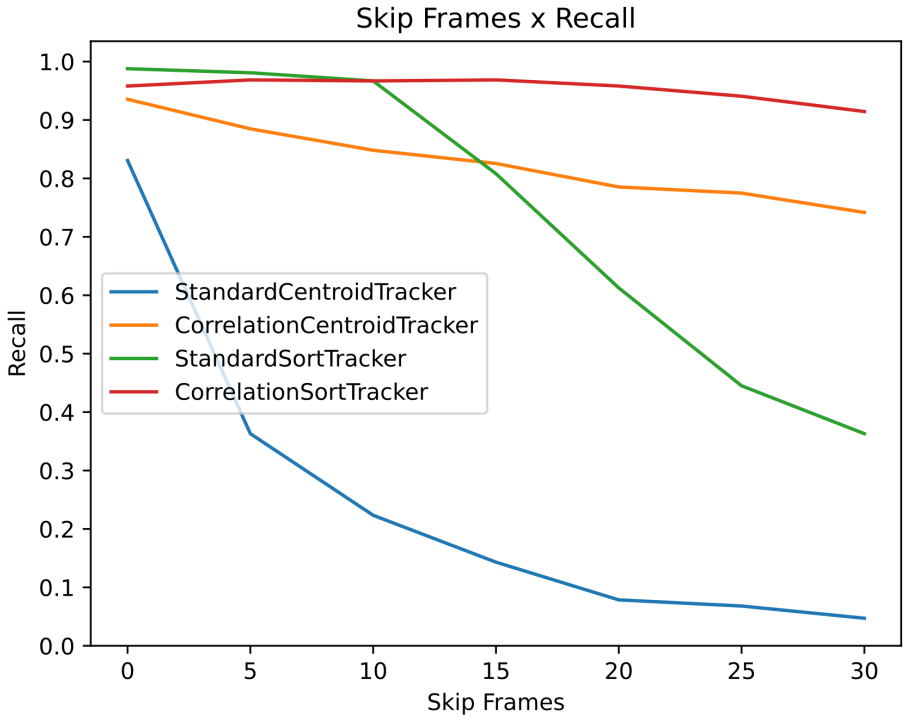
5.2.3 Recall

Neste trabalho, a avaliação do *recall* permite inferir a capacidade do algoritmo em identificar e contabilizar embarques e desembarques que de fato ocorreram. Conforme apresentado na Tabela 9 e na Figura 33, o *recall* do algoritmo tende a diminuir à medida que a quantidade de *skip frames* aumenta, especialmente nas configurações que não utilizam o *Correlation Tracker*. Sendo assim, para atenuar o número de contabilizações que podem ser perdidas, é preferível selecionar uma configuração que utilize um rastreador com *Correlation Tracker* ou uma configuração que utilize um rastreador *Standard* com o menor número possível de *skip frames*.

Tabela 9 – *Recall* do algoritmo de contagem em diferentes configurações.

Rastreador	**Skip Frames**						
	0	**5**	**10**	**15**	**20**	**25**	**30**
Standard CentroidTracker	83.07%	36.3%	22.34%	14.31%	7.85%	6.81%	4.71%
Correlation CentroidTracker	93.54%	88.48%	84.82%	82.55%	78.53%	77.49%	74.17%
Standard SortTracker	98.78%	98.08%	96.68%	80.8%	61.26%	44.5%	36.3%
Correlation SortTracker	95.81%	96.86%	96.68%	96.86%	95.81%	94.07%	91.45%

Figura 33 – *Recall* do algoritmo de contagem em diferentes configurações.



5.2.4 Análise das Métricas

Ao analisar as métricas de forma conjunta, é possível constatar que o método de rastreamento *StandardSortTracker* se destaca dos demais métodos. Isso fica evidente ao observar o desempenho do *StandardSortTracker* para as configurações com 5 e 10 *skip frames*, onde foi possível obter valores de precisão e *recall* superiores a 96% a uma taxa de processamento entre 28 e 49 FPS.

Foi possível observar também que as configurações de rastreamento que utilizam o *Correlation Tracker* são em média de duas vezes a quatro vezes mais rápidas que as suas versões *Standard* que realizam detecções em todos os frames, ou seja, versões configuradas com zero *skip frames*. Sendo assim, no contexto deste trabalho, é possível afirmar que existe um ganho em velocidade ao trocar a abordagem que realiza a detecção de objetos em cada frame por uma abordagem que intercala o papel da detecção de objetos entre o modelo de detecção e rastreador *Correlation Tracker*.

De modo geral, é necessário compreender o que é mais admissível para o operador de transporte quanto aos erros que podem ser cometidos durante a contagem de passageiros. Uma vez que, independente da abordagem de contagem, existe um *trade-off* entre precisão e *recall* nas abordagens estudadas. Em outras palavras, as soluções com a precisão maior que o *recall* podem perder contabilizações de embarque e desembarque. Em contrapartida, as soluções com o *recall* maior que a precisão podem contabilizar embarques e desembarques que não ocorreram.

5.3 Demonstrações

Para fins de demonstração da solução desenvolvida neste trabalho, uma *playlist* de vídeos não listados foi criada na plataforma YouTube. A *playlist* contém vídeos do dataset de teste que foram processados pela *Raspberry Pi 4*, bem como os resultados obtidos no teste em tempo real proposto na Seção 4.8. A *playlist* pode ser acessada em: <https://youtube.com/playlist?list=PLI4PUo1AxUoUW9XdCuPOmXhURVhIaALZu&si=Dbkm_5yZQBiochyd>.

6 Conclusão

Neste trabalho foi desenvolvido uma solução de contagem de passageiros em tempo real baseada em algoritmos de visão computacional. O hardware utilizado neste trabalho é composto por uma *Raspberry Pi 4* ligada a uma câmera por meio da interface CSI. Tendo isso em vista, foi adotada uma abordagem que combina técnicas de detecção e rastreamento de objetos voltadas para dispositivos embarcados e em tempo real. A solução desenvolvida foi dividida em cinco etapas, sendo elas: captura dos *frames* da câmera; detecção de passageiros; rastreamento de passageiros; contabilização de passageiros; envio de dados processados para um servidor na nuvem.

Para a detecção de passageiros, um modelo *Single Shot Detector (SSD)* foi treinado e testado a partir de um *dataset* produzido durante este trabalho de forma a ser constituído, sobretudo, de imagens contendo uma visão sobre portas de embarque e desembarque de veículos de transporte coletivo de passageiros. Como resultado, a segunda e melhor versão treinada do modelo de detecção SSD obteve um mAP@0,50:0,95 de 65,79%.

Para o rastreamento de passageiros, um conjunto de quatro versões de rastreamento de objetos foram desenvolvidas. Essas versões desenvolvidas consistem em combinar as técnicas de rastreamento *Correlation Tracker*, *Simple Online and Realtime Tracking (SORT)* e *Centroid Tracker*. E a partir do rastreamento realizado por esses algoritmos, são realizadas as contabilizações de embarque e desembarque de passageiros.

Diante disso, os dados de embarque e desembarque obtidos pelo processamento embarcado são enviados para um servidor construído utilizando a nuvem da *Amazon Web Services (AWS)*. Dessa forma, os dados de contagem podem ser disponibilizados em tempo real para o operador de transporte e para os próprios passageiros.

Após concluir o algoritmo de contagem de passageiros, foi construído um *dataset* de teste para avaliar o desempenho da solução desenvolvida. O *dataset* foi construído é composto por diferentes tipos de vídeos, sendo que cada vídeo foi rotulado de forma a contabilizar a quantidade de pessoas que entram e saem do veículo no qual foi filmado o vídeo. Então, o algoritmo de contagem de passageiros foi testado na *Raspberry Pi 4* a partir do *dataset* de teste. Dentre os resultados obtidos ao concluir o teste, a configuração do algoritmo que emprega o método de rastreamento SORT foi a que mais se destacou. Com essa configuração foi possível obter valores de precisão e *recall* superiores a 96% a uma taxa de processamento entre 28 e 49 FPS.

7 Trabalhos Futuros

7.1 Submódulo de GPS

Devido às limitações de tempo, não foi possível adquirir o módulo de GPS, bem como desenvolver o seu submódulo. O submódulo de GPS é indispensável para análise de demanda de embarque e desembarque por ponto de parada.

7.2 Abordagem com apenas um Processador por Veículo

Devido ao custo associado ter de mais de um processador por veículo, uma abordagem que utiliza apenas uma placa de processamento foi idealizada. A abordagem mantém o uso de uma camêra por porta, porém utiliza apenas uma placa para processar os vídeos de embarque e desembarque. Para tanto, seria utilizado apenas o tempo ocioso para processar os vídeos das portas, ou seja, utilizar apenas os momentos em que o veículo está em deslocamento. Dessa forma, é estimado que o processamento de mais de uma porta possa ser feito de forma a atualizar as contabilizações de passageiros antes mesmo do veículo chegar ao próximo ponto de parada.

Referências

AMIN, P. N. et al. Deep learning based face mask detection and crowd counting. In: *2021 6th International Conference for Convergence in Technology (I2CT)*. [S.l.: s.n.], 2021. p. 4–5. Citado na página 43.

BARCELLOS, P. R. M. Rastreamento de objetos em sequências de vídeo utilizando múltiplos filtros de correlação. 2021. Acessado em 20 de julho de 2023. Disponível em: <https://lume.ufrgs.br/handle/10183/225708>. Citado 3 vezes nas páginas 29, 30 e 31.

BELINAZI, L. Com necessidade de distanciamento, solução criativa surge para contagem de passageiros no transporte público. São Paulo, Brasil, 2020. Acessado em 18 de dezembro de 2022. Disponível em: <https://agoraesimples.com.br/negocios/contagem-automatica-de-passageiros/>. Citado na página 20.

BEWLEY, A. et al. Simple online and realtime tracking. *CoRR*, abs/1602.00763, 2016. Disponível em: <http://arxiv.org/abs/1602.00763>. Citado 2 vezes nas páginas 33 e 59.

BOLME, D. S. et al. Visual object tracking using adaptive correlation filters. p. 2544–2550, 2010. Citado 2 vezes nas páginas 29 e 31.

BOX, P.; OPPENLANDER, J. *Manual of Traffic Engineering Studies*. [S.l.]: Institute of Transportation Engineers, 1976. v. 4. 17 p. Citado na página 23.

BURKARD, R.; DELL'AMICO, M.; MARTELLO, S. *Assignment Problems*. Society for Industrial and Applied Mathematics, 2012. Disponível em: <https://epubs.siam.org/doi/abs/10.1137/1.9781611972238>. Citado na página 35.

CARVALHO, C. H. R. et al. Tarifação e financiamento do transporte público urbano. Instituto de Pesquisa Econômica Aplicada (IPEA), 2013. Acessado em 18 de dezembro de 2022. Disponível em: <https://repositorio.ipea.gov.br/handle/11058/1365>. Citado na página 19.

CHEN, Z.; HONG, Z.; TAO, D. An experimental survey on correlation filter-based tracking. 2015. Disponível em: <https://doi.org/10.48550/arXiv.1509.05520>. Citado na página 31.

CNI. Mobilidade urbana. In: *RETRATOS DA SOCIEDADE BRASILEIRA*. [S.l.: s.n.], 2015. v. 5, n. 27, p. 1–9. ISSN 23177012. Citado na página 19.

DANELLJAN, M. et al. Accurate scale estimation for robust visual tracking. p. 65.1–65.11, 01 2014. Citado na página 58.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. [S.l.]: Pearson Education, 2008. v. 4. Citado 3 vezes nas páginas 24, 26 e 27.

JONKER, R.; VOLGENANT, A. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, Springer-Verlag, Berlin, Heidelberg, v. 38, n. 4, p. 325–340, nov 1987. ISSN 0010-485X. Disponível em: <https://doi.org/10.1007/BF02278710>. Citado na página 38.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012. v. 25, p. 1097–1105. Disponível em: <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>. Citado 2 vezes nas páginas 25 e 26.

KUHN, H. W. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, v. 2, n. 1-2, p. 83–97, 1955. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109>. Citado na página 37.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, p. 4, 2015. Citado na página 23.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998. Citado 2 vezes nas páginas 23 e 25.

LI, Q. et al. Kalman filter and its application. p. 74–77, 2015. Citado na página 33.

LI, Z. et al. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, v. 33, n. 12, p. 6999–7019, 2022. Citado na página 23.

LIN, T.-Y. et al. Feature pyramid networks for object detection. 2017. Disponível em: <https://arxiv.org/abs/1612.03144>. Citado na página 57.

LINDAU, L. A.; ALBUQUERQUE, C.; CORRÊA, F. Sobreviver, renovar, prosperar: um caminho para o transporte coletivo de qualidade no brasil. São Paulo, Brasil, 2021. Acessado em 18 de dezembro de 2022. Disponível em: <https://www.wribrasil.org.br/noticias/sobreviver-renovar-prosperar-um-caminho-para-o-transporte-coletivo-de-qualidade-no-brasil>. Citado 2 vezes nas páginas 19 e 20.

LIU, C.-M.; JUANG, J.-C. Estimation of lane-level traffic flow using a deep learning technique. *Applied Sciences*, v. 11, p. 5619, 06 2021. Citado na página 39.

LIU, W. et al. SSD: Single shot MultiBox detector. In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016. p. 21–37. Disponível em: <https://doi.org/10.1007%2F978-3-319-46448-0_2>. Citado 3 vezes nas páginas 26, 27 e 28.

MOORE, K.; LANDMAN, N.; KHIM, J. Hungarian maximum matching algorithm. 2023. Acessado em 15 de julho de 2023. Disponível em: <https://brilliant.org/wiki/hungarian-matching/>. Citado 3 vezes nas páginas 36, 37 e 38.

MUNKRES, J. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, v. 5, n. 1, p. 32–38, 1957. Disponível em: <https://doi.org/10.1137/0105003>. Citado na página 37.

ROBINSON, A. Discriminative correlation filters in robot vision. n. 2146, p. 53, 2021. ISSN 0345-7524. Citado 2 vezes nas páginas 31 e 32.

ROSEBROCK, A. *Deep Learning for Computer Vision with Python*. [S.l.]: PyImageSearch, 2017. Citado na página 24.

ROSEBROCK, A. Simple object tracking with opencv. 2018. Acessado em 01 de fevereiro de 2023. Disponível em: <https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/>. Citado 2 vezes nas páginas 41 e 59.

SANDLER, M. et al. Mobilenetv2: Inverted residuals and linear bottlenecks. 2019. Disponível em: <https://arxiv.org/abs/1801.04381>. Citado na página 57.

SUN, S. et al. Benchmark data and method for real-time people counting in cluttered scenes using depth sensors. *IEEE Transactions on Intelligent Transportation Systems*, IEEE, 2019. Citado na página 50.

VERNER, K. Human detection dataset. 2022. Acessado em 01 de fevereiro de 2023. Disponível em: <https://www.kaggle.com/datasets/constantinwerner/human-detection-dataset>. Citado na página 51.

WELCH, G.; BISHOP, G. An introduction to the kalman filter. 2001. Acessado em 05 de julho de 2023. Disponível em: <https://courses.cs.washington.edu/courses/cse571/03wi/notes/welch-bishop-tutorial.pdf>. Citado 3 vezes nas páginas 33, 34 e 35.

YANG, Y. et al. Parallel correlation filters for real-time visual tracking. *Sensors*, v. 19, n. 10, 2019. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/19/10/2362>. Citado na página 32.