EdgeML aplicado ao Rastreamento de veículos e controle de tráfego

Hyago V. L. B. Silva; Davi Rosim; Felipe A. P. de Figueiredo; Samuel B. Mafra
Instituto Nacional de Telecomunicações - INATEL
Santa Rita do Sapucaí - MG
hyago.silva@mtel.inatel.br; davi.rosim@ges.Inatel.br; felipe.figueiredo@inatel.br; samuelbmafra@inatel.br

***Abstract*—Com o crescimento do número de estudantes, funcionários e visitantes, torna-se vital implementar medidas para supervisionar e manter o tráfego seguro. Este artigo propõe uma solução baseada em YOLO (You Only Look Once) para rastreamento de objetos em uma área específica, analisando fluxo, velocidade e distância. O sistema, desenvolvido em uma Raspberry Pi 4 Model B com câmera e utilizando YOLOv8n, atinge 97% de precisão. Caso a velocidade ultrapasse 25 km/h, uma foto é enviada por e-mail à segurança do campus para a tomada de medidas necessárias.**

***Index Terms*—Detecções e Rastreamento, EdgeML, gestão de tráfego, segurança em instituições.**

I. Introduction

As Universidades ou Institutos refletem a pequena complexidade dos desafios urbanos, especialmente no que diz respeito à mobilidade e ao gerenciamento de tráfego. O controle eficiente do tráfego e do fluxo de veículos é vital para garantir a segurança, a eficiência e a sustentabilidade dentro dos campos universitários [1]. A implementação de estratégias de gerenciamento de tráfego não só melhora a mobilidade no campus, mas fornece dados e informações cruciais para o aprimoramento do ambiente [2].

A detecção de objetos utiliza visão computacional (VC), com estratégias de alguma solução por trás, um viés através de detectar o objeto e o que fazer com esse dado. Atualmente o uso de algoritmos como YOLO (You Only Look Once) (YOLO) [3], permite identificar e rastrear veículos em tempo real, mesmo em dispositivos com recursos limitados, utilizando computação de borda (Edge Computing). Edge Computing, processa dados diretamente no dispositivo, reduzindo a latência e a necessidade de transmissão de grandes volumes de dados para servidores remotos [4]. Em um campus universitário, a coleta de informações sobre velocidade, distância e fluxo de veículos pode ser utilizada para monitorar e gerir o tráfego, prevenindo acidentes e melhorando a mobilidade.

Para converter aspectos digitais para o mundo real, é necessário técnicas e métricas em conjunto. Neste artigo, utilizando o rastreamento com YOLO, é possível avaliar a distância entre os objetos [5]. Uma das métricas utilizada, foi a conversão dos pixels de uma imagem para o mundo real, através da matriz homográfica. Posteriormente, é possível calcular, a distância euclidiana, para calcular a distância direta entre dois pontos em um plano, sendo este plano a matriz homográfica [6]. A velocidade pode ser determinada pela conversão de pixels para metros, através desta matriz homográfica, utilizando a proporção conhecida entre a medida real do cenário e a dimensão da imagem capturada pela câmera [7].

Com estes princípios, a VC utilizada em dispositivos embarcados em câmeras de trânsito estrategicamente posicionadas, equipados com modelos de detecção como o Yolov8, podem identificar e monitorar veículos, calculando sua velocidade e distância, bem como analisando o fluxo de tráfego. Utilizando câmeras com capacidade de processamento suficiente, é possível realizar essas análises em tempo real, melhorando a eficácia da gestão do tráfego. Esses dispositivos embarcados proporcionam uma infraestrutura robusta para a coleta e análise de dados, contribuindo significativamente para a segurança no trânsito dentro do campus universitário. Caso a velocidade que o veículo passe no percurso seja superior a um limite estabelecido no campus que é de 30km/h, é enviado um e-mail para o setor responsável tomar as medidas necessárias.

II. Trabalhos Relacionados

O autor, Danilo Leal, utiliza e detalha a aplicação de detecção e rastreamento de veículos em vídeos, visando contabilizá-los e estimar suas velocidades médias [8]. Utilizando os métodos de ponta YOLOv4 para detecção e DeepSORT para rastreamento. A base de dados foi adquirida de forma manual. Além disso, foi feito um mapeamento da área, facilitando a contagem e a estimativa de velocidades dos veículos. A combinação dos métodos YOLOv4 e DeepSORT com a ferramenta complementar desenvolvida mostrou-se eficiente na detecção e rastreamento de veículos, com precisão de aproximadamente 80%, em uma área desejada relativamente pequena.

O trabalho [9], propõe um sistema não intrusivo baseado em vídeo para medir a velocidade de veículos em vias urbanas, utilizando uma câmera de baixo custo para capturar vídeos e rastrear características distintivas das placas dos veículos. O sistema combina um detector de movimento otimizado e um detector de texto para localizar e rastrear placas de veículos, corrigindo distorções de perspectiva.

Testado em cinco horas de vídeos sob diversas condições climáticas, o sistema alcançou um erro médio de 0,5 km/h e manteve-se dentro do limite de [-3,+2] km/h em 96% dos casos. A detecção de placas, baseada em algoritmos avançados, obteve uma precisão de 0,93 e recall de 0,87. O conjunto

de dados usado inclui vídeos com velocidades verdadeiras verificadas por detectores de laço indutivo, e o sistema pode ser integrado a sistemas de vigilância para monitoramento de velocidade e identificação de veículos, potencialmente emitindo multas ou alerta para veículos roubados.

O artigo [10], propõe um sistema de medição de velocidade de veículos utilizando detecção de caixas delimitadoras 3D e transformação de perspectiva, combinando uma rede neural Faster R-CNN com algoritmos de regressão para calcular configurações 3D dos veículos. Testado com o conjunto de dados BoxCars116k, o sistema alcançou alta precisão, com melhorias significativas em relação aos métodos tradicionais de caixas 2D. Utilizando técnicas como subtração de fundo e rastreamento de movimento, o sistema pode ser integrado a câmera de vigilância de tráfego para medição de velocidade, contagem de veículos e detecção de congestionamentos, oferecendo uma solução eficiente para o monitoramento urbano.

Em comparação com este artigo, foi elaborado e divulgado toda implementação do cálculo de velocidade e distância, que foram utilizados, e não apenas fazer a medição destas métricas, mas exibir os valores, e restringir a partir do limite exigido dentro do campus das instituições ou locais das ruas, caso ultrapasse esta velocidade é retirado uma foto do *frame* e enviado ao e-mail responsável pela segurança pública, similar a um radar.

Durante o monitoramento ainda é feito a contagem e o fluxo de carros a cada 2 minutos. O custo em comparação com um radar atual é bem vantajoso, de acordo com este protótipo poderia ser muito bem aprimorado e desenvolvido para indústria de segurança pública. O modelo alcançou aproximadamente 97% de Precisão, mesmo com uma base de dados desbalanceada, perante as outras métricas, está bastante preciso. A gravação ficou com tempo de 1 hora aproximadamente, obtendo diversas velocidades, distâncias e o fluxo devido à quantidade de carros no percurso.

III. Materiais e Métodos

A. Base de Dados

A base de dados é composta por quatro classes principais: carros, caminhões, motos e ônibus. Para carros são incluídos vans, no caso entre outros veículos. Esses dados foram adquiridos e preparados por meio da plataforma Roboflow. A divisão dos dados foi dividida com 70% para o treinamento do modelo, 20% para validação durante o ajuste dos parâmetros e 10% para testes finais, no total de 1795 imagens. O tamanho da imagem é de 640 pixels (px) para comprimento e altura. A distribuição da base de dados é mostrada abaixo na tabela I. Esta base de dados foi adquirida através do site Roboflow [11].

TABLE I
Base de Dados.

Classe	Imagens
Moto	745
Caminhão	687
Carro	424
Ônibus	394

B. Arquitetura da solução

A Arquitetura proposta utiliza Raspberry Pi 4 Modelo B como dispositivo de borda com uma câmera representando uma câmera inteligente. A câmera utilizada é de 720 px, alimentada via USB. O sistema opera diretamente na placa Raspberry Pi, coletando dados e registrando eventos, como velocidades de veículos. O pré-processamento das imagens capturadas é realizado localmente na placa, o que permite uma resposta rápida e eficiente.

Por fim, um modelo de detecção de objetos, o YOLOV8n versão 8.2 mais especificamente, treinado com um conjunto de dados mencionado acima, durante 300 épocas. Este modelo, será utilizado para realizar inferências sobre os dados capturados, e enviados a um dashboard desenvolvido com Streamlit, onde mostrará as detecções, distâncias, velocidades e fluxo dos veículos. Veja a arquitetura na figura 1, abaixo.

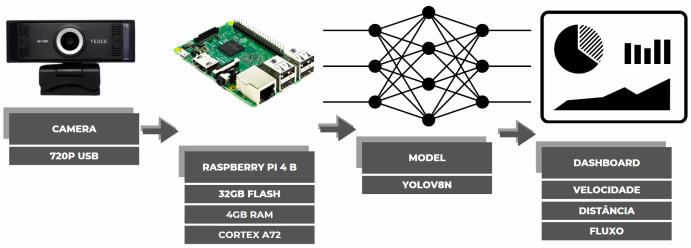

Fig. 1. Arquitetura

C. Transformação de pixels para metros

Para extrair a coordenada real de um veículo a partir de uma imagem, é preciso de um método de transformação algébrica, que será responsável por transformar os pixels em metros. De antemão, é definido a região da imagem em que será feita as detecções e rastreamentos, com os vértices dessa região é possível gerar uma matriz 4x2, comumente chamada *SOURCE*.

E para esse método funcionar é preciso de uma segunda matriz, também 4x2, que represente os pontos da matriz *SOURCE* em coordenadas no mundo real, chamada de *TARGET*. Estas transformações, foram baseadas em tutoriais do Open CV, entre outras referências, como Roboflow, [12].

Após a definição inicial, é utilizado o método de transformação homográfica resultando na matriz *H*. Agora é possível transformar agora qualquer pixel em coordenada real. O algoritmo 1, responsável pela transformação dos pixels é demonstrado a seguir.

Em seguida, a imagem 2, demonstra a região de interesse onde haverá as detecções e o rastreamento. Nessa região, com o algoritmo 1, irá ocorrer a transformação dos pixels para metros. A região se trata de um retângulo, de 30mx6.6m. É mostrado também os pixels correspondentes aos vértices do retângulo, considerando a resolução do vídeo de 1920x1080 px.

Algorithm 1 Transformação de Pixel para Metro

```
Class ViewTransformer
function ViewTransformer(source, target)
    self.matH ← cv2.findHomography(source, target)
end function
function transformPxToMt(pointsPixel)
    if pointsPixel == 0 ou self.matH == None then
        return pointsPixel
    end if
    pointsMeter ← cv2.perspectiveTransform(pointsPixel, self.matH)
    return pointsMeter
end function
```

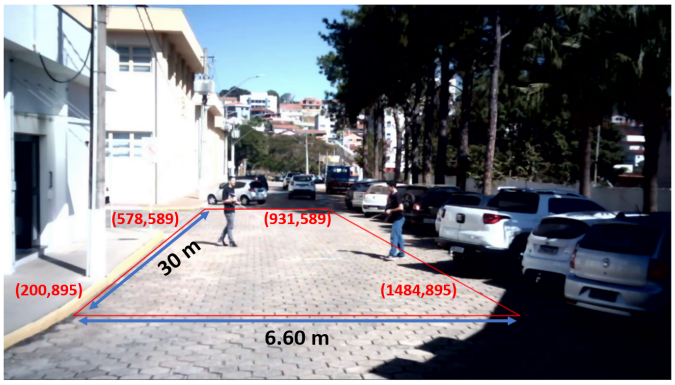

Fig. 2. Dimensão da área de detecção dos veículos em pixels e em metros

D. Cálculo da velocidade

A abordagem para o cálculo da velocidade analisa a área pré-definida mostrada acima. A detecção é seguida por um rastreamento que atribui um identificador único ao objeto. O método de cálculo da velocidade foi baseado em um tutorial do Roboflow e aprimorado para este projeto, visando aumentar a precisão da estimativa de velocidade. E para isso, segue os seguintes passos:

1. Ao obter a matriz de transformação, com o algoritmo 1, de pixels para metros, a cada frame, ou instante desejado, é obtido a coordenada atual no veículo e a mesma é armazenada em uma lista que possui: coordenadas em um espaço de tempo e o respectivo identificador do objeto.

2. Calcula-se a distância percorrida pelo objeto como a diferença, em módulo, entre a coordenada atual e anterior.

3. Determina-se o tempo decorrido durante o movimento do objeto, utilizando o número de quadros armazenados na lista de coordenadas e a taxa de quadros por segundo do vídeo.

4. Calcula-se a velocidade do objeto utilizando a fórmula da velocidade média e já convertido para quilômetros por hora.

$$\text{Velocidade} = \frac{\text{distância}}{\text{tempo}} \times 3.6$$

Caso a velocidade ultrapasse 30km/h, é retirado uma foto, e enviada por email utilizando o APP key do Gmail, com protocolo SMTP. Informando no e-mail, a imagem com o ID, velocidade e as distâncias entre outros objetos, se existirem.

5. Aplica-se a média móvel exponencial (MME) para suavizar a série temporal das velocidades. A fórmula da MME é dada por:

$$\text{MME}_t = \alpha \cdot \text{Vm}_t + (1 - \alpha) \cdot \text{MME}_{t-1}$$

Onde α é o fator de suavização. Quanto maior o valor de alfa, a velocidade é mais atrelada ao presente, quanto menor o valor de alfa, estaria mais corresponde ao passado.

6. Por fim é adicionado a velocidade suavizada ao *bounding box* do objeto.

O algoritmo 2, demonstra como foi elaborado a lógica para conseguir executar todos os passos anteriores.

Algorithm 2 Calcular Velocidade

```
const ALPHA = 0.5
function calculateEMA(previousEMA, currentSpeed)
    return ALPHA · currentSpeed + (1 - ALPHA) · previousEMA
end function
function calculateSpeed(labels, detections, coordinates, fps)
    for each id in detections.id do
        if id ∉ coordinates then
            labels.append(f"#{id}")
        else
            coordinate_anterior ← coordinates[id][-1]
            coordinate_atual ← coordinates[id][0]
            distance ← abs(coordinate_anterior - coordinate_atual)
            time ← len(coordinates[id]) / video_fps
            speed ← distance / time · 3.6
            if previous_ema is None then
                previous_ema ← speed
            else
                ema_speed ← calculate_ema(previous_ema, speed, ALPHA)
                if ema_speed > 30 then
                    sendEmail(frame, ema_speed, mensagem)
                end if
            end if
            labels.append(f"#{id} {int(ema_speed) km/h")
        end if
    end for
end function
```

E. Cálculo da distância

Para calcular a distância entre dois veículos, é utilizado a distância euclidiana, é necessário, as coordenadas dos pontos **BOTTOM_CENTER** (BC), de dois objetos distintos que foram detectados e rastreados. Importante notar que estes pontos já estejam transformados através do primeiro algoritmo. O algoritmo 3, demonstra como foi feito para se obter o cálculo da distância entre cada objeto.

Algorithm 3 Cálculo da distância entre veículos

function EUCLIDEAN_DIST(pointA, pointB)
  return $\sqrt{(pointA_x - pointB_x)^2 + (pointA_y - pointB_y)^2}$
end function
function DRAW_LINE(frame, pointA, pointB, distance)
  CV2.LINE(pointA, pointB)
  text_position ← $((pointA_x + pointB_x)//2, (pointA_y + pointB_y)//2)$
  CV2.PUTEXT(frame, distance, text_position)
  return frame
end function

Agora com estes algoritmos todos em conjunto, é possível agora unir cada um destes e obter os resultados de distância, velocidade e também, fluxo. Sendo o fluxo calculado de forma que a cada 2 minutos é mostrado a quantidade média de objetos detectados, sendo o gráfico mostrado juntamente com dashboard de Número médio de objetos detectados por minuto. O loop principal, está no algorítimo, 4.

Algorithm 4 Código principal

Require: algoritmos 1, 2 e 3
view ← **ViewTransformer**(SOURCE,TARGET)
model ← **YOLO**(MODEL_NAME)
previous_ema ← None
video_info ← SV.VIDEOINFO.FROM_VIDEO_PATH(VIDEO_PATH)
frame_gen ← SV.GET_VIDEO_FRAMES_GENERATOR(VIDEO_PATH)
tracker ← SV.BYTETRACK(frame_rate=video_info.fps, track_activation_threshold=CONFIDENCE_THRESHOLD)
polygon_zone ← SV.POLYGONZONE(polygon=SOURCE)
coords ← DEFAULTDICT(lambda:deque(maxlen=video_info.fps))

for each frame **in video_info.total_frames do**
  result ← MODEL(frame)[0]
  detec ← SV.DETECTIONS.FROM_ULTRALYTICS(result)
  detec ← polygon_zone.trigger(detec)
  detec ← DETEC.WITH_NMS(IOU_THRESHOLD)
  detec ← BYTE_TRACK.UPDATE_WITH_DETECTIONS(detec)
  pts ← DETEC.GET_ANCHORS_COORDINATES(BC)
  pts_mt ← VIEW.TRANSFORMPXTOMT(pts)
  for each id, [_, y] **in** ZIP(detec.tracker_id, pts_mt) **do**
    coords[id].append(y)
  end for
  labels ← []
  CALCULATESPEED(labels, detec, coords, video_info.fps)
  for each i **in range**(**len**(pts_mt) - 1) **do**
    for each j **in range**(i + 1, **len**(pts_mt)) **do**
      ptA ← **tuple**(pts[i])
      ptB ← **tuple**(pts[j])
      distance ← **EUCLIDEAN_DIST**(pts_mt[i], pts_mt[j])
      annotated_frame ← **DRAW_LINE**(frame, ptA, ptB, distance)
    end for
    Anota o frame no video alvo
  end for

IV. RESULTADOS

Através do campus e a concessão do órgão judiciário no INATEL, obtemos permissão de poder fazer gravações conforme a LGDP. Treinando o modelo Yolov8n, para detecção e rastreamento de objetos, foi utilizado 300 épocas, e com isto foi encontrado o maior em melhor valor possível das métricas qualitativas. Caso treinasse por mais tempo não haveria melhora, conforme testando. Abaixo é mostrado a matriz de confusão, com a figura 3, com o valor da acurácia obtida em cada classe. As classes estão numeradas de 0 a 4. Respectivamente, caminhão, carro, moto, ônibus e background (fundo).

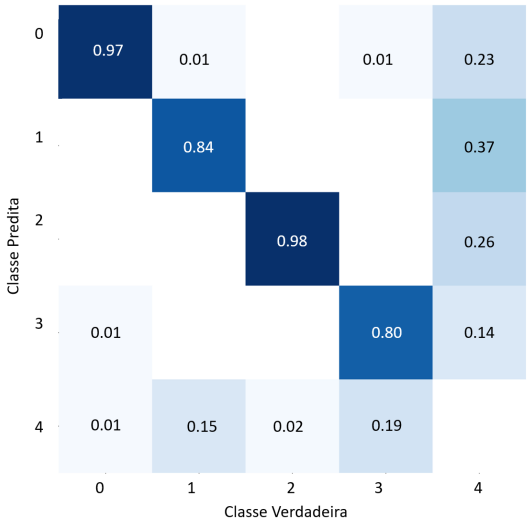

Fig. 3. Matriz de confusão Yolov8n

É importante notar que as classes não estão balanceadas na base de dados utilizada. Devido a isso, foram calculadas outras métricas, como F1-Score, Recall e Precisão, para avaliar o desempenho do modelo de forma mais abrangente. Na matriz de confusão, podemos ver que as classes ”carro” e ”ônibus” obtiveram valores inferiores, sendo as piores classes em termos de desempenho, com o ”carro” apresentando os piores resultados. Este treinamento foi realizado com uma base de dados pequena, o que limita a representatividade das classes.

Por outro lado, ao utilizar o modelo padrão do Yolov8n, treinado com a base de dados COCO, espera-se obter um desempenho significativamente melhor. A base de dados COCO é extremamente robusta e diversificada, contendo milhares de imagens e uma ampla variedade de contextos e objetos. Essa vasta quantidade de dados permite que o modelo aprenda a partir de uma gama muito maior de exemplos, resultando em um desempenho mais preciso e confiável, especialmente para as classes que estavam sub-representadas na base de dados menor utilizada inicialmente.

A Figura 4 mostra a curva de Precisão e Recall, essencial para avaliar o desempenho de modelos de aprendizado de máquina em classificação e detecção de objetos. Precisão mede a proporção de verdadeiros positivos entre todas as previsões

positivas, enquanto Recall mede a proporção de verdadeiros positivos entre todas as instâncias reais. A curva ajuda a encontrar o equilíbrio entre essas duas métricas.

A métrica mAP (Mean Average Precision) complementa essa análise, fornecendo uma média das precisões em diferentes níveis de recall. A área sob a curva (AUC) indica o desempenho geral do modelo; quanto mais próximo de 1, melhor o equilíbrio entre precisão e recall.

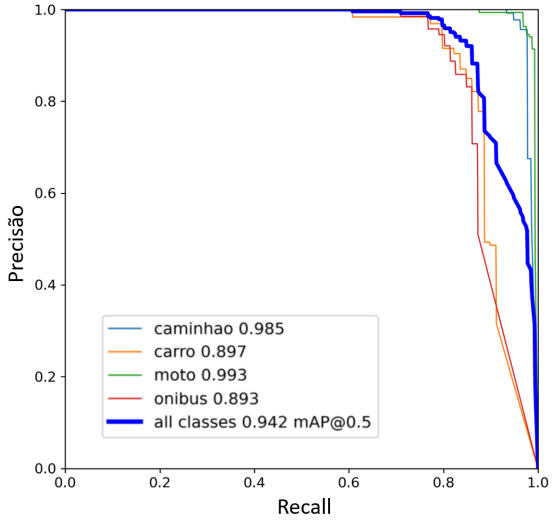

Fig. 4. Curva de Precisão e Recall Yolov8n

A Tabela II a seguir apresenta diversas métricas utilizadas para avaliar o desempenho do modelo em uma Raspberry Pi 4 Model B. Essas métricas incluem a inferência do modelo, o pré e pós-processamento das imagens e frames, o FPS (quadros por segundo), o consumo de CPU e RAM durante o processo, e a principal métrica de detecção de objetos, a mAP (Mean Average Precision). Além disso, são consideradas a precisão, o recall, o F1-Score e o tamanho do arquivo do modelo.

TABLE II
METRICS OF YOLOV8N.

	FPS	CPU	RAM	mAP50	Precision	Recall	F1	Size
float32	1,5	25%	1,5 GB	94%	97%	88%	92%	12 MB
float16	2,3	23%	1,2 GB	92%	93%	88%	90%	6 MB
int8	2,5	20%	1,1 GB	92%	94%	87%	90%	3 MB

É importante destacar que esta tabela permite identificar o melhor modelo para a tarefa utilizando especificamente a Raspberry Pi 4 Model B. Ao aplicar a quantização no modelo Yolov8n através da linha de comando com as opções *task* e *export*, conseguimos exportar o modelo no formato *tflite*, adicionando a quantização *int8*. Com a quantização, observou-se uma leve redução no consumo de CPU e RAM e uma significativa redução no tamanho do modelo. Notavelmente, não houve perda de precisão, recall, F1-score e mAP, mantendo um desempenho excelente mesmo após a quantização. No entanto, o problema persiste na inferência, pois não houve melhora no desempenho do FPS durante a inferência do modelo, mesmo após a quantização.

A figura 5, a seguir, possui alguns testes de distâncias, feitos de diferentes perspectivas, através da detecção de pessoas, para teste. Colocando uma trena entre as pessoas para ser medido a distância real em comparação e comprovando a distância calculada com o algoritmo durante as detecções. Em seguida, a figura 6, com o teste da velocidade dos objetos, desta vez com veículos, onde estes foram solicitados que estivessem e mantivesse uma velocidade constante, praticamente durante um percurso, dentro da região de interesse.

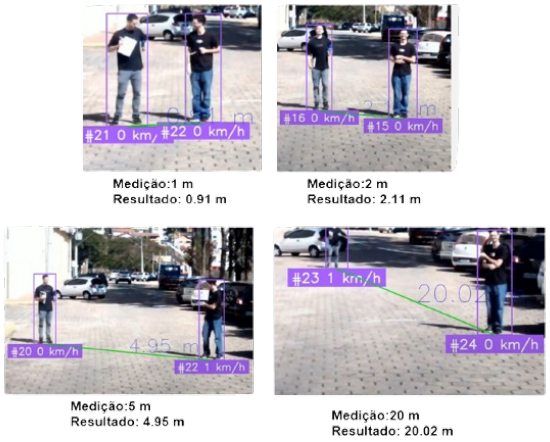

Fig. 5. Teste de distância

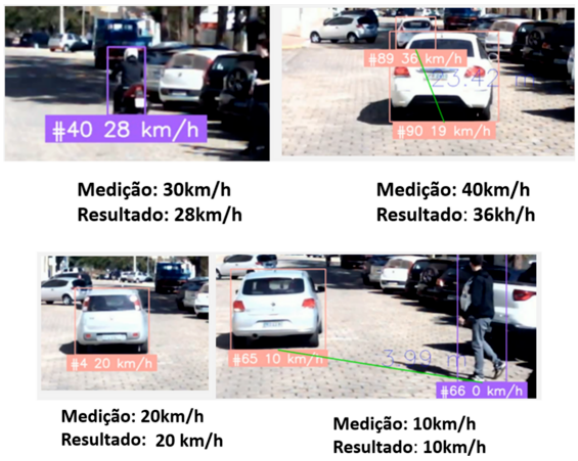

Fig. 6. Teste de velocidade

Posteriormente a figura 7, representa o dashboard construído com Streamlit, demonstrando os dados e as medições, de velocidade, fluxo e distância, sendo enviadas da Raspberry Pi está conectada na rede local, e disponibilizando um IP, identificado como outro host na rede, pelo próprio IP que a RaspBerry é reconhecida. Estes dados, são enviados diretamente para um dashboard, podendo ser vistos do celular ou computador, caso estejam na mesma rede. Os dados ficam na própria Raspberry Pi e são enviados notificações de e-mails caso a velocidade

Fig. 7. Dashboard

permitida seja ultrapassada, enviado a foto do frame que foi identificado a infração, junto a isto, o ID do objeto detectado, e datando a hora do e-mail naquele instante, registrando o dia do ocorrido, para posterior avaliação.

Durante a gravação de 1 hora de vídeo, foram registrados diversos veículos, sendo que o horário gravado é no final turno da manhã, às 11:40. Durante a captura dos veículos, obtemos velocidades, distâncias e fluxos. O valor obtido de velocidade, foi medido e validado, uma faixa de erro de $\pm$ 4 km/h. Agora o valor de distância está compreendido em uma faixa de erro de $\pm$ 0.11m. Foram avaliados cerca de 20 amostras de distâncias, e 20 amostras de velocidade. Analisando seus máximos e mínimos conforme a variação, sendo acima do valor real ou abaixo.

V. Conclusão

Em conclusão, a solução baseada em algoritmos de detecção de objetos YOLO e dispositivos de borda como o Raspberry Pi mostrou-se eficaz para gerenciar o tráfego, oferecendo dados precisos sobre o fluxo de veículos, velocidade e distância, e melhorando a segurança. Os testes revelaram uma margem de erro pequena para velocidade (±4 km/h) e distância (±0,11 m), com tempos de inferência razoáveis.

Para futuras melhorias, recomenda-se a otimização do código para melhorar o desempenho sem perda de frames, a implementação de aceleradores de redes neurais como PyArmn ou Google Edge TPU Coral, e o uso de uma perspectiva lateral da câmera para evitar sobreposição de bounding boxes e capturar uma área maior. Além disso, é importante explorar áreas maiores e utilizar matrizes de homografia para coletar mais pontos e avaliar métricas adicionais.

Também se sugere aumentar a quantidade de dispositivos e sincronizá-los com um servidor na nuvem, manter a alta precisão do modelo sem comprometer o FPS, e implementar mecanismos de criptografia em borda e transmissão, anonimizando rostos e placas dos carros. Por fim, a utilização de OCR para identificar e notificar placas de veículos, juntamente com a velocidade, é recomendada.

Essas melhorias, juntamente com a expansão da base de dados e a otimização dos algoritmos, têm o potencial de aprimorar significativamente a infraestrutura e sustentabilidade desta solução, e aprimorar para diversos cenários além deste em específico.

VI. Agradecimento

Este trabalho foi parcialmente financiado pelo CNPq (Processos nºs 403612/2020-9, 311470/2021-1 e 403827/2021-3), pela Fundação de Amparo à Pesquisa de Minas Gerais (FAPEMIG) (Processos nºs APQ-00810-21 e APQ -03162-24) e pelo projeto /2023, assinado com a EMBRAPII.

References

[1] M. C. de Araújo Urbano, *Mobilidade Ativa: condicionante da Sustentabilidade Urbana nas vias históricas do bairro de São José–Recife*. PhD thesis, PUC-Rio.
[2] D. R. PIRES, "Estratégias para políticas públicas de mobilidade urbana sustentável para cidades brasileiras de pequeno porte," 2020.
[3] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLO," Jan. 2023.
[4] D. Situnayake and J. Plunkett, *AI at the Edge*. " O'Reilly Media, Inc.", 2023.
[5] A. S. d. Macena, "Rastreamento de múltiplos objetos utilizando modelos de aprendizado profundo em hardware limitado," 2021.
[6] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.
[7] P. Skalski, "How to estimate speed with computer vision," January 19 2024. Roboflow Blog.
[8] D. M. Leal, "Detecção e rastreamento de objetos em vídeo via rede neural convolucional (cnn): Yolo e deepsort aplicados para contar veículos e estimar suas velocidades médias a partir de referencial fixo," 2023.
[9] D. C. Luvizon, B. T. Nassu, and R. Minetto, "A video-based system for vehicle speed measurement in urban roadways," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 6, pp. 1393–1404, 2016.
[10] V. Kocur and M. Ftáčnik, "Detection of 3d bounding boxes of vehicles using perspective transformation for accurate speed measurement," *Machine Vision and Applications*, vol. 31, no. 7, p. 62, 2020.
[11] D. IC, "tipos-de-veiculos dataset." https://universe.roboflow.com/dataset-ic/tipos-de-veiculos , apr 2023. visited on 2024-06-01.
[12] T. Opsahl, "Estimating homographies from feature correspondences," *UNIK. Available: Estimating homographies from feature correspondences*, 2017.