APLICAÇÃO DE ALGORITMOS DE VSLAM E RECONHECIMENTO DE OBJETOS PARA CLASSIFICAÇÃO DE VEÍCULOS E VAGAS DE ESTACIONAMENTO EM AMBIENTES EXTERNOS UTILIZANDO UM DRONE

Pedro Henrique de Oliveira Garcia[1]; Roberto Scalco[2]

[1] Aluno de Iniciação Científica da Escola de Engenharia Mauá (EEM/CEUN-IMT);
[2] Professor da Escola de Engenharia Mauá (EEM/CEUN-IMT).

Resumo. *A proposta deste projeto consistiu na aplicação de técnicas de visão computacional e classificação de imagens obtidas pela câmera monocular de um drone, realizando a localização e classificação de veículos e vagas em estacionamentos em campo aberto. A técnica de mapeamento e localização simultâneas, do inglês vSLAM (visual Simultaneous Localization and Mapping), possibilitou que o drone pudesse obter imagens dos estacionamentos contendo sua informação de localização espacial. O algoritmo de classificação ofereceu bons resultados na validação, o qual, foi capaz de identificar e exibir corretamente grande parte das classes das vagas (ocupada e vazia), daquelas que conseguiu identificar. A rede neural profunda YOLO foi aplicada para realizar a localização dos veículos e das vagas, permitindo determinar se as vagas estavam ocupadas ou não.*

Introdução

A robótica móvel é uma área em grande ascensão nas últimas décadas (MOREIRA, 2017). Pode ser dividida em robôs terrestres, aquáticos e aéreos, sendo esse último o objeto de estudo deste trabalho. Os drones têm grande relevância e popularidade no cenário comercial atual, pois seu custo é relativamente baixo, além de terem dimensões reduzidas (AL-SA'D et al., 2019).

Merino e seus colaboradores (2006) indicam que um dos avanços tecnológicos responsáveis pela ascensão dos drones foi a visão computacional, que permitiu a utilização de algoritmos de navegação, identificação e classificação de objetos, entre outras técnicas computacionais. Alguns usos de drone com essas tecnologias são: a agricultura de precisão (JORGE; INAMASU, 2014); busca e salvamento (DOHERTY; RUDOL, 2008) e navegação autônoma (ZINGG et al.,2010).

Ramos (2012), bem como Cadena et. al. (2016) apontam que os algoritmos de localização e mapeamento simultâneos SLAM possibilitam a criação de mapas de ambiente de forma dinâmica, a partir das informações da localização do drone. Uma metodologia do SLAM, que utiliza apenas sensores visuais, como câmeras, é a visual SLAM (vSLAM). Dessa maneira, outros sensores não são necessários para obter informações sobre o ambiente em que o drone está imerso (RAMOS, 2012). Alguns casos implementados são o de Gómez et al. (2016) que construíram um sistema multirrobótico para monitoramento e navegação em *greenhouses*, enquanto que Pestana e sua equipe (2015) utilizaram o vSLAM para que um drone pudesse navegar e mapear um ambiente utilizando marcadores terrestres.

O objetivo deste trabalho é a localização e classificação de vagas e veículos em ambientes externos, como por exemplo, estacionamentos de parques temáticos, shopping centers e *Campi* universitários, utilizando drone equipado com uma câmera monocular. A localização das vagas em ambientes externos é relevante para organização de eventos, pois possibilita melhor distribuição dos usuários no espaço. A classificação das vagas possibilita determinar irregularidades no ambiente monitorado, como veículos estacionados em local irregular, veículos especiais (como ambulâncias ou bombeiros) que precisam ter rotas de escoamento sempre disponíveis.

Material e Métodos

Esta seção traz a descrição detalhada dos equipamentos, tecnologias, métodos e modelos utilizados para o desenvolvimento do projeto.

Drone DJI Tello

O drone Tello, fabricado pela DJI, utiliza o processador Intel Movidius Myriad e oferece sistema de voo com estabilização avançada, sensores de colisão e o fabricante oferece ao usuário um SDK (*Software Development Kit*), para desenvolvimento de aplicações com o drone (SHENZHEN-RYZE, 2018). O DJI Tello, apresentado na Figura 1, foi escolhido devido ao bom desempenho na transmissão do vídeo, custo relativamente baixo (ordem de R$ 600,00) comparado a outros drones da mesma empresa e as seguintes características:

a) câmera: 5 megapixels;
b) qualidade do vídeo: HD (720p);
c) autonomia da bateria: 13 min;
d) alcance: 100 m;
e) velocidade máxima: 28 km/h.

Figura 1: Drone DJI Tello.
Fonte: Shenzhen-Ryze (2018).

OpenCV

A OpenCV (*Open Source Computer Vision*) é uma biblioteca de programação, de código aberto e inicialmente desenvolvida pela Intel com o objetivo de tornar a visão computacional acessível aos programadores. A biblioteca possui módulos de processamento de imagens e vídeo, GUI (interface gráfica do usuário) básica com sistema de janelas independentes, detecção de mouse e teclado, além de mais de 350 algoritmos de visão computacional como: filtros de imagem, calibração de câmera, reconhecimento de objetos entre outros (OPENCV, 2019).

Google Colaboratory

O Google Colaboratory ou Colab, como é chamado pela própria fabricante, é um serviço de nuvem gratuito hospedado para incentivar a pesquisa nas áreas de aprendizagem de máquina e inteligência artificial que executa a linguagem Python de programação na versão 3.6.9 atualmente. As principais características do Colab são: a não é necessidade de realizar configurações, disponibilidade gratuita ao acesso à GPU (*Graphics Processing Units*), simples de conectar o *notebook* de trabalho (ambiente de desenvolvimento do código-fonte) com o serviço Google Drive para troca de arquivos (GOOGLE, 2019).

YOLO

A YOLO (*You Only Look Once*) é uma rede neural convolucional que extrai características (*features*) para a detecção de objetos em passada única (*single pass*). Devido a essa característica, a YOLO é capaz de conseguir uma velocidade na detecção muito maior do que as técnicas concorrentes, sem perder em acurácia. Em seu funcionamento a YOLO utiliza um modelo de aprendizagem profunda, cuja arquitetura é chamada de **Darknet**, que é o mesmo nome do *framework* utilizado para implementar o detector (BOCHKOVSKIY; WANG; LIAO, 2020). Esse

framework, desenvolvido pelo próprio criador da YOLO, Joseph Redmon, possui código aberto e foi escrito na linguagem C, também possuindo suporte para execução em GPU.

A Figura 2 apresenta o funcionamento do *framework* YOLO, ajustando trechos da imagem para um tamanho padronizado, em seguida, a imagem é processada pela rede neural e, por fim, são gerados os *bounding boxes* e aplicado o algoritmo de *non-max suppression*, para indicar apenas os *bounding boxes* que possuam maior probabilidade de indicar a presença de uma classe (pessoa, animal ou outro elemento que foi inserido na rede), destacados na imagem com os retângulos.

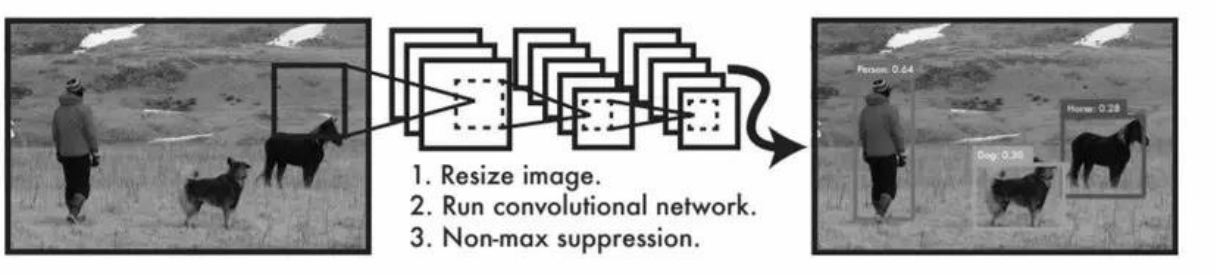

Figura 2 - Funcionamento do algoritmo de detecção de objetos do YOLO.
Fonte: Alves (2020).

Dataset Parking Lot

Para auxiliar no problema de classificação de vagas em estacionamentos, Almeida et al. (2015), apresentaram um conjunto de dados de imagens que em sua versão atualizada contempla distintas classes de vagas captadas em diferentes estacionamentos em condições climáticas variadas. O conjunto de dados PKLot contém 12.417 imagens de estacionamentos de duas universidades: Universidade Federal do Paraná (UFPR) e da Pontifícia Universidade Católica do Paraná (PUCPR), distribuídas em três condições climáticas distintas (ensolarado, nublado e chuvoso). O *dataset* fornece junto a cada imagem um arquivo com extensão xml (*eXtensible Markup Language*) contendo a localização de todas as vagas e sua condição. Ao total, 695.899 de vagas de estacionamento segmentados a partir de cada imagem estão disponibilizadas e foram verificadas e rotuladas manualmente.

Para validação do modelo são utilizados dois parâmetros de performance: a matriz de confusão e a acurácia, descritos a seguir:

A matriz de confusão, apresentada na Figura 3, representa uma tabela que possui as frequências de classificação para cada classe do modelo resultando nas seguintes situações:

a) verdadeiro positivo (VP): o modelo previu corretamente a classe;
b) falso positivo (FP): o modelo não previu corretamente a ocorrência da classe;
c) falso negativo (FN): o modelo previu incorretamente a existência do elemento da classe;
d) verdadeiro negativo (VN): o modelo previu corretamente a não existência da classe.

		Decisão do Modelo	
		Positivo	Negativo
Classificação	Positivo	Verdadeiro Positivo (VP)	Falso Positivo (FP)
	Negativo	Falso Negativo (FN)	Verdadeiro Negativo (VN)

Figura 3 - Matriz de confusão 2x2.
Fonte: Almeida et al. (2015).

A acurácia (ACC), indicada pela equação (1), representa um coeficiente que indica a proximidade de um resultado com o seu valor de referência real. Dessa maneira, quanto maior a acurácia, mais próximo da referência é o resultado obtido.

$$ACC = \frac{\sum VP + \sum VN}{População} \quad (1)$$

Open VSLAM

O *framework OpenVSLAM* é um sistema *SLAM* visual monocular, estéreo e RGBD (*Red Green Blue Depth*) que compreende abordagens *SLAM* conhecidas, encapsulando-as em vários componentes separados com interfaces de programação de aplicativos (API). Segundo Sumikura, Shibuya e Sakurada (2019) os principais recursos são: a compatibilidade com vários tipos de modelos de câmeras, facilidade de personalização para outros modelos de câmeras, além de permitir que mapas previamente criados possam ser carregados e armazenados.

Resultados e Discussão

A proposta deste trabalho consiste na implementação da arquitetura apresentada na Figura 4. Inicialmente deve ser estabelecida uma rede wi-fi entre o drone e o computador. Dessa maneira, é possível que a movimentação do drone seja realizada por comandos informados, via teclado, enquanto que o drone faz a captura e transmissão do vídeo para o computador. O vídeo recebido, na forma de *stream*, têm suas imagens são processadas e armazenadas individualmente pela biblioteca openCV. Além disso, essas imagens serão processadas em dois algoritmos: um de classificação de imagens (YOLO) e o outro de localização e mapeamento simultâneo (open vSLAM).

A partir da identificação das vagas disponíveis e do mapeamento do ambiente é possível avaliar as regiões em que existem vagas livres, além de indicar para o usuário a região do estacionamento em que isso ocorre.

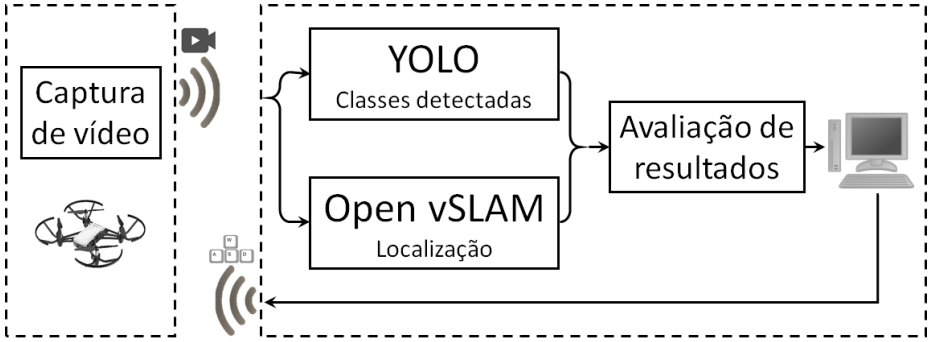

Figura 4 - Arquitetura do sistema.
Fonte: o autor.

A implementação da comunicação entre o drone e o computador foi realizada utilizando a linguagem de programação Python, em sua versão 3.6.9, no ambiente Microsoft Visual Studio Code 1.51.1. Tanto a YOLO quanto o open vSLAM foram utilizados no ambiente Google Colab, mantendo a mesma versão do Python.

A seguir, o funcionamento dos blocos do diagrama da Figura 4 serão detalhados.

Comunicação com o drone

Com a leitura da documentação do SDK do drone, fornecida pelo fabricante, os primeiros testes com a comunicação foram realizados com sucesso. A comunicação se inicia com a conexão à rede wi-fi que o próprio drone gera ao ser ligado, posteriormente é necessário o desenvolvimento de *software* para criar *threads* e *sockets* para monitoramento via protocolo UDP (*User Datagram Protocol*) das portas especificadas pelo fabricante sendo elas:

a) porta 8889: envio de comandos e recepção da resposta sobre o comando;
b) porta 8890: recepção da mensagem do estado do drone. Essa mensagem contém uma sequência de caracteres (string) com informações dos sensores do drone;
c) porta 11111: recepção de *stream* de vídeo.

Recepção e Exibição de Imagens

Assim que se estabeleceu corretamente a comunicação entre o drone e o computador, iniciaram-se os testes para recepção e exibição de imagem e para isso foi necessário o uso da biblioteca de visão computacional *OpenCV* que possui funções próprias para essa exibição, como mostra o resultado da Figura 5.

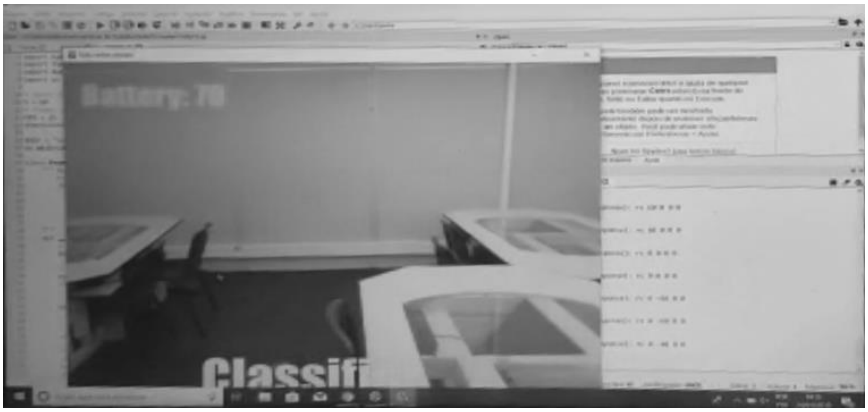

Figura 5 - Voo indoor de teste no Bloco P do Instituto Mauá de Tecnologia.
Fonte: O autor.

Controle de ações via Teclado

A documentação da *SDK* apresenta uma lista de comandos que podem ser enviados ao drone utilizando um *socket* criado para a porta de envio de comandos e recepção de respostas. Utilizando a contribuição de *software* livre de Escoté (2018), para auxílio no controle de envio de comandos via *software*, foi possível construir um programa para controlar a movimentação do drone, utilizando os comandos apresentados no Quadro 1.

Quadro 1 - Comandos para manipulação do drone.

Tecla	Legenda
W / S	Elevar ou abaixar
A / D	Rotacionar para esquerda ou direita
T / L	Levantar voo ou aterrissar
← / ↑ / → / ↓	Mover para esquerda, frente, direita ou trás

Fonte: O autor.

Gravação de vídeos

O OpenCV também conta com funções para armazenamento de vídeos e, para seu uso, foi necessária a criação de um novo *socket* para processamento paralelo, permitindo a recepção das imagens do drone e sua passagem como parâmetro para gerar um vídeo ao final do voo. Os vídeos armazenados foram utilizados nos testes dos algoritmos de detecção de objetos e localização e mapeamento.

YOLO

Posteriormente iniciou-se o trabalho com a detecção e classificação de elementos nas imagens utilizando redes neurais convolucionais. Para o treinamento e validação da rede neural YOLO, que possui pesos pré-treinados, foi utilizado o *dataset* COCO (*Common Objects in Context*) (Lin et al., 2014). Entretanto, para o objetivo do trabalho, foi necessário o treinamento da rede para detecção de apenas duas classes (vazia e ocupada) e, como as características de detecção para vagas desocupadas não estão presentes nesse *dataset*, houve a necessidade de treinar a rede neural com um *dataset* contendo essas duas as classes.

Utilizou-se o *dataset* ParkingLot de Almeida el al. (2015) que possui 12.417 imagens que no total contém 695.899 de vagas divididas entre vazias e ocupadas. A vantagem de sua utilização foi que junto a cada imagem, há um arquivo xml contendo a situação e localização de *bounding boxes* de cada vaga, porém para o treinamento da rede neural é necessária uma conversão de dados.

Tratamento do *dataset* e formatação para a YOLO

Os arquivos xml fornecidos com a localização da imagem possuem a seguinte estrutura, apresentada na Figura 6:

```
<parking id="pucpr">
  <space id="1" occupied="0">
    <rotatedRect>
      <center x="300" y="207" />
      <size w="55" h="32" />
      <angle d="-74" />
    </rotatedRect>
    <contour>
      <point x="278" y="230" />
      <point x="290" y="186" />
      <point x="324" y="185" />
      <point x="308" y="230" />
    </contour>
  </space>
```

Figura 6 - Estrutura do arquivo xml de cada imagem.
Fonte: Almeida et al. 2015.

Para o treinamento dessa versão do YOLO, os passos a seguir representam a estrutura do algoritmo desenvolvido para gerar os dados no formato que a rede neural YOLO aceita:

a) arquivo de extensão .txt de mesmo nome que a imagem;
b) cada vaga presente no arquivo xml deve ser convertida para o seguinte formato: <id_classe> <centro_do_objeto_x> <centro_da_objeto_y> <largura_objeto> <altura_objeto>;
c) o id da classe deve ser um número inteiro maior ou igual a zero;
d) os valores subsequentes ao id devem ser relativos às dimensões da imagem e estarem compreendidos entre 0 e 1. Ou seja, o produto entre esses valores e as dimensões da imagem os valores vai gerar a localização e dimensão exata independentemente do tamanho que a imagem esteja. Isso se deve pois durante o treinamento a rede neural redimensiona a imagem para a detecção de características.

Esse algoritmo é aplicado para mapear o caminho de todos os arquivos xml inicialmente, após o mapeamento realizar para cada arquivo a sua leitura, aquisição das informações necessárias, conversão para os padrões YOLO, escrita dos arquivos de texto e deleção do arquivo xml.

Com esses arquivos prontos, foi necessária a separação aleatória em diretórios de treino e de teste para a rede neural, garantindo que as diferentes condições climáticas estivessem distribuídas proporcionalmente nos diretórios. A proporção para a distribuição do total das imagens foi de 70% das imagens para treino e o restante para o teste da rede.

Treinamento da Rede Neural

Para o treinamento da rede neural, foi necessário utilizar o ambiente Colaboratory do Google, pois a YOLO v4 exige uma GPU para seu treinamento e neste ambiente é fornecido o acesso gratuito à uma Tesla T4 (16 GB RAM) por tempo limitado no plano gratuito.

Para o treinamento da rede neural é necessário clonar o repositório GitHub do autor da rede, alterar parâmetros de configuração para uso de GPU e compilar. Após a compilação é necessário

carregar os diretórios de imagens (treino e teste) e gerar um arquivo de texto (.txt) contendo os nomes das imagens para cada diretório. Também é necessário gerar um arquivo de dados (extensão .data) contendo a localização dos arquivos de treino, teste e um diretório para *backup*, que a cada 100 épocas de treino salva um arquivo de *last weights* para caso ocorra algum problema no treinamento e, por fim, uma cópia da estrutura da rede neural com um arquivo de extensão .cfg.

Após isso, foi utilizada uma técnica denominada transferência de aprendizagem, em que os desenvolveres Bochkovskiy, Wang e Liao (2020) disponibilizam um arquivo de pesos iniciais para e com alguns filtros previamente estabelecidos para acelerar algumas épocas dos treinamentos de outras redes neurais de mesma arquitetura.

Com todas as configurações prontas inicia-se o treinamento da rede neural executando o arquivo *darknet* do diretório e passando como parâmetro a palavra *"train"* e seus parâmetros obrigatórios. A Figura 7 representa uma época do treinamento da rede neural, indicando que ainda restam mais de 28 horas para a finalização.

```
[ ] !./darknet detector train data/vagas.data cfg/yolov4vagas.cfg /ic/recursos/yolov4vagas_last.weights -dont_show -map
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.745531), count: 134, total_loss = 882.854736
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, total_loss = 0.001273
 total_bbox = 644115, rewritten_bbox = 0.023754 %

 (next mAP calculation at 1239 iterations)
 741: 982.352356, 1167.759888 avg loss, 0.000392 rate, 20.794979 seconds, 47424 images, 28.182314 hours left
Loaded: 0.000034 seconds
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.734147), count: 190, total_loss = 3685.728516
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.711451), count: 129, total_loss = 717.935974
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.787725), count: 4, total_loss = 10.861202
 total_bbox = 644438, rewritten_bbox = 0.023742 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.748993), count: 188, total_loss = 3594.379883
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.730398), count: 119, total_loss = 687.170654
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.775967), count: 3, total_loss = 4.333868
 total_bbox = 644748, rewritten_bbox = 0.023730 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.692705), count: 98, total_loss = 1500.169434
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.700280), count: 70, total_loss = 336.386658
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.659291), count: 5, total_loss = 7.268089
 total_bbox = 644921, rewritten_bbox = 0.023724 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.766563), count: 270, total_loss = 5344.409668
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.710338), count: 169, total_loss = 939.752686
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.732208), count: 3, total_loss = 7.638243
 total_bbox = 645363, rewritten_bbox = 0.023708 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.713661), count: 140, total_loss = 2029.624756
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.756664), count: 128, total_loss = 600.745728
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.647702), count: 9, total_loss = 12.521272
 total_bbox = 645640, rewritten_bbox = 0.023697 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.745429), count: 231, total_loss = 5102.507812
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.714526), count: 86, total_loss = 580.222168
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, total_loss = 0.091850
 total_bbox = 645957, rewritten_bbox = 0.023686 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.692541), count: 113, total_loss = 1489.356812
```

Figura 7 - Google Colaboratory executando o treinamento da rede na época 749.
Fonte: O autor.

Avaliação do Modelo

Esse repositório compilado possui um método para a validação do treinamento chamado *"map"*, que irá realizar a validação do modelo com os arquivos de teste e, utilizando a matriz de confusão, irá gerar uma acurácia para o modelo a partir dos pesos fornecidos. Assim, acompanhando a evolução da matriz de pesos foi possível gerar um gráfico da acurácia em função das épocas, apresentado na Figura 8. Além de possibilitar a avaliação se os pesos do modelo chegaram a um estágio de *overfitting*.

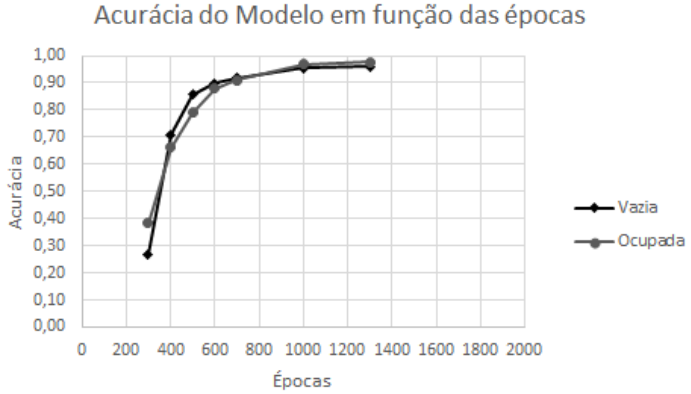

Figura 8 - Gráfico da acurácia do modelo em função das épocas de treinamento.
Fonte: O autor.

Assim, com alta acurácia no modelo, decidiu-se verificar o resultado utilizando uma imagem passada por parâmetro. O resultado, apresentado na Figura 9, possui retângulos indicando vagas ocupadas e livres, com a respectiva probabilidade de acerto:

Figura 9 - Processamento de uma imagem para identificação de vagas.
Fonte: o autor, com base na imagem de Almeida et al. 2015.

Passagem do vídeo para o algoritmo

A instalação do *framework* inicia-se clonando o repositório GitHub do grupo de autores Sumikura, Shibuya e Sakurada (2019), bem como *softwares* necessários para o seu funcionamento. Para o uso do *software* é necessário passar os parâmetros de configuração, salvamento para uso posterior e se há necessidade ignorar alguns *frames* para processamento mais rápido. O uso do *software* gerou o seguinte resultado:

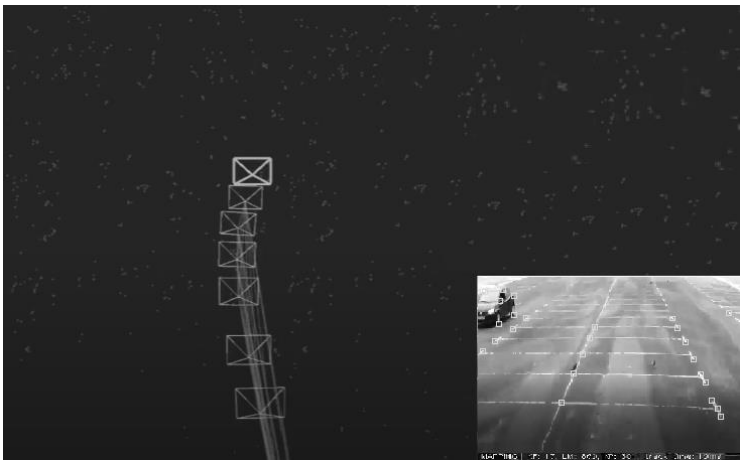

Figura 10 - Resultado do processamento de imagem pelo algoritmo de vSLAM.
Fonte: o autor.

A Figura 10 apresenta os seguintes elementos: à esquerda existem retângulos com retas ligando os vértices que representam as posições no espaço em que os *frames* foram retirados. As retas que ligam os retângulos representam o deslocamento no espaço entre os *frames* subsequentes. Os pontos representam as *features* que o algoritmo posicionou na imagem do voo e os transpôs para o espaço. A imagem ao lado direito é o *frame* do vídeo do voo do drone que está sendo lido pelo algoritmo no momento.

Para a implementação do projeto foi utilizado um computador Lenovo Ryzen 3, com 1,7 GHz, 8 GB RAM, 256 GB SSD e com adaptador de vídeo integrado AMD RADEON RX

Vega 8 com memória compartilhada, que foi fator limitante pelo fato do processamento de imagens exigir considerável desempenho do computador e o *hardware* utilizado não possuía capacidade para realizar as operações de forma satisfatória em tempo real.

Outro fator limitante foi a disponibilidade por curtos períodos do uso da GPU do Google, que fizeram com que o treinamento demorasse dias, e qualquer ajuste de parâmetro do treinamento levaria um tempo de treino extremamente alto para posterior validação do modelo.

Por fim, o *framework* de open vSLAM não oferece suporte para uso em tempo real, apenas para vídeos gravados e esse seria um fator que possibilitaria apenas as informações de localização ao usuário com o retorno do drone e o processamento do vídeo resultante.

Conclusões

Foi possível o desenvolvimento de um *software* de comunicação e controle do drone, a partir do computador, com um resultado conforme esperado tanto com relação ao envio de comandos, bem como recepção e exibição da *stream* de vídeo.

Além disso, foi possível realizar o treinamento e validação da rede neural profunda YOLO, e implementar a identificação de vagas livres e ocupadas do estacionamento, a partir de imagens de teste. Entretanto, para que o processamento da rede neural seja realizado em tempo real, em conjunto com o controle do voo do drone, é necessário o processamento em uma GPU.

As limitações do desempenho do *hardware* utilizado, bem como a falta de suporte do *framework* open vSLAM para uso em tempo real foram fatores de limitações do projeto.

Agradecimento

Ao professor Murilo Zanini de Carvalho por suas contribuições sobre algumas tecnologias que está estudando em seu projeto de Doutorado.

Referências Bibliográficas

Al-Sa'D, M. F.; Al-Ali, A. K.; Mohamed, A.; Khattab, T.; Erbad, A. (2019) RF-based drone detection and identification using deep learning approaches: an initiative towards a large open source drone database. *Future Generation Computer Systems*, **100**, 86-97.

Almeida, P. R. L. de; Oliveira, L. S.; Britto Jr., A. S.; Silva Jr., E. J. ; Koerichbc, A. L. (2015) PKLot - a robust dataset for parking lot classication. *Expert Systems with Applications*, **42**, n. 11, 4937-4949.

Alves, G. (2020) Detecção de objetos com YOLO – uma abordagem moderna. *IA Expert Academy*. 13. out. 2020. Disponível em: <https://tinyurl.com/ALVES-G-2020>. Acesso em: 29 nov, 2020.

Bochkovskiy, A.; Wang, C.-Y.; Liao, H.-Y. M. (2020) YOLOv4: optimal speed and accuracy of object detection. *Computer Vision and Pattern Recognition*. arXiv preprint arXiv:2004.10934.

Cadena, C.; Carlone, L.; Carrillo, H.; Latif, Y.; Scaramuzza, D.; Neira, J.; Reid I.; Leonard, J. J. (2016) Past, present, and future of simultaneous localization and mapping: toward the robust perception age. *IEEE Transactions on Robotics*, **32**, 1309–1332.

Doherty, P.; Rudol, P. (2007) A UAV search and rescue scenario with human body detection and geolocalization. In: *Advances in Artificial Intelligence*. Springer, Berlin, Heidelberg. **4830**, 1-13.

Escoté, D. F. (2018) DJI Tello drone python interface using the official Tello SDK. feel free to contribute!. Disponível em: <https://tinyurl.com/escote-2018>. Acesso em: 29 dez. 2018.

Gómez, J. J. R.; Auñón, P. G.; Garzón, M.; Rivas, J. L.; Cerro, J.; Barrientos, A. (2016) Heterogeneous multi-robot system for mapping environmental variables of greenhouses. *Sensors*, **16**, 163-190.

Google. (2019) Olá, este é o Colaboratory. Disponível em: <https://tinyurl.com/gColab-2019>. Acesso em: 26 jun. 2020.

Jorge, L. A. de C.; Inamasu, R. Y. (2014) Uso de veículos aéreos não tripulados (VANT) em agricultura de precisão. In: Bernardi, A. C. de C.; Naime, J. de M.; Resende, A. V. de; Bassoi, L. H.; Inamasu, R. Y. (Ed.). *Agricultura de precisão*: resultados de um novo olhar. Brasília, DF: Embrapa, 109-134.

Merino, L.; Wiklund, J.; Caballero, F.; Moe, A.; Dios, J.; Forssén, P. Nordberg, K.; Ollero, A. (2006) Vision-based multi-UAV position estimation: localization based on blob features for exploration missions. *IEEE Robotics & Automation Magazine*, **13**, n. 3, 53-62.

Moreira, A. H. (2017) *Sistema multirrobótico descentralizado no controle de posição e formação por quadricópteros*: uma integração entre o mundo virtual e real. Dissertação (Mestrado em Ciências da Computação) – Universidade Federal do ABC, Santo André, 118 p.

OpenCV (2019) OpenCV-Python Tutorials - v. 3.4.3. Disponível em: <https://tinyurl.com/openCV-2019>. Acesso em: 25 jun. 2020.

Pestana, J.; Sanchez-Lopez, J. L.; Puente, P.; Carrio, A.; Campoy, P. (2014) A vision-based quadrotor multi-robot solution for the indoor autonomy challenge of the 2013 international micro air vehicle competition. In: *International Conference on Unmanned Aircraft Systems (ICUAS)*. Orlando. 617–622.

Ramos, J. M. G. de R. (2012) *SLAM for drones*: simultaneous localization and mapping for autonomous flying robots. Dissertação (Mestrado em Ciências da Computação) – École Nationale Supérieure d'Arts et Métiers, Paris, 185 p.

Shenzhen-Ryze Technology Co.. (2018) Tello Official Website. Disponível em: <https://tinyurl.com/Shenzhen-Ryze-2018>. Acesso em: 25 jun. 2020.

Sumikura, S; Shibuya, M.; Sakurada, K. (2019). OpenVSLAM: a versatile visual SLAM framework. In: *Proceedings of the 27th ACM International Conference on Multimedia* (MM '19). Association for Computing Machinery, New York, NY, USA, 2292–2295.

Lin, T.-Y.; Maire, M.; Belongie, S. J. Bourdev, L. D.; Girshick, R. B.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C. L. (2014) Microsoft COCO: Common Objects in Context. *CoRR* abs/1405.0312.

Zingg, S.; Scaramuzza, D.; Weiss, S.; Siegwart, R. (2010) MAV navigation through indoor corridors using optical flow. In: *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, 3361-3368.