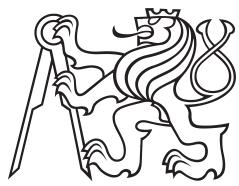


Bachelor Project



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Drone detection using neural networks from combined RGB camera and LiDAR data

Adam Škuta

Supervisor: Matouš Vrba
Supervisor–specialist: Martin Saska
Field of study: Mathematical Engineering
Subfield: Mathematical Modelling
February 2017

Acknowledgements

I would like to express my utmost gratitude to my supervisor Ing. Matouš Vrba for his guidance and great support during my time writing this thesis.

I would also like to thank my parents and friends for their huge support and help.

Declaration

Prehlasujem, že som predloženú prácu vypracoval samostatne a ,že som uviedol všetku použitú literatúru.

V Praze, 10. February 2017

Abstract

The detection of drones using neural networks from combined RGB and LiDAR data is tackled in this thesis. Multiple approaches for RGB and LiDAR data fusion into RGBD data are presented. The detection of the drone is realized via a Convolutional neural network. All methods are trained and tested and the results are compared with the RGB only method. The dataset used in this thesis was generated using multiple Virtual environments. The results prove that there is a difference between RGBD and RGB approaches and in some specific scenarios RGBD approaches prove to be better than RGB ones in drone detection. Further work on these methods is, therefore, a worthwhile study.

Keywords: Drone, Convolutional neural network, Detection, LiDAR, Sparse pointcloud

Supervisor: Matouš Vrba
Ústav X,
Uliční 5,
Praha 99

Abstrakt

Detekcia dron pomocou neurónových sietí z kombinovaných dát RGB kamery a LiDARu je popísaná v tejto práci. Viacero spôsobov kombinácie RGB kamery a LiDARu je prezentovaných. Samotná detekcia dron je realizovaná pomocou Konvolučnej neurónovej siete. Všetky metódy sú natrénované a otestované a následné výsledky sú porovnané s čistou RGB kamerou. Dáta používané v tejto práci boli vygenerované pomocou viacerých virtuálnych prostredí. Výsledky dokazujú, že v niektorých prípadoch RGBD metódy produkujú lepšie výsledky oproti RGB kamere v kontexte detektie dronov. Budúca práca preto predstavuje ďalšiu zaujímavú štúdiu.

Klíčová slova: Dron, Konvolučná neurónová sieť, Detekcia, LiDAR, Riedky pointcloud

Překlad názvu: Moje bakalárka se strašně, ale hrozně dlouhým předlouhým názvem — Cesta do tajů kdovíčeho

Contents

1 Introduction	1	3 Results	21
1.1 Related Work	1	4 Conclusion	25
1.2 Problem Statement	3	Bibliography	27
2 Methodology	5		
2.1 Sparse to dense.....	6		
2.2 YOLOv3	7		
2.3 Image inpainting method based on the Fast Marching Method	9		
2.4 Coordinate systems	11		
2.5 Camera Model	12		
2.6 Dataset.....	13		
2.6.1 Unreal Engine	14		
2.6.2 AirSim	15		
2.7 Training	16		
2.7.1 Sparse to Dense.....	16		
2.7.2 Image inpainting	16		
2.7.3 YOLOv3.....	17		

Figures

1.1 Schema of the process.	3
2.1 Example of different network architectures available. Taken from [1].....	6
2.2 YOLOv3 network architecture. Taken from [2].	8
2.3 Inpainting principle. Image from: [3]	10
2.4 Example of the inpainting technique. Image from: [3].	11
2.5 Visualization of transformation.	12
2.6 Pinhole camera model. Image from [4].....	13
2.7 Unreal Engine user interface....	14
2.8 Parrot AR.Drone 2.0. Image taken from [5]......	15
2.9 Sample photos from each environment.	15
2.10 Sparse to dense training.....	17
2.11 Applied filter on Sparse to dense output.....	18
2.12 Inpaint results.	18
2.13 Sample YOLOv3 outputs.	19
2.14 Training results.....	19
3.1 Recall based on distance of the drone.	22

Tables

2.1 Chosen weights. 19

3.1 Testing results..... 22

3.2 Inference speeds of all methods. 23

ctuthesis t1606152353

Chapter 1

Introduction

In this thesis, drone detection using neural networks from combined RGB camera and LiDAR data is studied. With the recent development of drone technology, drones have become more readily available to the public and could only be expected to rise in popularity in the future. Drone detection is an important problem to tackle when it comes to tasks such as interception of uncooperative drones [6] or localization of drones in swarm [7] [8]. All methods presented in this thesis belong to the relative localization category. Compared to absolute localization methods, relative localization does not need to rely on pre-existing ground infrastructure and can be used in more environments. A Light detecting and ranging (LiDAR) sensor has been utilized on drones on multiple occasions [9] [10] [11] and RGB camera provides an easily accessible, cheap and lightweight sensor. Therefore a fusion of data retrieved from both sensors and its impact on the overall drone detection problem is an interesting problem to tackle. The results can be useful for occasions where a drone is already equipped with a LiDAR sensor. Or they can provide further clarification, whether the addition of LiDAR sensor into the drone detection problem provides any advantage.

1.1 Related Work

Solving the problem of drone detection has been tackled in different ways. They mostly differ from the sensors that are utilized or by placing markers on flying drones.

- **Static Sensors** Current State of the art drone detection techniques utilizing static sensors such as Radars, Acoustic sensors or Wi-Fi rely on pre-existing infrastructure. In [12] Radar technology faces multiple challenges when detecting smaller aircrafts such as drones, because of their size and altitude of their flight, but shows promising results. When it comes to acoustic sensors, sound of the drone can be a very good source of information but depends heavily on the ambient sound of its environment[12]. In [13] drones are detected using Wi-Fi packets that are sent between the drone and its user. This method relies on the fact that most commercially available drones utilize Wi-Fi packets when it comes to communication with the end user. Only presence of the drone in the vicinity can be achieved and not its exact location.

- **Marker based detection** Part of relative localization techniques utilizes a set of markers that are placed on a target drone. In [14] an observer drone utilizing a spherical camera with 360 deg Field-of-View detects and localizes drones equipped with markers. This system is applied in real-time and provides accurate localization within 4cm. Another approach is provided in [15], where ultra-violet emitters are equipped on a drone and serve as markers for the detection of the ultra-violet sensor. This approach proved to be successful in real-life situations. The downside of utilizing visual markers on target drones is that a hardware modification of the target drone is required. Therefore this approach is not applicable to all situations, especially when the target drone is not cooperating.

- **Convolutional neural networks** Due to its popularity and advancements, object detection convolutional neural networks have become a very popular technique for relative drone detection such as YOLOv3 [16], CenterNet [17], RetinaNet [18] and FasterRCNN [19]. One approach of using a convolutional neural network is described in [20]. This approach is similar to the approach presented in this thesis in the sense that it fuses RGB and LiDAR data and use them as input into the modified YOLOv2 [21] convolutional neural network. This approach represents the LiDAR point cloud as a Birds-eye-view map and outputs 7 degrees of freedom for detected objects. Another approach described in [22] uses a convolutional neural network for the use of relative micro aerial vehicle detection. This approach uses tiny-YOLO architecture [21] and estimates the distance of the detected vehicle from the size of its bounding box. The results prove to be applicable in real-world scenarios. For this thesis a YOLOv3 architecture is utilized, due to its speed and simplicity making it easier for utilization of LiDAR data.

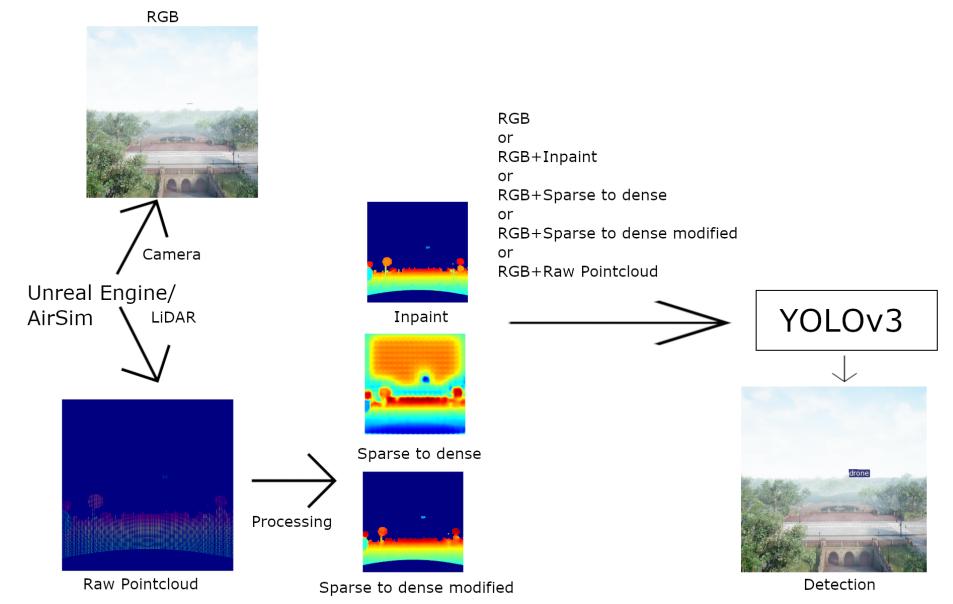


Figure 1.1: Schema of the process.

1.2 Problem Statement

The goal of this thesis was to examine whether usage of LiDAR data coupled with RGB images from camera is useful for the localization of drones in contrast to the usage of image data alone. The LiDAR and RGB camera were mounted on top of the observer drone, which took pictures and point clouds of the target drone. All the measurements were taken inside a virtual environment, with a realistic drone and sensor simulation. The dataset was then processed and used as the input for training and testing a Convolutional neural network for the object detection as can be seen in Figure 1.1. The preprocessing was as follows:

- coordinate transformation for the non-matching coordinate systems,
- projection of 3d points into a 2d image,
- utilizing different processing methods on sparse LiDAR point cloud to make it denser,
- fusing RGB images and LiDAR data into RGBD images.

The output metrics were then compared with the RGB trained convolutional neural network metrics.

Chapter 2

Methodology

In this chapter, various methods used in this thesis are discussed. A video game engine Unreal Engine¹ paired with plugin AirSim² was used for simulating real-life environments and for generating the dataset.

One approach in this thesis was to input generated RGB data from the dataset and use it as an input into the YOLOv3 Convolutional neural network for drone detection.

Another approach was to use Sparse to dense Convolutional neural network, which takes sparse LiDAR pointcloud and RGB image as an input and outputs dense LiDAR pointcloud. The neural network was pre-trained and its output was concatenated with RGB images and used as an input into the YOLOv3 for drone detection

Another approach was to use the output of Sparse to dense network and further process the data, eliminating sections of the depth image that were very sparse and using the output concatenated with RGB images to use as input for YOLOv3.

A separate approach was to use the inpainting method from the OpenCV Python library, which takes sparse LiDAR pointcloud and fills in unknown values based on the nearby known values, outputting dense pointcloud. The output of the inpainting method is further processed to filter inpainted values that were very far from known values and concatenated with RGB images to use as an input into YOLOv3.

¹<https://www.unrealengine.com>

²<https://microsoft.github.io/AirSim/>

2. Methodology

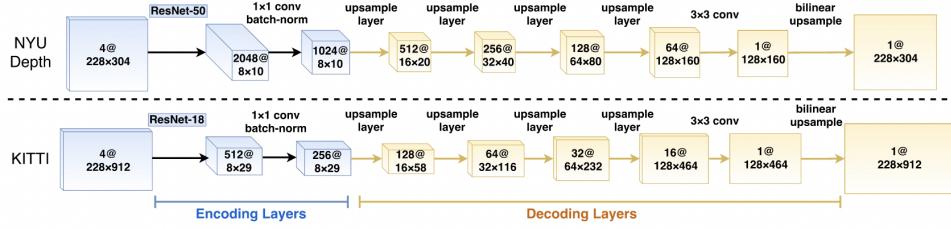


Figure 2.1: Example of different network architectures available. Taken from [1].

The final approach was to concatenate sparse LiDAR pointcloud with RGB images and use it as an input into YOLOv3.

2.1 Sparse to dense

Sparse to dense is a Convolutional neural network written in PyTorch. [1] It predicts the depth measurements from a sparse depth dataset. The size of the network is modifiable and can be chosen as training parameters. For the encoding and therefore input layers a ResNet-50 or ResNet-18 can be chosen, depending on the size of the input image for memory constraints as can be seen in Figure 2.1. The decoding layers consist of 4 upsampling layers and a deconvolutional layer with either stride 2 or 3 or upprojection layer or upconvolutional layer as a choice for training. The default loss function is least absolute deviations also known as L_1 error:

$$L_1 = \sum_{i=1}^n |y_{true} - y_{predicted}|, \quad (2.1)$$

where:

- n is batch size,
- y_{true} are real depth values,
- $y_{predicted}$ are predicted depth values.

The input to the network are RGBD images and the output is a depth map with the same dimensions as input. The depth input D is sampled from the

ground truth depth map D^* with the following formula:

$$D(i, j) = \begin{cases} D^*(i, j), & \text{with probability } p, \\ 0, & \text{otherwise,} \end{cases} \quad (2.2)$$

where:

- i, j are coordinates of the input image,
- $p = \frac{m}{n}$, where m number of depth samples to be chosen at the start of training and n is the total amount of available depth samples.

During training several input data augmentations take place. These augmentations include:

- scaling the input image by a random number $s \in [1, 1.5]$,
- rotating the input image by a random degree $r \in [-5, 5]$,
- scaling the brightness, contrast and saturation of the RGB component of the image by a random number $k \in [0.6, 1.4]$,
- normalizing the RGB component of the image,
- flipping the image horizontally with a 50% chance.

The output of the network is a dense depth image with the dimensions of the input. Every pixel contains predicted depth measurement in meters. The output of the Sparse to Dense network will be used for further training later.

2.2 YOLOv3

You only look once (YOLO) is a convolutional neural network model used mainly for object detection and recognition[23]. The main advantage is its simplicity in comparison to similar convolutional neural networks, resulting in faster detection speeds. It belongs to the state of the art convolutional neural networks for object detection and recognition. The version used in this work is the third version YOLOv3[16]. The backbone called Darknet53 consists of 53 convolutional layers. The original detector consists of 3 detection layers

2. Methodology

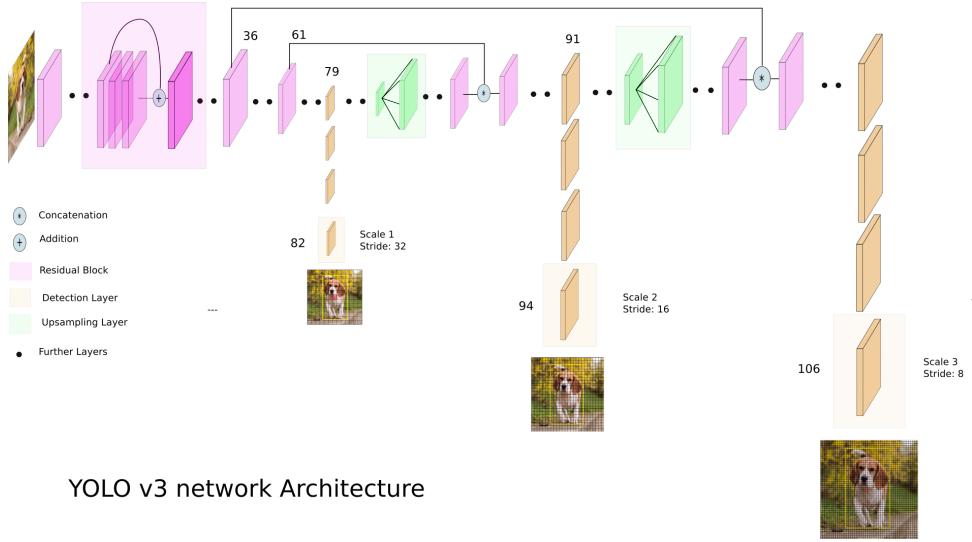


Figure 2.2: YOLOv3 network architecture. Taken from [2].

each responsible for detecting objects of various sizes as referred in Figure 2.2. At each detection layer, the image is divided into multiple grid cells, where each grid cell detects three bounding boxes. YOLOv3 takes n -channel images as the input. Each of the detection layers outputs three bounding boxes for each of the cells. The content of one bounding box is as follows:

- t_x, t_y, t_w, t_h are bounding box co-ordinates,
- p_O is objectness score,
- p_c is class score for each class in the dataset.

The bounding box t_x and t_y coordinates are relative to the upper-left corner of its respective cell, while t_w and t_h are relative to one of the anchors. Anchors are pre-defined default bounding box sizes and can be modified before training. To transform these bounding box coordinates to be relative to the image following transformation is applied:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x, \\ b_y &= \sigma(t_y) + c_y, \\ b_w &= p_w e^{t_w}, \\ b_h &= p_w e^{t_h}, \end{aligned} \tag{2.3}$$

where:

- $\sigma(x)$ is a sigmoid function,

2.3. Image inpainting method based on the Fast Marching Method

- c_x, c_y are grid cells offsets from the top left corner of the image,
- p_w, p_h are anchors width and height respectively,
- b_x, b_y, b_w, b_h are bounding box coordinates relative to the image size.

The loss function used during the training is sum squared error loss or L_2 error described as follows:

$$L_2 = \sum_{i=1}^n (\hat{\mathbf{t}} - \mathbf{t})^2, \quad (2.4)$$

where:

- n is batch size,
- $\hat{\mathbf{t}}$ is vector of predicted bounding box co-ordinates,
- \mathbf{t} is vector of ground truth bounding box coordinates which can be obtained by inverting transformation in (2.3).

2.3 Image inpainting method based on the Fast Marching Method

Image inpainting is a method used for reconstructing missing values in the image. One such method based on [3] called the Image inpainting method based in the Fast Marching Method is presented in this section. A depth image is provided as the input. The same method can be extended for RGB images, but won't be needed for this thesis

The depth value of a pixel to be inpainted is determined by the known neighbouring pixel values. To compute a depth value from one close pixel the following formula is used:

$$I_q(\mathbf{p}) = I(\mathbf{q}) + \nabla \mathbf{I}(\mathbf{q}) \cdot (\mathbf{p} - \mathbf{q}), \quad (2.5)$$

where:

- \mathbf{q} is a vector of pixel coordinates with known depth value,
- \mathbf{p} is a vector of pixel coordinates with unknown depth value,

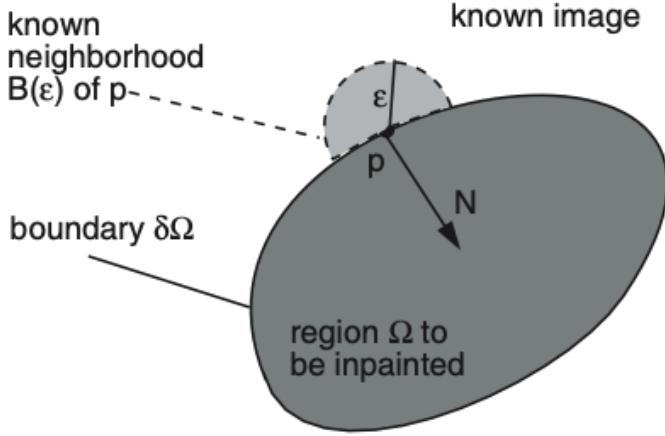


Figure 2.3: Inpainting principle. Image from: [3]

- $I(x)$ is a depth value at pixel coordinates \mathbf{x} ,
- $\nabla I(\mathbf{x})$ is a gradient vector at pixel coordinates \mathbf{x} .

To get a final value for the unknown pixel the Equation (2.5) is applied on all known pixels in a specified region $B_\varepsilon(p)$ from the unknown pixel as can be seen in Figure 2.3. The function is as follows:

$$I(\mathbf{p}) = \frac{\sum_{\mathbf{q} \in B_\varepsilon(\mathbf{p})} w(\mathbf{p}, \mathbf{q}) I_q(\mathbf{p})}{\sum_{\mathbf{q} \in B_\varepsilon(\mathbf{p})} w(\mathbf{p}, \mathbf{q})}, \quad (2.6)$$

where $w(\mathbf{p}, \mathbf{q})$ is a weighting function designed for propagating sharpness of the image and is obtained as the product of the following equations:

$$\begin{aligned} dir(\mathbf{p}, \mathbf{q}) &= \frac{\mathbf{p} - \mathbf{q}}{\|\mathbf{p} - \mathbf{q}\|} \cdot \mathbf{N}(\mathbf{p}), \\ dst(\mathbf{p}, \mathbf{q}) &= \frac{1}{\|\mathbf{p} - \mathbf{q}\|^2}, \\ lev(\mathbf{p}, \mathbf{q}) &= \frac{1}{1 + |T(\mathbf{p}) - T(\mathbf{q})|}, \end{aligned} \quad (2.7)$$

where,

- $\mathbf{N}(\mathbf{x})$ is a normal vector of the boundary to be inpainted at pixel \mathbf{x} ,
- $T(\mathbf{x})$ is distance of pixel \mathbf{x} to the inpainting boundary.

The (2.6) is iteratively applied to all pixels on the inpainting boundary and advances inside the region to be inpainted until the whole region has been filled. This is implemented via the Fast Marching Method algorithm. Example results of inpainting method can be seen in Figure 2.4



Figure 2.4: Example of the inpainting technique. Image from: [3].

2.4 Coordinate systems

To correctly label the data for training, a position of the target drone in relation to the sensor mounted on the observer drone is required. The AirSim API returns the position of each drone in respect to their starting points. The starting point is a point where the drone spawns in the map. Therefore a transformation from the starting point of the target drone to the camera mounted on the observer is required. This transformation is written as follows:

$$\mathbf{T} = \mathbf{T}_o^c \mathbf{T}_{os}^o \mathbf{T}_{ts}^{os}, \quad (2.8)$$

where:

- \mathbf{T}_{ts}^{os} is transformation from the starting point of the target drone to the starting point of the observer drone,
- \mathbf{T}_{os}^o is transformation from the starting point of the to the body of the first drone,
- \mathbf{T}_o^c is transformation from the body of the first drone to the cameras coordinate system.

Transformation matrix \mathbf{T} is generally be described as follows:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}^T & 1 \end{bmatrix}, \quad (2.9)$$

where:

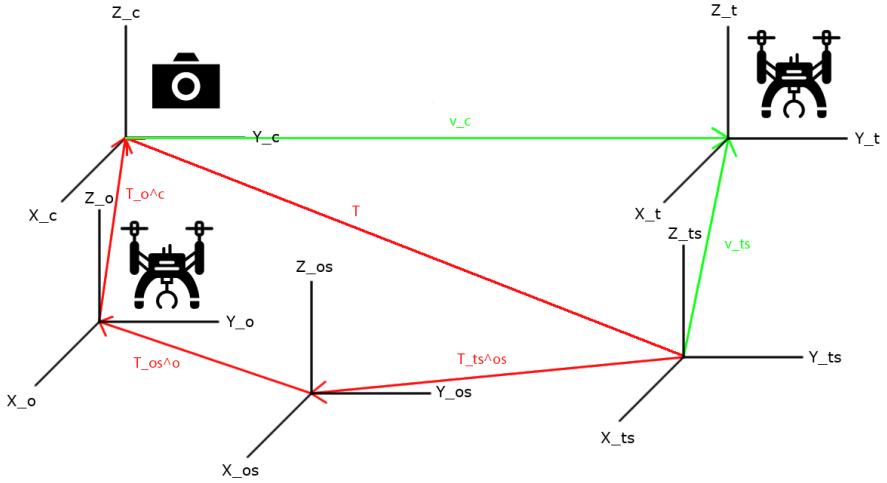


Figure 2.5: Visualization of transformation.

- \mathbf{R} is a 3×3 rotation matrix,
- \mathbf{p} is a 3×1 translation column vector,
- $\mathbf{0}^T$ is a 1×3 row vector of zeros.

To transform a location of the drone represented by vector \mathbf{v}_{ts} into the coordinate system of the camera, as can be seen in Figure 2.5, the following transformation is applied:

$$\mathbf{v}_c = \mathbf{T}\mathbf{v}_{ts} \quad (2.10)$$

2.5 Camera Model

For the creation of the bounding boxes used for training and for processing raw data from LiDAR sensor a transformation from coordinate system of the camera to the pixel values of the image needs to be defined. For this task a pinhole camera model is used as seen in Figure 2.6

The transformation is then defined as follows:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f \frac{X_c}{Z_c} + c_x \\ f \frac{Y_c}{Z_c} + c_y \end{bmatrix}, \quad (2.11)$$

where:

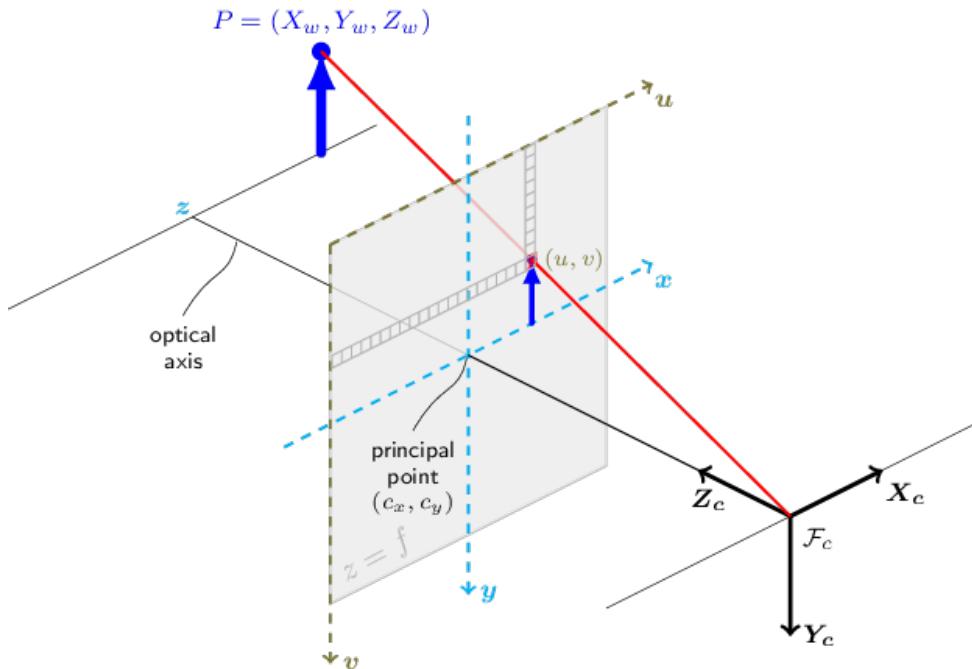


Figure 2.6: Pinhole camera model. Image from [4].

- u and v are pixel coordinate values on the image,
- X_c, Y_c, Z_c are coordinate values of a point in the coordinate system of the camera,
- f is focal length of the camera,
- c_x, c_y are offsets on the image plane.

Pinhole camera model is only idealization of a real life camera and no lens distortion needs to be considered. The AirSim simulator simulates pinhole camera so no other processing needs to be done.

2.6 Dataset

The dataset for this work can be generated in two ways. The first is real-life drone shots mixed with point clouds from LiDAR mounted on top of a drone. The second is generating a dataset using a realistic virtual environment where a drone, camera and LiDAR are being emulated very close to their real-life counterparts. An advantage to this approach is that a great variety of

2. Methodology



Figure 2.7: Unreal Engine user interface.

environments can be chosen a lot of them often inaccessible otherwise (power plant, airport, snowy mountains out of season etc.). Therefore this approach was chosen for the task.

2.6.1 Unreal Engine

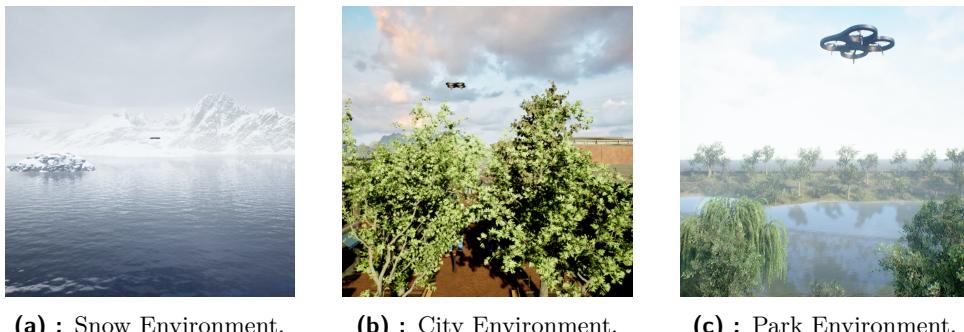
Unreal Engine is a software tool used for creating realistic 3d environments, most often used as a video game engine. It is written in C++ and open-source supporting a variety of pre-built environments and assets. Example of Unreal Engine interface can be seen in Figure 2.7. For this work three different environments were used for the creation of the dataset 2.9:

- city Park Environment Collection (2256 samples taken),
- automotive Winter Scene (1813 samples taken),
- downtown West Modular Pack(1120 samples taken).

Together 5320 pictures and labels were generated using two drones. The observer drone was equipped with an RGB camera and LiDAR sensor and was responsible for taking the pictures and pointclouds from LiDAR. The Parrot AR.Drone 2.0 shown in Figure 2.8 was used as a model for drone detection.



Figure 2.8: Parrot AR.Drone 2.0. Image taken from [5].



(a) : Snow Environment.

(b) : City Environment.

(c) : Park Environment.

Figure 2.9: Sample photos from each environment.

2.6.2 AirSim

An open-source plugin for Unreal Engine called AirSim was used for the generation of the dataset. It simulates realistic flight motions of drones as well as seven types of sensors, including RGB camera and LiDAR used for this task. AirSim supports both a C++ API as well as Python API, the latter of which was used for controlling the motion and capturing the dataset. The location of the second drone was generated through API call, which produces a location of the drone in local coordinates relative to its starting point, which is later transformed to the local coordinates of the first drone carrying the LiDAR and RGB sensors using (2.11). The bounding box required for the Convolutional neural network was generated the same way but the edge points of the 3D bounding box were transformed using (2.11). The bounding box dimensions were set to (0.6, 1.0, 0.3). The LiDAR pointcloud was generated via an API call, which returned 3D pointcloud in relation to the observer drone. Therefore transformation (2.11) was utilized. The capturing drone travelled on each map on a 3D cube grid.

2.7 Training

For training purposes, 5320 samples were taken using the AirSim simulator. Each sample consists of:

- 640x640 RGB image from the camera,
- 640x640 sparse depth image from the LiDAR sensor,
- label file containing ground truth bounding boxes co-ordinates.

This dataset was split into 3662 training, 407 validation, 1251 testing samples.

2.7.1 Sparse to Dense

The data were trained using Sparse to Dense neural network to receive dense depth images. The training was done for 15 epochs using a batch size of 8. The backbone was Resnet18 and the decoder was set to Deconv3 as described in section 2.1. The network was trained on Kitti³ dataset. A processing algorithm was applied to the output depth map, further filtering points that were not in the vicinity of the ground truth depth points. Both filtered and unfiltered depth maps were used for further training and testing to clarify their overall impact and the results can be seen in Figure 2.10 and Figure 2.11.

2.7.2 Image inpainting

An alternative algorithm for making sparse pointcloud into dense ones was applied as well. The function is provided in the OpenCV Python library. The input into the function was a sparse depth image 2.12, same as for the Sparse to dense network except for the RGB part. The algorithm introduced in 2.3 was used with the radius of 1 pixel. The results were processed with the same filtering method as for the results of Sparse to dense method. For further training, only the filtered depth map was used.

³<http://www.cvlibs.net/datasets/kitti/>

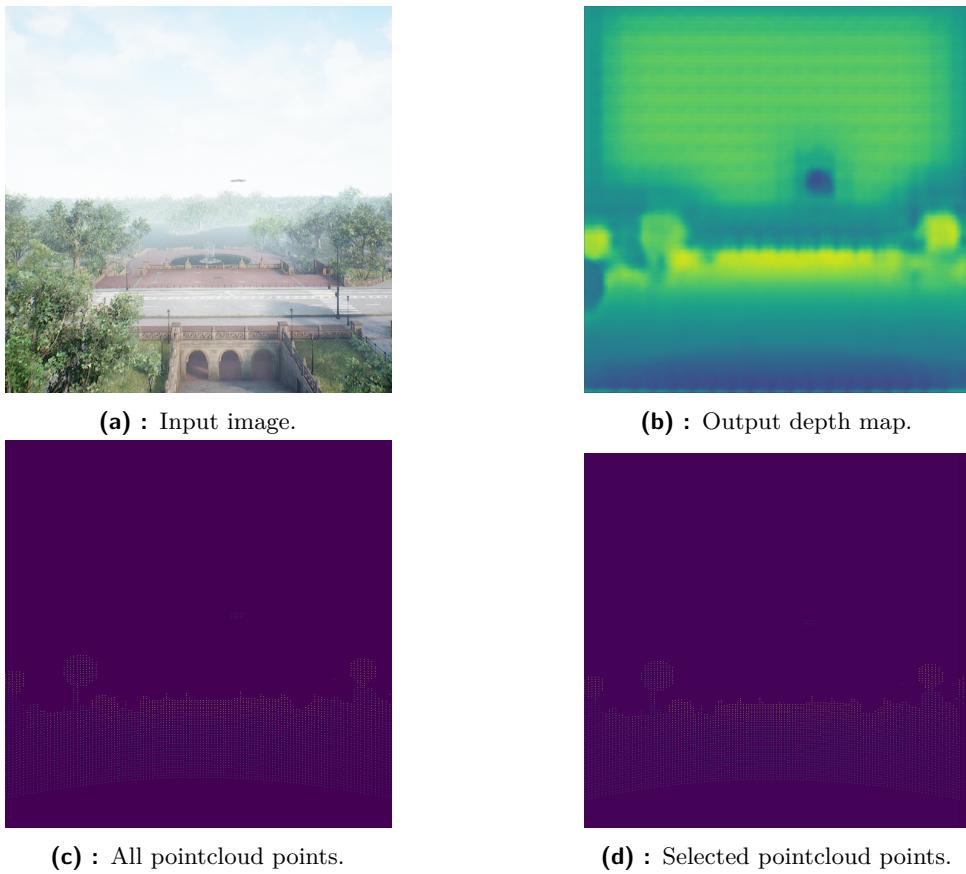


Figure 2.10: Sparse to dense training.

2.7.3 YOLOv3

The following parameters were used for training all inputs:

- number of epochs was set to 120,
- batch size was set to 128,
- size of the input images was $416 \times 416 \times n$, where $n \in \{3, 4\}$,
- learning rate was set to 0.001.

YOLOv3 PyTorch implementation was used for training⁴. Further modifications were required to be made. The original implementation of YOLOv3 supports 3-channel RGB images as inputs. For the sake of this work an

⁴<https://github.com/eriklindernoren/PyTorch-YOLOv3>

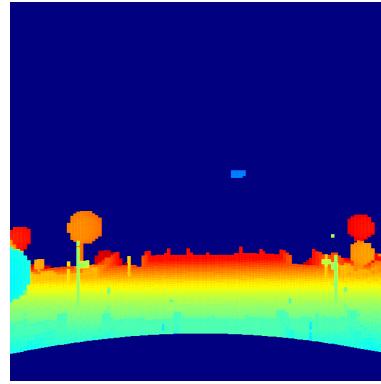


Figure 2.11: Applied filter on Sparse to dense output.



(a) : Non filtered result.

(b) : Filtered result.

Figure 2.12: Inpaint results.

RGBD input option using the H5 file system was implemented. The dataset consisted of drones of various sizes ranging from very small (few pixels) to very large closeups. Therefore 5 detection heads were implemented instead of the original 3. This ensured much higher detection confidence of smaller far away drones as well as bigger more close ones. For the training results 2.14 validation mAP followed the training loss and started converging after around the 8th epoch. It fully stopped rising after the 12th epoch. Considering this the chosen weights ensured that the network does not overfit the training and validating dataset. Table 2.1 shows selected weights for each method.



Figure 2.13: Sample YOLOv3 outputs.

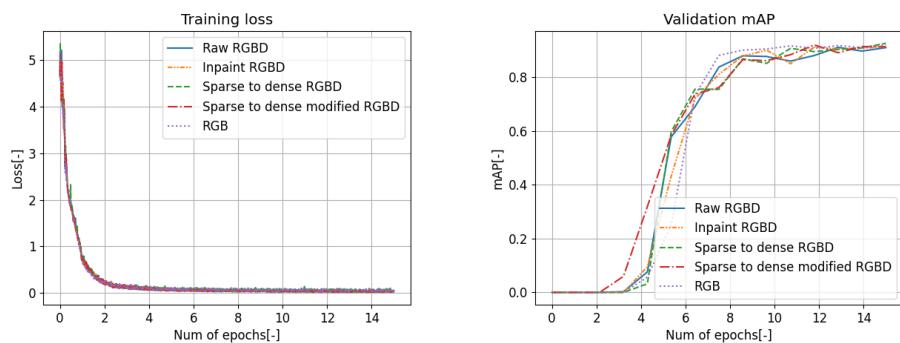


Figure 2.14: Training results

	Raw RGBD	Inpaint RGBD	Sparse to dense RGBD	Sparse to dense modified RGBD	RGB
Weights[epoch]	9	9	11	12	8

Table 2.1: Chosen weights.

Chapter 3

Results

After the training was completed the different metrics on the validation dataset were compared. These metrics include:

- precision,
- recall,
- intersection over Union,
- mean Average Precision (mAP).

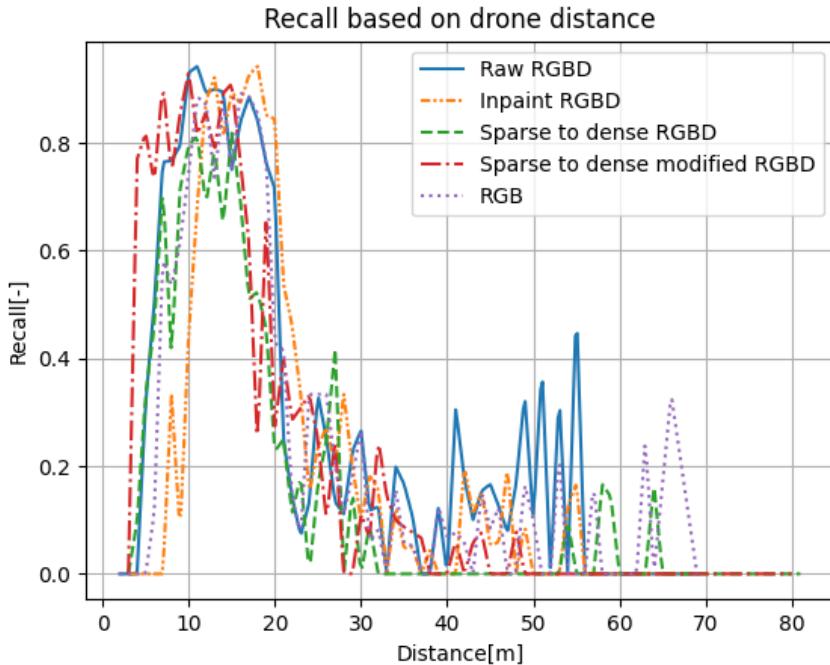
These metrics are given by the following formulas:

$$\begin{aligned} \text{Precision} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}, \\ \text{Recall} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}, \\ \text{IoU} &= \frac{\text{Area of overlap of two bounding boxes}}{\text{Area of union of two bounding boxes}}, \\ mAP &= \frac{1}{N} \sum_{i=1}^N AP_i. \end{aligned} \tag{3.1}$$

where:

- N is number of classes,

Results	Raw RGBD	Inpaint RGBD	Sparse to dense RGBD	Sparse to dense modified RGBD	RGB
mAP	0.48	0.46	0.36	0.43	0.41
Precision	0.59	0.89	0.92	0.60	0.79
Recall	0.53	0.48	0.37	0.48	0.48
IoU	0.79	0.84	0.84	0.83	0.83

Table 3.1: Testing results**Figure 3.1:** Recall based on distance of the drone.

- AP is Average Precision, which is calculated by plotting Precision against Recall and calculating the area under the resulted curve.

In this case, the dataset consisted of only one class, which is a drone. Therefore $mAP = AP$. After the weights were chosen the network was validated on the test dataset. The confidence threshold was chosen to be 0.2 and IoU threshold was set to 0.5. These settings showed best overall results for all the methods. From the results 3.1, Raw RGBD offers the best improvement in terms of mAP by 7%. Every method except unmodified Sparse to dense offers some improvement in terms of mAP. In terms of precision unmodified Sparse to dense and Inpaint offer improvement over RGB by 13% and 10% respectively. Other methods namely Raw RGBD and modified Sparse to dense offer a decrease of 20% and 19% respectively. When it comes to recall not a big improvement is made. The best method is Raw RGBD with an

Speed	Raw RGBD	Inpaint RGBD	Sparse to dense RGBD	Sparse to dense modified RGBD	RGB
time[ms]	61	542.4	66.9	75.9	39

Table 3.2: Inference speeds of all methods.

increase of 5% in comparison to RGB. Unmodified Sparse to dense suffers a decrease of 11% in comparison to RGB, while the remaining methods are unchanged. This can be observed in Figure 3.1 where all methods perform the best in the range from 4 to 19 meters reaching a maximum recall of around 0.95. Modified Sparse to dense offers the highest recall in around 4 meters but starts to fall after 18 meters. Raw RGBD shows a few spikes in the range from around 48 to 55 meters, the highest reaching 0.5 recall. RGB shows a spike at around 67 meters with recall of around 0.7. In terms of IoU, all methods perform very similarly with best-performing methods Inpaint RGBD and unmodified Sparse to dense showing improvement of 1%.

The inference speed 3.2 was measured as a sum of all processing times that needed to be done before the detection was made simulating real-life applications. From the results, the 4th channel depth input almost doubles the inference time of YOLOv3, while Sparse to dense network alone performs quite fast. The longest time for inference belongs to the Inpaint RGBD method with 0.5424 seconds.

Overall Raw RGBD provides an increase in mAP and Recall but suffers a decrease in precision. The inference time is longer, which may prove to be a problem in real-life applications.

Inpaint RGBD produces overall better results in every metric compared to RGB but suffers heavily in long inference times, making it unusable in real-life applications.

Sparse to dense RGBD offers a decrease in mAP and Recall but provides overall the best Precision out of any method tested. Sparse to dense network output time is slowing down the inference speed by only a small amount.

Sparse to dense modified RGBD provides little improvement in terms of mAP and suffers a decrease in Precision. Out of any other method, it provides the highest recall in close detection distances. The modifying algorithm is responsible for a very little slowdown compared to Sparse to dense RGBD.

Chapter 4

Conclusion

In this thesis, four different methods of fusion of RGB data with LiDAR data were proposed and compared along with an RGB only method. Virtual environments were used for the generation of the dataset. The detection Convolutional neural network was trained from scratch and tested generating various metrics that were compared with RGB data results. The advantage of these methods is that they can be easily implemented on drones already carrying these sensors, or on drones already carrying LiDAR sensor as RGB camera is very lightweight and cheap. Since these methods provide relative localization of drones, there is no need to rely on pre-existing sensor infrastructure as in the case of absolute localization methods. These methods also do not rely on markers placed on target drones, so they can be used in a wider variety of scenarios, where f.e. the target is an uncooperative drone.

The dataset used in this method was generated using a variety of Virtual environments as described in section 2.6. The engine used for the creation of the dataset proves to be useful when trying to simulate real-life situations and environments. The main advantage is that environments can be created based on specific drone scenarios, without the need to physically simulate such scenarios. Expanding on the number of different environments and the number of different drone models could be an improvement for the overall generalisation and size of the dataset in future works.

The main detection Convolutional neural network was described in section 2.2. The network was modified from its original implementation to input RGBD data and to detect smaller objects as flying drones that are farther away are only a few pixels in size. A more specific custom-designed architecture could prove to bring an overall improvement on the problem and should be further studied and compared.

4. Conclusion

Overall from the results, it can be seen that no single approach excels over others in all tested metrics. In situations where high precision is required, the best method is the one described in section 2.1. It suffers no significant slowdowns in terms of inference time and provides the best precision out of all tested methods. A modified method from section 2.1 proves to provide a higher recall compared to other methods in terms of close distance detection. Compared to other methods RGB only method provides the most balanced results in all the metrics and the best result in terms of inference speed, making it the best default method for drone detection out of all the tested methods. When it comes to more specific scenarios the fusion of LiDAR and RGB data will excel more.

This thesis aimed to compare different approaches of fusion of LiDAR and RGB data and compare them with RGB only method. This assignment was satisfied and proved that some approaches of LiDAR and RGB fusion can have better results in some specific scenarios. The fusion of LiDAR and RGB has got potential for future works and with larger dataset methods tested in this thesis could overall be better than RGB methods.

Bibliography

- [1] F. Ma and S. Karaman, “Sparse-to-dense: Depth prediction from sparse depth samples and a single image,” 2018.
- [2] A. Kathuria, “What’s new in yolo v3?,” Apr 2018.
- [3] A. C. Telea, “An image inpainting technique based on the fast marching method,” *Journal of Graphics Tools*, vol. 9, pp. 23 – 34, 2004.
- [4] “Camera calibration and 3d reconstruction.”
- [5] P. Martinez, *Vision-based Algorithms for UAV Mimicking Control System*. PhD thesis, 11 2017.
- [6] M. Vrba, D. Heřt, and M. Saska, “Onboard marker-less detection and localization of non-cooperating drones for their safe interception by an autonomous aerial system,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3402–3409, 2019.
- [7] T. Krajník, M. Nitsche, J. Faigl, P. Vanundefinedk, M. Saska, L. Přeučil, T. Duckett, and M. Mejail, “A practical multirobot localization system,” *J. Intell. Robotics Syst.*, vol. 76, p. 539–562, dec 2014.
- [8] A. Tahir, J. Böling, M.-H. Haghbayan, H. T. Toivonen, and J. Plosila, “Swarms of unmanned aerial vehicles — a survey,” *Journal of Industrial Information Integration*, vol. 16, p. 100106, 2019.
- [9] S. Lee, D. Har, and D. Kum, “Drone-assisted disaster management: Finding victims via infrared camera and lidar sensor fusion,” in *2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, pp. 84–89, 2016.

4. Conclusion

- [10] H. Kramer, S. Mücher, and H. van der Hagen, “Hotspot vegetation structure and terrain monitoring of dutch coastal dunes with lidar and optical camera’s mounted on drones,” in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pp. 739–742, 2021.
- [11] D. R. Alves de Almeida, E. Broadbent, A. M. Almeyda Zambrano, M. P. Ferreira, and P. H. Santin Brancalion, “Fusion of lidar and hyperspectral data from drones for ecological questions: The gatoreye atlantic forest restoration case study,” in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pp. 714–715, 2021.
- [12] X. Shi, C. Yang, W. Xie, C. Liang, Z. Shi, and J. Chen, “Anti-drone system with multiple surveillance technologies: Architecture, implementation, and challenges,” *IEEE Communications Magazine*, vol. 56, no. 4, pp. 68–74, 2018.
- [13] I. Bisio, C. Garibotto, F. Lavagetto, A. Sciarrone, and S. Zappatore, “Unauthorized amateur uav detection based on wifi statistical fingerprint analysis,” *IEEE Communications Magazine*, vol. 56, pp. 106–111, 04 2018.
- [14] S. Holter, A. Tsoukalas, N. Evangelou, N. Giakoumidis, and A. Tzes, “Relative visual localization for unmanned aerial systems,” 2020.
- [15] V. Walter, N. Staub, A. Franchi, and M. Saska, “Uvdar system for visual relative localization with application to leader–follower formations of multirotor uavs,” *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2637–2644, 2019.
- [16] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” 2018.
- [17] X. Zhou, D. Wang, and P. Krähenbühl, “Objects as points,” 2019.
- [18] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” 2018.
- [19] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” 2016.
- [20] W. Ali, S. Abdelkarim, M. Zahran, M. Zidan, and A. E. Sallab, “Yolo3d: End-to-end real-time 3d oriented object bounding box detection from lidar point cloud,” 2018.
- [21] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” 2016.
- [22] M. Vrba and M. Saska, “Marker-less micro aerial vehicle detection and localization using convolutional neural networks,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2459–2466, 2020.
- [23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” 2016.