Vision Based Localization of Drones in a GPS Denied Environment

Abhimanyu Chadha

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Ryan K. Williams, Chair
Jia-Bin Huang
A. Lynn Abbott

August 13, 2020
Blacksburg, Virginia

Keywords: Autonomous UAVs, Stereovision, Localization

Vision Based Localization of Drones in a GPS Denied Environment

Abhimanyu Chadha

(ABSTRACT)

In this thesis, we build a robust end-to-end pipeline for the localization of multiple drones in a GPS-denied environment. This pipeline would help us with cooperative formation control, autonomous delivery, search and rescue operations etc. To achieve this we integrate a custom trained YOLO (You Only Look Once) object detection network, for drones, with the ZED2 stereo camera system. With the help of this sensor we obtain a relative vector from the left camera to that drone. After calibrating it from the left camera to that drone's center of mass, we then estimate the location of all the drones in the leader drone's frame of reference. We do this by solving the localization problem with least squares estimation and thus acquire the location of the follower drone's in the leader drone's frame of reference. We present the results with the stereo camera system followed by simulations run in AirSim to verify the precision of our pipeline.

Vision Based Localization of Drones in a GPS Denied Environment

Abhimanyu Chadha

(GENERAL AUDIENCE ABSTRACT)

In the recent years, technologies like Deep Learning and Machine Learning have seen many rapid developments. This has lead to the rise of fields such as autonomous drones and their application in fields such as bridge inspection, search and rescue operations, disaster management relief, agriculture, real estate etc. Since GPS is a highly unreliable sensor, we need an alternate method to be able to localize the drones in various environments in real time. In this thesis, we integrate a robust drone detection neural network with a camera which estimates the location. We then use this data to get the relative location of all the follower drones from the leader drone. We run experiments with the camera and in a simulator to show the accuracy of our results.

Dedication

To my family.

Acknowledgments

To begin with, I would like to thank Dr. Ryan Williams for giving me this opportunity. His esteemed guidance and patience with me and my mistakes is what has lead the way to this thesis becoming a success. His quick-thinking is what made it possible to pivot the experiments as soon as the pandemic stuck. His passion and his jolly nature is an inspiration and I am grateful to call myself his student.

I would also like to thank Dr. Jia-Bin Huang and Dr. Lynn Abbott for being on my advisory committee and providing their valuable insights. Dr. Jia-Bin Huang's courses have paved the foundations of my basics in the field of computer vision and I would like to thank him for the same.

I would like to thank my friends, Deeksha, Dhar, Sose, Karan, Vanssh, Kartikey and Vijay for providing a constant support mechanism and cheering me up whenever needed.

Lastly, but most importantly, I would like to thank my family. They have bent over backwards to enable me to pursue this opportunity. This would not have been possible without their sacrifices and constant support.

Contents

List of Figures

List of Abbreviations

FD Follower Drone

FPS Frames Per Second

LD Leader Drone

PC Point Cloud

SDK Software Development Kit

UAV Unmanned Aerial Vehicle

Chapter 1

Introduction

1.1 Motivation

The evolution of transportation has come a long way. From the invention of a wheel, to animal powered carts to self-driving cars that we see today by companies such as Tesla and Waymo. The last advancement has mainly been due to the rapid rise and success of fields such as Deep Learning and Computer Vision. They in turn have been dependent upon the vast amount of data available and enormous computing power.

Along with cars, the UAVs have also received quite some attention because of heir practicality in daily life. UAVs have proved their usefulness in a variety of areas, especially for military purposes such as search and rescue, monitoring, surveillance, as well as applications such as 3D reconstruction, transportation, and artistic photography. However, most drones or UAVs are not truly autonomous and are operated remotely by a human controller from the ground. Therefore, the next generation UAV requires a self-controlling function to fly in an unstructured environment. Some popular examples of autonomous UAVs being used as delivery agents can be seen by the Wing[1] and Amazon Prime Air[22].

Another popular area for autonomous UAVs, especially a swarm of UAVs, is Search and

Figure 1.1: Wing Drone Delivery in Christiansburg, VA[1]

Rescue operations. In 2017 alone, the United States National Park Service deployed almost 3,500 search and rescue missions in national parks[23]. And while search and rescue personnel tend to be highly effective and skillfully trained, they still face a daunting task since a lost person's chances for survival drop dramatically after the first eighteen hours. In [24] detect for humans in a real-world outdoor environment using thermal and color imagery. The detected human positions are then geolocated on a map which can then be used to narrow down the search area for the rescue teams.

All of these applications, whether they're a swarm of UAVs travelling together for a particular task or separate UAVs minding their own business, require a good detection and estimation method. This is necessary to avoid collisions and achieve formation control in GPS-denied environments. The traditional approach for detection of UAVs has been with April tags[25]. This approach works perfectly in controlled laboratory settings such as uniform light distribution and absence of motion blur but both these ideal conditions are rarely met in the real environment. This problem is can be solved by the state-of-the-art object detection networks based on CNNs. An alumni of CAS Lab, Sudha Ravali Yellapantula along with Dr. Ryan

Williams, did exactly that in [8].

In this thesis, we take her work forward. We implement the detection in real-time on embedded systems which can be placed on UAVs and combine it with a stereo camera system. The stereo camera system, along with the detection of UAVs, also helps us estimate its location. We then exploit this information for localization of that UAV. Such a localization solution can then be used directly for cooperative control, as a supplement to other forms of localization, or given the lead agent's position is known globally, it can be used to globally localize the entire team of agents.

1.2 Thesis Outline

The thesis is organized into six chapters.

Chapter 2, Preliminaries, gives an overview of the concepts used in this thesis.

Chapter 3, Related Work, reports the work done in the fields related to the detection and the location estimation of UAVs.

Chapter 4, Solution Pipeline, elaborates on the techniques used in the proposed approach to the problem statement.

Chapter 5, Experiments, describes the different experiments carried out and shows the results of the same.

Chapter 6, Conclusion, gives the closing remarks and describes the scope for future work.

Chapter 2

Preliminaries

2.1 Neural Networks

The origin of neural networks dates back all the way to 1943 [26] when a a simple neural network was modeled with electrical circuits by neurophysiologist, Warren McCulloh, and a mathematician, Walter Pitts and thus described the idea of how a neurons might work. Jumping to the year 2020, the research of neural networks and its applications to various fields has come a long way. They range from images, text, autonomous driving, medical field, recommendations etc.

A lot of factors have to come to the aid of the basic idea of a neuron ranging from a vast availability of data to tremendously powerful computing resources. A lot of different styles and types of neural networks have also come up such as convolutional networks, RNNs[27], LSTMs[28], GANs[29] etc. The simple idea of neuron still remains the same though and has been adapted in creative ways in the above stated examples.

2.1.1 A Simple Neuron

A simple neuron, which is the building block of the deep learning networks, consists of certain number of inputs and just one output. As it can be seen in the figure each input has a weight

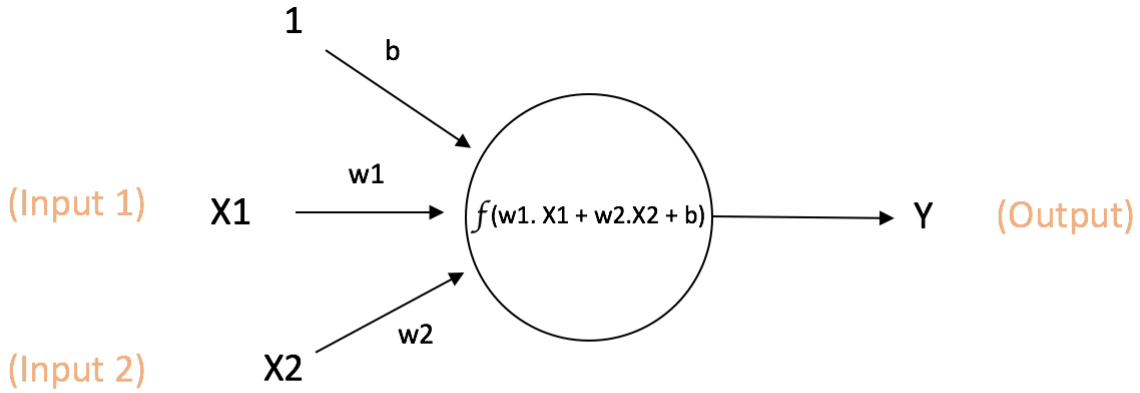

Figure 2.1: A Simple Neuron[2]

associated with it thus deciding the importance of that particular input. They are made to pass through an activation function. There is almost always a bias value present (usually set to one) which is helpful when all the input values to the activation function are zero.

Activation Function

An activation function is needed to introduce non-linearity in a neural network. It is usually applied element-wise, with $h_i = g(x^T W_{:,i} + c_i)$[30] This helps one decide whether to activate a neuron on the basis of certain threshold values of input and not just activate the neuron whenever an input value is present. There are a variety of activation functions[31] such as Sigmoid, ReLU, Leaky ReLU, tanH etc.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

$$\text{ReLU}(x) = \begin{cases} 0, & \text{if } n < 0 \\ x, & \text{otherwise} \end{cases},$$

$$\text{LeakyReLU}(x) = \begin{cases} \alpha x, & \text{if } n < 0 \\ x, & \text{otherwise} \end{cases}.$$

Figure 2.2: Common activation functions[3]

Back Propagation

Back propagation is the commonly known as the way with which a network learns and thus improvises. It draws a very good analogy to real-life where one is expected to practice a certain skill repeatedly and improvise every time.

At the output or the final layer of a neural network we obtain the outputs of that certain network. In a supervised learning problem it is then compared to the expected output. Over here we use loss functions to decide how good or bad the output of the network is or how closely acceptable it is to the expected output. One of the most common loss function is Binary Cross-Entropy function[32]. The approach for back-propagation is pretty straightforward, to minimize this loss function. It is done by taking the gradient of this loss function and equating it to zero. The gradient in the output layer is linked to layer before it and that layer is connected to the layer before it, thus the error or the loss function is propagated through all the layers and the weights are adjusted accordingly. It is with this reference that the network learns how much importance should a particular input be given and which neurons should be activated and when.

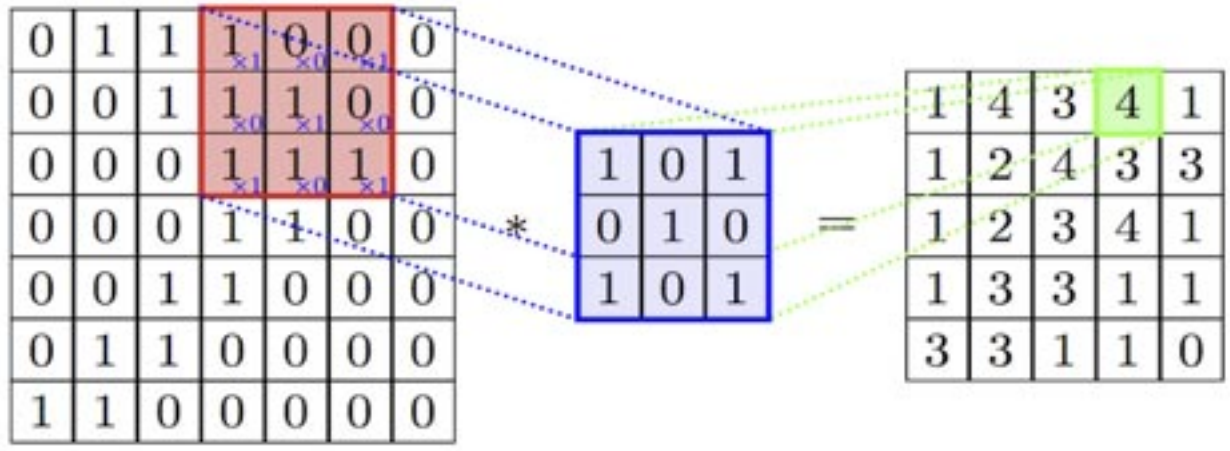

Figure 2.3: A convolutional filter [4]

2.2 Convolutional Neural Networks

The convolutional neural networks revolutionized the way images are perceived by a network. They were first seen in [33] and were used to recognize handwritten digits from the database[34]. Their first most notable application was seen in [35] as an entry for an object recognition competition. The authors utilized the prior knowledge of convolutions and combined it with the recent advances in GPU memory utilization. Thus with a huge data set and considerable computational power at their hands they were able to train and obtain a test error rate as low as 17 percent on the ImageNet data set. These networks use a kernel or filter that is convolved with the input, usually an image. At each step, the dot product of the corresponding cells of the image and the filter are computed and these values become the part of the feature map of that particular filter. The weights of this filter act as the parameters which we aim to learn when the network is trained.

Stride

Stride is defined as the number of pixels by which the kernel or filter moves at every iteration. It helps in deciding the size of the feature maps produced at every convolutional layer. It is thereby used to reduce the granularity of convolution. Occasionally, stride value of 1 and 2 are used. The winning entry[35] in ILSVRC competition of 2012 used a stride of 4 and in the following year a stride of 2 was used by the winning entry[36] to improve the accuracy.

Padding

One disadvantage of strides can come into effect when the filter captures a lot of information from the middle of an image and ends up loosing a lot of information on the edges of the same. This happens because the kernel slides over the middle part of the image more number of times than its edges. Thus to prevent this kind of information loss, a simple but efficient solution of padding[5] is introduced. In this method pixels are introduced towards the edges of the image. If the value of those pixels is zero, it is referred to as zero padding.

Pooling

Pooling[37] is the process that takes place when the filter or kernel slides over a set of pixels. It is used to reduce the size of representation so that the process can be speeded up. There are two kinds of pooling, namely Max-Pooling and Average-Pooling. As their respective names suggest, when a Max-Pooling kernel slides over a set of pixels, simply the maximum value out of those pixels is chosen and the rest are discarded. On the other hand, when an Average-Pooling kernel slides over a set of pixels, the average of all those pixels is taken.

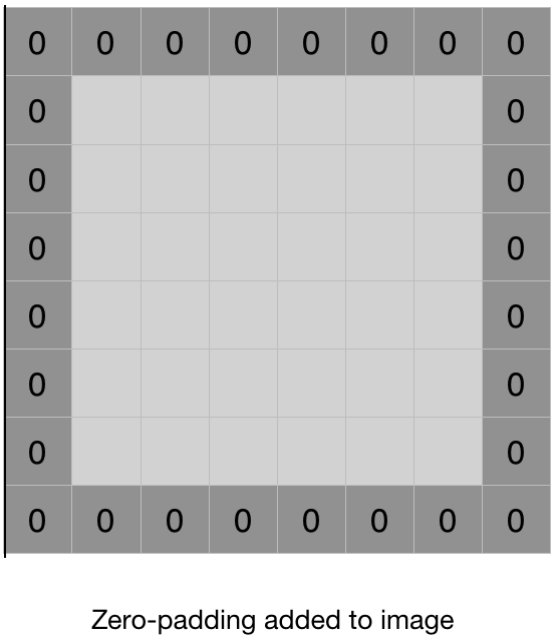

Figure 2.4: An example of Zero Padding[5]

2.3 Stereo Vision

Stereopsis, the perception of depth, was first described by Sir Charles Wheatstone in [38]. Stereo matching is the process of taking two or more images and estimating a 3D model of the scene by finding matching pixels in the images and converting their 2D positions into 3D depths[39]. It's goal is to estimate the depth of an object by super-imposing two or more pictures captured from a different viewpoint in space. The images can be captured via the same camera or multiple cameras. We always use multiple views because structure and depth are inherently ambiguous from single views. The stereo camera systems have been around since the 1970's and were inspired by humans and how they perceive the depth. Since humans use two eyes to perceive the depth, the stereo camera system was made to do the same. Figure 2.5 shows the first stereoscope from the year 1838.

Figure 2.5: Stereoscope[6]

2.3.1 Depth from Disparity

The stereo camera model can be seen in figure 2.6. In this very simple but resourceful example we use the concept of similar triangles to estimate the depth of point P. The general setup of stereo camera system, as can be seen in 2.6, is two camera's with their image planes and optical axes parallel to each other and they're generally of the same focal length as it makes the calculations easier.

The quantities represented by x_l and x_r are the distances of the projection points from the optical axes. x_l is towards the right of its origin so it will be represented as a positive value whereas x_r is on the left of its origin so it'll be represented as a negative value. These are distances so their magnitude is positive and the sign just represents the direction. B represents the baseline that is the distance between the two optical axes. Z is the distance between the point P and the baseline. It is this distance that the stereo camera model

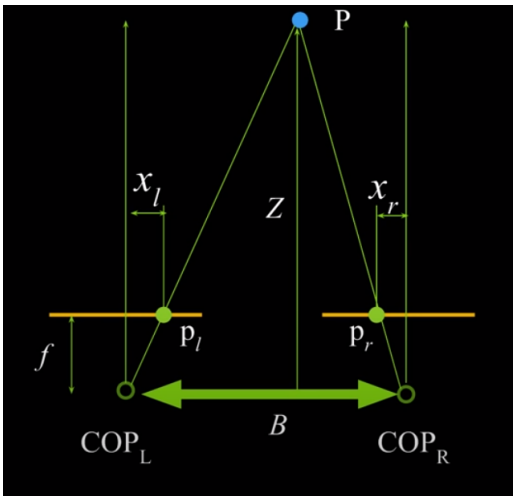

Figure 2.6: Depth from Disparity

estimates.

We can now see two triangles being formed in the figure. The smaller one can be seen in 2.7 and is labelled by vertices (p_l, ***P***, p_r). The larger one can be seen in 2.8 and is labelled by vertices (COP_L, ***P***, COP_R).

On applying the properties of similar triangles we can deduce that:-

$$\frac{B - x_l + x_r}{Z - f} = \frac{B}{Z} \tag{2.1}$$

On rearranging the terms, we get:-

$$Z = f * \frac{B}{x_l - x_r} \tag{2.2}$$

In equation 2.2 the difference between x_l and x_r is called the disparity. The depth of a point

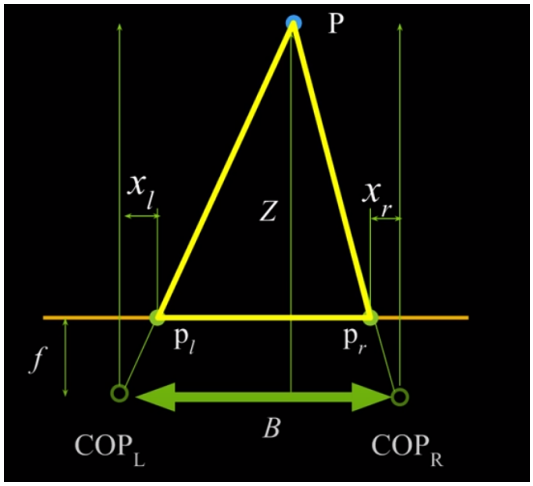

Figure 2.7: Depth from Disparity: Inner Triangle

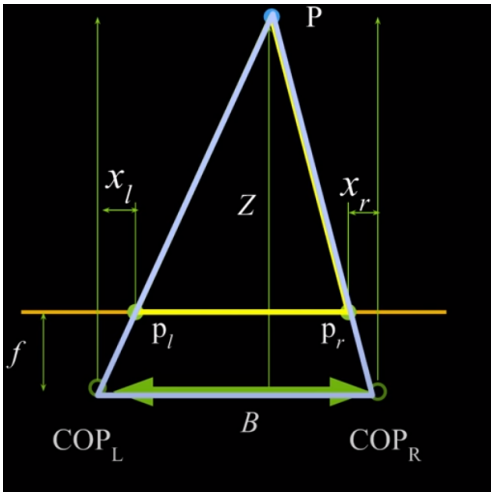

Figure 2.8: Depth from Disparity: Outer Triangle

is always inversely proportional to the disparity. Thus, if there is no disparity, the depth would come out to be infinite.

The disparities are calculated by something called the epipolar constraints which basically reduces the a correspondence problem to a one dimensional estimation problem where we just have to look for the corresponding point along the epipolar line. The epipolar geometry between two views is essentially the geometry of the intersection of the image planes with the pencil of planes having the baseline as axis. This geometry is usually motivated by considering the search for corresponding points in stereo matching [40].

2.4 Least-Squares

Estimators that are based on least squares do not require a probabilistic assumption on the noise signal that corrupts the underlying variable or the estimated state, and are therefore easily implementable and applicable for a broad class of estimation problems [41]. The underlying model involves the observation of a linear function of a variable $\theta \in \mathbb{R}^q$ that is additively corrupted by noise v,

$$z = H\theta + v \tag{2.3}$$

where z,$v \in \mathbb{R}^q$ and $H \in \mathbb{R}^{pxq}(p > q)$; we refer to each component of the vector z as a *measurement channel* and H is the *observation matrix* that is assumed to be of row rank q.In a centralized setting and absence of information about the noise statistics, the least squares estimation proceeds by minimizing the cost function

$$J(\theta) = (z - H\theta)^T(z - H\theta) \tag{2.4}$$

J in 2.4 is a differentiable and convex function of the underlying state θ, its optimal value is found by setting its gradient to zero, and declaring its optimum, that is, the least squares estimate, as

$$\hat{\theta} = (H^T H)^{-1} H^T z \tag{2.5}$$

2.4 is a general form of the least-squares problem and 2.5 is a closed-form solution. For our project, as we shall see in the further chapters, we solve it as an optimization problem to obtain the best explanation for the location of each of the follower drones.

Chapter 3

Related Work

The use of vision for for formation control is a two-part problem, namely detection followed by location estimation. The first part is identifying a drone in an image or a video. This can be done with the help of convolutional networks trained specifically on a dataset of images of drones.

In [7] the authors prepare a synthetic labelled dataset of drones and use the residual information between consecutive image frames to form an integrated detection and tracking pipeline. To target the problem of a sparse dataset, they collect a dataset of their own and then use data augmentation techniques such as geometric transformations, illumination variation and image quality to make the dataset more diverse. The authors then use [42], which replaces the usage of external object proposals with region proposal networks, for drone detection. The FasterRCNN achieves nearly cost-free region proposals and it can be trained end-to-end by back-propagation. For the tracking part, the authors utilize [43] which is able to separate the domain independent information from the domain specific information in network training. They thus form an integrated pipeline of drone detection and tracking.

The authors of [44] re-train YOLOv2[45] on a combination of the publicly available USC Drone-Dataset[7] with a manually labelled dataset of their own. The USC Drone-Dataset is which consists of thirty YouTube video sequences and captures a variety of different drone

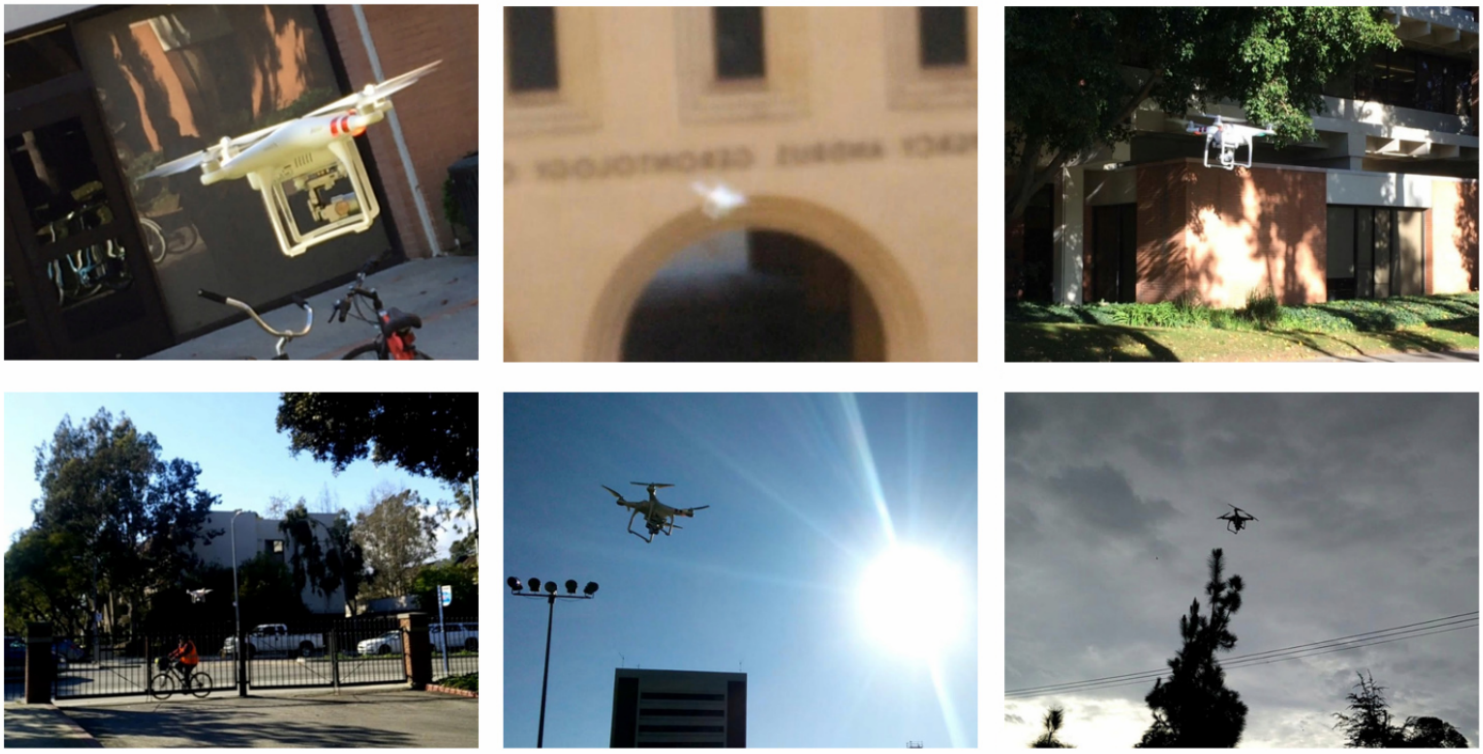

Figure 3.1: Examples from the USC Drone-Dataset [7]

models. The videos have the frame resolution of 1280 x 720 and include different environments, both outdoor and indoor, such as grassland, courtyard, warehouse etc. They play around with the input resolution of the network to figure out a trade-off between high-resolution detection and the time taken per each detection. They use k-means++[46] algorithm to adjust the anchor box's pre-defined dimensions. This helps in making the network more prone to detect drones.

In [47] the authors re-train Zeiler and Fergues [36], Visual Geometry Group (VGG16)[48] and VGG16 with Faster R-CNN[48] to detect drones. Zeiler and Fergues is an eight layered architecture with five convolutional layers and three fully-connected layers. On the other hand, VGG16 is a sixteen layered architecture with thirteen convolutional layers and three fully connected layers. They carried out experiments on the Bird-vs-Drone dataset which contains five mp4 videos of resolution 1920 x 1080. The objective is to teach a network to identify the difference between a bird and drone.

In all of the previous works the authors stated the absence of a good dataset as a shortcoming to making a robust drone-detection pipeline. In [8] this problem is tackled head-on. The author captures a sparse dataset and then uses Cycle-GANs[9] to generate a dataset of drone images. Cycle-GAN uses image-to-image translation for data generation, as seen in Figure 3.4. It builds on Pix2Pix[49]. This network just requires two unpaired collection of images and then finds a mapping between them. It introduces Cyclic-Consistency loss, as seen in Figure 3.3 which makes it different from all the other GANs out there. This is followed by re-training YOLOv3[50] to form a robust drone detection network. We utilize this network to a great extent in this thesis too. Some example of this work can be seen in Figure 3.2

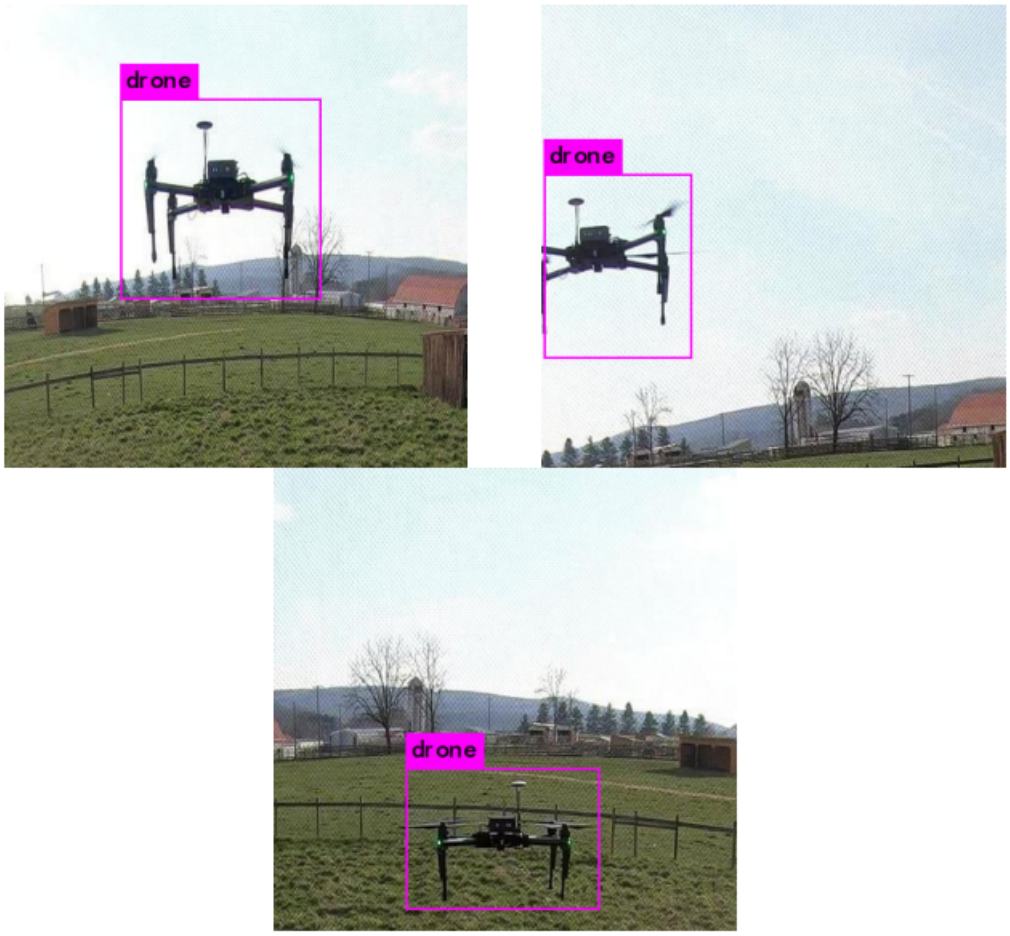

Figure 3.2: Drone Detection results[8]

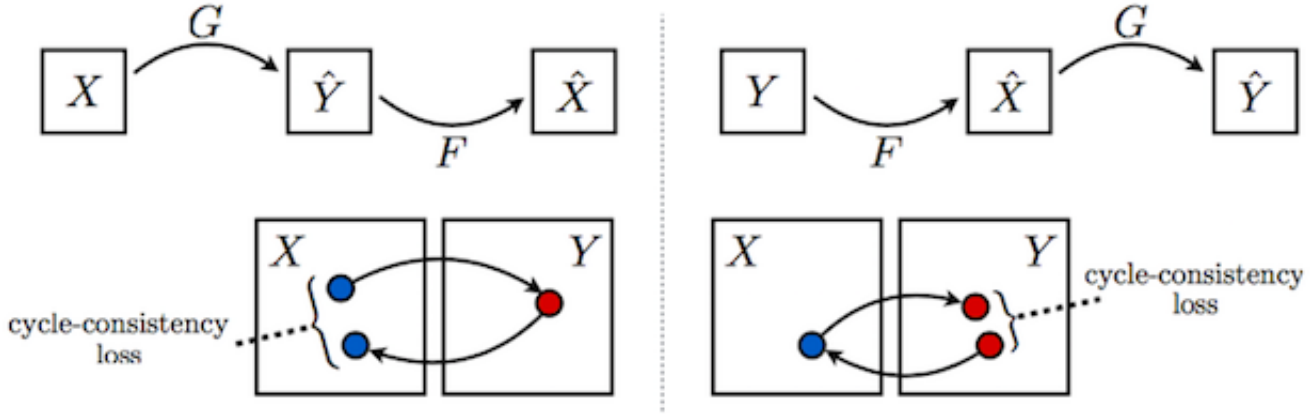

Figure 3.3: Cyclic-Consistency Loss [9]

Figure 3.4: Example usage of Cycle-GAN [9]

The second part is the location estimation of the drone and that problem hasn't popularly been tackled with just vision-based techniques. They've usually been combined with other sensors such as Ultra-Wideband or IMU sensors. In [10] the authors use the concept of feature matching to localize the drones in the same reference frame. Kept in the same planar field, they conduct two experiments as seen in Figure 3.5. The first one with a forward-facing camera and the second one with a downward-facing camera on on each of the drones. The relative position measurements are computed using the images captured from the on-boarding camera and they're distributed between the neighbouring drones. The authors extract SURF[51] features from these images, which are more robust when compared to the famous SIFT[52] features. These SURF features are then fed into the QuEst[53] Algorithm which estimates the relative positions of the drones with respect to each other. The authors don't calculate the exact positions since they only aim to achieve planar formations and thus don't face the possibility of collisions. This approach works perfectly well in a feature rich environment but has the possibility to fail when the images captured by the drones can't find common features. One example would be when the drones are flying over a water body, where conducting accurate feature matching can prove to be a daunting task. Another hard requirement for this to work is that there should always be some overlap between the pictures captured by the drones thus limiting the inter-distance between the drones. In [11] estimate

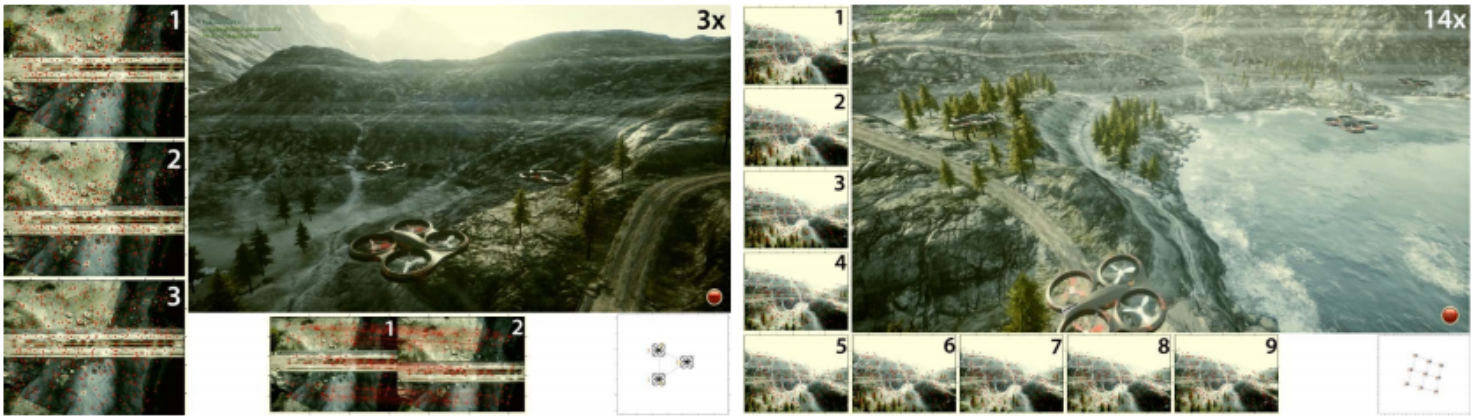

Figure 3.5: Snapshots of simulations from [10]

the relative poses of the drones with a sensor fusion algorithm which captures the features from a poster of part of the Boston University map, as seen in Figure 3.6, attached it to the floor of the their laboratory.

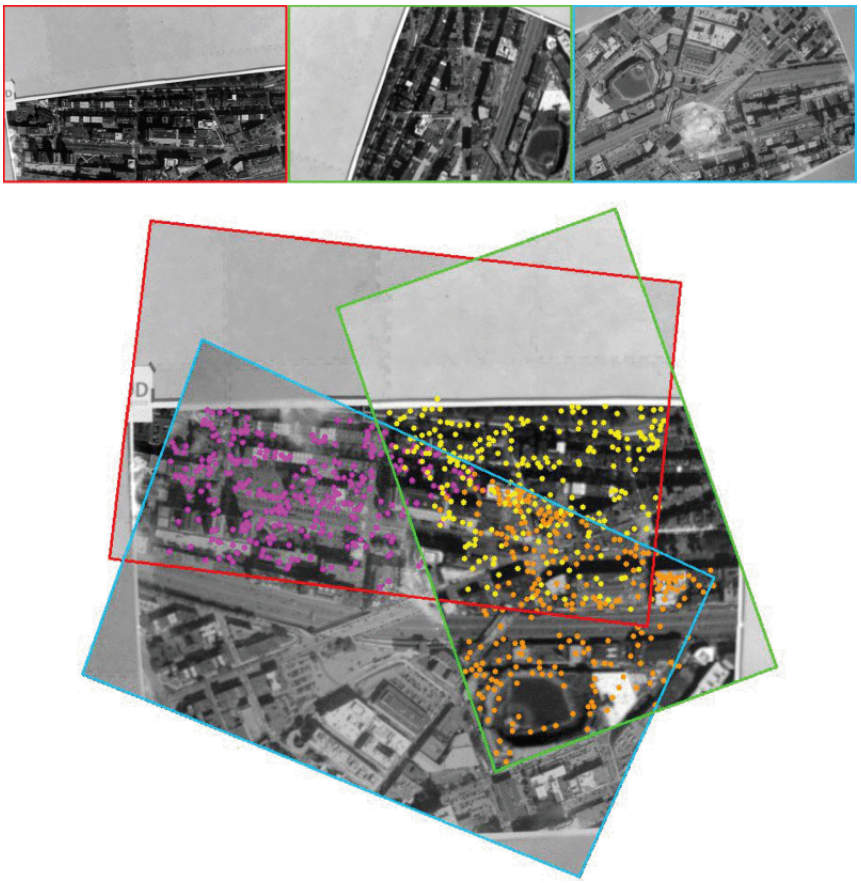

Figure 3.6: Feature detection and matching of Boston University Map[11]

Chapter 4

Solution Pipeline

4.1 YOLO: You Only Look Once

YOLO [12] is one of the best real-time object detection networks out there. The first version of YOLO was released in 2015. Since then, considerable improvements have been made to the algorithm and multiple versions have been released, the latest one being YOLOv4 [54].

4.1.1 Approach

Unlike previous object detectors, such as R-CNN[27], Fast R-CNN[55] which used to re-purpose classifiers, YOLO decided to adapt a single convolutional neural network which could do all the work. The authors re-framed the problem as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities.

The authors present a unified detection approach where they look at the whole picture in a single-go. This is different from its predecessors which used region based approaches or sliding window techniques. This approach helps save the time but also affects the accuracy to some extent.

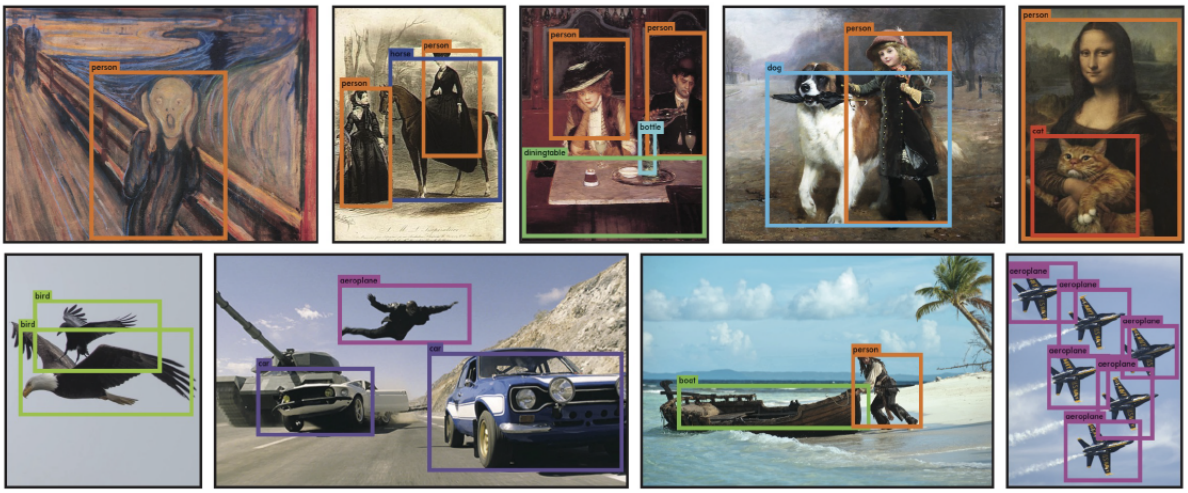

Figure 4.1: Some examples of detection by YOLO[12]

4.1.2 Algorithm

The YOLO network divides a picture into a grid of size S x S and if the center of the object falls in that grid cell then that grid is responsible for detecting that object. What they mean by this is that an object might span up to multiple grids but they can't have each one of them detecting it, thus this makes things easier and faster. The rest of the grids help in the formation of a color map, which ultimately helps in defining the bounding box around the object. Here each grid goes onto predict B bounding boxes along with a confidence score. The confidence score helps the model eliminate the boxes which have no object in them. The confidence score is defined as the product of the probability of an object and the IOU(Intersection Over Union). Each bounding box gives out five measurements namely x,y,w,h and confidence. The (x,y) give the center of the object while (w,h) give the width and height of the box.

4.1.3 Datasets

The orignal version of YOLO was trained on the ImageNet 1000-class competition dataset[56] and was evaluated on the PASCAL VOC Dataset[57]. The ImageNet training dataset contains over a million images and is one of the most standard datasets used to train networks in the computer vision community. These networks are usually fine-tuned with the help of other datasets. The PASCAL VOC dataset has many versions but the authors used the versions from 2007 and 2012 for inference of their YOLO Network.

In [8] the author trained YOLOv3 on a custom dataset which contained both real and synthetic images of drones. In further chapters we shall see how We utilized this object detection network for drones in our project.

4.1.4 Network Design and Training

The authors based the architecture (Figure 4.2) on GoogLeNet which was used for image-classification. YOLO is made up from twenty-four convolutional layers along with two fully-connected layers. They use 1x1 reduction layers followed by 3x3 convolutional layers. All the layers in the network use the leaky rectified linear activation except the last one which uses a linear activation.

The authors also released another version of YOLO, known as Fast YOLO and it had nine convolutional layers instead of twenty four. This version, as the name suggests, was faster than YOLO but compromised on the accuracy.

At first the authors don't train the whole network but rather just the first twenty con-

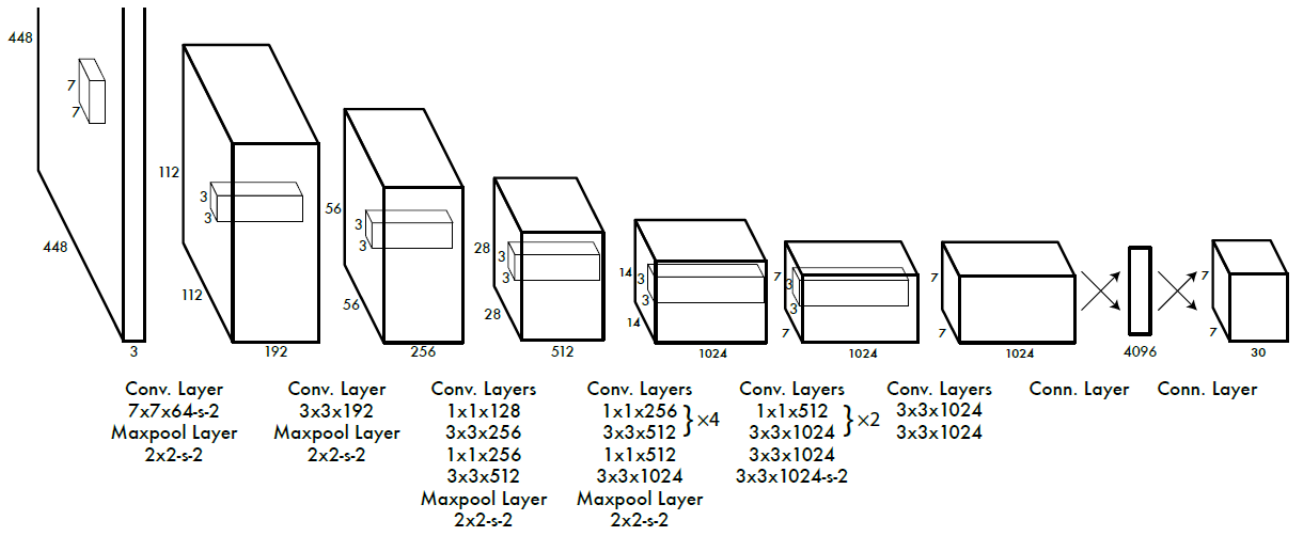

Figure 4.2: YOLO's Architecture[12]

volutional layers followed by an average-pooling layer and a fully connected layer. They then add four more convolutional layers and two fully connected layers.

4.1.5 Loss Function

The authors use sum-squared error because it is easy to optimize but they make a few changes to it. That is because gradients from the zero confident score from the grids which don't contain any object might overpower the grids which actually do, leading to model instability. Thus, they increase the loss from the bounding box coordinate predictions and decrease the loss from the confidence predictions for boxes that don't contain any object. They use λ_{coord} = 5 and λ_{noobj} = 0.5, to achieve this.

The overall loss function which is used in training comes out to be:-

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{\text{ij}}^{\text{obj}}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2]$$
$$+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{\text{ij}}^{\text{obj}}[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2]$$
$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{\text{ij}}^{\text{obj}}(C_i-\hat{C}_i)^2$$
$$+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{\text{ij}}^{\text{obj}}(C_i-\hat{C}_i)^2$$
$$+\sum_{i=0}^{S^2}1_{\text{ij}}^{\text{obj}}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2$$

The loss function only penalizes classification error if an object is present in that grid cell and only penalizes the bounding box coordinate error if that predictor is responsible for the ground truth box.

The authors used a batch size of 64, a momentum of 0.9 and a decay of 0.0005. They used a variable learning rate which helped their model from avoiding divergence. To prevent overfitting they used dropout and data augmentation.

4.2 ZED2: Stereo Camera System

The ZED2 stereo camera system is the latest product by StereoLABS and is one of the best stereo camera systems in the market. We chose this camera for our experiments because its lightweight(120 grams) and seamlessly integrates with both Jetson and YOLO. It has a 120 degree wide-angle field of view with built in IMU, Barometer and Magnetometer sensors.

Figure 4.3: ZED 2 Stereo Camera System[13]

It comes with an SDK[58] compatible with Windows, Linux and Jetson systems making it quite easy to access. It also has compatible APIs with YOLO, Tensorflow, PyTorch, ROS and OpenCV. For our project we couple the camera with our customized YOLO as we shall see in further chapters.

4.3 DJI Drones

DJI is one of the leading manufacturer of drones in the world. They offer a wide variety of commercial and industrial drones. Their products are used in a diverse number of fields ranging from agriculture monitoring[59], bridge inspection[60], search and rescue operations[61] to even wedding photography. For our experiments we use the DJI Matrice 100[14] and the DJI Matrice 600[15].

4.4 Embedded systems for AI at the Edge

NVIDIA Jetson is the world's leading embedded AI computing platform. Its high-performance, low-power computing for deep learning and computer vision makes it possible to build

Figure 4.4: DJI Matrice 100[14]

Figure 4.5: DJI Matrice 600[15]

software-defined autonomous machines. To run the trained YOLO networks mentioned above in real-time on our drones we chose two embedded systems namely the Jetson Nano TX2i and the Jetson Xavier. Both these systems came decent computational power along with compatibility with the ZED2 cameras thus making them the perfect choice for our experiments.

4.4.1 JetPack SDK

The JetPack[62] SDK's are comprehensive solutions provided by NVIDIA to interface with the Jetson Platforms. They're based on Linux operating system include the L4T packages along with CUDA-X accelerated libraries. These are the perfect combination for deep learning, computer vision and accelerated computing applications on the edge.

4.4.2 Jetson TX2i

The Jetson TX2i's come fitted with two hundred and fifty six NVIDIA CUDA cores, a Dual-core Denver two 64-bit CPU and quad-core ARM A57 Complex. It can operate in the temperature range of -40 degree celsius to 85 degree celsius. Combined along with the Orbitty carrier this forms a lightweight powerful embedded system which can be easily mounted on small drones.

4.4.3 Jetson AGX Xavier

The Jetson AGX Xavier's come fitted with 512-core Volta GPU with Tensor Cores, an 8-core ARM v8.2 64-bit CPU. With dimensions of 105 mm x 105 mm x 65 mm, this embedded system is a little heavier than the TX2i's and thus can only be supported by bigger drones

Figure 4.6: Jetson TX2i with Orbitty[16]

but at the same time exceed the TX2i's by leaps in terms of computational power.

4.5 AirSim

AirSim[18] is an open-source simulator platform launched by Microsoft in 2017. It is built on Unreal-Engine and offers a large variety of conditions and environments for the process of data collection, an essential part to the recent advances in machine intelligence and deep learning. It has two default vehicle modes namely a car and a quadrotor mode. It follows the software-in-the-loop (SITL) principle. The simulator includes PX4 SITL flight controller, which allows the simulation code to be directly imported to the commercially available quadrotor platforms.

This extensive simulator gives the user to modify a number of variables and provides a few by default to make it easier for a researcher to focus on their experiments instead spending much time on the development of the their environments. They provide robust vehicle

Figure 4.7: Jetson AGX Xavier Developer Kit[17]

Figure 4.8: A snapshot from AirSim[18]

models, accurate physical phenomenon in the environment such as gravity, air-density, air pressure and magnetic field.

The original version was launched with the integration of basic sensors such as GPS, IMU, Barometer and Cameras. The recent releases of the simulator have added the functionality of other sensors. We focus our attention on Lidar sensors which end up playing a crucial role in our simulation experiments.

4.6 Lidar

Lidar stands for Light Detection and Ranging and it is a remote sensing method[63]. It uses light in the form of a pulsed laser to measure distances in three dimension. A lidar emits these lasers into its surrounding environment. These rays travel and bounce off the surrounding objects thus finding their way back to the sensor. Once they come back, the lidar calculates the distance on the basis of time taken by the pulse to come back. On repeating this process millions of time, a lidar is able to create a precise 3-D map of its environment and the objects in it. The model for Lidar was readily available in AirSim and gave us a point cloud similar to what we obtain from the ZED2 stereo camera system.

Figure 4.9: Velodyne Alpha Prime Lidar[19]

Chapter 5

Experiments and Results

Our experiments, for this thesis, were divided into two parts. The first part was running the YOLO network, trained by Sudha, in real time on an embedded device and combining it with the ZED Stereo camera system. We measure a relative vector between an agent and one of its neighbors represented in a local body-fixed frame rigidly attached to the agent. Our objective is to use these measurements to estimate the position of all agents in a single fixed frame attached to a leader agent.

The second part is transforming the coordinates obtained by every ZED Camera to a single frame, in this case the leader drone's frame of reference. The leader drone is chosen beforehand. Since the same drone would be detected by multiple neighbors and not all the locations would be exactly the same, we would thus apply the method of least squares approximation to estimate the location of every drone in the leader drone's frame.

In the beginning we had decided for putting this network on multiple drones and going out in the field and conducting the experiments but due to the COVID-19 pandemic we had to pivot some part of our experiment. We implement the first part, as planned, in real time on an embedded system with decent accuracy and frame rate. For the second part, since we could not go out in the field, we shifted to the simulator AirSim. We had to change our approach to suit the features available in this open-source simulator but we draw parallels to

the real world experiments at every point. This would enable the next person to seamlessly transfer the experiment onto the real drones whenever it is safe to do so.

5.1 Part 1: Detection and Estimation

For this part, we combined the YOLO network trained by Sudha, mentioned in Chapter 3 with the ZED Stereo camera system. The YOLO network was used for detection of the drones and the ZED Camera was used for the estimation of the relative vector.

We chose the Jetson TX2i and Jetson Xavier as the embedded platforms to run our network in real-time. They were to be put upon the the DJI Matrice 100 600, respectively. The Linux based Jetpack[62] SDK was installed on both of them. For the TX2i the Orbitty Carrier version was used while for the Xavier the default package provided by NVIDIA was used. The reason we chose this operating system was because of its compatibility with the ZED Camera's SDK.

The two platforms are state-of-the-art embedded systems for running deep learning networks in real-time. Coupled with the best object detection network, YOLO, the speed and the accuracy are amazing. But in our case, along with the detection of the drone we also wanted the location of the same. Thus the CUDA cores provided by the embedded platforms had to be divided among the YOLO network and the ZED2 stereo camera system. This led to a bottleneck where due to limited computational power the FPS rate of the detection network was going low and thus in turn was affecting the accuracy of both detection and the estimation. To increase the FPS we reduced the quality of the detection from 2K to 720p on the Xavier and VGA on the TX2i. This is one trade-off we had to suffer to get a

Figure 5.1: Results on TX2i with a VGA picture quality

decent FPS of 5 and 10 on the TX2i and the Xavier, respectively. With this adjustment, we got the relative vectors for the drone with respect to the left camera of the ZED2 stereo camera system. To draw a comparison we also ran the pipeline on a PC. The results for the TX2i, Xavier and the PC can be seen in figure 5.1, figure 5.2, figure 5.3 respectively. The coordinate frame's orientation can be seen in figure 5.4

5.2 Part 2: Localization

We implemented this part in AirSim and ran multiple simulations in different environments to measure the performance of our method. To begin with, we launch five drones in a straight line. Launching them in a straight line is very important since this simulator gives each individual drone its own origin at the launch position. Thus, when we have to transform the

Figure 5.2: Results on Xavier with a 720p picture quality

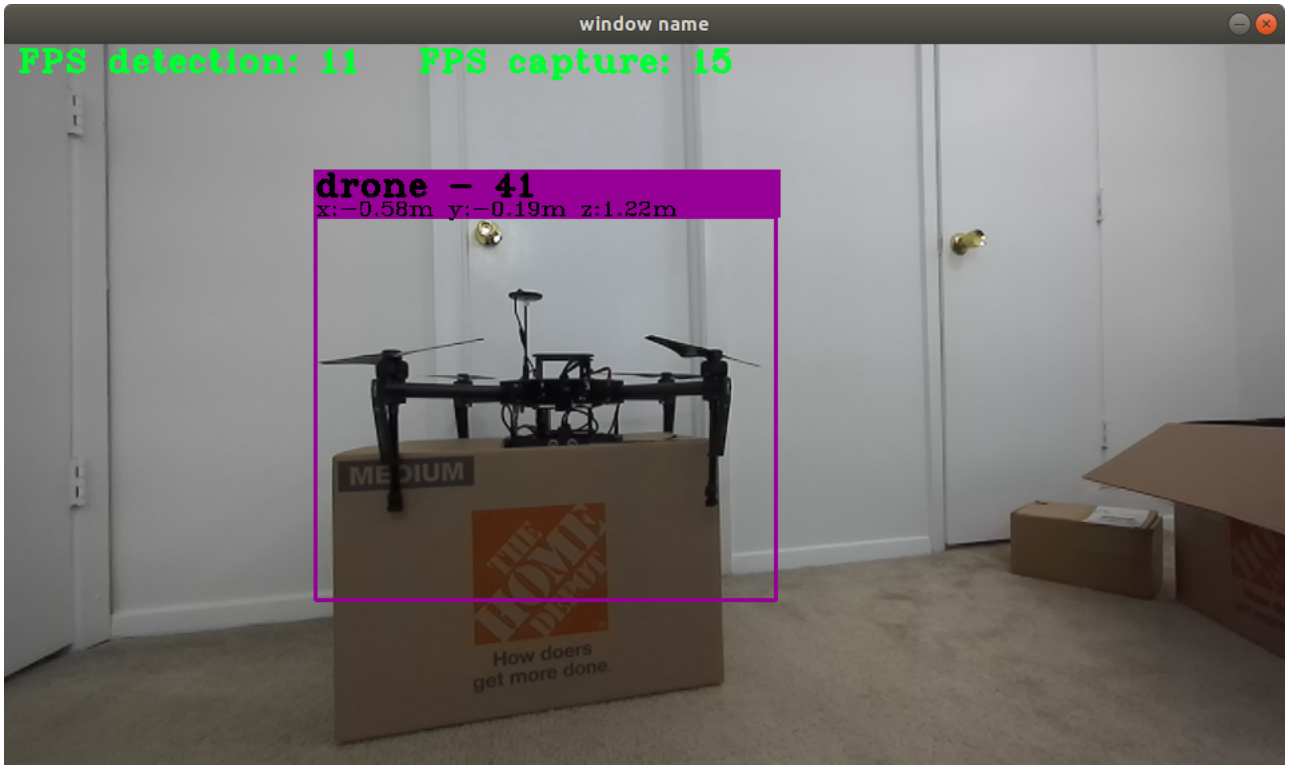

Figure 5.3: Results on PC with a 2K picture quality

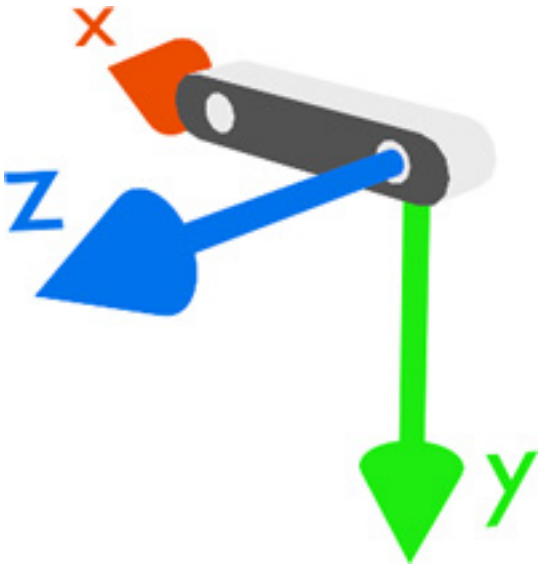

Figure 5.4: ZED Camera's Coordinate Frame of Reference[20]

current location to a single drone's frame, this kind of a starting position makes it easier. With Python API's provided along with the simulator, we give the individual drones to go to particular coordinates and form a formation.

Our first function is to transform the positions of the all the follower drones into the leader drones' frame. We use these as our ground truth. In the real world, we shall be using UWB's for estimating the ground truth. The function goes like:-

Algorithm 1 Transform States (`LD_worldFrame, FD_worldFrame`)

```
FD_leaderFrame.x_val = FD_worldFrame.x_val - LD_worldFrame.x_val
FD_leaderFrame.y_val = FD_worldFrame.y_val - LD_worldFrame.y_val
FD_leaderFrame.z_val = FD_worldFrame.z_val - LD_worldFrame.z_val
return FD_leaderFrame
```

The Algorithm Transform States is better visualized in figure 5.5.

Once we have the location in the leader drones' frame, we call the API for the LIDAR on each drone and thus obtain the point cloud. In the real world, we shall obtain the coordinates in the sensor frame from the stereo camera system itself. Every point cloud obtained is, naturally, in the sensor's frame and needs to be transformed to the leader drone's frame. The Transform Point Cloud function goes like:-

Algorithm 2 Transform PC (PC_sensorFrame, LD_worldFrame, FD_worldFrame)

```
for Each point in the PC do
    PC.z_val += 0.2                    ▷ To transform from sensor to drone's frame

    PC.x_val += (FD_worldFrame.x_val - LD_worldFrame.x_val)
    PC.y_val += (FD_worldFrame.y_val - LD_worldFrame.y_val)
    PC.z_val += (FD_worldFrame.z_val - LD_worldFrame.z_val)
return Transformed PC
```

Figure 5.5: Algorithm 1: Transform States

The Algorithm Transform PC is better visualized in figure 5.6 and figure 5.7.

Line 2 is to transform the point cloud from the sensor's frame to their respective drones frame to which it is rigidly attached.

Line 4 to 6 is where the point cloud is transformed from the follower drone's frame of reference to the leader drone's frame of reference.

Once we have all the point clouds, we need to filter out the clutter and retain only the points which have a greater probability of being the coordinates of the follower drones. This would be the equivalent of drawing a bounding box in the real experiments where we would need to discard random object surroundings and focus just on the pixels that have the drone. For the simulator, since we have the ground truth, we shall discard the points that are not in the 0.5m range of these locations.

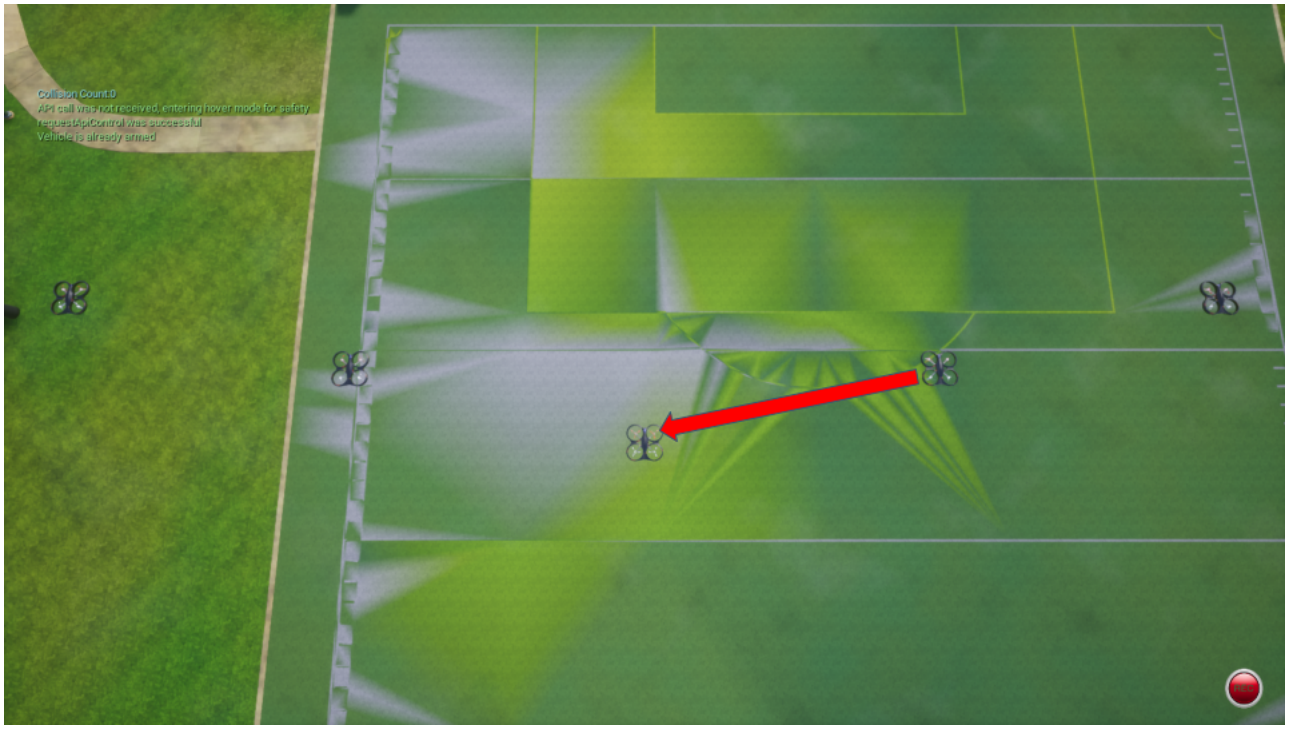

Figure 5.6: Algorithm 2: Transform PC (for FD_2)

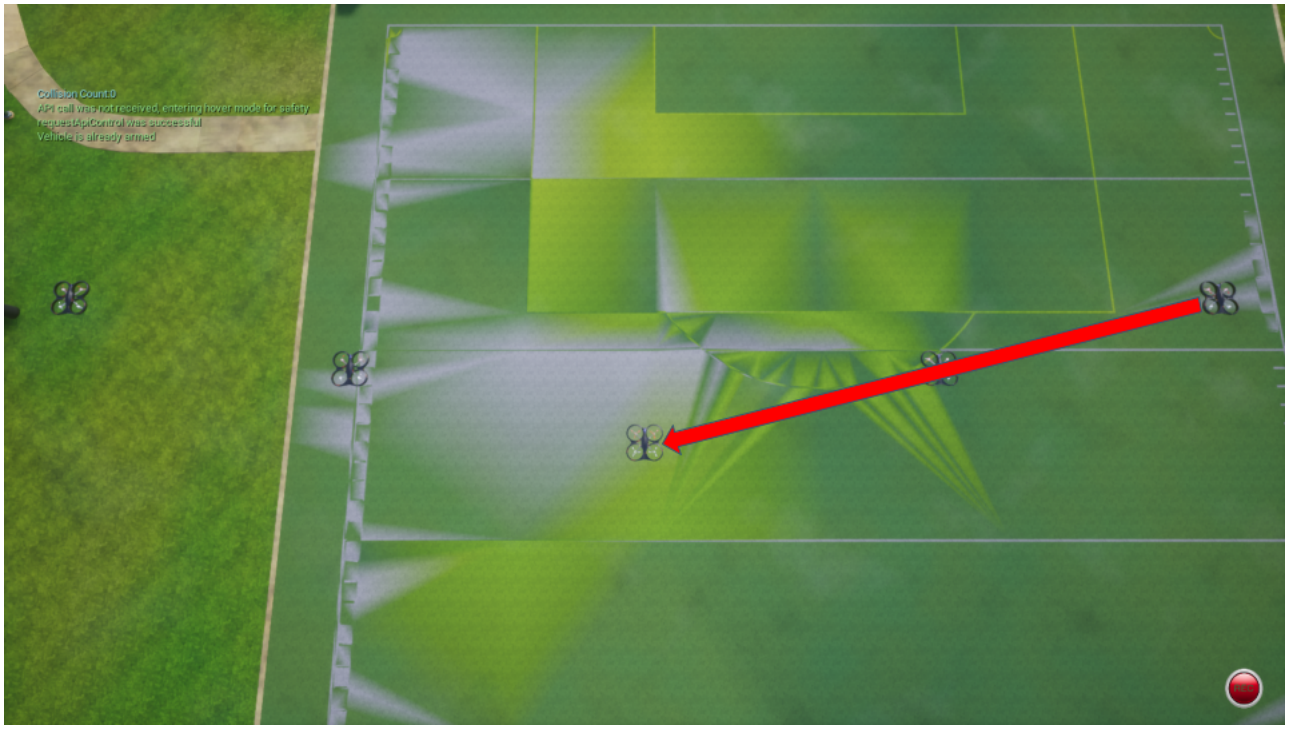

Figure 5.7: Algorithm 2: Transform PC (for FD_3)

Figure 5.8: Algorithm 3: Filter PC (before filtering)

The Filter Point Cloud function goes like:-

Algorithm 3 Filter PC (Transformed PC, N, droneStates, estimatedStates)

```
for Each point in the PC do
    for Each of the FD do
        if Point is in range of location FD ± 0.3 then
            Save the point as a possible location for that FD in estimatedStates
        else
            Discard the point
return estimatedStates
```

The Algorithm Filter PC is better visualized in figure 5.8 and figure 5.9.

Once we have narrowed down the location of each of the follower drones to a few coordinates, which represent the estimates for the same drone via different paths, we use the Least Square Estimate Optimization to get the best fit for the location of the follower drones. The Least Squares Estimate function goes like:- The Algorithm Linear Least Squares is better visualized

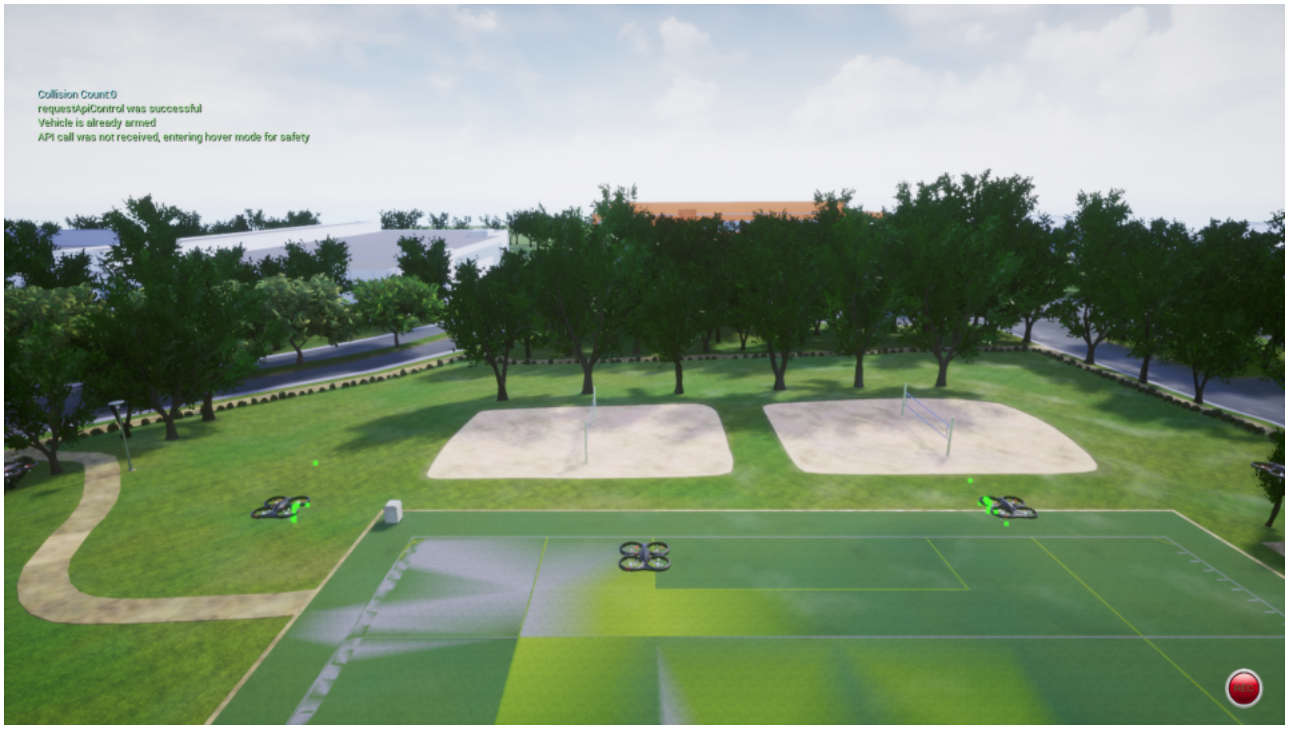

Figure 5.9: Algorithm 3: Filter PC (after filtering)

Algorithm 4 Linear Least Squares Estimate for Localization

1: **for** Each path to FD **do**
2: $\hat{x}_{FD} = ArgMin(\sum x - x_{FD})$
3: $\hat{y}_{FD} = ArgMin(\sum y - y_{FD})$
4: $\hat{z}_{FD} = ArgMin(\sum z - z_{FD})$
5: return $\hat{x}_{FD}, \hat{y}_{FD}, \hat{z}_{FD}$

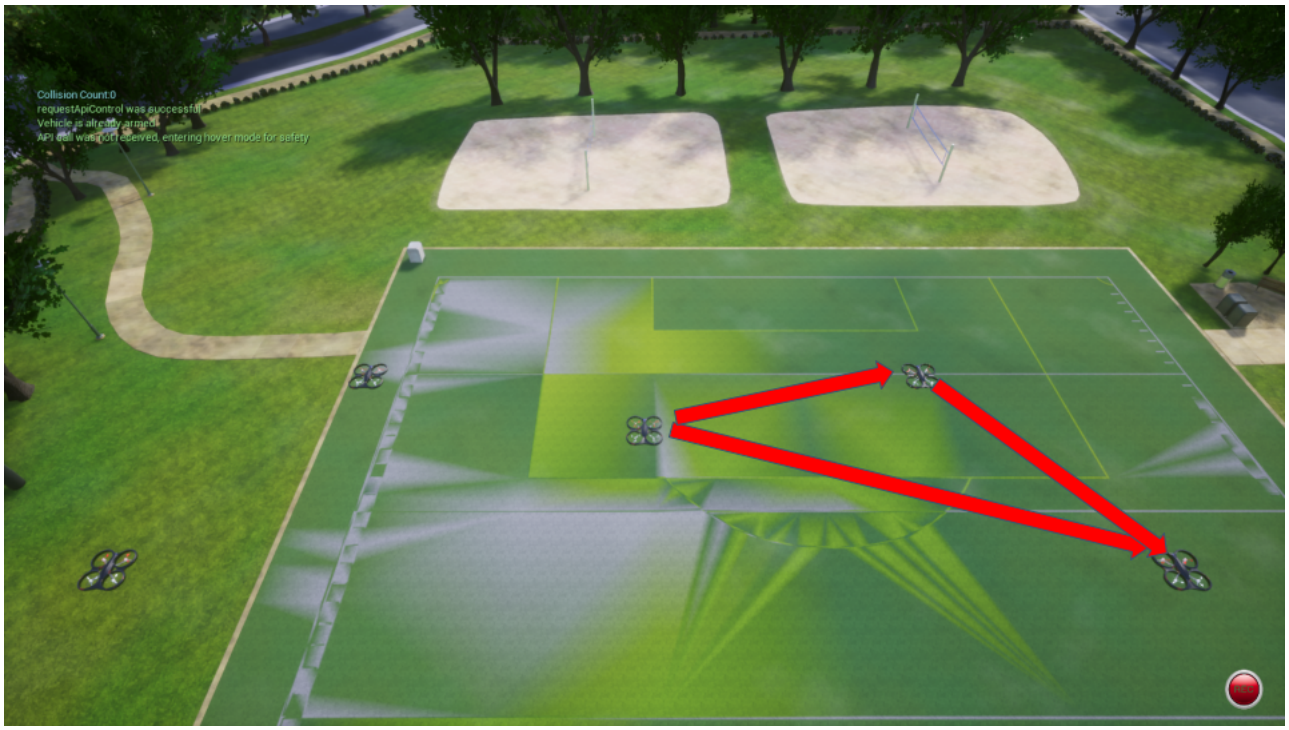

Figure 5.10: Algorithm 4: Linear Least Squares

in figure 5.10.

5.3 Results

For the second part, in AirSim, five drones were equipped with a LIDAR to detect two hundred thousand points per second at a range of twenty metres. We ran a monte-carlo simulation for thousand iterations in which each of the five drones were given random co-ordinates. One of the core requirements of our approach was that the graph formed with the drones as nodes and the inter-distance as edges should be connected. That is each node should have at least one edge another one. Thus, if in the random allocation of coordinates, a lone edge was formed, that simulation was discarded and run again. For every we show the estimation of the X and the Y coordinate separately.

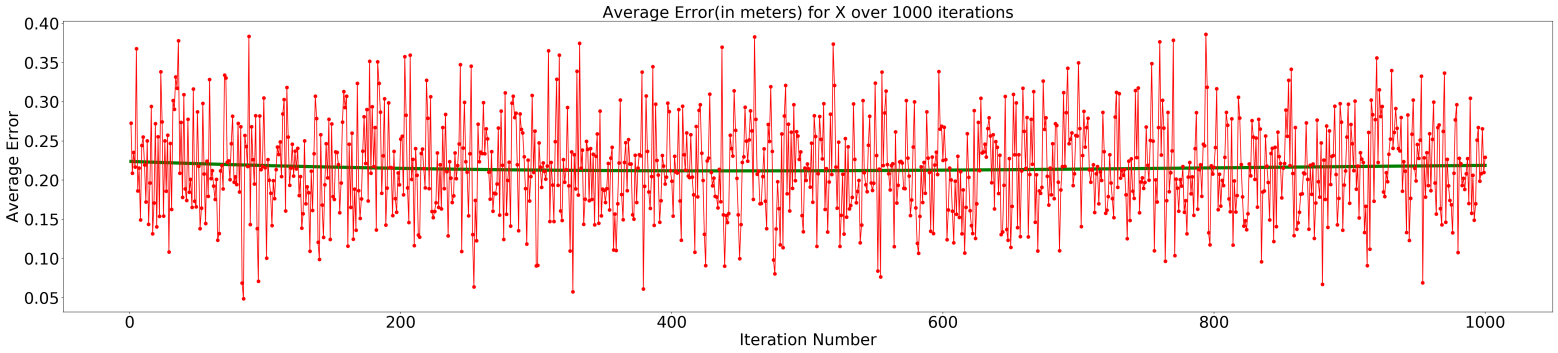

Figure 5.11: Average Error(in meters) for X 1000 iterations

In Figure 5.11, for every iteration we show the estimation error for the average of the predicted X coordinates of the four follower drones in the leader drone's relative frame.

In Figure 5.12, for every iteration we show the estimation error for the average of the predicted Y coordinates of the four follower drones in the leader drone's relative frame.

In Figure 5.13, for every iteration we show the difference between the average of the estimated X coordinates and the ground truth X coordinates of the four follower drones in the leader drone's relative frame.

In Figure 5.14, for every iteration we show the difference between the average of the estimated Y coordinates and the ground truth Y coordinates of the four follower drones in the leader drone's relative frame.

In Figure 5.15, we show the distribution of the Average Error for X over thousand simulations.

In Figure 5.16, we show the distribution of the Average Error for Y over thousand simulations.

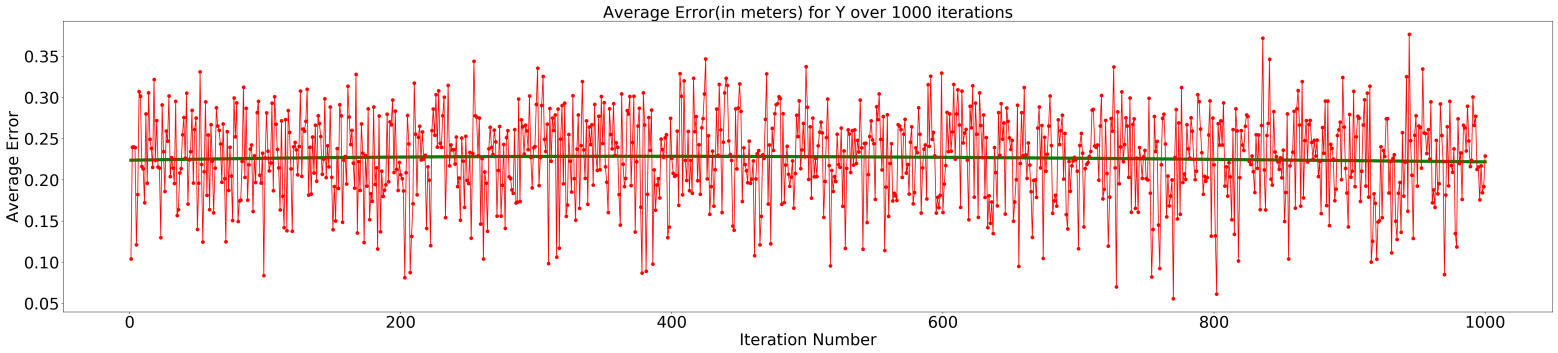

Figure 5.12: Average Error(in meters) for Y 1000 iterations

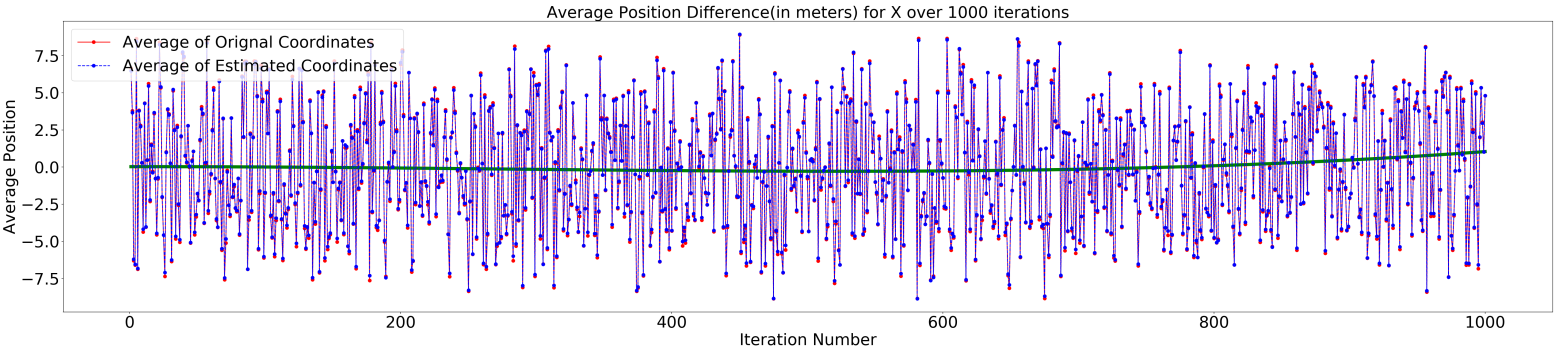

Figure 5.13: Average Position Difference(in meters) for X over 1000 iterations

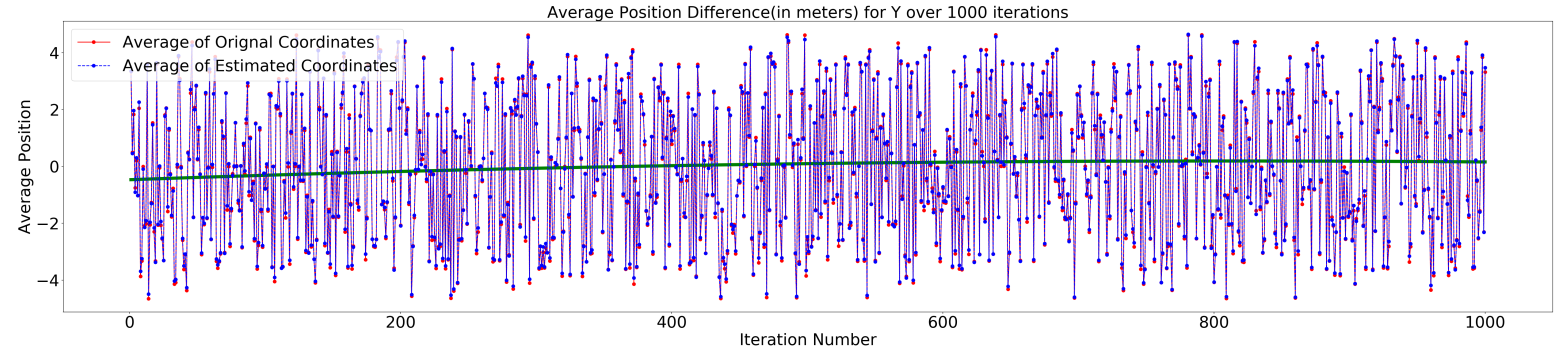

Figure 5.14: Average Position Difference(in meters) for Y over 1000 iterations

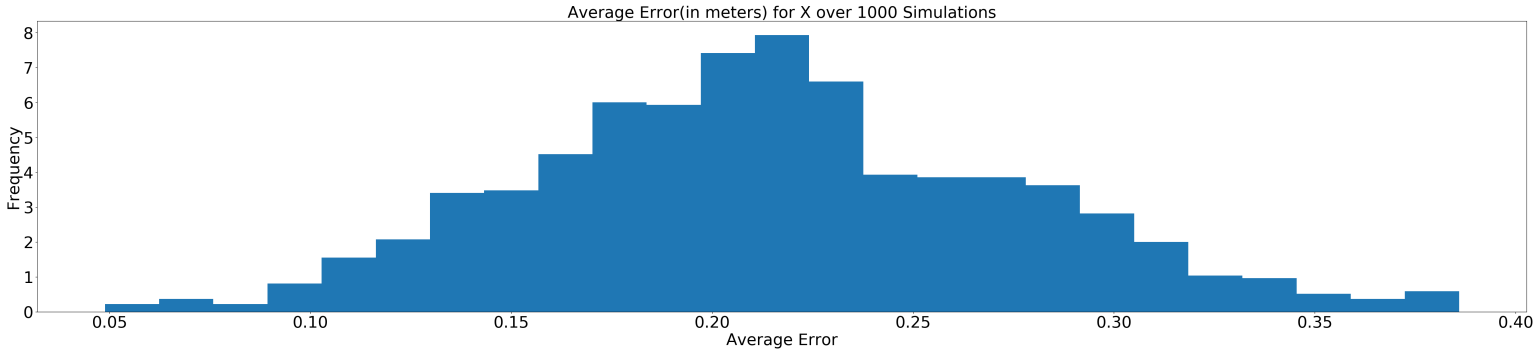

Figure 5.15: Distribution of Average Error for X over 1000 simulations

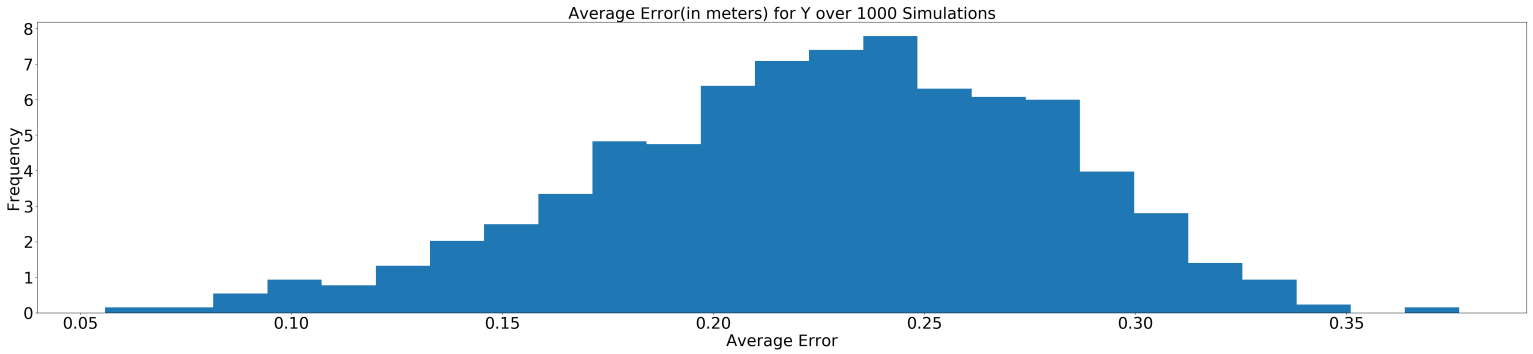

Figure 5.16: Distribution of Average Error for Y over 1000 simulations

Chapter 6

Conclusions and Future Work

In this thesis we implement a novel localization method, for multiple drones, in a GPS-denied environment with just the on-board sensors (Stereo camera system and IMU). This method can be further exploited in a number of different situations such as bridge inspection, search and rescue as described above.

We partly implemented it real-life and partly in a simulator, we have been able to implement a robust pipeline to detect, estimate and localize multiple drones in a GPS-denied environment.

We tested the versatility of the YOLOv3[50] network to be able to work with different cameras and even in different environment settings. It passed with flying colors. The flexibility that the ZED stereo camera system gives with various deep learning networks and compatibility with various NVIDIA embedded systems makes it a perfect choice for a sensor to be attached to a UAV. Along with that there are other advantages such as being lightweight and economical when compared to a velodyne lidar[19].

With AirSim[18], we were able to test the precision and the accuracy of our localization approach. Along with that, as an added bonus, we were also able to play around with the inbuilt LIDAR sensor and were able to draw a direct comparison to the stereo camera

system.

6.1 Future Work

To implement the localization in the field as soon as it is safe to do so. The simulations in AirSim lead us to conclude that irrespective of the method we use to obtain the point cloud, we would need to take into consideration the geometry of the UAV. This can be done with segmentation networks which can be combined easily with the ZED2 stereo camera system. This would lead to the estimation of the location being more accurate as we would be able to determine the location closer to the centre of mass. Another area which would need some fine-tuning is to generate an even more diverse data and training either YOLO or any other object-detection network on it. This would enable the network to detect various other kinds of UAVs other than just the DJI Matrice 100 and 600. To get rid of the bottlenecks we could use the backpack computing unit[21] as shown in Figure 6.1. Lastly, we should extend the methodology to detect missed measurements with a filter such as missing measurements kalman filter.

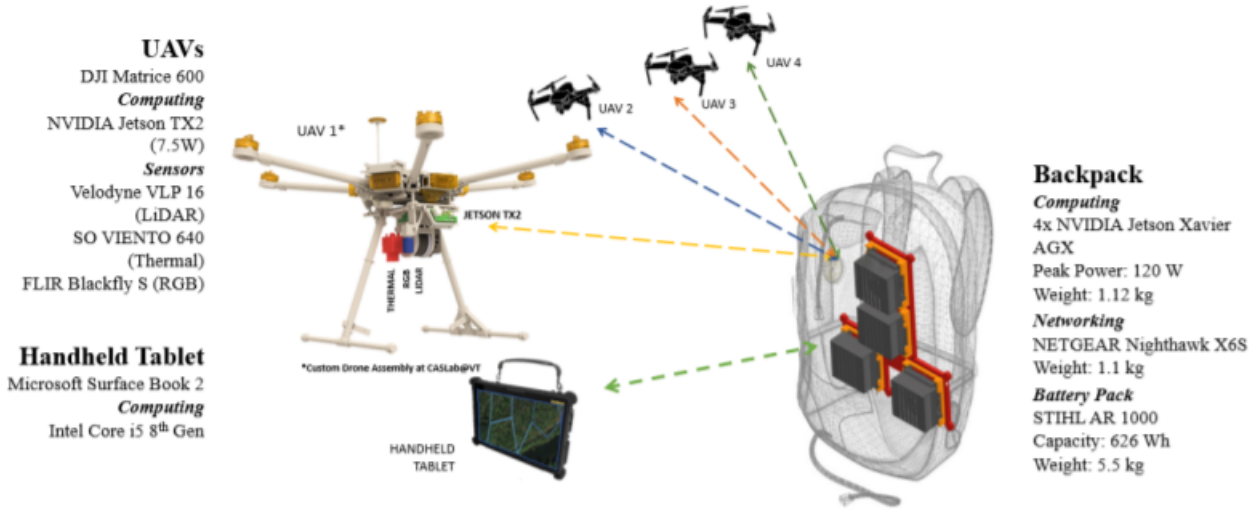

Figure 6.1: Back Pack Computing Unit [21]

Bibliography

[1] Wing drone delivery https://medium.com/wing-aviation/wing-launches-americas-first-commercial-drone-delivery-service-to-homes-in-christia

[2] Ujjwal Karn. A quick introduction to neural networks, November 2016.

[3] Robin Elbers and Tom Heskes. On the replication of cyclegan. 2018.

[4] Soham Chatterjee. Different kinds of convolutional filters, December 2017.

[5] https://www.geeksforgeeks.org/cnn-introduction-to-padding/.

[6] Stereoscope https://collection.sciencemuseumgroup.org.uk/objects/co8207343/holmes-type-stereoscope-stereoscope.

[7] Yueru Chen, Pranav Aggarwal, Jongmoo Choi, and C.-C. Jay Kuo. A deep learning approach to drone monitoring. *CoRR*, abs/1712.00863, 2017.

[8] Sudha Ravali Yellapantula. Synthesizing realistic data for vision based drone-to-drone detection. Master's thesis, Virginia Tech, 2019.

[9] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.

[10] Kaveh Fathian, Emily Doucette, J. Willard Curtis, and Nicholas R. Gans. Vision-based distributed formation control of unmanned aerial vehicles, 2018.

[11] E. Montijano, E. Cristofalo, D. Zhou, M. Schwager, and C. Sagüés. Vision-based distributed formation control without an external positioning system. *IEEE Transactions on Robotics*, 32(2):339–351, 2016.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[13] Zed 2 https://www.stereolabs.com/zed-2/.

[14] Dji matrice 100 https://www.dji.com/matrice100.

[15] Dji matrice 600 https://www.dji.com/matrice600.

[16] Orbitty carrier for tx2i http://connecttech.com/product/orbitty-carrier-for-nvidia-jetson-tx2-tx1/.

[17] Jetson agx xavier https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit.

[18] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[19] Velodyne alpha prime https://velodynelidar.com/products/alpha-prime/.

[20] Zed coordinate frame https://www.stereolabs.com/docs/positional-tracking/coordinate-frames/.

[21] James McClure Nathan Lau Larkin Heintzman Amanda Hashimoto Ryan K. Williams, Nicole Abaid. Collaborative multi-robot multi-human teams in search and rescue.

[22] Amazon prime air https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011.

[23] 2017 annual sar dashboard https://nps.maps.arcgis.com/apps/opsdashboard/index.html#/b526c87ae21f4a669eb6c9238c2c4bcf.

[24] P. Rudol and P. Doherty. Human body detection and geolocalization for uav search and rescue missions using color and thermal imagery. In *2008 IEEE Aerospace Conference*, pages 1–8, 2008.

[25] John Wang and Edwin Olson. Apriltag 2: Efficient and robust fiducial detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4193–4198. IEEE, 2016.

[26] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[27] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[29] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[31] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Springer, Cham, 2018.

[32] J. Shore and R. Johnson. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on Information Theory*, 26(1):26–37, 1980.

[33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[34] Mnist datatset http://yann.lecun.com/exdb/mnist/.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[36] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[37] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. pages 92–101, 01 2010.

[38] Charles Wheatstone. Xviii. contributions to the physiology of vision.—part the first. on some remarkable, and hitherto unobserved, phenomena of binocular vision. *Philosophical transactions of the Royal Society of London*, (128):371–394, 1838.

[39] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[40] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, USA, 2 edition, 2003.

[41] Mehran Mesbahi and Magnus Egerstedt. *Graph Theoretic Methods in Multi-Agent Networks*. 07 2010.

[42] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[43] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking, 2015.

[44] Manjia Wu, Xie Weige, Xiufang Shi, Panyu Shao, and Zhiguo Shi. *Real-Time Drone Detection Using Deep Learning Approach: Third International Conference, MLICOM 2018, Hangzhou, China, July 6-8, 2018, Proceedings*, pages 22–32. 07 2018.

[45] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[46] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, 2006.

[47] M. Saqib, S. Daud Khan, N. Sharma, and M. Blumenstein. A study on detecting drones using deep convolutional neural networks. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–5, 2017.

[48] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[49] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2016.

[50] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[51] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[52] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[53] Kaveh Fathian, Juan-Pablo Ramirez-Paredes, Emily Doucette, Jess Curtis, and Nicholas Gans. Quest: A quaternion-based approach for camera motion estimation from minimal feature points. *IEEE Robotics and Automation Letters*, PP:1–1, 01 2018.

[54] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[55] Ross Girshick. Fast r-cnn, 2015.

[56] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[57] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, Jan 2015.

[58] Zed sdk https://www.stereolabs.com/developers/release/.

[59] Vikram Puri, Anand Nayyar, and Linesh Raja. Agriculture drones: A modern breakthrough in precision agriculture. *Journal of Statistics and Management Systems*, 20(4):507–518, 2017.

[60] Najib Metni and Tarek Hamel. A uav for bridge inspection: Visual servoing control law with orientation limits. *Automation in construction*, 17(1):3–10, 2007.

[61] Marzena Półka, Szymon Ptak, and Łukasz Kuziora. The use of uav's for search and rescue operations. *Procedia Engineering*, 192:748 – 752, 2017. 12th international scientific conference of young scientists on sustainable, modern and safe transport.

[62] Jetpack sdk https://developer.nvidia.com/embedded/jetpack.

[63] What is a lidar? https://web.archive.org/web/20130530144617/http://oceanservice.noaa.gov/facts/lidar.html.