DECEMBER 2019 M.Sc. in Mechanical Engineering ENDER AYHAN RENCÜZOĞULLARI

REPUBLIC OF TURKEY

GAZİANTEP UNIVERSITY

GRADUATE SCHOOL OF NATURAL & APPLIED SCIENCES

AUTONOMOUS DRONE NAVIGATION WITH DEEP LEARNING AND COMPUTER VISION

M.Sc. THESIS

IN

MECHANICAL ENGINEERING

BY

ENDER AYHAN RENCÜZOĞULLARI

DECEMBER 2019

Autonomous Drone Navigation with Deep Learning and Computer Vision

M.Sc. Thesis

in

Mechanical Engineering

Gaziantep University

Supervisor

Assist. Prof. Dr. Ali KILIÇ

Co-Supervisor

Prof. Dr. Sadettin KAPUCU

by

Ender Ayhan Rencüzoğulları

December 2019

Ender Ayhan
Rencüzoğulları

REPUBLIC OF TURKEY
GAZİANTEP UNIVERSITY
GRADUATE SCHOOL OF NATURAL & APPLIED SCIENCES
MECHANICAL ENGINEERING

Name of the Thesis : Autonomous Drone Navigation using Deep Learning and Computer Vision

Name of the Student : Ender Ayhan RENCÜZOĞULLARI

Exam Date : 29.01.2020

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. A. Necmeddin YAZICI
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Melda ÇARPINLIOĞLU
Head of Department

This is to certify that we have read this thesis and that in our consensus/majority opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Sadettin KAPUCU
Co-Supervisor

Asst. Prof. Dr. Ali KILIÇ
Supervisor

Examining Committee Members:	Signature
Prof. Dr. Sedat BAYSEÇ	……..
Asst. Prof. Dr. M. Veysel ÇAKIR	……..
Asst. Prof. Dr. Ali KILIÇ	……..

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Ender Ayhan Rencüzoğulları

ABSTRACT

AUTONOMOUS DRONE NAVIGATION USING DEEP LEARNING AND COMPUTER VISION

RENCÜZOĞULLARI, Ender Ayhan
M.Sc. in Mechanical Engineering
Supervisor: Assist. Prof. Dr. Ali KILIÇ
Co-Supervisor: Prof. Dr. Sadettin KAPUCU
January 2020, 78 Pages

In this study, a configurable autonomous UAV system and platform are presented for operations that require autonomy and inference ability. The system is capable of performing autonomous take-off, planned flight and landing while also making inference about objects. Autonomous flight system is built with open source flight stack of Pixhawk and Robot Operating System (ROS). If navigation task is known as waypoints, controller node in robot operating system sends predefined route to flight stack via communication protocol. That allows the drone navigates in open fields as desired. Also, object detection is implemented with widely used DNN algorithm. This algorithm is not only retrained by applying fine-tuning, also optimized with TensorRT engine so that system performance is improved in terms of accuracy and speed. That is to say, developed UAV is aware of environment during flight. Entire system runs on Jetson TX2 with high GPU performance. Also, SITL method is utilized within a simulator for debugging and fast control prototyping purpose so that most probable crashes can be prevented or mitigated before occurring during real tests. In addition to software a UAV platform is produced for running the developed system in real environment. Design consideration of UAV, hardware selection and production are presented.

Key words: Autonomous UAV, Drone navigation, Deep neural network, Deep learning, Object detection, Drone production with 3D printer.

ÖZET

DERİN ÖĞRENME VE BİLGİSAYARLI GÖRÜ KULLANARAK OTONOM DRON DOLAŞIMI

RENCÜZOĞULLARI, Ender Ayhan
Yüksek Lisans Tezi, Makine Mühendisliği
Danışman: Dr. Öğretim Üyesi Ali KILIÇ
Eş-Danışman: Prof. Dr. Sadettin Kapucu
Ocak 2020, 78 Sayfa

Bu çalışmada, çıkarım yapma yeteneğine ve otonomiye gerek duyulan operasyonlarda kullanılmak üzere yapılandırılabilir bir İHA sistemi sunulmuştur. Sistem; otonom kalkış, planlı uçuş ve iniş yapma yeteneğine sahip olup, aynı zamanda nesneler hakkında çıkarımda bulunabilmektedir. Otonom uçuş sistemi, Pixhawk'ın açık kaynaklı uçuş yığını yazılımı ve Robot İşletim Sistemi ile inşa edilmiştir. Dolaşım görevi yol noktaları olarak biliniyorsa, robotik işletim sistemindeki denetleyici düğümü iletişim protokolü aracılığıyla uçuş yığınına önceden tanımlanmış bir rota gönderir. Bu, dronun açık alanlarda istenildiği gibi hareket etmesini sağlar. Ayrıca, nesne tespiti yaygın olarak kullanılan bir DNN algoritması ile gerçekleştirilmektedir. Bu algoritma sadece rastgele ayarlamalarla yeniden eğitilerek değil, aynı zamanda TensorRT motoruyla optimize edilerek düzenlenmiştir. Böylece sistem performansı doğruluk ve hız açısından iyileştirilmiştir. Diğer bir deyişle, geliştirilen İHA, uçuş sırasında çevrenin farkındadır. Tüm sistem Jetson TX2 üzerinde çalışmaktadır. Ayrıca, SITL yöntemi hata ayıklama ve hızlı kontrol benzetim amacıyla bir simülatör içersinde kullanılabilir. Böylece muhtemel çarpışmaların gerçek testler sırasında gerçekleşmesi engellenmekte veya hafifletilmesi sağlanmaktadır. Geliştirilen yazılıma ek olarak, sistemi gerçek ortamda çalıştırmak için bir İHA platformu üretilmiştir. İHA'nın tasarım değerlendirmesi, donanım seçimi ve üretimi sunulmuştur.

Anahtar kelimeler: Otonom IHA, Drone dolaşımı, Derin sinir ağı, Derin öğrenme, nesne tespiti, eğitim veriseti, 3B yazıcı kullanarak drone üretimi.

Dedicated to my lovely family

ACKNOWLEDGMENTS

I would like to thank to my supervisor Assist. Prof. Dr. Ali Kılıç for his patience, supportive attitude and guidance from the very beginning of this project. I have learnt many things from him, but in particular, being practical as required by engineering.

I also would like to thank to my co-supervisor Prof. Dr. Sadettin Kapucu who is eye-opening, kind and reassuring. Attending his classes was fascinating and fun because his narrative skill is powerful.

There was also another unofficial supervisor for me at Providence University in Taiwan, Assist. Prof. James Wu. We have met on Github while I was seeking a solution to a problem, afterwards we have worked together online for almost one year. He has helped me a lot during my project and after. Thanks James.

Beyond all, I would love to convey my sincere gratitude and love to my family for their endless support, love and faith on me. I would not achieve many things without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS and SYMBOLS

DL	Deep learning
DNN	Deep neural network
NN	Neural network
ML	Machine learning
AI	Artificial intelligence
ReLU	Rectified linear unit
ROI	Region of interest
IoU	Intersection over union
NMS	Non-max suppression
mAP	Mean average precision
RCNN	Region-based convolutional neural network
YOLO	You only look once
SSD	Single shot detector
VOC	Visual object classes
COCO	Common objects in context
ONNX	Open neural network exchange
SLAM	Simultaneous localization and mapping
DSO	Direct sparse odometry
VO	Visual odometry
SITL	Software in the loop
UDP	User datagram protocol
FCU	Flight controller unit
SoC	System-on-chip
ECU	Electrical controller unit
RC	Radio controller
GPS	Global positioning system
IMU	Inertial measurement unit
GCS	Ground control station
QGC	QGroundControl

ROS	Robot operating system
GUI	Graphical user interface
ESC	Electronic speed controller
PLA	Polylactic acid
PETG	Polyethylene Terephthalate Glycol-modified
I2C	Inter-integrated circuit
PWM	Pulse width modulation
LiPo	Lithium polymer
$\boldsymbol{K_v}$	Velocity constant
$\boldsymbol{\omega^T}$	Transpose of weight
$\hat{y}$	Label
$\boldsymbol{b_x}$	Center of bounding box along x-axis
$\boldsymbol{b_y}$	Center of bounding box along y-axis
$\boldsymbol{b_w}$	Width of bounding box
$\boldsymbol{b_h}$	Height of bounding box
$\boldsymbol{c}$	Object class
$\boldsymbol{\sigma}$	Sigmoid function
$\boldsymbol{\alpha}$	Learning rate
$\boldsymbol{\omega}$	Weight
$\boldsymbol{b}$	Bias

CHAPTER 1
INTRODUCTION

1.1 Introduction

Autonomous vehicles (AVs) are rising like a new era. They can fly, drive or swim in natural environments including harsh conditions and rough terrains. That capabilities offer both conveniences and safety to human life. For example, autonomous cars can detect obstacles on the path and execute braking for collision mitigation or avoidance. On the other hand, unmanned aerial vehicles (UAVs) may fly hundreds of miles for surveillance, military purposes and field fertilization.

UAVs are a kind of mobile robots that mainly consist of controller(s), communication system, propulsion system and other complementary components. Flight can be executed by both a human remotely and autonomously with an onboard computer. Compared to manned aircrafts, UAVs are used more in hard-to-reach and dangerous areas for human to work. Although they are mostly used in military applications, their use in civil, scientific and commercial applications is tremendously increasing for a decade. For example, music and movie industries uses drone, which is universally the most used name for rotary-wing vehicles, for shooting video from different perspectives (i.e. bird's eye view). Therefore, an operation that is needed to be completed with UAVs defines the requirements.

Analyzing and determining the requirements (e.g. size, hardware and software) is crucial for designing and integrating a UAV system successfully. In that regard, system design of UAVs depends on requirements. Environment and type of operation define vehicle, system and performance requirements. These requirements create sub-requirements of the system up to units.

These units consist of electronic hardware, mechanical structure and software system in general aspect. Speaking of electronic hardware, sensor technology provides cost

effective sensing elements day-by-day; such as camera, gyroscope and GPS. Thus, manned aerial vehicles quickly evolve to unmanned vehicles by virtue of sensors that provide vision, location, position and speed information. Also, use of powerful ECU is another fundamental requirement for processing complex and huge data, especially for vision-based applications (i.e. object detection with deep learning) which requires extremely high computing power [15].

Selecting appropriate software programs is another challenging task that affects development process of a UAV. Operating of robotic systems require various programs and libraries. For example, OpenCV is probably the most used library for image processing which can be utilized with Python and C++ programming languages. Beyond that, there are a number of dependencies for developing autonomous systems. In this project, a few open source software is used through developing autonomous flight system with Jetson TX2. This onboard computer has high computing capability by virtue of its GPU processor for deep learning applications [3].

For implementation of deep learning in this project, autonomous navigation system of a UAV is, not only duplicated but also improved, from an open source project which is Redtail of NVIDIA [3]. The UAV of Redtail has capability to fly autonomously through unseen trails/path in forest by use of trained model. Also, in Redtail project, object detection is carried out to avoid crashes while UAV is navigating autonomously throughout its path. However, speed and accuracy of object detection model of is extremely low (1 fps) which may cause imminent collision in case of sudden appearance of dynamic objects (i.e. person) in front of the UAV. To improve, object detection and classification are applied by training an open source dataset. Some different optimized DNN architectures are trained and benchmarked to reach higher accuracy and speed in detecting and classifying objects for performing better real-time detection. Thus, this study shows that a UAV can fly safely at high speeds.

As for structure, UAVs can be classified as Fixed-Wing Aerial Vehicles (FWAV) and Rotary-Wing Aerial Vehicles (RWAV), which are shown in Figure 1.1 and Figure 1.2. FWAVs are designed to soaring high above (e.g. 10.000 meters) through wide ranges with high speed. On the other hand, RWAVs (i.e. multi-copters, helicopters) are designed to operate with high agility while avoiding obstacles in narrow areas.

Moreover, most distinguishing specifications are, but not liovering, vertical take-off and landing abilities because of their structure. Multi-copters are the top choice for confined areas since they are naturally more agile, compact and easy to launch than FWAVs [18]. There are various types of multi-copters, such as quadcopter, hexacopter and octocopter, according to number of propeller-motor couple.

Figure 1.1: Rotary-wing UAV

Figure 1.2: Fixed-wing UAV

1.2 Research and Development Objectives

In this thesis, main focus is not only design consideration of UAV platform, but also integration of a system architecture for autonomous flight with object detection and classification capabilities in unknown environments. There are two fundamental objectives of this study which are described in the following paragraphs.

Mechanical aspect: As structure, quadcopter is selected, developed and produced from scratch. Thus, this thesis offers a suitable quadcopter platform for tasks that rely on autonomous flight. The quadcopter is produced by 3D printer for rapid prototyping purpose and cost effectiveness.

Software aspect: The developed platform can be controlled manually, semi-automatically and autonomously according to requirement of user. Entire UAV system uses open source software so that it can be modified and developed in flexibility

aspect. Moreover, architecture of system is modular by use of Robot Operating System (ROS). Such kind of system can be improved by adding new capabilities which don't affect other capabilities. But in the first place, DNN-based object detection with high speed and accuracy is carried out for collision avoidance while navigating real-time in unknown and complex outdoor environment. Consequently, this study presents a design consideration of quadcopter type UAV configuration with its pros and cons for smart applications.

1.3 Thesis Outline

Chapter 1 provides basic intiution about study that explaines research and development objectives. Chapter 2 provides concise brief about literature that has been surveyed before starting this project. That chapter is split into two main topics which are UAV systems and platforms and object detection method. Chapter 3 explains basic concepts of deep learning, convolutional neural networks and computer vision within the boundary of this project. Chapter 4 is the method section that has been applied for making a drone smart through this project. Object and obstacle detection, its implementation and the theory underlying object detection were explained. Also, there is another method section which is Chapter 5 that explains how autonomous flight system has been developed and integrated by using open sources by virtue of community. That chapter also provides some intitution about software-in-the-loop method because it is one of the most important steps while developing a software system, in particular aerial vehicles. Chapter 6 has been dedicated to give information about how to configure UAVs, select frame, choose hardware and produce the selected frame through production with 3D printer. Finally, Chapter 7 concludes the study in terms of performance and failure.

CHAPTER 2
LITERATURE SURVEY

2.1 Introduction

For autonomous mobile robots, especially aerial vehicles, there are many requirements in many aspects such as software and hardware. Furthermore, perception and navigation are the most important capabilities that make robots mobile. Developing an autonomous UAV which is perceiving environment and navigating around can be achieved with following methods.

2.2 UAV Systems and Platforms towards Autonomy

Lorenz Meier et al. [1] have presented low weight quadrotor structure with carbon fiber base plates. Each motor with 8 inches propellers enables the quadcopter to carry nearly 400g. Total system capacity as payload around 1.2 kg including battery. Continues flight time of that quadcopter is around 8 minutes. Essential hardware setup consists of pxCOMEx and pxIMU as well as two USB cameras. That system enables autonomous flight depending on onboard image processing. Hardware and software setup provide high-speed and low-latency.

Pierre-Jean Bristeau et al. have reviewed platform design, control and navigation technologies of AR Drone which is one of the most known commercial drones. AR Drone is a conventional quadcopter that consists of four brushless motors with propellers. Motor carriers are attached together with carbon fiber tubes while central body is plastic. The battery package is Lithium-Polymer with 3 cells supplying 11.1 V and 1000 mAh. That capacity refers to 15 minutes flight approximately. There are two main onboard electronics which are mother board and navigation board. Real-time operating system and entire calculations are run on mother board. System includes two camera that may be used for object recognition. On the other hand, navigation board is dedicated to provide interface between embedded sensors and the mother board. These sensors are gyroscopes, accelerometers and ultrasonic sensors. The

control technology of AR Drone consists of two main loops which are attitude and angular rate controls. While attitude control loop calculates angular rate from the attitude set point and estimated attitude, angular rate loop manages the motors with proportional controllers.

Smolyanskiy et al. [3] has provided a novel approach to autonomous flight with deep learning and a commercially available drone platform to implement autonomous system in real environment. 3DR Iris quadcopter is slightly modified for extra electronic hardware such as camera and lidar. Jetson TX2 is utilized as companion computer for running entire software system while Pixhawk is used as flight controller unit. In GPS-denied environments like forest, open source IDSIA dataset is used for training TrailNet architecture. Dataset consists of three views of the same scene. These views show left, center and right side of the path in forest. Images of views are labeled according to their orientation so that TrailNet model is trained on this dataset to infer orientation angle of drone to navigate. Afterwards, estimated orientation result is fed into the controller. On the other hand, obstacle avoidance has been implemented to avoid collisions during flight. Direct sparse odometry has been carried out for obstacle avoidance as well as object detection with YOLO. Software system runs in robot operating system in terms of flexibility within a single system.

Weaver et al. [4] has proposed implementation of autonomous take-off and landing with UAVs onto boats. The UAV which is used in this project is quadcopter type drone. Software system works in robot operating system as a framework. Scope of this project is to achieve that quadcopter takes-off from a launch pad on the boat, follows the predefined waypoint sequences from GPS, lands onto a predetermined location to pick up object, navigates back and lands onto the boat. DJI F450 frame has been used to optimize cost. Landing gear is ABS that absorbs collision impact in case of crash. LiPo battery with 6000 mAh is the power source for quadcopter so that it allows quadcopter to operate for 15 minutes approximately. Ardupilot Mega, like Pixhawk, is dedicated to provide stabilization and navigation. ODROID, installed with Ubuntu operating system and robot operating system, receives GPS and other flight attributes from Ardupilot. A camera, which is Logitech c920, is used for detecting landing spot on boat. Quadcopter receives commands from ground station vehicle so that it follows waypoints from GPS. As soon as the camera detect landing spot which is circle, quadcopter aligns itself over the spot, lands and pick up the object. Lastly, quadcopter

starts to follow sequences of waypoints towards the home position. This system is promising for search-and-rescue operations.

Siebert et al. [5] have designed and built a UAV to evaluate its performance in real environment for surveying purpose. The developed UAV is based on quadcopter which mainly consists of GPS, camera and flight controller. Structure of quadcopter has been made of aluminum and carbon fiber reinforced plastic to obtain lightweight platform. Flight trajectory is transmitted to the quadcopter via wireless from ground station, mobile phone or a mobile pc. After trajectory is submitted to the system, quadcopter follows autonomously a predefined trajectory. Trajectory file, which is used for path planning, consists of global position, altitude, speed and camera trigger. If a waypoint, which is predefined in trajectory file, is dedicated to be trigger for camera to capture image, system executes this command and take image(s). Captured images are evaluated for 3D mapping surveillance of earthwork projects.

Kan et al. [6] have developed autonomous drone which consists of Raspberry Pi 2.0 for flight control and GPS. Type of UAV is AR Drone which is quadcopter. Idea of this project is to evaluate success of planned flight with GPS about outdoor autonomous flight. The proposed system for evaluation of GPS-based navigation has been experimented by calculating derivation between predefined waypoint with GPS and current position measured with GPS. Proposed method to decrease misalignment between calculated current position and predetermined position is that longitude and latitude deviations are measured and used in calibration of GPS. Proposed method has been assessed according to planned routes.

Benito et al. [7] have presented design consideration for UAV platforms carrying payload. Indoor applications have been considered so that UAVs should not exceed the width of a classic door which is 750 mm. Agility and maneuverability have been highlighted in accordance with video capturing for surveillance purpose. Afterwards, when combining these requirements for configuring a UAV, quadcopter has been preferred for their project.

2.3 Object Detection with Deep Learning

Lee et al. [8] have proposed a method for implementing object detection offline by moving computation to off-board computing cloud. RCNN algorithm, which is a region proposal method, has been applied in this project. The goal is to reduce

computational load on lightweight and low-cost UAVs because object detection with convolutional neural networks requires powerful graphical processor units. The system mainly consists of position estimator, proportional integral derivative (PID) and object detection with convolutional neural network. Entire system runs within robot operating system on a laptop which is connected to drone via driver of robot operating system over Wi-Fi. While position estimator and PID controller runs on computer, object detection runs on a remote server that is connected to laptop via internet. Thus, RCNN algorithm is applied for detecting objects remotely. Although this method makes sense about reducing computational load on UAVs, it does not seem efficient for real-time applications because of latency.

Girshick et al. [9] have developed promising and widely known convolutional neural network approach for object detection. RCNN detector consists of three units. One of units generates region proposals which are independent from classes. Subsequent unit is responsible for extracting features with convolutional neural network from each region. At the end, according to object classes, linear support vector machines operate to infer. Although this algorithm is accurate and robust for object detection, it runs slower than single-shot detectors. That is to say, convolution operations are time consuming because of sliding window operations.

Redmon and Farhadi [10] have proposed one of the fastest object detection algorithms as well as high accuracy. Unlike region proposal algorithms, this detector split images into grids. Afterwards, algorithm passes each split cell for predicting a bounding bow with help of anchor boxes. That allows algorithm predicts faster than sliding-window-based algorithms. This method is used as detector in this project so that it will be explained in detail.

Liu et al. [11] have presented YOLO-like detector which is called as single shot detector (SSD). That means a single deep neural network detects objects in images. Also, anchor box logic is used in this method. This algorithm discretizes predicted bounding boxes according to the default/predefined boxes (anchor box) with different aspect ratios according to classes/objects in dataset. In the same time, SSD also produces scores about existence of object classes in associated bounding box. This step is followed by non max suppression to estimate final detection.

Everingham et al. [12] have presented a publicly available dataset including images and their ground truth labels. This dataset is named as PASCAL VOC which stands for PASCAL Visual Object Classes. This dataset has been collected and annotated to provide large amount of standardized data for computer vision and machine learning community to compare different methods such image segmentation and object detection. This dataset consists of 20 different object classes. This dataset has been collected from 2005 to 2012.

Lin et al. [13] have collected and presented a dataset to advance state-of-the-art in scene understanding. This dataset is called as Microsoft Common Objects in Context (MS COCO). These images have been collected from everyday life in natural environment. MS COCO has two releases. First grouped has released, in 2014, with 80 classes. On the other hand, other dataset has been released, in 2015, with 91 classes. 2014 release contains approximately 160.000 images which have been distributed as training, validation and test. Training folder takes 50 percent while remaining images are for validation and test.

Vandersteegen et al. [14] have developed a solution for running object detection fast and accurate on UAVs because real-time detection is needed for surveillance operations. This study claims that the proposed solution may cope with different flight altitudes and angel of view in object detection. This method has been implemented on both Jetson TX2 and Xavier platforms with 10 times increased speed. YOLOv3 has been used as base architecture. YOLOv3 is generally used for detecting 80 classes and taking three different input resolutions as 320x320, 416x416 and 608x608. Thus, with this solution, only required set of classes are used, such as person, car, bicycle, motorbike, bus and truck. Also, not only anchor boxes for selected classes have been recalculated to improve accuracy, but also input resolution has been modified as 608x352 which stands for 16/9 aspect ratio. After the detector has been trained on both MS COCO and Visdrone2018 dataset, TensorRT has been used in optimizing the architecture that has caused boost in performance.

Hossain and Lee [15] have declared that target detection and tracking from drones are demanding in recent years. They have proposed an effective deep-learning-based method for these demands so that proposed method has been run on both Jetson TX, Jetson Xavier and Intel Neural Compute Stick because of real-time calculations.

Comparison of these onboard computing platforms are compared so that a benchmark table has been offered when selecting the appropriate hardware. YOLOv2, YOLOv3, tiny-YOLOv2, tiny YOLOv3, Faster-RCNN, SSD, DeepLab-v3 and YOLOv3 + Deep SORT have been compared in terms of speed for assessing performance of object detection, tracking and semantic segmentation at pixel level on-board embedded GPU systems.

Nils Tijtgat et al. [16] have presented an evaluation result of performing real-time object detection on UAVs with Jetson TX2. Evaluation has been based on various detection algorithms. To select cutting-edge detector, accuracy and speed requirements for a given frame rate have been considered according to benchmark. Consequently, YOLOv2 architecture has been recommended for object detection in contrast to CPU-based algorithms or other GPU-based algorithms because results showed that YOLOv2 has higher accuracy and speed for a given frame rate.

Song Han et al. [17] have proposed a framework, which is called as Deep Drone, to boost visual performance of drones in detecting and tracking objects. Implementation of the proposed framework has been carried on Jetson TX and TK platforms as well as a desktop GPU which is NVIDIA GTX 980. Processed frame rate per second, power consumption and accuracy have been evaluated. System architecture that has been proposed consist of detection algorithm with convolutional neural network and tracking algorithm using HOG feature. For object detection, first version of YOLO and Faster RCNN have been used and analyzed. YOLO have not been used since it is bad at detecting small objects or far objects. On the other hand, KCF algorithm have been used for tracking objects. Consequently, this study has presented a new architecture that combines object detection and tracking to be used on UAVs.

CHAPTER 3
DEEP LEARNING for COMPUTER VISION

3.1 Introduction

Deep learning (DL) is a particular kind of machine learning technique that is used to teach computers natural phenomena that human can learn and apply. DL is the key technology underlying a number of AI-based high-tech solutions in, but is not limited to, engineering, cybersecurity, medicine and agriculture. For example, DL enables driverless car to recognize a traffic light and detect a pedestrian accurately. In addition, researchers accelerate diagnosis of cancer by segmenting out detected cancer cells. Moreover, DL can achieve impressive results that are exceeding human level performance sometimes. A computer model can learn recognition or detection directly from image sequences, sound or text. Such kind of models are trained with large numbers of labeled data and neural network (NN) architectures.

Until the advent of DL, another dominant option was manual engineering instead of learning approach in computer vision. Although it requires extreme effort for separating task and extracting features from images or videos, yet, autonomous systems cannot be thought without manual processes (i.e. image processing) which are applied by human directly. Yet, computer vision and deep learning are not discrete disciplines. As can be understood from the name, it is vision of computers that turns out image-based deep learning is a computer vision application since it requires vision to learn images by use of NN architecture. Deep learning is an application in the field of computer vision (excluding, for example, speech and text recognition). There is no any strict boundary between computer vision and deep learning. In similar manner, fundamental goal of computer vision is to emulate human vision. It enables making inferences and performing actions by using visual input [19].

3.2 Neural Network Basics

Inventors dreamed about creating machines that think for long time ago. Although this desire dates back to the time of ancient Greece, actual beginning of artificial

intelligence (AI) appeared around Second World War. Alan Turing, who solved Enigma machine, was the one to put forward the question of whether machines can think [20]. Speaking of boundary of AI, ML and DL, while AI is the most comprehensive naming and broadest boundary that involves ML as subset, ML includes DL as subset through AI development. Figure 3.1 shows the boundary of these technologies and milestones

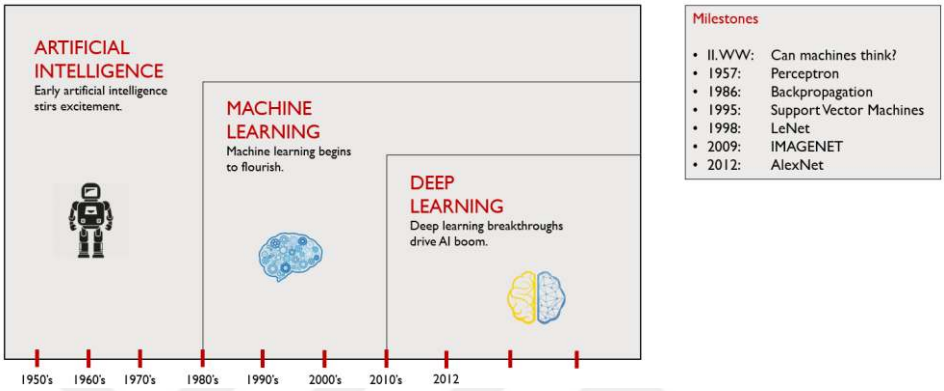

Figure 3.1: Deep learning in AI with milestones [50].

For a decade, AI is promising field not only for practical applications, but also for research subjects. To automate routine works, analyze images or understand speech, intelligent software is increasingly developed by engineers, developers and researchers. This approach is beneficial in many aspects. For example, need for human to specify necessary information or extract required feature from data that computer needs are avoided [19].

Like human or animal brain, AI has neurons which are represented with circles that are inter-connected each other as shown in Figure 3.2. This composition turns out to a neural network with a number of layers.

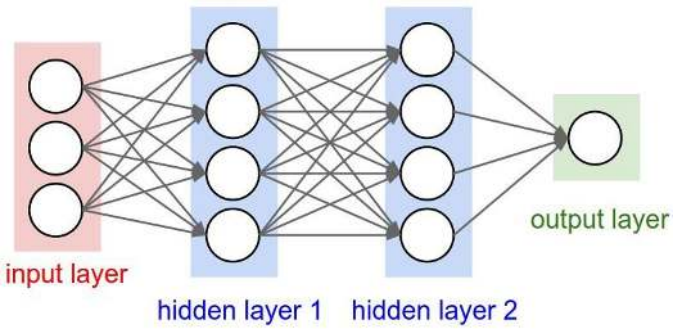

Figure 3.2: Neurons are represented with circles. A layer consists of neuron(s) depending on network architecture.

A neural network is a computing model and can learn from data so that it can recognize pattern, classify data and predict future events. It generally consists of two or three connected layers and operates on them. Such kind of networks are called shallow neural network [19]. On the other hand, deep networks have many layers according to the task. They may have deeper layers which can reach hundreds. These both are machine learning approach that learn from input data directly.

As is seen in Figure 3.2, layers are working in parallel like a nervous system. Illustrated network, for example, consists of an input layer, two hidden layers and one output layer. Calculation result of each layer is an input for subsequent layer. Input layer receives data from world. It passes the inputs to the first hidden layer. Afterwards, the hidden layer performs mathematical computations on inputs to send the calculation result to next layer for another calculation, and at last to output layer for the final evaluation. For example, to predict the price of an airplane ticket, departure date, flight company and distance to destination (flight range) are essential factors which are more important (heavy in terms of weight) than other factors to consider, such as race or age of pilot and flight attendants.

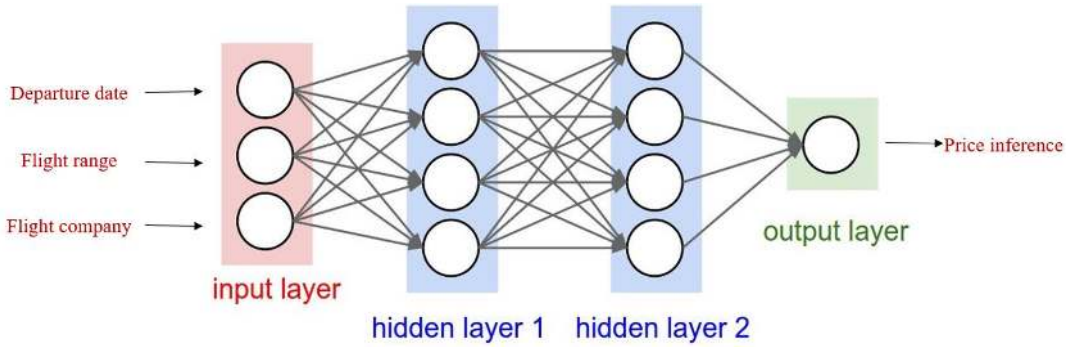

Figure 3.3: Representative price estimation network

The input layer receives input data which are departure date, flight range and flight company in this example. These three-input data pass to the first hidden layer which performs mathematical computations on the inputs. For example, after passing departure date to first hidden layer, the related neuron may provide the detailed information about the date which may be holiday. As can be predicted, holiday causes more expensive ticket price compared to ordinary days. On the other hand, if flight company has not a good reputation, output of the hidden layer passes a result which may cause lower price information since, for instance, the customer care of this company is comparably worse than reputable companies. Thus, these results from the layers passed to next layer by affecting the estimation result. The output layer is used to estimate price of ticket after all computations. This information is called as weight. Each connection between neurons is associated with a weight that dictates the importance of the received input, shown in Figure 3.4. Initial weights are generally set randomly [19, 21].

Logistic regression is a learning algorithm for binary classification which is to define the output if the result is ‘1’ or ‘0’. Figure 3.4 illustrates the classification result whether it is a dog

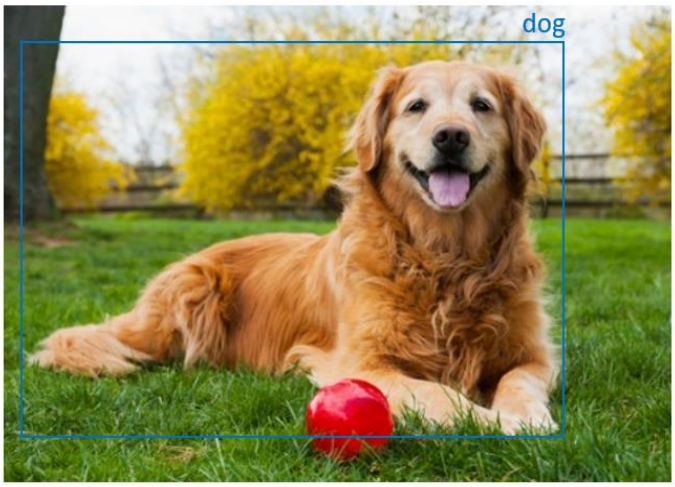

Figure 3.4: Input image for binary classification to predict whether it is a dog.

Logistic regression solves regression problem which can take an input vector $x \in \mathbb{R}^n$ and predict the value $\hat{y} \in \mathbb{R}$ as scalar output which is defined as

$$\hat{y} = \omega^T x + b \tag{1}$$

where $b \in \mathbb{R}$, which is bias, and $\omega \in \mathbb{R}^n$ is a vector of parameters which are called as set of weights that controls the behavior of the system. Each parameter (e.g. departure date) in weights determine how feature affects the prediction. If a feature x_i receives a positive weight, that increases the value of the prediction $\hat{y}$. If a feature x_i receives a negative weight, that decreases the value of the prediction. The larger magnitude of weight, the much more the result is affected. Opposite situation is also valid. Figure 3.5 shows positive and negative weights that are passing to next layer.

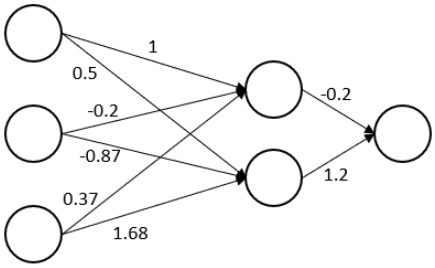

Figure 3.5: Neurons pass weights to subsequent neurons.

For given x, it is desired that $\hat{y} = P(y = 1 \mid x)$ in probability aspect. However, $\hat{y}$ may be much bigger than 1 or even be negative which doesn't make sense for probability. Thus, it is desired to be $0 < \hat{y} < 1$. In this regard, each neuron should have an

activation function which defines the output of that neuron for given set of inputs x. There are a numerous activation functions, such as binary step, sigmoid, TanH and ReLU. There is not a rule or specific guide for selecting best activation function since it is a trial-error process. For example, when implementing logistic regression model, that must learn parameters ω and b, sigmoid function from diverse type of activation functions can be applied to $\hat{y}$ for binary classification as shown with dog example above [3, 19, 21, 22].

$$\hat{y} = \sigma(\omega^T x + b) \tag{2}$$

where σ stands for sigmoid function. Let $z = \omega^T x + b$. Sigmoid function then is equal to:

$$\sigma(z) = \frac{1}{1+ e^{-z}} \tag{3}$$

In this case, $\sigma(z)$ approaches 1 as z gets larger. Conversely, if z is small, $\sigma(z)$ approaches 0. That can be seen clearly in Figure 3.6.

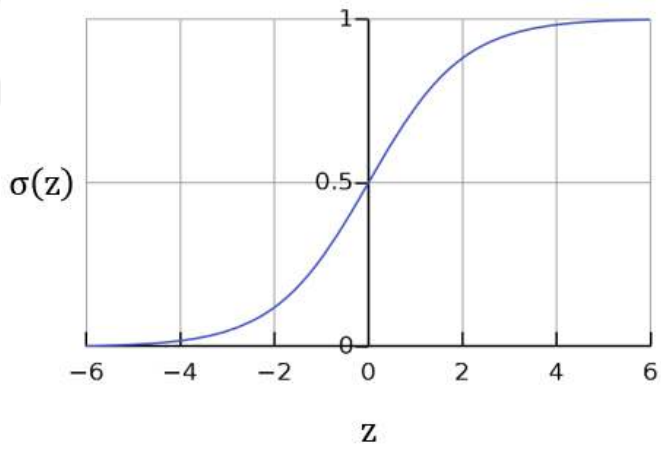

Figure 3.6: Sigmoid function commutes between 0 and 1 as z changes.

To train the parameters of logistic regression, it is required to define a cost function. It is a function to define in measuring how good the output $\hat{y}$ when the ground truth label is y. The ground truth label y is the desired result which is known and desired value that computer does not know at the very beginning. For logistic regression, the following equation which is a loss function regarding a single training example. Thus, it measures how good training is implemented on a single input [3].

$$L(\hat{y}, y) = -(y log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$ (4)

However, the actual goal is to measure training on entire training set. Thus, cost function, J, is implemented to parameters which must have average value for entire set. It means that summation of each loss function is divided by number of training examples in the set, which is m. Consequently, cost function is:

$$J(w, b) = -\frac{1}{m}\sum_{i=1}^{m} y^i log(\hat{y}^i) + (1 - y^i) \log(1 - \hat{y}^i)$$ (5)

Cost is required to be minimized as much as possible. It is overall sum of loss (error) through training on entire set of input. Gradient descent is a powerful and general optimization method to increase the success of estimation result. Gradient descent can be visualized as in Figure 3.7 [21].

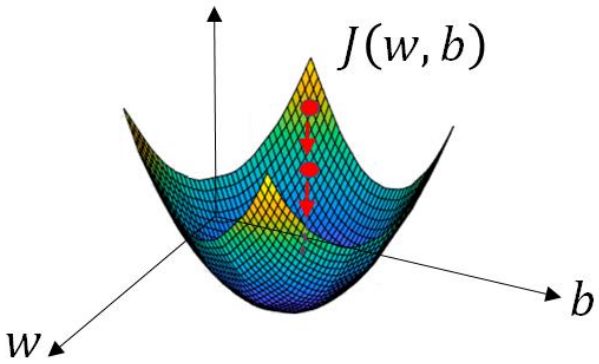

Figure 3.7: Cost is optimized with gradient descent method.

When implementing gradient descent, this method is repeatedly update parameters w and b until the algorithm converges. Gradient descent is a step-by-step update which uses learning rate α on partial derivative of cost function. Thus, parameters are optimized for optimum result of logistic regression model. In order to reduce loss, gradient descent is implemented as follows

$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$ (6)

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$ (7)

The two steps, which are described above as logistic regression and loss calculation, are called as forward propagation which computes the loss on a single training example. After loss calculation, backward propagation (gradient descent) that goes

backwards is used to compute derivatives with respect to loss. Derivative calculations regarding variables starting from loss (very end of forward propagation) to beginning update the parameters [21]. Then, gradient descent of parameters w and b is updated to use as input in next forward propagation calculation until logistic regression converges optima.

3.3 Deep Neural Network Basics

In the context of neural network above, forward and backward propagation is defined with a shallow network architecture as well as one of the learning algorithms which is logistic regression. While logistic regression is a shallow model, a neural network with five-hidden-layer is a deeper model. Both can be visualized as in Figure 3.8 and 3.9 respectively.

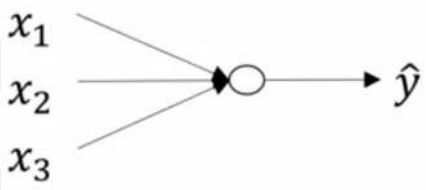

Figure 3.8: Logistic regression model is a shallow network that has one layer.

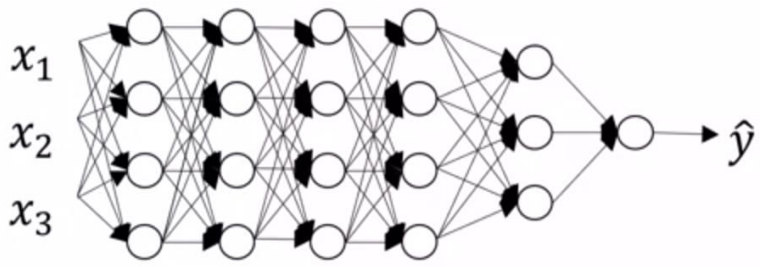

Figure 3.9: Neural network with 5 hidden layers is a deeper neural network.

As number of layer increases, neural network becomes deeper so it is called deep neural network. DNNs can solve complex problems whereas shallow neural networks cannot. It is challenging and not a rule-based method to decide how many layers to use at the beginning when starting to solve a problem. Thus, it is reasonable to start with training of logistic regression, afterwards, try one, two and deeper architecture until achieving desired result. Thus, number of hidden layers is a hyperparameter.

In developing DNN, not only parameters must be organized well, but also hyperparameters. While parameters are w and b, hyperparameters are used in learning algorithms, such as learning rate, number of iterations, number of hidden layers, number of hidden units, batch size and choice of activation functions. These are parameters, which are controlled by human, that modify and update w and b. These can be thought as setting parameters when training a DNN. Setting hyperparameters is an empirical process. For example, learning rate for a training can be set as 0.001. After trying this idea by coding and experiment, result may not be sufficient or successful. Thus, learning rate can be set with different value (e.g. 0.007) for next trial until achieving desired result [10, 23].

DNN is spreading because it achieves high accuracy and speed in solving problems, especially object detection. Before explaining object detection method in next chapter, which is implemented in this study, it would be better to provide some intuition and information about what CNN and computer vision are.

3.4 Convolutional Neural Network for Computer Vision

Computer vision is advancing rapidly by virtue of deep learning. This method is the key for smart vehicles, which are self-driving cars and autonomous aerial vehicles, to figure out where the other objects around and what they are. Thus, obstacle avoidance as a safety function helps preventing collision or reduces the severity of imminent crash by mitigating speed or changing direction of the vehicle.

To avoid collision and surveying particular object with a UAV, object detection is a problem to be solved in many aspects in computer vision. These are generally speed and accuracy of object detection algorithm working on SoC as a challenging trade-off. These will be explained in next chapter that presents object detection method in this study.

Moreover, there is another big challenge in computer vision problems that input size may get really large. Working with large images not only consumes high processing power, also causes the algorithm to run longer since an image may have millions of many pixels to be processed. In this regard, large images are not feasible to work with since computation load may inflate memory. That does not satisfy the computing requirement to training neural networks for real-time applications. On the other hand, working with tiny images is not sufficient as well because small images have not

enough data for extracting features by DNN. The solution is to implement the convolution operation [21].

Convolution is a function derived from two given functions by integration which expresses how the feature is modified. Three important requirements are needed for convolution operation which they are input image, feature detector and feature map.

To understand convolution operation, a simple edge detection operation is explained. For example, to detect vertical edges in Figure 3.13a which is a grayscale image, a simple convolution operation, Figure 3.10, can be explained with that a 5-by-5 matrix of grayscale image is convolved with a 3-by-3 filter, also called as kernel.

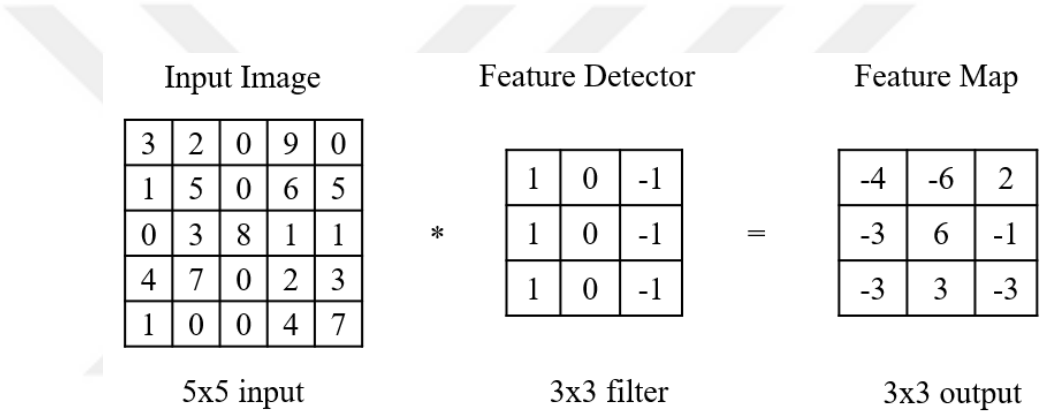

Figure 3.10: 5x5 input image is convolved with a 3-by-3 vertical edge detector filter.

3x3 output presents the value of vertical edge feature in the input image. As for how filters convolve the image to extract its feature map, following steps apply:

1. Filter is placed over the input matrix beginning from top-left corner so that they overlap each other as 3-by-3, as in Figure 3.11a. Afterwards, matched elements are multiplied (dot product), then multiplication results are summed and the output is written as a first element of feature map matrix starting from top-left, shown in Figure 3.11b.

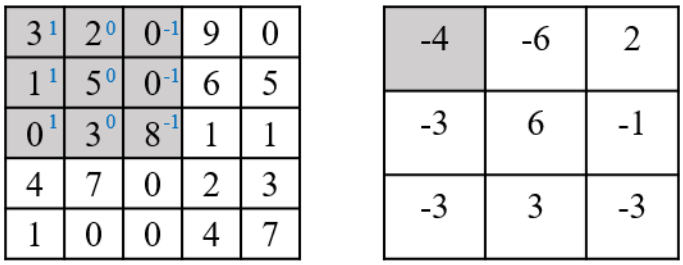

Figure 3. 11: (a) Convolution starting from top-left corner. (b) First extracted value of feature map of vertical edge detection is -4.

2. Filter is moved one column to the right and the same calculation is repeated, visualized in Figure 12.

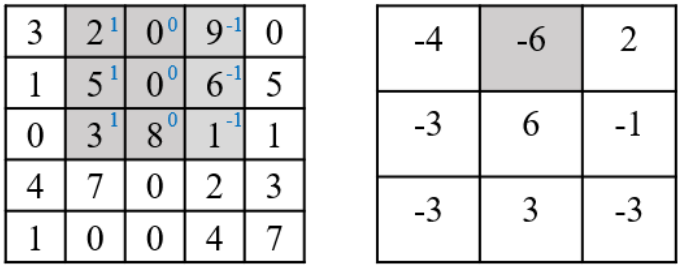

Figure 3.12: (a) Filter is shifted one column to right. (b) Second element of feature map is calculated with the same method in step 1, as -6.

3. This iteration continues until the rightmost column of input matrix. When it reaches the right-end of input matrix, filter is moved one row down and convolution starts from first column-second row, in this case from 2-by-1. This operation repeatedly continues until the last element of feature map is calculated.

Movement of filter is called as stride which is a hyperparameter that controls size of the output. Thus, stride is 1 in this example. That means filter is moved one pixel at a time. If stride were 2, the filter would jump two pixels at a time. This would produce smaller output volumes, and less computation power [25].

As another important hyperparameter, padding the input volume with zeros around the border may be useful according to case. That helps controlling the spatial size of output volumes. This is generally used for preserving the input size so that the input and output size are the same.

Figure 3.13: (a) Input image for vertical edge detection. (b) Vertical edges are detected with convolution operation [21].

In addition, it is common to use pooling layers for reducing the size of input images, speed up the computation and reduces number of parameters in network to learn. There are different pooling types which are general pooling, average pooling and max pooling. Max pooling has more advantages compared to others since it assesses the maximum, or largest, value in each region of feature map. Pooled feature maps present maximum feature in the region [21, 26]. For example, when pooling a 4-by-4 input matrix with a 2-by-2 filter and stride is 2, input matrix is divided into 4 regions so that the maximum value of each region is taken into calculation. That results a 2-by-2 output matrix including only the dominant value (maximum) while down sampling, illustrated in Figure 3.14.

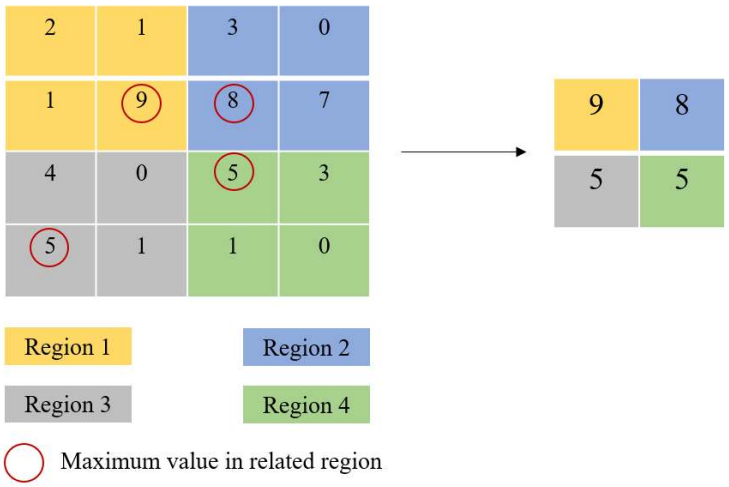

Figure 3.14: Max pooling reduces the input size from 4x4 to 2x2

CHAPTER 4

OBJECT and OBSTACLE DETECTION

4.1 Introduction

Navigating in unknown environment is of paramount importance for mobile robots. Solving such kind of challenge, by using sensors and computer vision techniques, offers environmental awareness capability to mobile robots in many applications. The scope of this chapter is to explain both a fast running object detection with DNN and obstacle detection with monocular SLAM. Those provide the information of objects' location, class and obstacles.

Collision with objects such as people, cars and so on are avoided by virtue of object detection. In this study, not only an open source and widely used DNN algorithm is used, but also it is accelerated for running whole system in real-time. That turns out low run time. Therefore, while object detection is running slow and inflating the memory causing system failure in NVIDIA's project, using accelerated DNN offers much more reliability in detection of objects for real-time flight applications [3, 26]. On the other hand, monocular SLAM contributes to navigation task by computing semi-dense 3D maps of environment.

4.2 Object Detection

In computer vision, object detection is a method that determines where recognized objects are in an image. Every object is located with their classes, such as person, dog and car in front of the vehicle. In order to understand object detection algorithm, it is extremely important to know about object classification and localization.

4.2.1 Object Classification

Classification, also known as object recognition, is a method to figure out the type of object is in frame. It is used to distinguish object classes from each other. For example, autonomous cars need to classify objects because braking conditions are different regarding the object type such as pedestrian, car, truck and bicyclist. Thus,

classification is a process of taking an input image and outputting a class with a probability value (i.e. 98% sports ball, 100% dog). Figure 4.2 shows recognized objects in the image.

Figure 4.1: Image is classified with YOLOv3 to define object labels in image so that sports ball and dog are recognized with probability values.

4.2.2 Object Classification with Localization

Object detection algorithms not only label the object with its class, also localize the object in the image. Localization refers to estimating the position of the recognized object in the image. Position of an object is visualized by drawing a predicted bounding box around the recognized object. In particular, detection algorithm outputs four numbers as attribute of bounding box for localization as well as value of class label. These four numbers are b_x, b_y, b_h, b_w which compose bounding box to be drawn around the detected object as visualized in Figure 4.2. The attributes of bounding box are described as:

- (b_x, b_y) : center of bounding box
- b_w: width of bounding box
- b_h: height of bounding box.

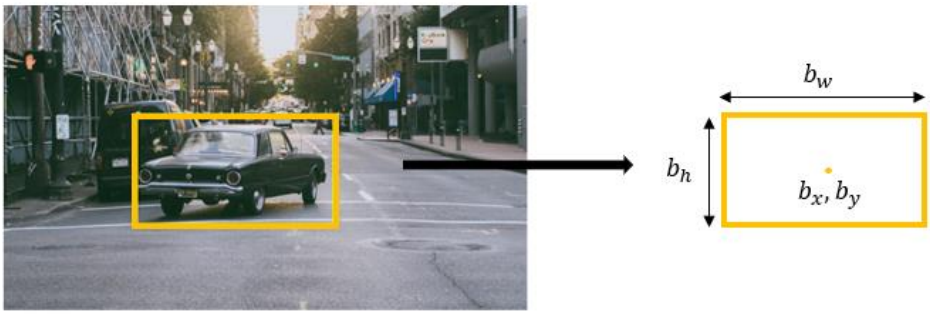

Figure 4.2: Object is localized by predicting the bounding box.

Consequently, object detection consists of localization and classification of multiple objects. In other words, classification and classification with localization concentrates on single object whereas object detection classifies and localize multiple objects in an image [10].

4.2.3 Object Detection Algorithms

In this study, a few open source object detection algorithms are compared for using in UAV system. There are diverse types of object detection algorithms and they can be grouped according to working principle as follows:

- Based on classification: these algorithms work in two stages. At first stage, regions of interest (ROI) is selected. Afterwards, these regions are classified using CNN. This method is slow because it is required to apply prediction for every selected ROI. RCNN and its faster versions, which are Fast-RCNN and Faster-RCNN, are some of the most known region proposal algorithms [9, 27].
- Based on regression: This method does not look for ROI, but whole image to predict classes and bounding boxes at a one stage. That means these algorithms are defined as single-shot and fast algorithms, such as YOLO and SSD [10, 11].

Two of the most used algorithms are YOLO and SSD among many others, for example RetinaNet, DetectNet and Faster R-CNN [28, 29]. Almost every algorithm has faster and/or more accurate versions such as MobileNet-YOLO, YOLOv3, MobileNet-SSD and SSD-TensorRT [30]. As a base algorithm, YOLOv3, which is 3rd version of YOLO, is used in this project because it is a unified algorithm for classification and localization of multiple objects, so object detection. Algorithm speed and accuracy in terms of mean-Average-Precision (mAP) are different for each architecture. YOLOv3 is fast and highly accurate compared to others in general since a single network predicts classes and bounding boxes from whole image in one evaluation [10, 21]. Moreover, Darknet is a framework which is written in C language and CUDA-which is the core technology underlying GPU technology of NVIDIA [31]. Thus, Darknet implementation allows YOLOv3 to make computations on a GPU, which runs it faster as well. Figure 4.3 shows mAP and inference time comparison between YOLOv3 and some other most used algorithms.

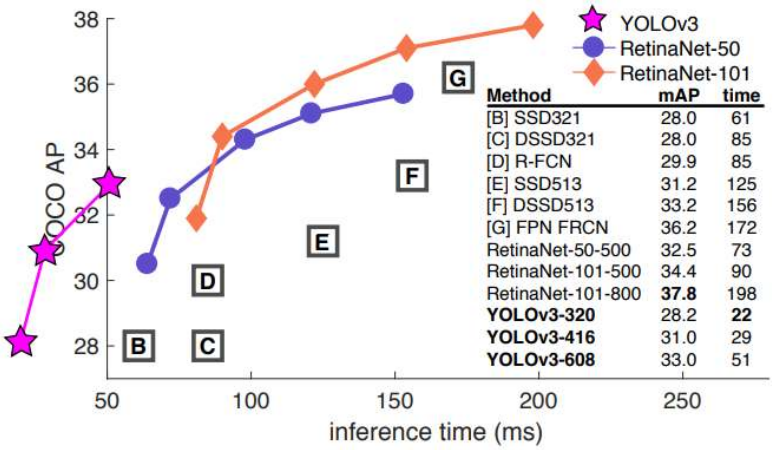

Figure 4.3: Comparison of YOLOv3 with some other object algorithms [10].

4.2.4 The Theory Underlying YOLOv3

YOLO and its updated versions are almost the most used algorithms by open source community, start-up companies, researchers and developers because it is quite fast and accurate. Network of YOLOv3 detects objects at one evaluation in contrast to sliding-window-based algorithms. Before explaining how YOLOv3 is used in this project, understanding the theory behind it is essential. Next sub-sections are explaining the basics of YOLO algorithm.

4.2.4.1 Bounding Box Prediction

This algorithm offers a good method to predict accurate bounding boxes in detecting objects. As mentioned previously, YOLO does not look for ROI but entire image at once, thanks to convolutional neural networks. Thus, generally 19-by-19 grid is placed on image to split. However, this size is not a must since grid size depends on use case. For simplicity of illustration, 3-by-3 grid is placed on image for explaining bounding box predictions, as illustrated in Figure 4.4. Object localization and classification with YOLO algorithm, so object detection, are applied to each of the nine cells of grid in the image for training [10, 21, 23].

Figure 4.4: 9-by-9 grid is placed on image to apply detection algorithm once.

Label, y, is specified for each of grid cells. For each, label y consists of:

$$y = (p_c, b_x, b_y, b_h, b_w, c) \tag{8}$$

p_c is predicted value which gives probability of that if there is an object associated with that grid cell or not. In other words, it outputs a probability value according to existence of an object. b_x, b_y, b_h, b_w are components of bounding box which are already explained previously. As for the last value of label, c stands for recognized object class in that grid cell if exists, such as pedestrian and truck. To provide better understanding, as for first grid cell which is shown with red rectangle in Figure 4.5, y label turns out zero for p_c and algorithm does not care about the rest:

$$y = (0, -, -, -, -, -) \tag{9}$$

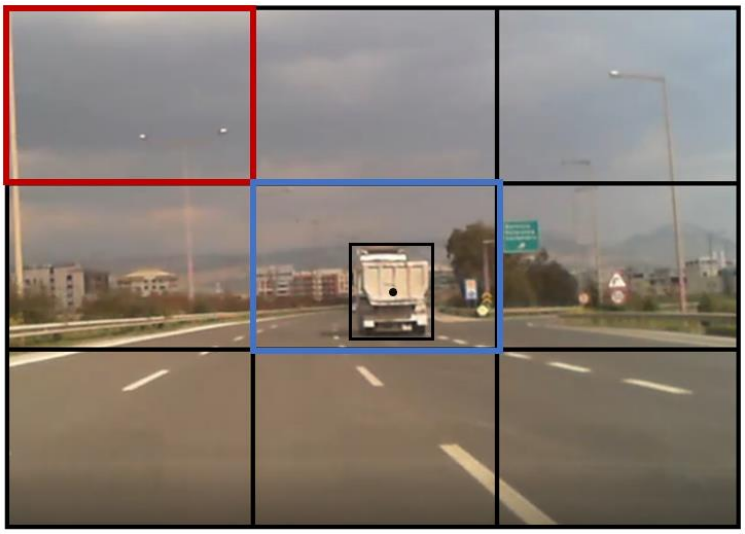

Figure 4.5: 9-by-9 grid is placed on image to apply detection algorithm for each cell.

Therefore, the other grid cells which don't include any object output label y as same as first (red) grid cell does. On the other hand, YOLO takes midpoint of object and assigns the recognized object to associated grid cell. In this regard, fifth grid cell with blue rectangle, shown in Figure 4.5 above, has an object which is a truck. (assume that algorithm is trained for one class, in this case for truck). Then, label y turns out:

$$y = (1, b_x, b_y, b_h, b_w, 1) \tag{10}$$

As for specifying the bounding box in Figure 4.6, center (b_x, b_y) is calculated according to the top-left corner of the associated grid cell starting from $(0,0)$ and bottom-right corner ending at $(1,1)$. Thus, b_x and b_y are between 0 and 1. On the other hand, b_h and b_w are specified as fraction of the overall height and width of associated grid cell, respectively. They can be bigger than 1 since object size may exceed the associated grid cell.

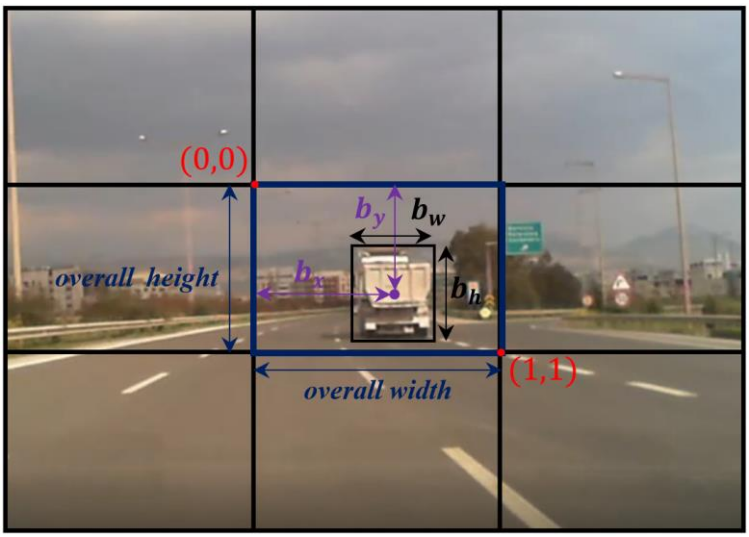

Figure 4.6: Specifying center, width and height of bounding box.

4.2.4.2 Intersection over Union (IoU)

IoU function is used to check that object detection algorithm runs well. Figure 4.7 is a good example to explain IoU function. YOLO may predict bounding box as orange box in Figure 4.7 although ground-truth bounding box is the blue one. The more overlapped area of ground-truth and predicted bounding box, the more accurate result is predicted. However, these bounding boxes are not always overlapped perfectly. Thus, a threshold value is defined to accept that the prediction is correct. Threshold for IoU is generally taken as 0.5. That is to say, if IoU is equal to or greater than 0.5, prediction can be judged as correct. In contrast, if IoU is lower than 0.5, then the predicted bounding box is ignored. Last but not the least, 0.5 as threshold value is not a must since it is a convention so that it can be defined higher when more accurate result is needed for strict use cases.

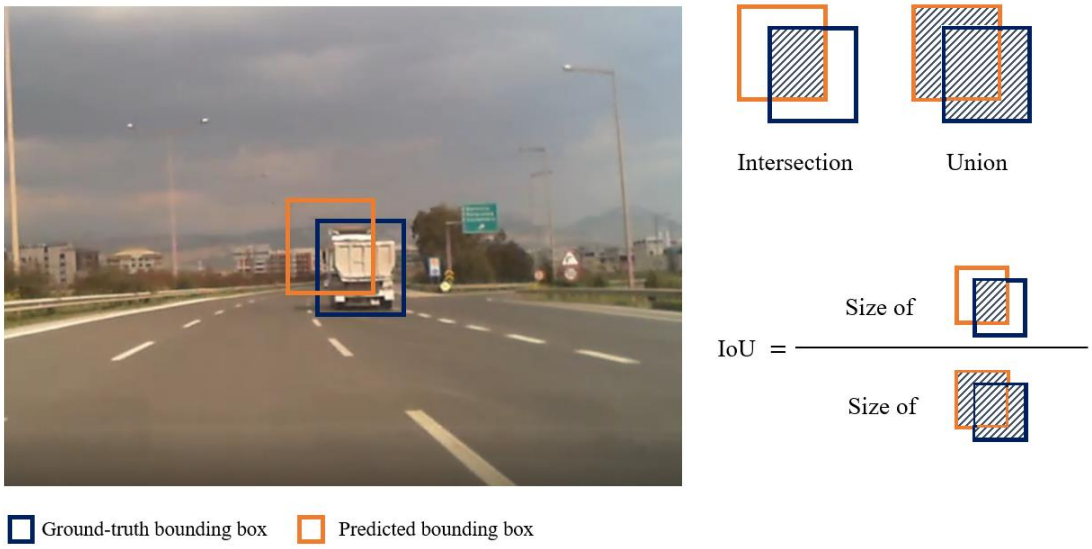

Figure 4.7: Intersection over Union function is used to predict more accurate results.

4.2.4.3 Non-max Suppression (NMS)

YOLO or any other algorithm may detect the same object multiple time so that this is undesirable situation. Non-max suppression helps the algorithm detect each object only once, not more. That is to say, it is used to remove multiple detections for one object. Figure 4.8(a) shows multiple detections, in this case 3 times, with probability values. This is undesired situation since there is only one object in the image. Thus, after applying NMS, Figure 4.8(b) shows desired output of YOLOv3 which turns out only one detection for one object. As can be understand from the name convention of Non-max suppression, bounding boxes with low possibilities are suppressed. In other words, maximum probability value, $p_c = 0.9$, determines the most confident bounding box associated with the detected object.

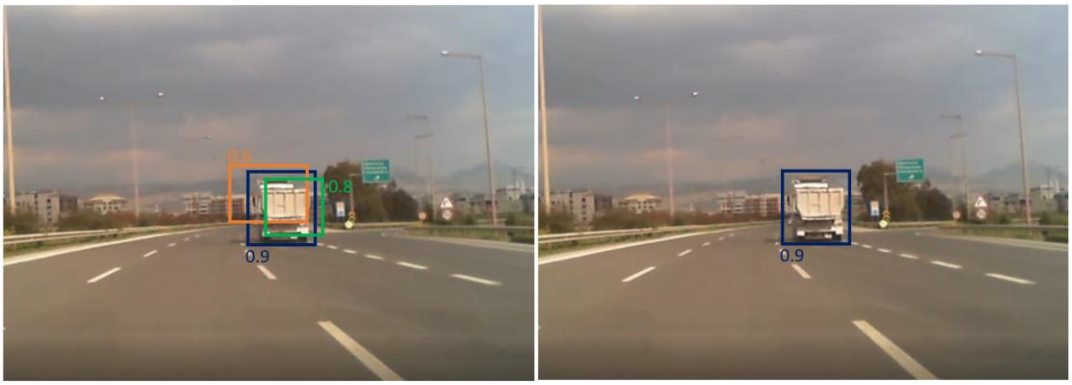

Figure 4.8: (a) Before Non-max Suppression applied, there are multiple detections per one object. (b) After Non-max Suppression applied, there is only one bounding box with maximum probability per one object.

Algorithm of non-max suppression applies as below [21]:

- Discard all bounding boxes with that predicted p_c is lower than the specified threshold (i.e. when $p_c \leq 0.5$).
- While there still exist any remaining boxes;
    - Select the bounding box with the highest p_c.
    - Discard any remaining box if IoU is equal to or greater than the selected bounding box with the highest p_c in previous step.

4.2.4.4 Anchor Boxes

It is explained to detect one object per one grid cell until here. When the case turns out to detect multiple objects per one grid cell in an image, idea of anchor boxes is to help to do that. Thus, anchor boxes are used to detect multiple objects which are overlapped in a one grid cell. Anchor boxes are predefined bounding boxes with a certain height and width. These boxes are tiled across the image to capture the scale of specific object classes to be detected. Size of anchor boxes is calculated, according to the shape and size of objects in training set, by using K-means algorithm. There can be a few anchor boxes in algorithm to detect multiple objects. If there are three object classes to be detected, there might be 3 or more anchor boxes according to the desired accuracy of predicting bounding boxes. For example, Figure 4.9 illustrates pre-defined anchor boxes to predict bounding boxes for both person and car. While size of anchor box 1 is chosen as close as to the human shape and size, anchor box 2 is chosen with respect to the shape and size of car [19]. As can be seen from the Figure 4.9, center point of

objects is in the same grid cell so that they are overlapped. To detect both overlapped objects, anchor boxes are defined before detecting bounding boxes directly. The more grid cell is placed across image, the less overlapped objects exist. Nevertheless, there may be overlapped objects in the same grid cell although size of grid is huge, for example 19-by-19. However, when the issue appears, anchor boxes are used to get rid of overlapping issue so that detection is achieved.

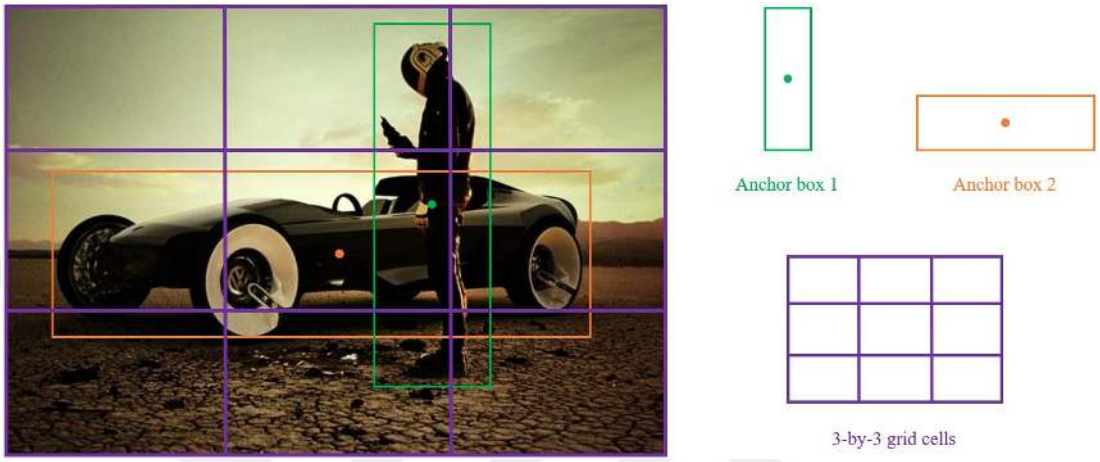

Figure 4.9: Anchor boxes are calculated with K-means algorithm regarding the size of objects in training set so that prediction accuracy can be improved.

Notice that if there are a greater number of object classes than number of anchor boxes in the same grid cell, algorithm does not work well. This is important to not to forget. In addition to this disadvantage of anchor box concept, if there are two objects in the same grid cell and both of them has the same shape of anchor box, algorithm will not achieve to detect successfully. Last but not the least, defining anchor boxes is one of the most important hyperparameters to be considered since it affects the accuracy of YOLO. As a final statement, when training a custom dataset including different object classes than default one for YOLO, which is PASCAL VOC dataset, anchor boxes must be recalculated from scratch according to the shape of objects in a custom dataset [12].

4.2.5 Object Detection Implementation with YOLOv3

Implementation of YOLOv3 is a challenging process since it consists of many software components and requirements not only for running, but also for preparing dataset, fine-tuning and training. Moreover, the biggest struggle is to achieve high speed model without losing accuracy of the model. Trade-off between speed and accuracy is still

debated among community researchers and developers for a long time, however, there is not a well-defined and strict solution to overcome this problem. To provide better understanding, for example, if a DNN model runs with fp32 precision instead of fp16, this model outputs more accurate results while it runs with lower speed since high precision models have more computational load.

4.2.5.1 Preparing Dataset

The more data, the more accurate results that DNN models output. Data is the bone of deep learning operations since models are generated by training DNN on dataset. A DNN model can detect a person in an image with accuracy of 50% or 78%. That success depends on, but not limited to, number and diversity of data as well as organizing dataset carefully. For example, if YOLOv3 or any other algorithm is trained on a dataset which is collected on road during summer so that this dataset does not include any information about snow, wet road, person with sweatshirt and any other information associated with winter. Therefore, the trained model does not achieve good result in winter. That is to say, this model most likely fails to detect a car which is partially occluded with snow. Lack of information in a dataset outputs model with low accuracy. Therefore, dataset should be chosen and/or collected carefully according to use cases. Importance of data as a performance metric is generally illustrated as Figure 4.10.

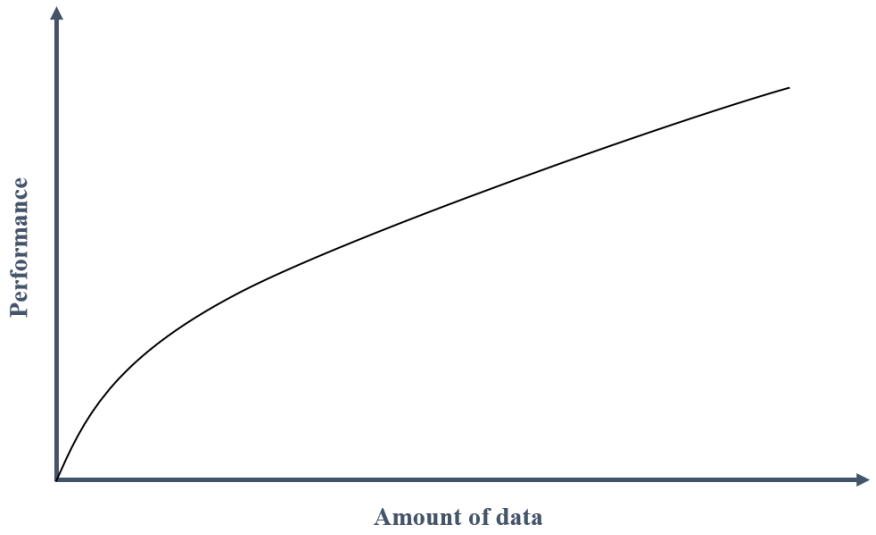

Figure 4.10: Data is input for algorithm so that it affects performance extremely.

There are many dataset including different classes which may be person, car, dog, pencil, cancer cells and so on [13, 32]. Use cases are the main consideration for

deciding which dataset, so classes, is required. In this project, PASCAL VOC dataset is used to train YOLOv3 [12]. It consists of 20 classes such as person, animal (cat, dog, horse, sheep), car, bicycle and so on. Although YOLOv3 is configured and trained according to various dataset, for example COCO and KITTI, it is originally trained on ImageNet and PASCAL VOC dataset. A UAV system mostly encounters person, animals and cars. Moreover, this project is intended to use in surveillance and search-and-rescue operations so that UAV must detect person in particular. Thus, PASCAL VOC dataset is suitable for the use cases of this project. Since more data results better performance in detecting an object, more person images are added into person class as well as annotations. Thus, COCO dataset is used for extra data [13].

Annotation is unavoidable component of a dataset because it defines the ground-truth of objects. If one desires to generate custom dataset, labeling a ton of images is not only the most time-consuming process, but also requires high attention. Fortunately, there are open source labeling tools for generating annotations [23, 33]. Figure 4.11 and Figure 4.12 show how labeling is implemented with LabelImg tool and output files [33]. Label defines the coordinates of ground-truth bounding box associated with the object class in an image so that after drawing rectangle, class is defined according to the object class. Thus, there is created an annotation file in txt format. This is the official source for generating custom dataset and training it with YOLOv3. There are other file formats which are json, csv and xml for other dataset and algorithms. This file includes coordinates of each object and their class with respect to the image. YOLOv3 requires txt format [23].

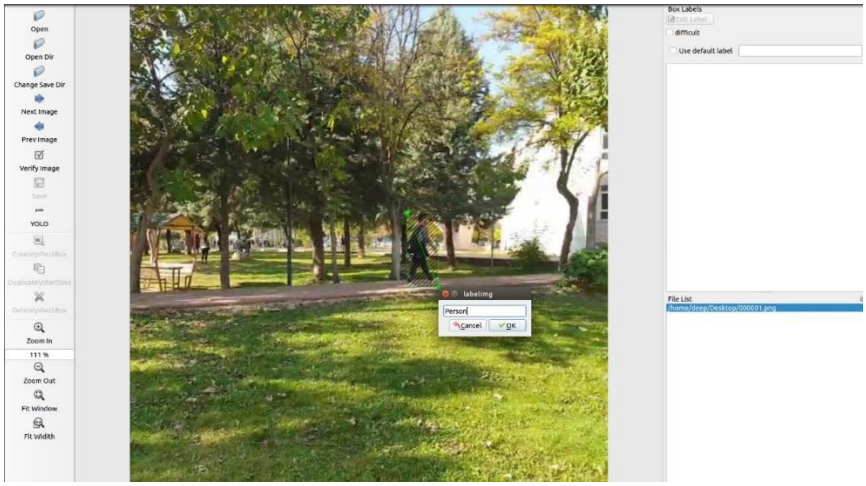

Figure 4.11: LabelImg is annotation tool to label objects.

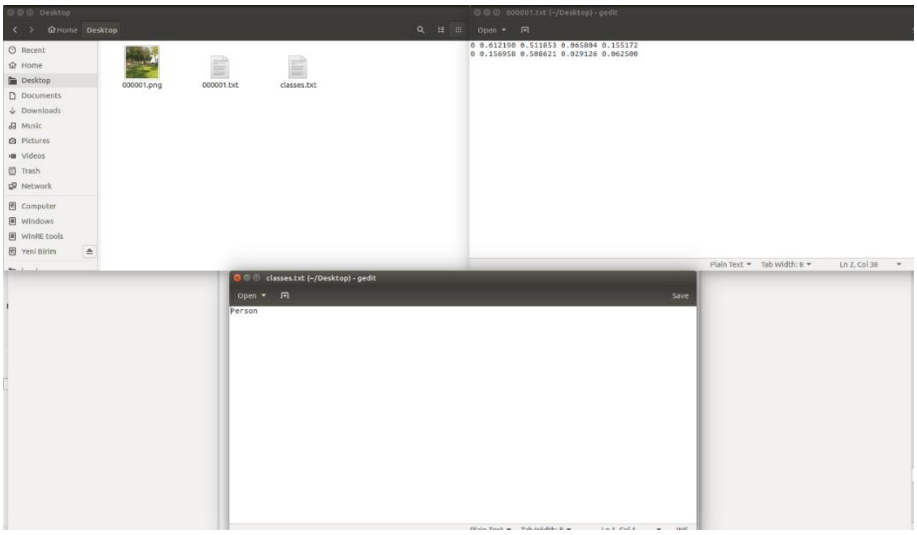

Figure 4.12: Extracted files of images in YOLO format.

Folder hierarchy for YOLOv3 is illustrated with Figure 4.13.

```
Main Folder
--- data
    --- dataset name
        --- images
            --- img1.jpg
            --- img2.jpg
            ..........
        --- labels
            --- img1.txt
            --- img2.txt
            ..........
        --- train.txt
        --- val.txt
```

Figure 4.13: Dataset structure for YOLOv3

Therefore, label file consists of individual txt file for each image in the training dataset according to YOLO format. Moreover, each txt file consists of y label for each object class in the image as:

$$(c, b_x, b_y, b_h, b_w) \tag{11}$$

```
1 0.716797 0.395833 0.216406 0.147222
0 0.687109 0.379167 0.255469 0.158333
1 0.420312 0.395833 0.140625 0.166667
```

The file *train.txt* has name of each image with its exact path in the computer system. Each image keeps one line in the file as follows:

/home/deep/darknet/data/dataset_name/images/image1.jpg

/home/deep/darknet/data/dataset_name/images/image2.jpg

/home/deep/darknet/data/dataset_name/images/image3.jpg

The structure of *val.txt* file is also the same as *train.txt*. Validation images are used for validation purpose only, not for training. They are used by algorithm per predefined epoch number.

Finally, dataset is prepared by implementing following steps:

1. Person class with training images and annotations of COCO dataset are downloaded.
2. Label format is converted to YOLO format for each label.
3. Dataset structure as Figure 4.13 is built manually.

4.2.5.2 Training

After dataset is prepared, there are other files to be modified for training. These files are configuration file (i.e. yolov3.cfg), class name file (i.e. coco.names) and data file (i.e. coco.data). *yolov3.cfg* file is responsible to, but not limited to, define number of classes, batch size, learning rate, size of anchor boxes, padding size, activation functions and so on. Configuration file is modified according to the official guide of darknet.

Class name file consist of only class names in dataset. In this project, 20 classes of PASCAL dataset are used by adding more person data from COCO into person class of PASCAL. Class name file, like *coco.names,* consists of 20 classes name for each line as Figure 4.14:

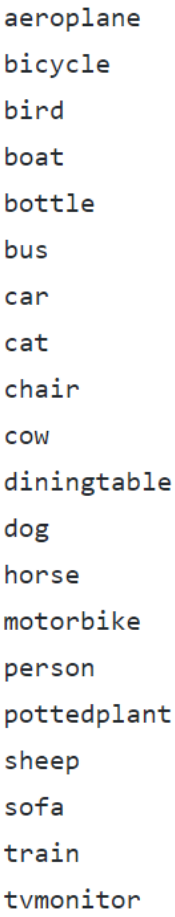

Figure 4.14: Screenshot of class.names file which is the same as the name file of PASCAL VOC.

Lastly, data file with *.data* extension defines the path of required files for training and validation such as train.txt, val.txt and name file (class.names) as Figure 4.15:

```
classes= 20
train = /home/deep/darknet/kitti-voc/train.txt
valid = /home/deep/darknet/data/kitti-voc/test.txt
names = data/class.names
backup = /home/deep/backup/
```

Figure 4.15: Screenshot of data file that defines path.

After configuring the essential files and the directory structure, the pretrained model of YOLOv3 on ImageNet dataset is downloaded to train YOLOv3 on this model [23]. Thus, object detection algorithm is not trained from scratch because a huge number of data would be collected to train YOLOv3 scratch. On the other hand, fine-tuning is a good option to train a pretrained model when data is limited. ImageNet consists of

nearly 15 million of images so that it would not be possible to collect and label more image during this project. In this regard, official pretrained model is trained with 8000 iterations on a computer with GTX 1070 8 Gb RAM. Training is implemented according to the official repository and other developers' repository in Github community [23, 34]. The training took almost 10 hours. Figure 4.16 shows loss during training. It is extremely important to decrease loss as much as possible. To decide when the training should be stopped, there is not a strict answer. However, community agrees on that training may be stopped as soon as loss is not decreasing much more. Finally, a trained model is obtained to optimize with TensorRT and deploy the optimized model in JetsonTX2. Training and inference results from drone camera will be provided and discussed in conclusion chapter.

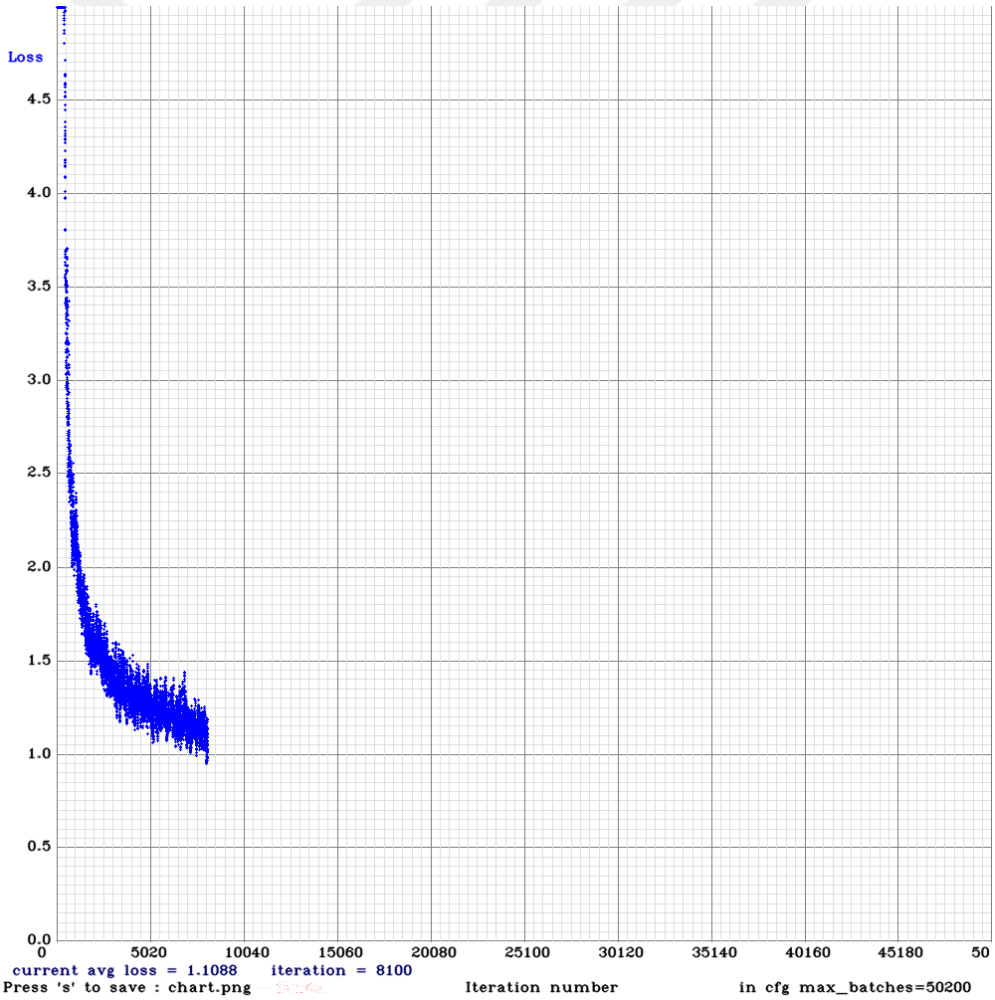

Figure 4.16: Loss is calculated per iteration during training

4.2.5.3 GPU and Speed

GPU processors are crucial in running DNN applications since they have parallel computing capability which makes them faster than CPUs working as serial. GPU as a hardware is also critical for training, as well as runtime of detection algorithm because training of a DNN model is an expensive process. Because model training is carried out on huge dataset which may consist of thousands or even millions of images. Although CPU can be used for training, it can take months to train a model. Because of the parallel computing capability of GPUs, training time is reduced from months to days. For training YOLOv3, NVIDIA GTX GeForce 1070 GPU is used in this project.

Although speed of YOLOv3 is quite well on advanced GPUs, such as TitanXp and GTX 1080, YOLOv3 can process 2 to 4 fps on Jetson TX2, which that speed is not sufficient for real-time applications with UAVs. In addition to running only YOLOv3 on Jetson TX2, there are other programs running on the system for autonomous flight, for instance robot operating system (ROS). In this case, speed of object detection model reduces to 1-to-2 fps, sometimes system fails, because of shared memory of Jetson TX2. To cope with that, TensorRT is used as an optimizer which is developed by NVIDIA to accelerate GPU-based models. With TensorRT, YOLOv3 reaches 7 to 9 fps which is enough for the UAV that is developed in this project.

However, speed is not the only criterion in selecting favorable algorithm. For example, SSD-MobileNet architecture, that is optimized with TensorRT, is tested. It is much faster than optimized YOLOv3 with TensorRT. Also, optimized SSD-MobileNet with TensorRT can operate at 50 fps on Jetson TX2, which that makes it faster than selected detection method in this project [35]. However, SSD-MobileNet is not as accurate as YOLOv3. Accuracy is quite important because classification is crucial in case of surveillance or search-and-rescue operations.

4.2.5.4 Speed Optimization with TensorRT

Object detection algorithms work on images so that computational load is extremely heavy. Thus, running a DNN model, which is not optimized, on Jetson TX2 causes low performance in terms of speed. Thanks to TensorRT that speeds up CUDA-based DNN models to run. TensorRT is a C++ library that results high performance inference time on NVIDIA GPUs. TensorRT is not applied by developing a model since it is an optimization tool. It takes trained network including set of parameters and produces

an optimized runtime engine. Although TensorRT provides API, there are a few framework parsers that helps optimization process. ONNX is used to convert darknet weights to ONNX. It is neural network exchange format to move models between tools. For example, the model which is obtained from training of YOLO cannot be fed into TensorRT optimizer directly. Instead, trained model is converted to ONNX and then fed to the optimizer. Process flow is as Figure 4.17. Following steps providing reference source code to optimize trained YOLOv3 weights for faster inference on Jetson TX2 or any other platform which has NVIDIA GPU supporting CUDA architecture [26, 30, 34, 36].

1. YOLOv3 is trained on PASCAL and COCO dataset so that a trained model is obtained
2. Obtained model is converted to ONNX so that yolov3.onnx is generated [37].
3. Yolov3.onnx is processed with TensorRT engine so that yolov3.trt is generated

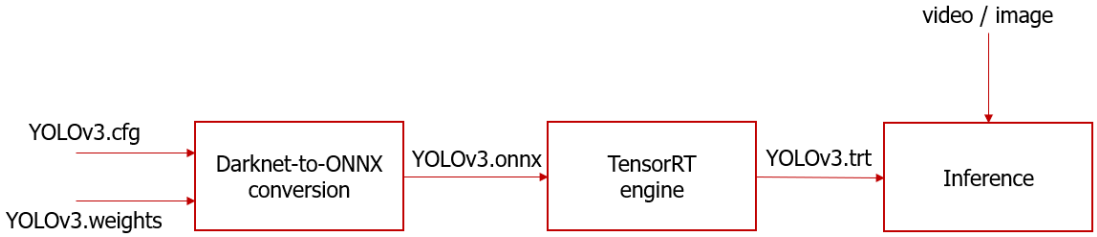

Figure 4.17: YOLOv3 optimization steps.

4.3 Obstacle Detection

In this project, monocular visual odometry technique, rather than stereo VO, is used to minimize system complexity. VO is used in robotics to determine position and orientation of robot by considering acquired sequential image data from camera so that travelled distance can be estimated. While traditional VO methods rely on visual information which is obtained with feature-based methods, on the other hand direct method uses pixel intensity in sequential image data directly as visual input [48]. Thus, direct sparse odometry method is used in this project [49]. Open source code of DSO is implemented on the developed UAV. To reach real-time processing in detecting obstacles, DSO feeds 30 fps to the system and updates keyframes at 3 Hz. Figure 4.18 shows four different scenes (from top to bottom) with both raw image and sparse depth map from DSO. Colors mean distance of detected point to camera. Red is close while blue is far. Color is changing according to the distance. For example, yellow is at a

distance between red and blue so that yellow represents middle range. Figure 4.18 represents dense sparse 3D map of associated environment around camera. On the other hand, Figure 4.19 illustrates trajectory of camera between start and end positions.

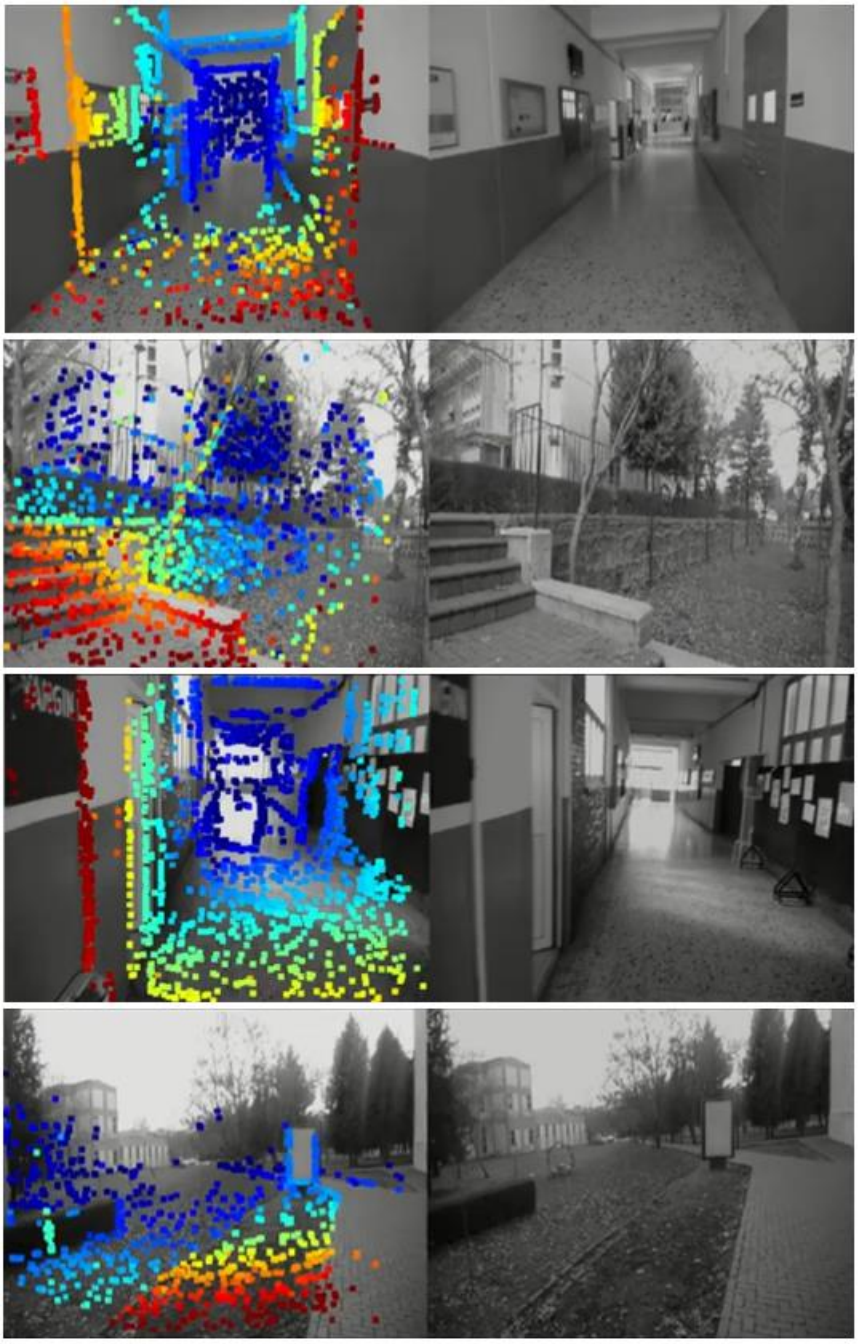

Figure 4.18: Images show sparse map of both indoor and outdoor environment.

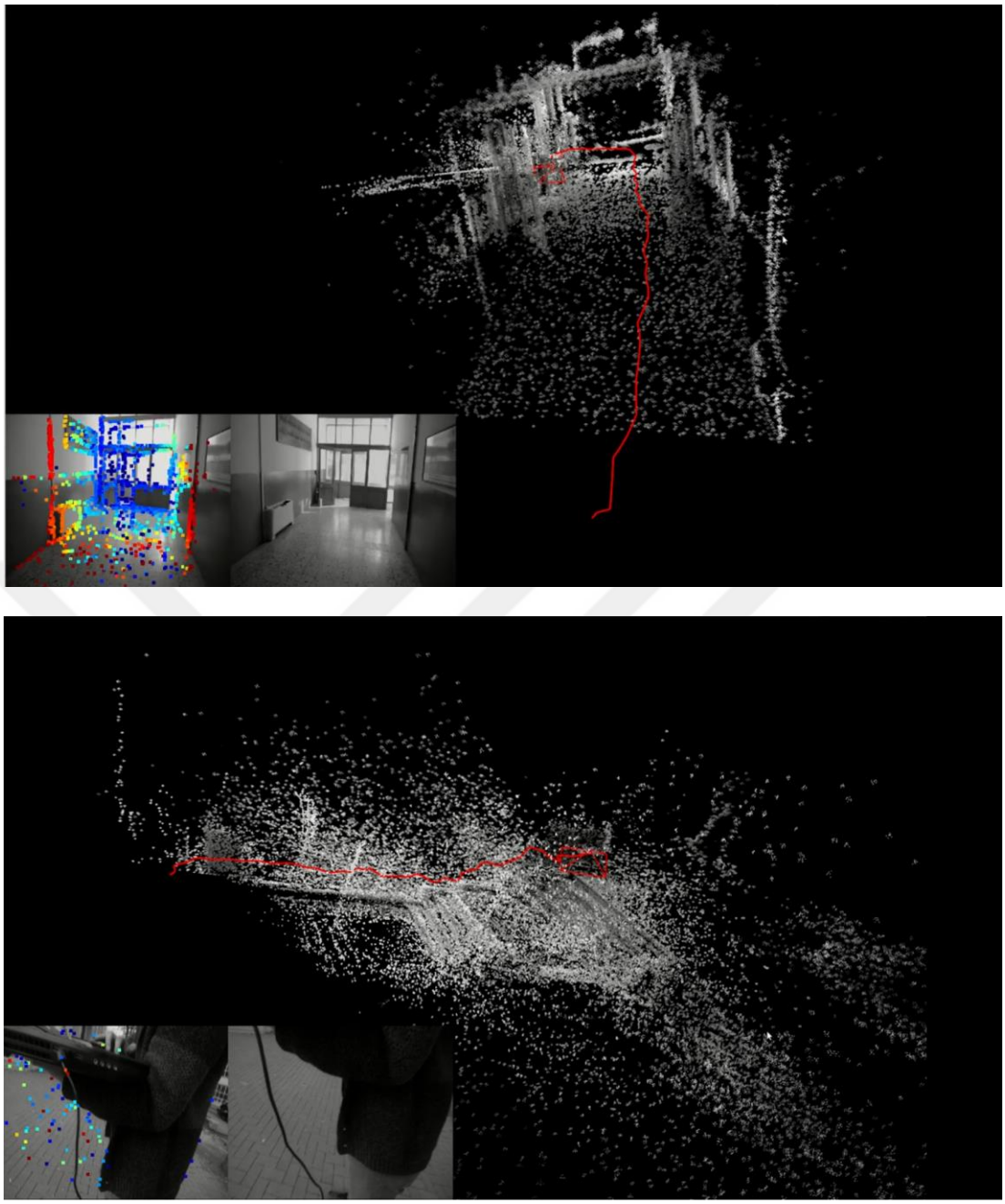

Figure 4.19: (a) 30 meters camera trajectory with red color through corridor of mechanical engineering laboratory (b) camera trajectory in front of Gaziantep University mechanical engineering laboratory.

CHAPTER 5
AUTONOMOUS UAV SYSTEM

5.1 Introduction

For autonomous flight, an open source project that is described in [3] is motivation source for navigating UAVs autonomously. This method is adapted to work on Jetson TX2 for faster computing capability purpose. Autonomous flight software module is developed for navigation among specified coordinates in case of that GPS signal is available for open fields, like mining area. To test the system, SITL method is carried out to prevent possible failures (i.e. crash) before flying the system in real world. This chapter explains how autonomous flight system is built. In general, software system of the UAV in this study consists of GPS-based autonomous flight and object detection for environmental awareness purpose.

5.2 System Description

To build a flexible quadcopter platform for autonomous applications according to the requirements, considerably inexpensive off-the-shelf hardware are used. This setup for building a quadcopter is explained in detail in next chapter. As for software system, PX4 flight stack is used as firmware in flight controller unit (FCU, which is Pixhawk). It provides communication link between Jetson TX2 and Pixhawk via MAVLink protocol [1]. Robot operating system (ROS) nodes run on Jetson TX2, which runs Ubuntu Linux4Tegra as operating system. ROS has GScam node for reading visual input from camera. Joy node is used for joystick. Keyboard package is used to allow receiving keyboard events. MAVROS is used as communication driver for autopilot that provides UDP MAVLink bridge to ground control station (GCS). As for vision process, object detection is applied within the DNN node to detect objects.

Smolyanskiy et al. have presented autonomous flight along trails in forest, but not autonomous take-off and landing capability. Autonomous take-off and landing capability are developed for having complete autonomous flight without manual

control from joystick. Also, in some scenarios, navigation path is known and it may not be a trail in forest. Thus, planned autonomous flight capability is developed. Although the system is completely autonomous in all discussed cases above, the pilot with RC controller has the priority in controlling of drone for safety reasons. Thus, a user can override the planned flight task by intervening the system with a single touch over the RC controller, joystick or ground station computer. System architecture is illustrated with Figure 5.1 in general.

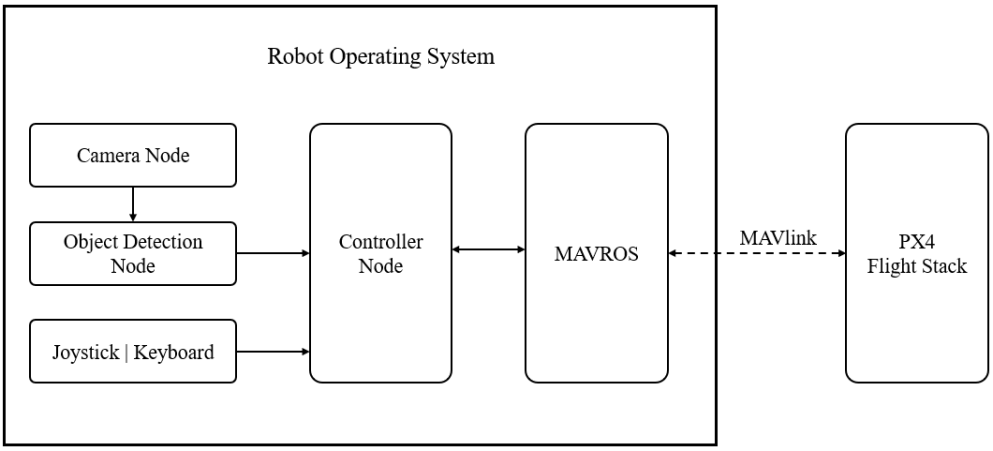

Figure 5.1: Key components of system architecture.

5.3 PX4 Autopilot Overview

Pixhawk is an open source flight controller unit that provides many capabilities for autonomous mobile robots, especially for aerial vehicles. Its source code is completely open to community who want to develop UAVs. The source code can be forked from the official repository [38]. PX4 source code is the autopilot of Pixhawk that contains two main layers which are flight stack and middleware. Flight stack is flight control system. On the other hand, middleware is a general robotics layer that supports any type of autonomous robots.

Flight stack, shown in Figure 5.2 as overview of building blocks, consists of navigation and control algorithms for autonomous drones. It provides an end-to-end pipeline for operating drones. Thus, this stack allows to use sensors, RC inputs, actuators (i.e. motor) for autonomous navigation.

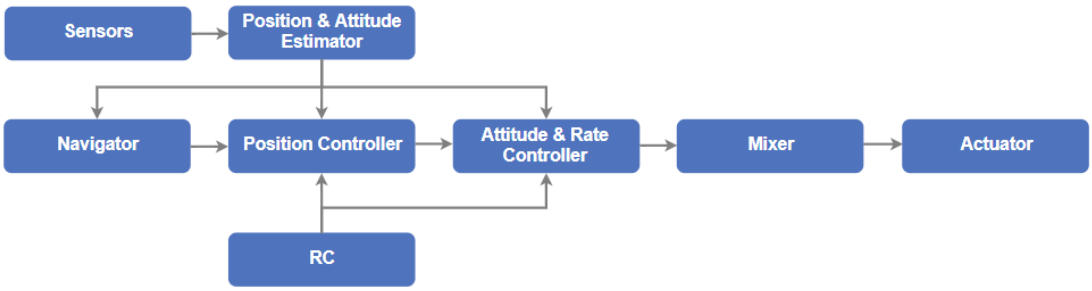

Figure 5.2: General overview of flight stack [39].

Pixhawk contains embedded sensors which are gyroscope, IMU, accelerometer and barometer. Estimator block of flight stack is responsible for taking sensor inputs to combine and/or fuse them for computing the quadcopter's current state (i.e. altitude). Controller is the component of flight stack that takes measurements and estimations as input so that it adjusts the value to match the position setpoints to move quadcopter towards the desired position. As for mixer, it takes turning commands (i.e. turn right) and translate them to understandable commands for motors. As for middleware, it contains drivers of embedded sensors on Pixhawk so that communication between sensors, companion computer and GCS can be achieved.

In short, PX4 flight stack receives information from companion computer, GCS and user with RC controller to move the quadcopter according to the system running within ROS environment. It also sends feedback to the system to correlate calculations.

5.4 ROS Description

Robot operating system is not an operation system in computers. Moreover, it requires operation system, such as Linux, to run. Essentially, ROS is used for providing communication between user, computer and sensors (i.e. Camera). One of the biggest advantages is that it provides flexibility. Thus, Python and C++ scripts can work together in different nodes simultaneously for moving a robot (i.e. quadcopter) [40,41].

In this project, entire system runs in ROS because there are different tasks to be run so that each script associated with another script needs to receive and/or deliver information each other. To do that, communication protocol of ROS is benefited. The architecture of a ROS system consists of five fundamental components which are ROS master, nodes, publishers, subscribers and topics. First, a *node* is executable file that uses a ROS client library, which allows nodes written in different programming languages such as Python and C++, to communicate each other. Second, *topic* is the

channel where nodes publish messages or are subscribed to read the published messages. ROS messages are sent and received between nodes to provide communication on topics. Third, a message which is transmitted by a node or topic is known as *publisher*. Fourth, a message which is received by a node or topic is known as *subscriber*. Lastly, the essential component among five is ROS *master* which is responsible for managing names and registration services to the nodes. If it is not started and run in the beginning, no matter what, ROS system does not work at all. The following Figure 5.3 shows a concise graphical structure to visualize how communication takes place throughout ROS [3, 4, 40, 41, 42].

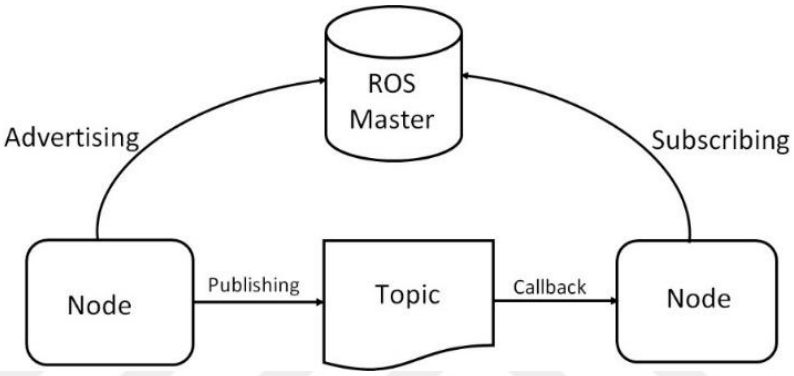

Figure 5.3: System components of a typical ROS model.

5.4.1 Camera Node

To run a webcam in ROS, *gscam* package is used as a camera driver that utilizes *gstreamer* to connect to camera devices [3, 42]. Thus, the camera node *gscam* publishes images to the */camera/image_raw* topic. Consequently, this node is the vision source of quadcopter which provides information (i.e. image) to the object detection node by sending standard image message in ROS [42]. This node enables camera feed to publish real-time video stream into ROS.

5.4.2 Object Detection Node

This node receives published video stream from camera node to perform object detection. Afterwards, object detection node publishes the output of the result. That is to say, YOLOv3 DNN processes each frame of the received video respectively so that it outputs detected object's class (label), probability, coordinates of the detected objects in x and y axes, and width and height of the object. For example, if a person (label is 16) and a car (label is 8) are detected, DNN node publishes an information as follows [3, 43]:

Label (class)	Probability	X	Y	Width	Height
16	0.8	180	50	32	48
8	0.7	250	140	110	60

Table 5.1: 6x2 output matrix is published from object detection node.

5.4.3 Joy Node and Keyboard Package

Joy is a ROS driver for using a generic Linux joystick. This package contains a node which is *joy_node* that is responsible enabling communication between ROS and a suitable joystick as a controller for simulation or real world. This node publishes the current state of joystick's buttons and axes as a ROS message. Microsoft Xbox Wired/Wireless and Logitech Wireless Gamepad F710 can be used as a generic joystick depending upon the user for Linux systems. Consequently, Xbox Wired joystick is used as SITL controller in this project. On the other hand, keyboard package is also implemented into ROS system because joystick may not be available every time. Thus, one can use just the keyboard of personal computer to control the robot, in this case quadcopter. Keyboard package publishes keyboard events to allow to receive presses, similar to *joy_node.* [42].

5.4.4 Controller Node

The controller node is responsible for conveying the desired actions and computation results according to the waypoints. Thus, this information is sent to the Mavros for controlling the quadcopter with PX4 flight stack in Pixhawk. This node runs at 20 Hz because system must achieve high speed for real-time controlling purpose.

To develop a completely autonomous flight, community uses px4 flight controller source code and modify it according to the requirements. In this project, px4_controller code of Redtail project is modified for two essential features which are the most important capabilities for autonomous quadcopters which are take-off and landing capabilities. In [3], quadcopter both takes-off manually before autonomous flight and land it manually after the flight. Their quadcopter flies autonomously but not fully. To develop complete autonomy, three new cases are added into the px4 flight stack for planned autonomous flight. The developed code which is incorporated into the open source code of px4 controller can be found author's repository on Github [24].

5.4.5 Mavros Package

As a driver, this package enables communication for autopilot systems using MAVLink communication protocol so that a companion computer (i.e. Jetson TX2) can send the required information to PX4 flight stack over the MAVLink communication protocol. It also provides UDP MAVLink bridge between GCS and PX4. In this regard, it is possible to control quadcopter remotely and/or configure flight parameters depending on flight cases. This package contains three main nodes which are *mavros_node, gcs_bridge* and *event_launcher* [42].

In this project, *mavros_node* is utilized as main communication node which is subscribed to *mavlink/to* topic for streaming MAVLink to PX4 autopilot by sending *mavros_msgs/Mavlink.msg* as message built-in Mavros. On the other hand, this node publishes *mavlink/from* topic for receiving *mavros_msgs/Mavlink.msg* stream from PX4 autopilot to companion computer. Last but not the least, *gcs_bridge* node is subscribed to and publish *mavlink/to* and *mavlink/from* topics respectively with *gcs_url* parameter to enable connection between GCS and PX4. This parameter is to define UDP link of GCS and PX4 as connection URL.

As for what Mavros utilizes in this project, movement commands (waypoints) and teleoperation commands from controller node are sent to *mavros_node* which submits the waypoints PX4 autopilot via MAVLink protocol. Thus, quadcopter operates autonomously according to the received take-off command, waypoints and landing command. Teleoperation command of user has the highest priority in controlling because of safety reasons in case of that system may fail and act strange.

5.5 Simulation with Gazebo

Flying drones without testing is dangerous and expensive in two aspects. First, some may be seriously injured in case of bug in the code. Thus, quadcopter may act strange and fly randomly towards someone or system may fail so that motors may stop during flight. Examples can be increased. Second, even if someone is not injured in order to any failure, drones are prone to break easily when crashes into obstacles or falls down on ground from high altitude. When considering these aspects, testing a model in simulation environment is essential to decrease the number of failures or eliminate them if possible. SITL method is widely used around the world because a developed

model can be easily tested and visualized in computer environment. This is not only safe but also fast to test the model.

In this regard, an open source simulation program which is Gazebo, among many others, is used [44]. It is user friendly because of its graphical user interface (GUI). Also, there are lots of documentation, support and discussion forums when help is needed.

ROS is compatible with Gazebo so that integration with PX4 is not challenging. Figure 5.4 shows general simulation environment. PX4 communicates with Gazebo to receive sensor data from the simulated environment and send computed values of actuators. Thus, the same scripts for running on quadcopter are tested in simulation environment.

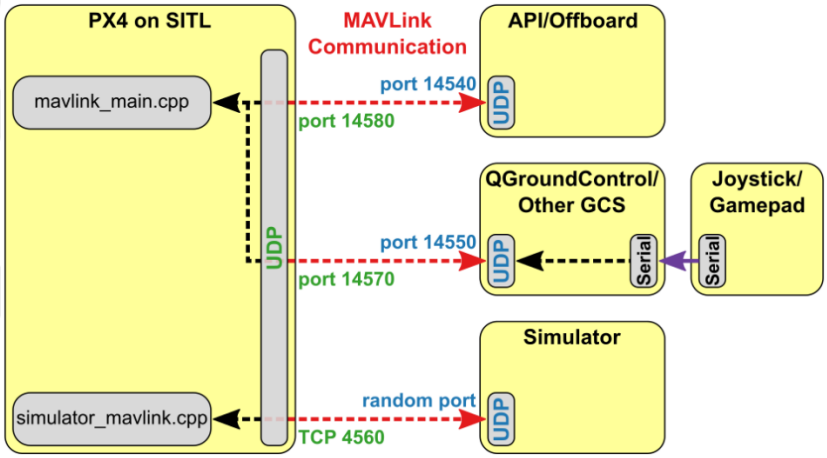

Figure 5.4: Followed pattern of PX4 when integrating simulation environment [39].

In Gazebo, a desired environment can be created according to the development requirements. Figure 5.5 shows the created an empty Gazebo environment including only a quadcopter model in this project. The quadcopter model is already built-in within Gazebo. Figure 5.6 shows what happens in terminal when controller node is started to test implementation. Figure 5.5 also shows the state which is reached altitude as 2 meters after running controller node.

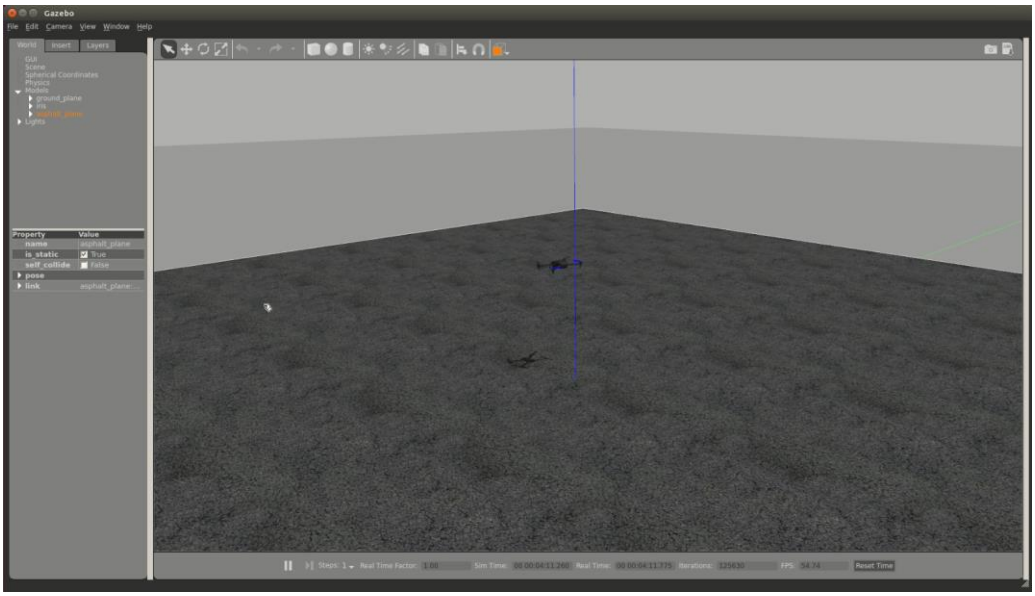

Figure 5.5: Empty simulation environment with a quadcopter within Gazebo.

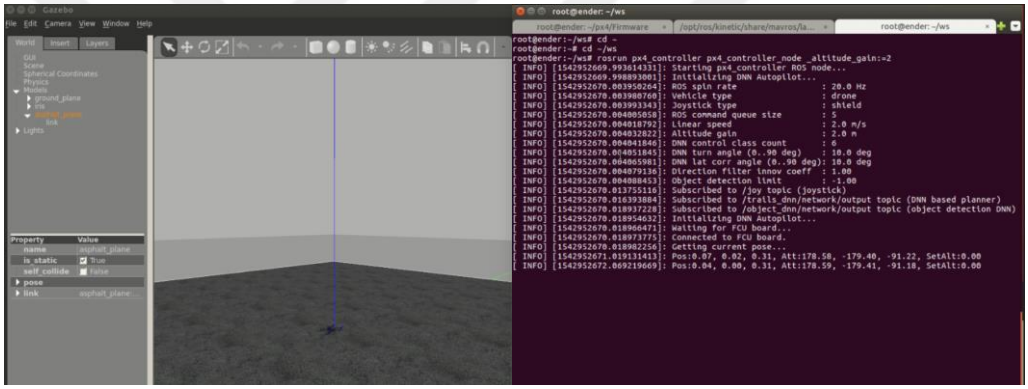

Figure 5.6: PX4 sends control commands to simulator and receives associated data to monitor status of the quadcopter.

5.6 Ground Control Station

GCS is control center allowing human to control UAV systems remotely. It is hardware setup that mainly consists of computer and telemetry. In this project, a personal computer is used as GCS hardware setup. On the other hand, to control quadcopter system, monitor the system status, update the firmware of PX4 software and calibrate embedded sensors, a GCS software is required to be connected with Pixhawk on quadcopter via MAVLink. Although there is many GCS software, QGroundControl (QGC) is preferred in this project since it provides full control capability and aerial vehicle setup for quadcopters with PX4 firmware. Nevertheless, QGC is not a must so that Mission Planner or APM Planner also can be used. Figure 5.7 shows a GPS-based image of quadcopter's location in the Gaziantep University.

Figure 5.7: QGroundControl provides GUI streaming location of quadcopter.

5.7 Sensor Calibration

This section explains why and how calibration is implemented for both embedded sensors in Pixhawk and camera which feeds camera node in ROS. Calibration of any sensor is crucial and avoidable because calibration defines the accuracy, confidence and quality of measurements and calculations using. PX4 system uses sensors to determine vehicle state for autonomous control. Vehicle state is the sum of, but not limited to, position/altitude, heading, speed, orientation/attitude and rates of rotation.

PX4 controller requires gyroscope, accelerometer, barometer and magnetometer and GPS. Also, external sensors can be connected to Pixhawk so that different control systems can be run. Also, calibration of embedded sensors is called as Pixhawk calibration among community. A USB camera for object detection and a lidar for distance measurement are connected to the system. To upload PX4 firmware and calibrate embedded sensors in Pixhawk, QGC is used. On the other hand, camera calibration is carried out because distorted images are needed to be corrected as much as possible. Lidar Lite v3 does not require any calibration process.

5.7.1 Pixhawk Calibration

Calibration steps are easily applied because QGC provides user-friendly GUI for it. However, it is crucially important to complete calibration successfully. Otherwise, embedded sensors may acquire low accurate measurements that can cause unreliable flight with low confidence.

Pixhawk can be calibrated both before installing on quadcopter and after installing on quadcopter. In this project, it is calibrated before installing on quadcopter. Calibration steps are described concisely with associated images as follows:

1. Firmware must be uploaded into the system. If it is already uploaded, QGC may warn for upgrade is needed as in Figure 5.8. After associated instruction is followed to upgrade the system, QGC downloads latest version of software and installs it.as in Figure 5.9.

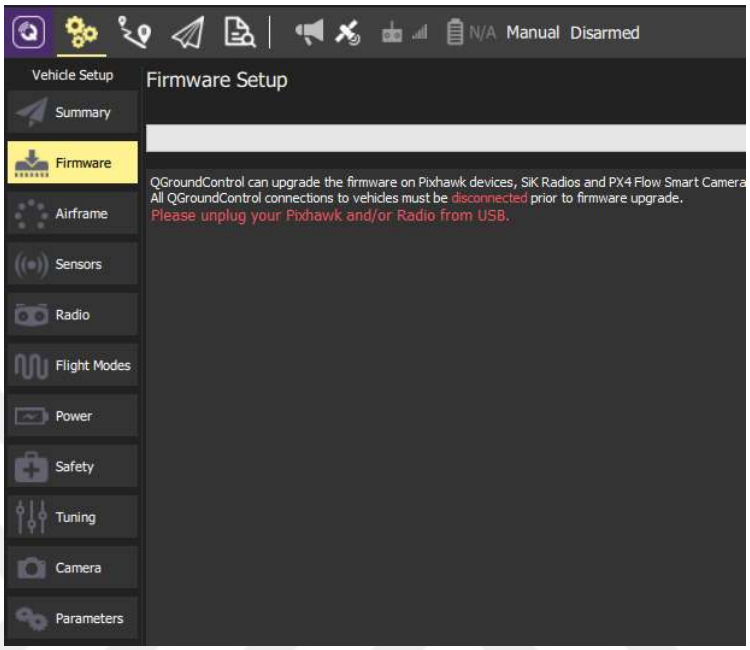

Figure 5.8: All system is monitored with subsystem blocks on the left.

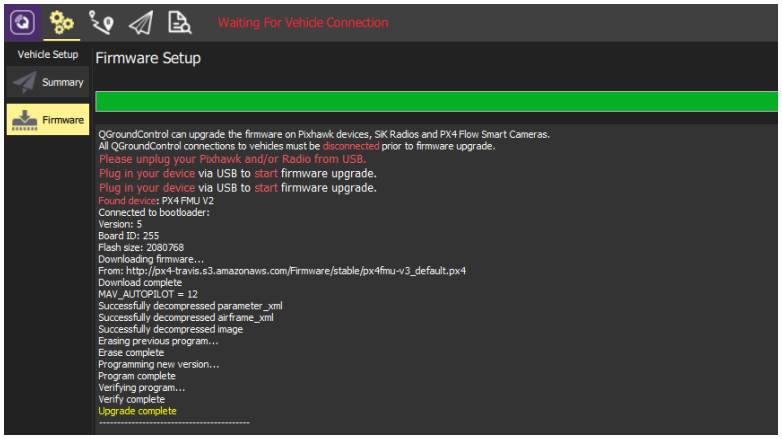

Figure 5.9: Firmware upgrade is successfully completed.

2. Airframe type of UAV must be chosen among others. Generic quadcopter type is selected since 4-propelled X-frame type UAV is developed in this project.

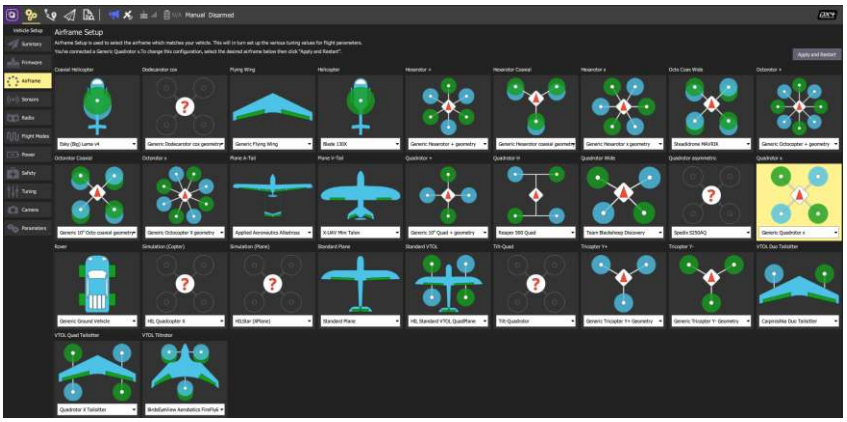

Figure 5.10: Quadcopter type is selected from various configuration options.

3. Sensor section lists out embedded sensors to be calibrated. First, compass (magnetometer) is calibrated by following instructions visual guidance when clicking on it. QGC guides instructions step-by-step with images. For example, when selecting the compass to calibrate, QGC shows what to do as following Figures 5.11 and 5.12. Thus, Pixhawk must be rotated around stated axes or held still in specific direction. For visualization purpose, only compass calibration step is provided with Figure 5.11 and Figure 5.12. Calibrating other embedded sensors follows the same logic so that it can be easily implemented.

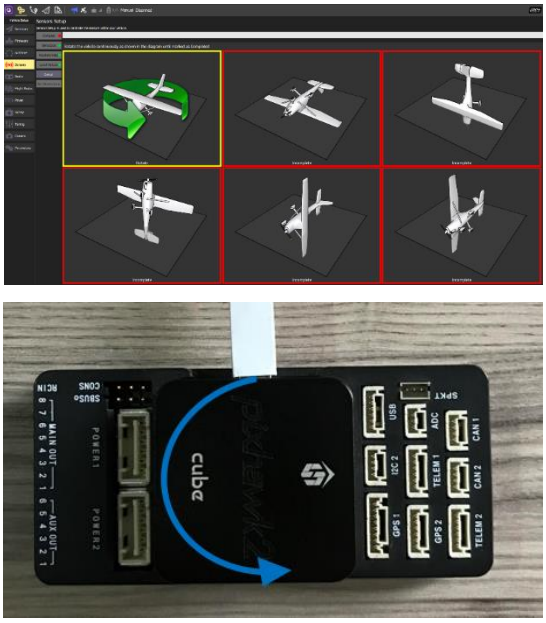

Figure 5.11: (a) Compass calibration instruction to rotate Pixhawk according to visual guide respectively. (b) Pixhawk is rotated around the stated axis.

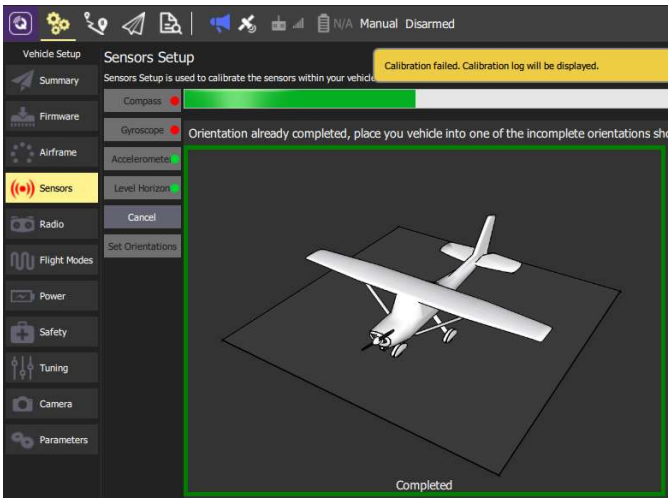

Figure 5.12: First step of compass calibration is completed. QGC visualizes it with completed label and green square.

After sensor calibration is completed, Radio for RC controller is calibrated according to the instructions. Thus, buttons sensitivity can be adjusted in radio calibration section. For flight mode, buttons' assignments to specific control tasks, such as kill switch, throttle and so on is implemented. There are other parameters that can be modified according to the system requirements. For example, lidar usage is enabled in parameters section for the quadcopter since it is used as an external sensor.

5.7.2 Camera Calibration

Camera technology is reliably developing since computer vision techniques are essential focus for autonomous systems, especially MAVs and ground vehicles. Vision ability is almost the most important characteristic for robots to sense and aware their environment like human being. For example, camera feeds view of a scene into onboard computer or processor so that developed software processes raw data for interpreting through image processing algorithms (i.e. edge detection, segmentation). In this regard, camera should be calibrated to get rid of distortion on images according to the external and internal intrinsic of camera. Calibration result as yaml file is required to rectify raw images which is fed from camera to the ROS which requires calibration file as yaml shown in Figure 5.13:

```
# oST version 5.0 parameters

[image]

Width
2304

Height
1536

[narrow_stereo]

camera matrix
1398.144534  0.000000  1114.822253
0.000000  1397.531805  736.299104
0.000000  0.000000 1.000000

distortion
0.053570  -0.110850  0.000941  -0.003266  0.000000

rectification
1.000000  0.000000  0.000000
0.000000  1.000000  0.000000
0.000000  0.000000  1.000000

projection
1371.306885  0.000000  1097.017897  0.000000
0.000000  1407.048706  737.157307   0.000000
0.000000  0.000000  1.000000  0.000000
```

Figure 5.13: Calibration parameters as yaml file format to be used in ROS.

Calibration can be implemented with MATLAB, ROS and OpenCV. Calibration tool of ROS is sufficient but not handy as MATLAB as because MATLAB offers much more information about camera. In this regard, camera calibration toolbox of MATLAB is used. Camera calibrator tool is provided within image processing and computer vision toolbox in MATLAB.

For calibration, there are diverse types of reference board. In this project, checkerboard, also called as chess board, is used as reference measurement tool. Square width and size of chessboard must be known since calibration algorithm detects corners at first, then calculate their distance each other according to the square width of checkerboard. Figure 5.14 shows detected corner of checkerboard.

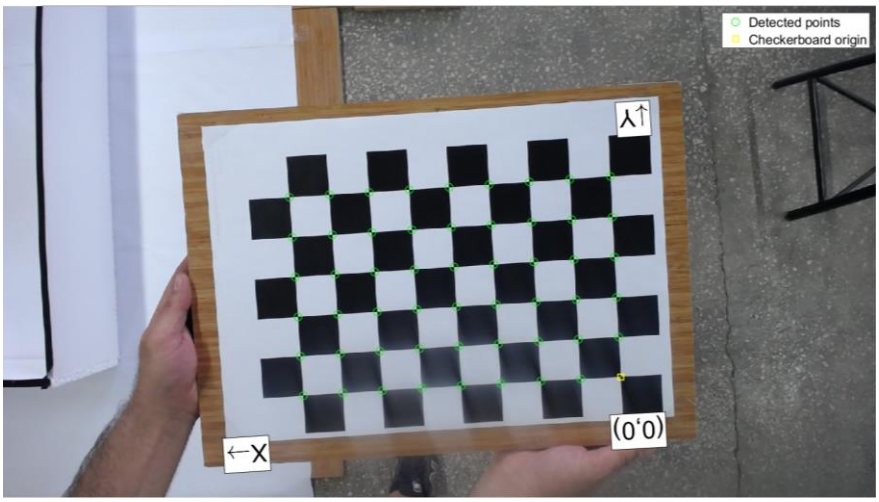

Figure 5.14: Checkerboard image is detected with its corner points.

10 to 20 images should be enough to calibrate camera for high accuracy. Also, MATLAB offers very useful GUI to separate images according to the provided chart for showing mean error in pixels. Thus, it is possible to remove images with high mean error and recalibrate because this provides more accurate camera parameters. In this regard, 32 images are captured and uploaded to calibrator so that images with high error are discarded. Figure 5.15 shows the projection errors (mean error in pixels) of each image with chart. Highlighted blue (first vertical line) in Figure 5.15 belongs to Figure 5.14. After calibration, overall mean error is calculated as 0.18mm per pixel. However, according to get more accurate parameters, the images with individual mean error above 0.15 are discarded so that there remain 16 images. These 16 images are recalibrated and mean error is decreased to 0.12mm per pixel as shown in Figure 5.15. Consequently, video stream (image sequences) can be rectified by using camera's intrinsic parameters in ROS as described previously. For example, Figure 5.16 shows comparison between distorted image (raw) and undistorted (rectified) image by using calibration parameters.

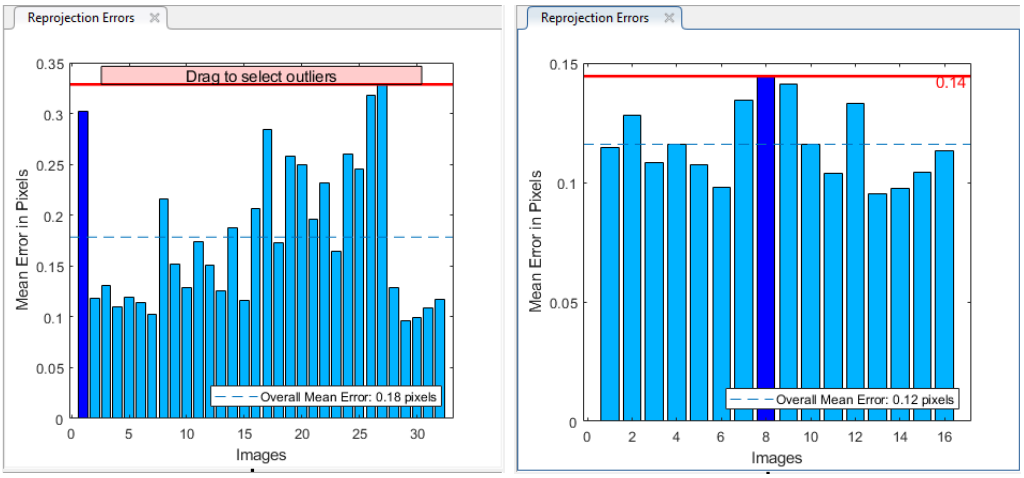

Figure 5.15: (a) overall mean error is 0.18mm per pixel for 32 images. (b) overall mean error is 0.12mm per pixel for 16 images.

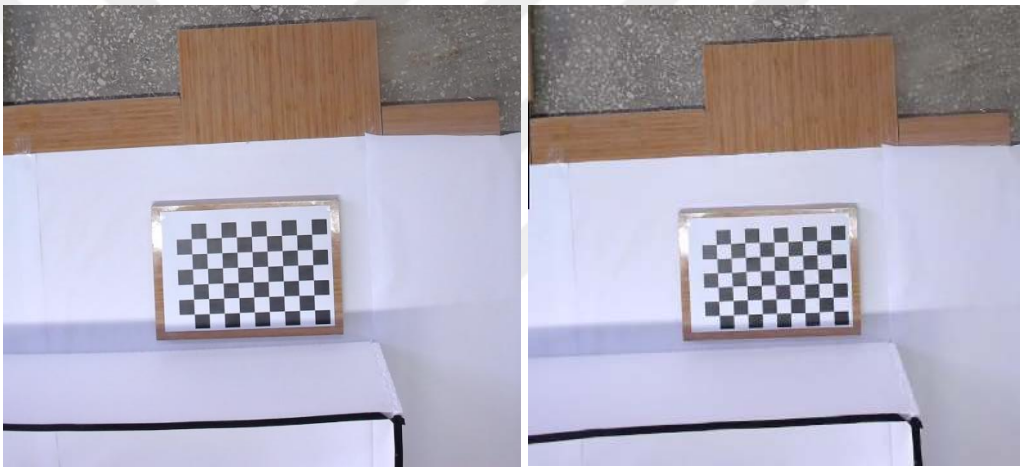

Figure 5.16: (a) Distorted raw image. (b) Undistorted (rectified) image.

Distortion in raw image can be recognized at a first glance by looking bottom-left of (a) and (b) in Figure 5.16. Finally, camera parameters are exported to workspace of MATLAB. Afterwards, calibration file in yaml format is generated with required values as in Figure 5.13.

CHAPTER 6
UAV CONFIGURATION

6.1 Introduction

UAV systems are flexible platforms for many real-time applications in civil and military operations such as surveillance, autonomous navigation and precision agriculture. Their most significant benefit is the ability to easily operate in places where it would turn out more dangerous and time-consuming for human to work. Therefore, system architecture of a UAV should be meticulously designed for particular tasks. The scope of this chapter is the design consideration of a UAV platform that is produced, with a 3D printer, to work autonomously in wide variety of areas, and integration of entire platform for smart applications.

6.2 Mechanical Design Consideration

There are on-going studies on mechanical structure for UAVs. Deciding which one is the most suitable is, indeed, a challenging trade-off between the requirements. Thus, there is not a strict answer for which type of UAV is most suitable in general. In this project, first consideration is size because flight is an expensive operation in terms of energy. It must be appropriate for the type of operation. For example, if a UAV is intended to be designed for agricultural spraying, it should carry nearly 7 kg for pesticides and fertilizers. Consequently, a UAV should have sufficient propulsion system and flight time for maximum efficiency**.** The study [18] explains that energetical problems arise as size of device is reduced. Scaling down a UAV may result an autonomy problem because lower size limits the ability to maintain flight for required time throughout the operation. To cope with these challenges, UAV is designed at system-level to balance trade-offs when selecting components.

Maneuverability is also an important characteristic, in addition to agility and stability. UAVs should have the ability to make rapid maneuvers when it suddenly encounters obstacles, considering environmental awareness, during flight. In this sense, frame

type of structure must be meticulously chosen from diverse types which are multi-copter, fixed-wing aerial vehicles, helicopter, flapping-wing and some others. The selection plays a crucial role for giving the UAV specific aerodynamics features, beyond appearance.

The structure should be light as much as possible, otherwise, system will require much more energy. The most important reason of designing light-weight structure is that propulsion system takes respectable amount of space and is heavier than other components. The heavier structure, the more required energy storage. For example, for a heavier structure, system needs bigger battery pack.

Multi-copters have some advantages over helicopters and fixed-wing aerial vehicles because of their nature. Although fixed-wing aerial vehicles can reach highest speed among them, it cannot hover. Helicopters can hover but they not only don't have sufficient payload capacity if a small UAV is desired, but also, they are not as agile as multi-copters. Table 6.1 compares the benefits of each air frame type [5]. All things considered; multi-copter structure is the scope of this project in mechanical aspect.

UAV Type	Efficiency	Flexibility	Payload capacity	Simplicity	Speed
Fixed-wing	Good	Poor	Good	Average	Good
Helicopter	Average	Average	Average	Poor	Average
Multicopter	Poor	Good	Average	Good	Average

Table 6.1: Comparison of advantages and limitations of frame types

As for selecting frame type of multi-copter, X-frame and H-frame, shown in Figure 6.1, are considered since they are mostly used ones and easiest to build compared to tricopter, octo coax wide, quadcopter wide, stretched X-frame and many others. One of the most important characteristics of X-frame is light-weight structure because it has a smaller central area than H-frame has. Because of the less space, some components also can be chosen as all-in-one type, for instance, 4-in-1 ESC instead of using individual ESC modules for each motor. That results a compact structure and prevents redundancy. Thus, it offers higher payload capacity.

Figure 6.1: (a) X-Frame is fully symmetrical along two-axes, (b) H-Frame is symmetrical along only one-axis

X-frame is naturally symmetrical so that motor propeller units are spaced the same distance from each other, and center of mass as well. It provides high stabilization. On the other hand, H-frame is not completely symmetrical structure since it has a long body along the center. Hardware components must be spread-out along the long body. Thus, it causes not only less compact structure, also high moment of inertia on pitch axis. H-frame offers less agility which is not a desired characteristic for multi-copter systems. Accordingly, the most suitable multi-copter type is the X-frame quadcopter for this project. Figure 6.2 and Figure 6.3 illustrate render image and technical drawing of the developed design respectively, that is based on an open source project from quadcopter design community [45].

Figure 6.2: Render image of quadcopter

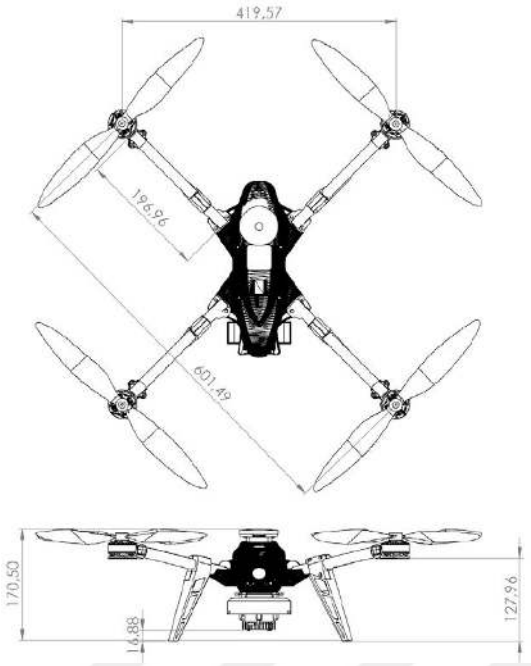

Figure 6.3: Technical drawing gives dimension.

6.3 Manufacturing of Quadcopter Structure

There are two particular considerations about manufacturing the main structure, which they are material and manufacturing type. Both are interrelated subjects. There are a few materials that can be used for the structure of quadcopter. Those are aluminum, carbon fiber and 3D printing filaments. Selecting an appropriate material is decisive for producibility. Therefore, not only for fast prototyping to get empirical results, but also for cost reduction, 3D printing may be the best choice among other conventional manufacturing techniques, for example molding of composite materials. Figure 6.4 illustrates the manufactured structure just before the post-processing [47].

PLA and PETG are commonly used filaments in 3D printing. They are melted between particular temperatures while coming out from the extruder of 3D printer. Afterwards, they are solidified once cooled down. Table 6.2 provides pros, cons, melting/extruder and bed temperature. Although PLA is cost-effective, because it is produced from renewable resources, it is brittle. However, the essential point is to obtain the optimal structure as much as possible. As for PETG, most importantly, it offers much more strength than PLA. Heat resistance, durability and flexibility are essential features that make PETG preferable [47].

Material Type	PLA	PETG
Pros	Easy to print Biodegradable Stiff	Durable Flexible Heat resistant
Cons	Degrading over time Low temperature tolerance Brittle	Prone to scratches More expensive Slow to print
Extruder temperature	205 ± 15°C	240 ± 10°C
Bed temperature	40 ± 15°C	60 ± 15°C

Table 6.2: Comparison of advantages and limits of PLA and PETG materials [47].

In addition, post-processing is a required step for appearance and obtaining high-quality surface. After production, body has rough surface and hollows that may cause fracture. To avoid such kind of failure and obtain strengthened structure, surface is rubbed with emery until it becomes smooth. However, there still remains undesired gaps on the surface that cannot be levelled with rubbing. A suitable paste is implemented to fill the gaps. After pasting, operation of smoothing with emery is repeated until the desired surface is obtained. Afterwards, the body is painted for cosmetic purpose. Figure 6.4 shows the state during post-processing and final state of the quadcopter respectively [47].

(a) (b) (c)

Figure 6.4: (a) Produced quadcopter structure, (b) during post-processing, (c) Final state of quadcopter.

6.4 Hardware Setup

The developed UAV is designed to favorably fulfill surveying and navigation missions in both military and civil applications. As previously discussed, the system is based on quadcopter principle since they have distinguishable benefits compared to others [5]. To build a flexible quadcopter for described purposes in this project, hardware

selection is based on [3]. The quadcopter consists of open source Pixhawk Cube (2.1) for autopilot control, NVIDIA Jetson TX2 module with Orbitty carrier board for running deep learning applications in real time as well as running entire software system. These main components are shown in Figure 6.5 respectively. Pixhawk is a flight controller unit (FCU) which has GPS, inertial measurement unit, gyroscope, accelerometer, barometer and magnometer with the capability of sensor fusion as open source.

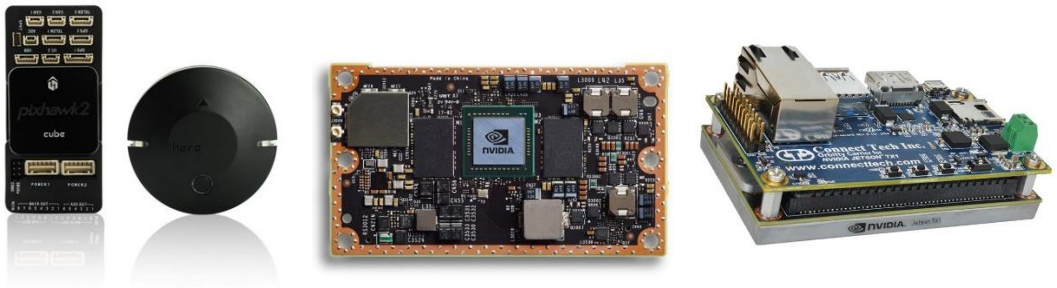

Figure 6.5: (a) Pixhawk 2.1, (b) Jetson TX2, (c) Orbitty carrier with Jetson TX2.

For vision-based processes, the system uses a monocular Logitech c930e USB camera with 90º field of view (FOV). Quadcopter is also equipped with Lidar Lite V3, which is a light-weight laser range finder with 40-meter range for estimating distance to ground with high accuracy, nearly ± 0.0025 meters. It sends data to Pixhawk autopilot controller via either I2C or PWM connections [47]. Camera and Lidar are shown in Figure 6.6.

Figure 6.6: (a) USB Webcam, (b) Lidar Lite V3.

In essence, both designing and integration of electric propulsion system are crucial factors in building of the desired quadcopter [46]. Battery, ESC, brushless dc motors and propellers, which are the components of electric propulsion system on quadcopter as in Figure 6.7, constitutes approximately 60 percent of entire hardware. While

selecting suitable brushless dc motor, there are some parameters that is evaluated. Those are motor Kv, weight-to-power ratio and size of propellers as diameter and pitch. Total weight of the quadcopter cannot be known before completing the assembly. Thus, individual weight of components is summed in order to technical specifications that are published by manufacturers. Total weight of the quadcopter is calculated as 2.5 kg, excluding payload. Afterwards, propeller is chosen with respect to the parameters which are material, pitch value, thrust, shape and area of blades [7]. All things evaluated, four pieces of 580 Kv brushless dc motors of Sunny Sky and 30.48 cm long carbon-fiber propellers with 9.65 cm pitch are used. Thus, the quadcopter can carry 5.5 kg at maximum, including itself [47].

Figure 6.7: (a) Battery pack, (b) 4-in-1 ESC, (c) Brushless DC Motor.

Battery provides required energy for entire system. Since LiPo batteries provide high performance and energy with long life cycle, it is used for power supply unit (Gur and Rosen, 2008). In order to get sufficient flight time and performance, some key concepts are considered. Number of cells (4), voltage value of each cell (14.8 V), capacity (4000 mAh), discharge rate (45C), total weight (484 g) and, of course, physical dimensions (144 x 51 x 34 mm) are the main determinants as optimal battery specifications for the developed quadcopter. Selected battery provides 15 to 18 minutes hovering [47].

ESC is a device that allows the FCU to drive brushless dc motors by arranging the amount of amperage that motors require. It has to be able to handle maximum current

that the motors might require. ESC selection mostly depends on current range that it can handle. Consequently, ESC must be able to handle the voltage of LiPo battery. 4-in-1 type ESC module is used for building a compact platform [47].

For arm, propeller and landing gear as shown in Figure 6.8, it is also worth to mention about material selection. To obtain light-weight, efficient and tough structure, carbon-fiber is used for those parts. Aluminum and fiber-glass are not as light as carbon-fiber. Furthermore, shape of carbon-fiber does not deteriorate over time so that it offers required toughness. As for rigidity, also known as stiffness, carbon-fiber is more rigid. It is nearly three times stiffer than aluminum and fiber-glass for a given weight. Last but not the least, carbon-fiber has negative coefficient of thermal expansion so that it does not expand or shrink as much as fiber-glass and aluminum when temperature changes [47].

Figure 6.8: (a) Carbon propeller, (b) Carbon tube for arm and landing gear.

Electric-Electronic hardware architecture diagram is presented in Figure 6.9.

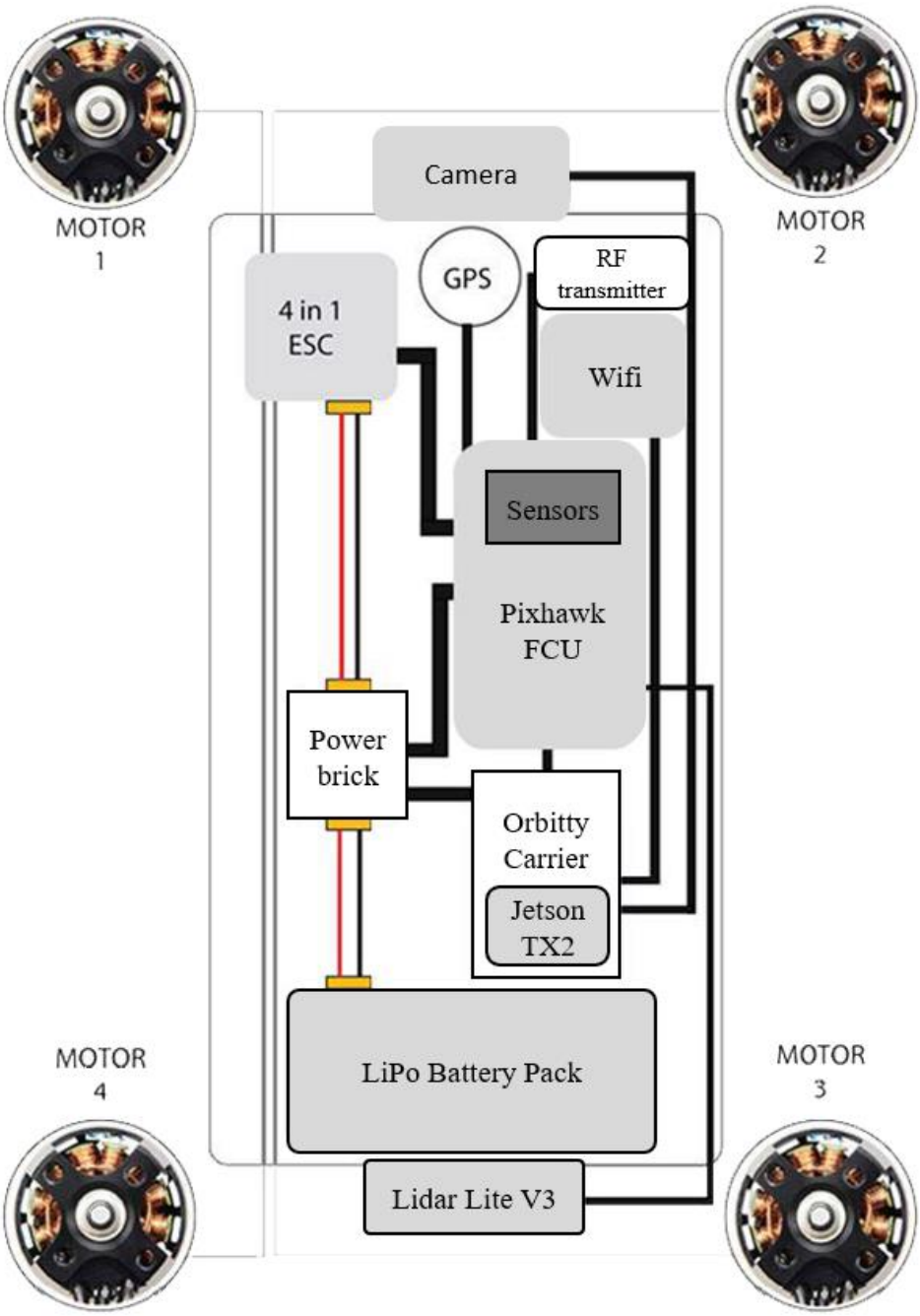

Figure 6.9: Hardware architecture diagram of Drone

CHAPTER 7
RESULTS and CONCLUSION

7.1 Experimental Results

For assessment purpose, autonomous take-off, navigation, landing, optimized object detection model has been evaluated.

7.1.1. Autonomous Flight

Figure 7.1 and Figure 7.2 respectively show captured images from the same flight scenario which is deployed in both software-in-the-loop within gazebo and real flight around the mechanical engineering laboratory.

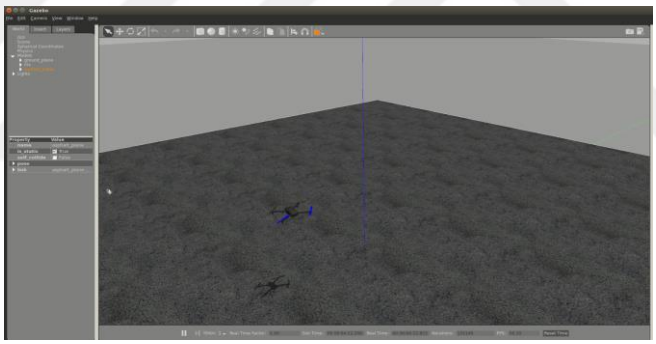

Figure 7.1: Simulation of autonomous navigation is executed according to the predefined waypoints as (x, y, z) in cartesian coordinates within Gazebo.

Figure 7.2: Autonomous navigation is executed according to the predefined waypoints as (x, y, z) in cartesian coordinates.

Both autonomous flights including take-off and landing in simulation and real environment could not be compared quantitatively since it has been almost impossible to get exact position information with pinpoint accuracy from GPS. Nevertheless, it was not hard to decide, by watching only, about that whether drone has reached the predefined waypoints. Reference coordinate system which is used in determining waypoints and the flight attitudes is illustrated with Figure 7.3. For safety reason because of geofenced area around mechanical engineering laboratory, low altitude and close distances (coordinate points) in meters have been defined for autonomous take-off, navigation/flight and landing as follows:

- $\bar{z} = 2$: flight altitude in meters along z-axis towards sky.
- $\bar{x} = [0, 2, -2, -2, 0]$: coordinate points in meters along x-axis which is forward according to the front face of drone.
- $\bar{y} = [2, 2, 2, 2, 0]$: coordinate points in meters along y-axis towards left of drone.

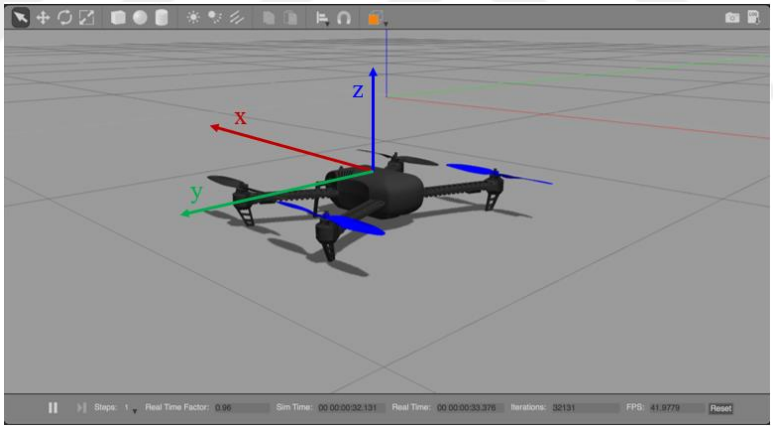

Figure 7.3: Reference coordinate system for defining waypoints along 3-axes.

The developed autonomous flight system can be completely built from scratch so that this project can be forked from Github/earcz, which is based on Redtail [3], [24], [43].

7.1.2. Object Detection

YOLOv3 object detection algorithm with its own framework, which is Darknet, is selected from many successful algorithms. Open source, fast and accurate algorithms have been compared study results and applied some empirical practices before selecting the most appropriate one. YOLOv3 have been performed well in terms of

accuracy with PASCAL and COCO datasets. Evaluation has been performed with respect to the official mAP calculation of YOLOv3.

class_id = 0, name = person, ap = 71.81%
class_id = 1, name = bicycle, ap = 50.64%
class_id = 2, name = car, ap = 60.52%
class_id = 3, name = motorbike, ap = 67.69%
class_id = 4, name = aeroplane, ap = 75.55%
class_id = 5, name = bus, ap = 85.08%
class_id = 6, name = train, ap = 81.67%
class_id = 7, name = truck, ap = 54.42%
class_id = 8, name = boat, ap = 46.03%
class_id = 9, name = traffic light, ap = 49.89%
class_id = 11, name = stop sign, ap = 79.13%
class_id = 12, name = parking meter, ap = 55.34%
class_id = 13, name = bench, ap = 34.16%
class_id = 14, name = bird, ap = 44.98%
class_id = 15, name = cat, ap = 79.73%
class_id = 16, name = dog, ap = 79.69%
class_id = 17, name = horse, ap = 77.59%

Table 7.1: Accuracy precision of detected objects with 0.5 IoU

Figure 7.4 also proves that trained model with more data has offered better result than the original model. That image has been taken with the developed UAV in front of the laboratory. Figure 7.5 also shows a couple detection results on captured images with different resolution from diverse environments. On the other hand, DNN have not been able to detect two persons in another captured image from drone, shown in Figure 7.6. That is most probably because of the following reasons:

- Image captured with low resolution.
- Persons sit on the table. However, datasets that are used consist of standing or walking person with a clear view.
- Persons were occluded with table so that they have been searched over the image partially. That has caused low intersection-over-union (i.e. 0.2).

Figure 7.4: (a) Captured image including one person and car before DNN applied.

Figure 7.4: (b) Default YOLOv3 model has detected only the car, but not the person.

Figure 7.4: (c) Generated new model of YOLOv3 on more data and with more iteration has successfully detected both the person and the car at 0.5 IoU.

Figure 7.5: Images captured in miscellaneous environments to test detection model.

Figure 7.6: Object detection fails when objects are occluded.

TensorRT has been used as an optimizer engine in obtaining accelerated DNN model. Before optimization, trained model has been running with nearly 3 fps on Jetson TX2. After optimization, model has achieved around 50 fps on Jetson TX2. Optimized model has been tested on COCO dataset so that the result is illustrated graphically as Figure 7.7.

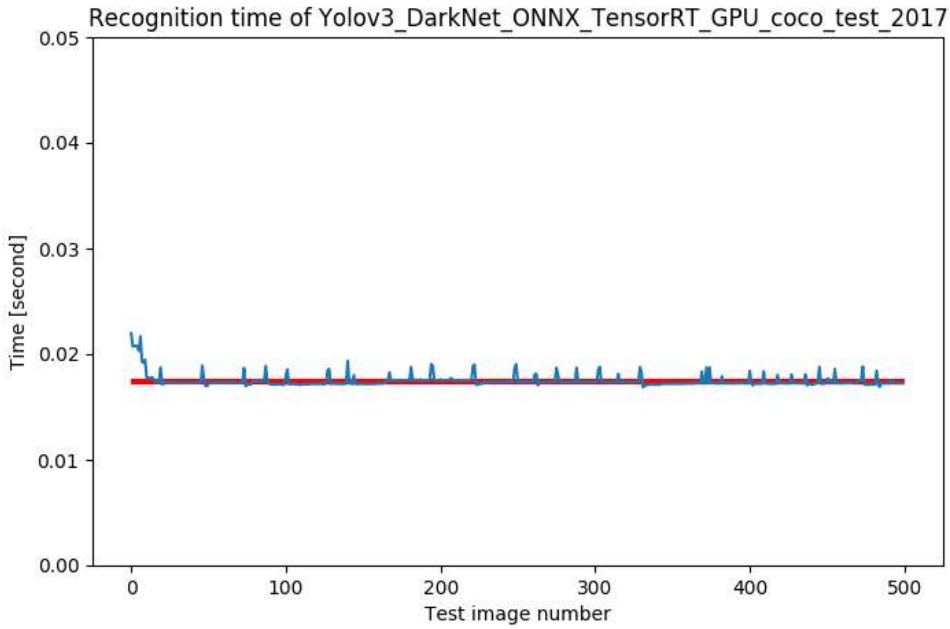

Figure 7.7: Optimized YOLOv3 model with TensorRT runs around 0.017 seconds.

7.2 Discussion and Conclusion

An autonomous UAV system that is capable of detecting objects has presented as well as building a drone. Autonomous system, which is based on PX4 flight stack, has

offered flexibility in GPS-based navigation through open fields for drones. In flexible system point of view, not only planned autonomous flight has been made applicable, but also autonomous take-off and landing features have increased autonomy level as fully autonomous. Also, the system which runs in ROS environment has provided the chance to contribute new capabilities without corrupting the base software system if developing new nodes within ROS. For example, while autonomous navigation system, which is based on planned flight, exists in ROS without running, a DNN-based navigation through fields lacking of GPS signals can be implemented as a ROS node to provide control input over Mavros to flight stack via MAVLink. Thus, ROS may consist of two or more different flight options according to the use cases. SITL model has also assisted in testing the entire system and debugging.

As for object detection with DNN, experiments and studies have been showed that PASCAL and COCO datasets are not the best fit for the developed UAV system. because they have more classes than required. For example, a bench is not an object which is required to be detected by drones in surveillance or search-and-rescue operations. In this regard, dataset should not include redundant classes, but only objects of interest. That allows not only to train more relevant data, but also preventing DNN to calculate unnecessary weights because of redundant data. It is also worth to mention that if a custom dataset which is obviously distinct from the dataset that has been used to obtain the pretrained model is collected, anchor boxes must be recalculated according to the objects and their labels. That improves accuracy because anchor boxes are the essential building block of YOLOv3.

For building a UAV platform, PETG and PLA materials have been used with 3D printing method to produce drone body and complementary components. The main body have taken nearly 44 hours to print in addition to post-processing applications which have taken about 4 days. During this project, 5 drones have been crashed and broken. According to that fact, 3D printing method for robotics applications is an innovative and feasible option in fast prototyping. In contrast, PLA and PETG are brittle materials with respect to the carbon structures so that producing drone with carbon or other stronger materials would have been better option. Producing a drone from scratch and post-processes are time consuming. Following images in Figure 7.8 show working four drone setups and damaged drones afterwards, respectively.

Figure 7.8: Captured images show drone setup and damaged structures after tests.

REFERENCES

[1] Meier, L., Tanskanen, P., Fraundorfer, F., Pollefeys, M. (2011). PIXHAWK: A system for autonomous flight using onboard computer vision, *IEEE International Conference on Robotics and Automation*, 2992-2997.

[2] Bristeau, P.J., Callou, F., Vissiere, D., Petit, N. (2011). The Navigation and Control technology inside the AR.Drone micro UAV, *Proceedings of the 18th World Congress The International Federation of Automatic Control,* **44(1)**, 1477-1484.

[3] Smolyanskiy, N., Kamenev, A., Smith, J., Birchfield, S. (2017). Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 4241-4247.

[4] Weaver, J.N., Frank, D. Z., Schwartz, E.M., Arroyo, A. A. (2013). UAV PERFORMING AUTONOMOUS LANDING ON USV UTILIZING THE ROBOT OPERATING SYSTEM. *Proceedings of the ASME District F Early Career Technical Conference, **12**,* 119-124.

[5] Siebert, S., Teizer, J. (2014). Mobile 3D mapping for surveying earthwork projects using an Unmanned Aerial Vehicle (UAV) system, *Automation in Construction*, **41**, 1-14.

[6] Kan, M., Okamoto, S., Lee, J. H. (2018). Development of Drone Capable of Autonomous Flight Using GPS, *Proceedings of the International Multi Conference of Engineers and Computer Scientists,* **2**, 665-669.

[7] Benito., JA., Glez-de-Rivera, G., Garrido, J., Ponticelli, R. (2014). Design considerations of a small UAV platform carrying medium payloads, *Proceedings of the 2014 29th Conference on Design of Circuits and Integrated Systems*, 1-6.

[8] Lee, J., Wang, J., Crandall, D.P., Sabanovic, S., Fox, G. (2015). Real-Time Object Detection for Unmanned Aerial Vehicles based on Cloud-based Convolutional Neural Networks, *2017 First IEEE International Conference on Robotic Computing (IRC),* 36-43.

[9] Girshick, R., Donahue, J., Darrell, T., Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation, *2014 IEEE Conference on Computer Vision and Pattern Recognition,* 580-587.

[10] Redmon, J., Farhadi, A. (2018). YOLOv3: An Incremental Improvement, *ArXiv,* abs/1804.02767.

[11] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., Berg, A.C. (2016). SSD: Single Shot MultiBox Detector. *European Conference on Computer Vision,* **9905**, 21-37.

[12] Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J., Zisserman, A. (2010). The PASCAL Visual Object Classes (VOC) Challenge, *International Journal of Computer Vision,* **88**, 303-338.

[13] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., Dollár, P. (2014). Microsoft COCO: Common Objects in Context, *ArXiv,* 1405.0312.

[14] Vandersteegen, M., Beeck, K.V., Goedemé, T. (2019). Super accurate low latency object detection on a surveillance UAV. *16th International Conference on Machine Vision Applications (MVA)*, 1-6.

[15] Hossain, S., Lee, D.-J. (2019). Deep Learning-Based Real-Time Multiple-Object Detection and Tracking from Aerial Imagery via a Flying Robot with GPU-Based Embedded Devices, *Sensors*, **19**, 3371.

[16] Tijtgat, N., Ranst, W. V., Volckaert, B., Goedemé, T., Turck, F. D. (2017). Embedded Real-Time Object Detection for a UAV Warning System. *ICCV Workshops,* 2110-2118.

[17] Han, S., Shen, W., Liu, Z. (2016). Deep Drone: Object Detection and Tracking for Smart Drones on Embedded System.

[18] Floreano, D., Wood, JR. (2015). Science, technology and the future of small autonomous drones, Nature, 521. 460-466.

[19] Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep Learning. Cambridge: MIT Press.

[20] Alan Mathison Turing. 2018. Available at: https://en.wikipedia.org/wiki/Alan_Turing

[21] Neural Networks and Deep Learning. 2018. Available at: https://www.coursera.org/learn/neural-networks-deep-learning

[22] Activation Function. 2018. Available at: https://en.wikipedia.org/wiki/Activation_function

[23] You only look once algorithm with framework of Darknet. 2017. Available at: **https://github.com/AlexeyAB**

[24] Repository of author of this project. 2018. Available at: **https://github.com/earcz**

[25] Convolutional Neural Networks for Visual Recognition. 2018. Available at: **http://cs231n.stanford.edu**

[26] Deep learning inference optimizer. 2018. Available at: **https://developer.nvidia.com/tensorrt**

[27] Ren, S., He, K., Girshick, R. B., Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *NIPS*, 91-99.

[28] Nagaraj, S., Muthiyan, B., Ravi, S., Menezes, V., Kapoor, K., Jeon, H. (2017). Edge-based street object detection, *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, 1-4.

[29] Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollár, P. (2017). Focal Loss for Dense Object Detection, arxiv:1708.02002.

[30] Repository of optimized single shot detector algorithm with TensorRT. 2018. Available at: **https://github.com/Goinggs/TensorRT-SSD**

[31] CUDA technology. 2018. Available at: **https://developer.nvidia.com/cuda-zone**

[32] Griffin, G.S., Holub, A., Perona, P. (2007). Caltech-256 Object Category Dataset.

[33] Annotation tool. 2018. Available at: **https://github.com/tzutalin/labelImg**

[34] Optimized you look only once algorithm with TensorRT. 2019. Available at: **https://github.com/eric612/TensorRT-Yolov3-models**

[35] Single shot detector algorithm with MobileNet. 2019. Available at: **https://github.com/chuanqi305/MobileNet-SSD**

[36] Open Neural Network Exchange. 2018. Available at: **https://onnx.ai**

[37] Optimized you only look once algorithm with TensorRT. Available at: **https://github.com/xuwanqi/yolov3-tensorrt**

[38] Pixhawk flight control unit. 2018. Available at: **https://github.com/PX4/Firmware**

[39] Pixhawk development guide. 2018. Available at: **https://dev.px4.io/master/en/index.html**

[40] Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., Stryk, O.V. (2012). Comprehensive Simulation of Quadrotor UAVs Using ROS and Gazebo, *Proceedings of the Third international conference on Simulation, Modeling, and Programming for Autonomous Robots*, 400-411.

[41] Fairchild, C., Harman, T.L. (2016). ROS Robotics By Example. Birmingham: Pack Publishing.

[42] Robot operating system. 2018. Available at: **https://www.ros.org**

[43] Redtail project of NVIDIA. 2018. Available at: **https://github.com/NVIDIA-AI-IOT/redtail**

[44] Gazebo simulation environment. 2018. Available at: **http://gazebosim.org**

[45] **https://www.thingiverse.com/thing:2202502**

[46] Gur, O., Rosen, A. (2008). Optimizing Electric Propulsion Systems for UAVs. *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference.* 5916.

[47] Rencüzoğulları, E. A., Kılıç, A., Kapucu, S. (2018). Design Consideration: UAV System Architecture for Smart Applications, *International Eurasian Conference on Science, Engineering and Technology,* 2503-2509.

[48] Visual odometry. 2018. Available at: https://en.wikipedia.org/wiki/Visual_odometry

[49] Engel, J., Koltun, V., Cremers, D. (2016). Direct Sparse Odometry, ArXiv:1607.02565v2.

[50] Deep learning. 2018. Available at: https://developer.nvidia.com/deep-learning.