UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

HAND GESTURES FOR DRONE CONTROL USING DEEP LEARNING

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

SOUBHI HADRI
Norman, Oklahoma
2018

HAND GESTURES FOR DRONE CONTROL USING DEEP LEARNING

A THESIS APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY

Dr. James Sluss, Chair

Dr. Samuel Cheng

Dr. Gregory MacDonald

To my parents who made me who I am today:

Fatima & Mohamad

To my beloved sisters

Samar & Rama & Nour & Eman

To my dear brother

Mustafa

To my friends

I dedicate this work to you all

Soubhi

Acknowledgments

I would like to extend my heartfelt appreciation and gratitude to my advisor, ***Dr. James J. Sluss***, for the endless support and guidance he has provided throughout my *Master Degree* at *The University of Oklahoma*.

I would also like to express my sincere gratitude to the distinguished members of my thesis committee, ***Dr. Samuel Cheng***, and ***Dr. Gregory MacDonald*** for their time and effort in reviewing this thesis.

I want to thank my friends that inspired and supported me during my journey at Oklahoma University.

Special appreciation goes to the *OU-Tulsa* family of students, professors, and staff for everything they have provided me.

TABLE OF CONTENTS

List of Figures

Abstract

Commercial drones, also known as unmanned aerial vehicles (UAVs), are rapidly becoming more prevalent and are used in many different applications, such as surveillance for sporting events, transportation for emergency equipment and goods, filming and aerial photography and many other activities. The number of drones in United States of America are forecast to double and grow from an estimated 1.1 million units in 2017 to reach 2.4 million units by 2022 according to the Federal Aviation Administration (FAA) [1]. The fact that most of the drones can carry payloads has encouraged many of the drone manufacturing companies to add different types of sensors to drones, and the most basic one is the camera. The previous reasons have led to open a new field of study which is called Drone-Human Interface (HDI) and user interface researchers have started studying the different possible ways to interact with drones ranging from the traditional devices like Radio Controller (RC) to controlling the drones using human body postures.

This thesis presents research detailing the use of hand gestures as a HDI method to control the drones. This work consists of three main modules: Hand Detector, Gesture Recognizer and Drone Controller. A deep learning method is incorporated and utilized in the first module to detect and track the hands in real-time with high accuracy and from a single Red-Blue-Green (RGB) image. Image processing algorithms and techniques are introduced as a dynamic way to identify the hand gestures and motions. Finally, the Drone Controller module is responsible for communicating with the drones. It sends and receives the messages between the proposed system and the drone which is connected to the system.

Chapter 1: Introduction

Hand gestures and body postures are common and natural forms of communication for humans to interact with machines. Due to the huge widespread usage of drones in a variety of applications, such as, surveillance for sport events, transportation for emergency equipment and goods, filming and aerial photography-, a lot of studies have been done to find the most appropriate way for humans to interact with drones. Using hand gestures in drone manipulation is becoming a popular way to interact, this thesis will explore that concept and propose a complete system to control the drones using hand gestures. The proposed system should function in real-time and the accuracy in detecting and recognizing the gestures is an essential factor to care about.

1.1. The Human-Computer Interface (HCI)

The Human-Computer interface can be defined as the area of study to design methods or systems that users utilize to communicate and interact with computers. The first traditional methods that may come to mind as HCI input/output devices are the keyboard and mouse. The importance of HCI rises as everyone in the world can interact with computers during their daily life. HCIs have been studied extensively by researchers and they have tried to set metrics and properties to define and determine the efficiency of HCI methods.

According to Alex Roney Mathew, A. Al Hajj and A. Al Abri [2], "*The most important concepts in HCI are functionality and usability. Services provided usually by a system are called functions. Usability is when a user utilizes the system's functions easily, properly and clearly. Functionality and usability may vary from one* *system to another. A system is said to be successful if there is a balance between both functionality and usability*."

Human-Robot Interaction (HRI) is a branch of HCI and it is the area of study that focuses on the interaction between robots and humans. The HRI Conference is an annual IEEE International Conference [3] which is dedicated to present applied research related to the HRI domain.

Although drones are considered as robots, the rapid increase in the popularity of drones has pushed to launch a new branch of research which is called Human-Drone Interaction (HDI) and it is dedicated to the drone world. Most of the time, it is not possible to interact with drones by touching them directly, so they need special methods. Quite a bit of research has been done in HDI and many investigations studied hand gestures or postures to interact with drones. Some researchers have studied the interaction with drones using the front-facing camera that is connected directly to a drone, as shown in Figure 1. Others have studied the interaction with drones from the perspective of the camera which is connected to a ground-station computer.

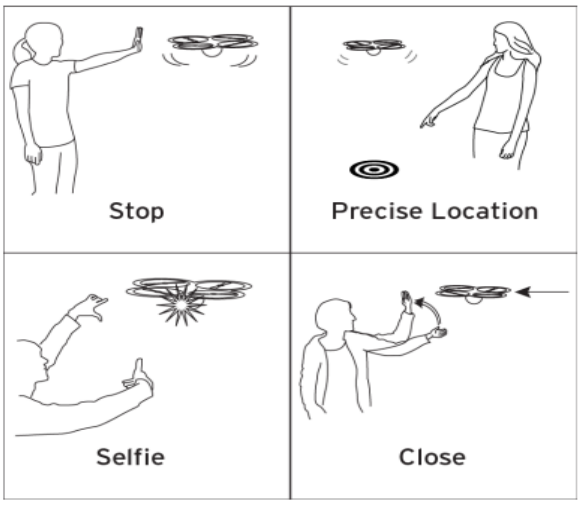


Figure 1. Example of user-defined gestures as HDI [4]

1.2. Existing and related research work

The rapid evolution and spread of drones make them a popular area for researchers in many different domains, either for commercial or personal usage. Controlling the drone is one of those domains that has been the target for many groups of researchers. Most of the current research has been done using third-party hardware devices to recognize the hand gestures or postures like in [5], [6] and [7]. The authors in [5] used the Microsoft Kinect Sensor [6] connected to a computer to detect the body frame and pose of the operator (human) and then they mapped their gestures to specific commands. The commands were then sent to a Parrot AR Drone [9] which was connected to a ground-computer over Wi-Fi. In [6], the researchers used a Parrot AR Drone connected via Wi-Fi to a ground station which employed a Leap Motion Controller [10]. The Leap Motion Controller recognizes gestures and can-do tracking using its two stereo cameras and three infrared LEDs. A Robotic-Operating system (ROS) was used on the client side. On the other hand, the authors in [7] used a front-facing camera on a Parrot AR Drone. The images taken from video streaming were then processed before doing the classification. The Haar feature-based [11] AdaBoost mainly was used for classifications in [7]. The authors mentioned "*A set of five gestures were carefully selected to build a dataset of 8302 images.* " Each of those five gestures was mapped to one of the drone's basic functionalities.

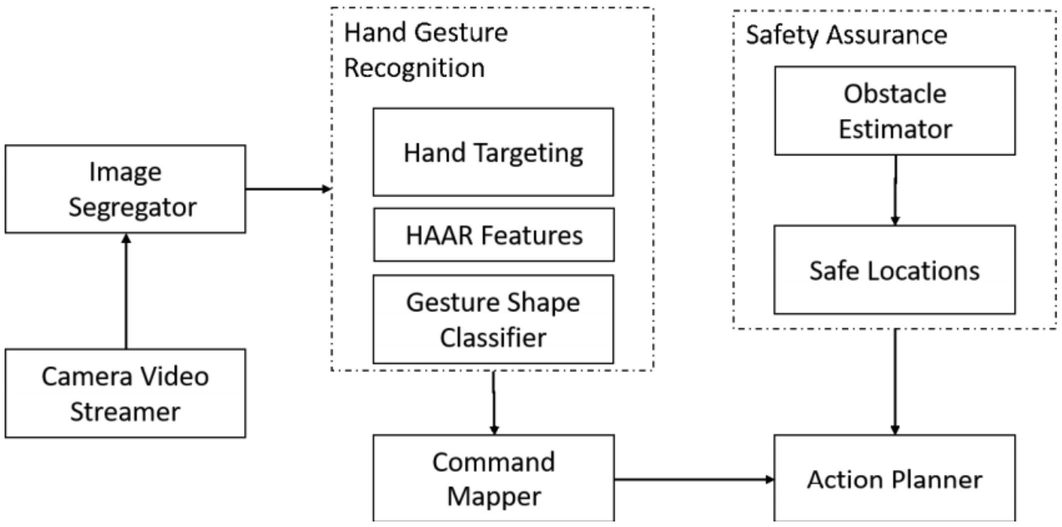


Figure 2. Gesture-based drone control framework [7].

Researchers in [12] also used a Kinect Sensor to analyze gestures and the definition for the gestures was given first *"For example, if one finger represents one aircraft, then one person can operate 10 aircraft."*

To the best of the author's knowledge, none of the existing systems to control the drone using hand gestures, either one or two hands, in real-time with high accuracy, and from only one RGB image, uses deep learning as a solution to detect and track.

For the hand gestures detection part, many investigations have been carried out and the most notable one is by the Nvidia Team [13]. The authors used depth, color and stereo-IR sensors to acquire the data. They also used a 3D-CNN (recurrent three-dimensional convolutional neural network) with connectionist temporal classification (CTC) to classify 25 gestures. They achieved zero or negative lag and the accuracy was 83.8% which is very close to human accuracy 88.4%.

1.3. The contribution of thesis

This thesis presents research to build a complete system to control a drone using hand gestures in real-time with high accuracy in recognizing the gestures and executing the drone’s commands.

The following are the main contributions of this thesis:

- Built a novel and robust system in software to control the drone using hand gestures.
- To the best of the author’s knowledge, this work is pioneering because it uses deep learning and Single Shot MultiBox Detector (SSD) as a solution to get high accuracy in real-time to detect hands from a single RGB image, so there is no longer a need for any special hardware. Moreover, the author acquired and labelled the entire dataset during this work.
- The gesture recognition module is dynamic and was achieved using image processing techniques, so there is no need to re-train the model if a new gesture is needed. This work also allows the use of either one or two hand gestures.
- This work is compatible with the Ardupilot [14] flight stack which is very popular and open source. The system also works with the PX4 [15] or any flight stack using the MAVLink (Micro Air Vehicle Communication protocol) [16] messages.

1.4. Content Summary

This research focuses on using hand gestures as a HDI method. The images are acquired from a camera connected to ground-station computer. The proposed system is purely software and it does not need any special hardware. A deep learning solution was applied in this thesis to get high accuracy hand detection in real-time.

This thesis is organized in a way to explain deeply each module in the system and it can be summarized as the following:

- Chapter 1 includes a literature review of the existing research work related to controlling the drone using hand gestures. It also explains the contributions of this thesis.
- Chapter 2 explains in detail the first module in the system, Hand Detector. Moreover, it demonstrates how a deep neural network is utilized in the module.
- Chapter 3 introduces the defined gestures that this system can understand. It also describes the problems the system had and how they were solved.
- Chapter 4 focuses on the drone as a system element and highlights some of the important concepts and terminologies in the drone area. It also explains the drone controller module in the system.
- Chapter 5 shows results that have been acquired using simulators and describes the system Graphical User Interface (GUI).
- Chapter 6 provides conclusions and suggests ideas for future research.

Chapter 2: Hand Detection

The main purpose for this module is to detect the hands that are shown in each frame. This goal can be achieved in many different ways with different accuracies in detecting the hands. For this project, it is extremely critical to a get real-time behavior with high and acceptable accuracy.

2.1. Hand detection using skin color thresholding

One of the simplest ways to detect the hands can be done using color range thresholding. Using the color range of human skin, all objects outside the range of skin color should be removed, keeping the objects inside the range, as shown in Figure 3. After extracting the hand from the other objects, some geometric solutions can also be applied to detect the fingers.

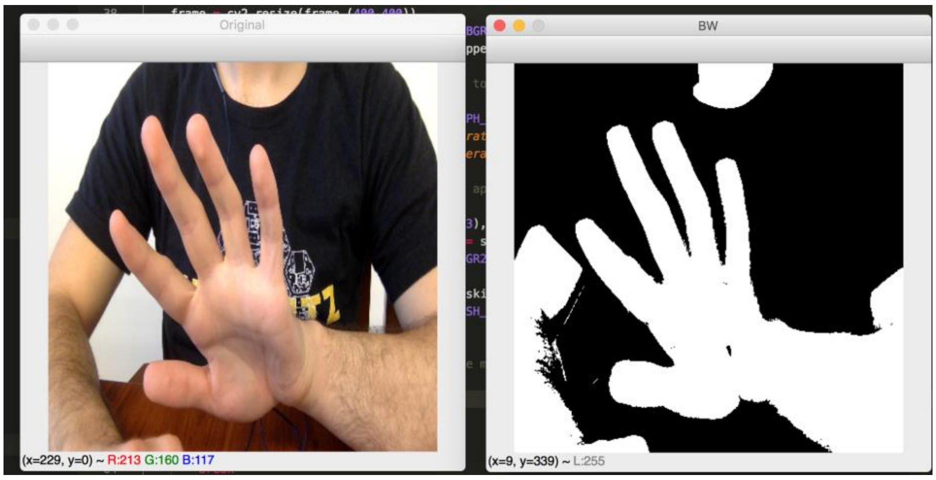


Figure 3. Skin color range thresholding to detect hands in HSV color space.

The previous method is weak and most likely fails to work for systems that require high accuracy because of many reasons:

1. The skin color range for humans is wide which means that after thresholding, many objects will still exist in the image and it will be difficult to extract and distinguish the hands from other objects, as shown in Figure 4.
2. Any change in the environmental conditions may corrupt the whole system because the change in the illumination and shade of the environment will change the color of the hands' skin slightly. The previous problem can be partially solved by moving from RGB (Red, Green and Blue) color space to HSV (Hue, Saturation and Value) or HSL (Hue, Saturation and Lightness) color spaces. In RGB color space, when the lightness changes, the change will be reflected directly on all the color components R-Red, G-Green, B-Blue, unlike the HSV color space which separates the illumination element V-Value from color elements H-Hue and S-Saturation. The problem is shown in Figure 4.

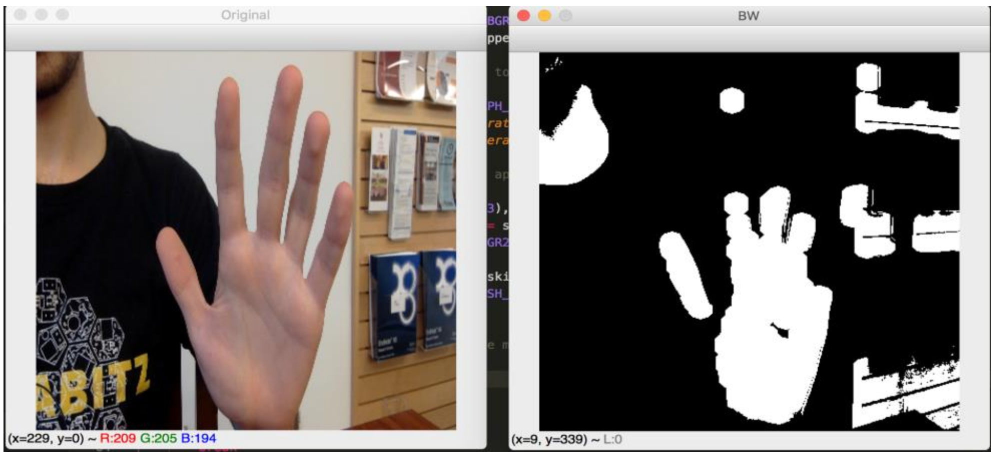


Figure 4. Two problems for color thresholding:

1. **The extraction fails when the environment's conditions change.**
2. **Objects within the same color range appear after thresholding.**

3. The final and most important reason is when the hand is shown in front of any background object with the same color range. Such as, when the hand appears in front of the person’s face. This method does not work at all and both objects will be considered as one object, as shown in Figure 5.

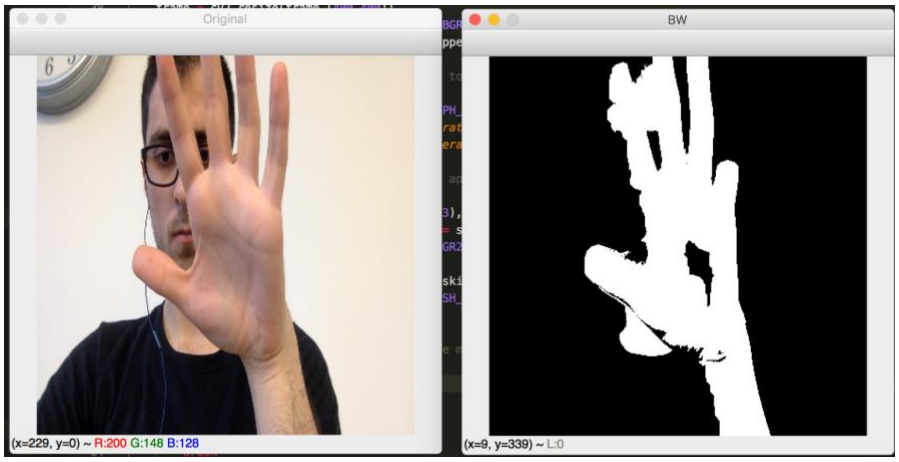


Figure 5. The hand and face are considered as one object.

2.2. Hand detection using Deep Learning

For this system, the accuracy should be high. There is no way for the problems mentioned in the previous method to happen because either losing the hands or detecting fault hands may lead to sending incorrect commands to drone, causing it to crash.

Another way that can be used to achieve hand detection is utilizing Deep Learning. A substantial amount of work has been done in the computer vision field using deep learning and the results are impressive. After considerable research, there are many known network architectures which give good accuracy in image recognition problems like AlexNet [17], VGG [18] and ResNet [19].

The detector network should do more than classification. It should do object recognition, localization and classification. There are many networks that work as detectors and the differences between them are in accuracy and speed. Some of the neural networks are very accurate but cannot work in real-time. In this project, all the system should work in real-time with good accuracy, i.e., it very rarely missed recognizing the hands in images. If there is a tiny error, like the localization of a hand is slightly shifted by few pixels in one frame, it will not impact the result because the system does not depend on just one frame to take the decision. YOLO v.2-v.3 (You-Only-Look-Once) [20] and SSD (Single Shot MultiBox Detector) [21] are good solutions for this problem.

In this project, SSD: Single Shot MultiBox Detector (by C. Szegedy et al.) is used to detect the hands. For 300 × 300 input, SSD reaches a 74.3% mAP (mean Average Precision) on the PASCAL VOC [22] dataset with 300*300 input image and 59 FPS (frame per seconds). In one frame or image, SSD can detect multiple categories. The output of a SSD is a bounding box around each class appearing in the input image. If there are many objects belonging to the defined classes, the output will be many bounding boxes. VGG-16 is the base network architecture which SSD is built on, as shown in Figure 6.

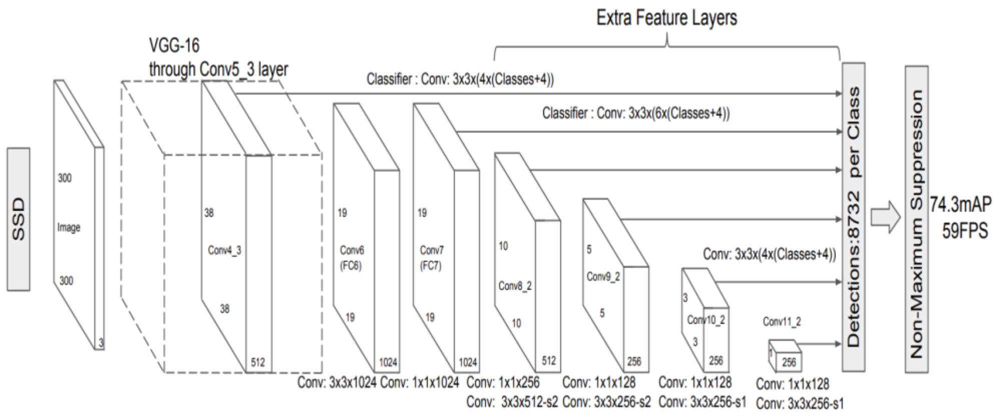


Figure 6. Single Shot MultiBox detector architecture with 300*300 input [21].

One of the main challenges in Deep Learning methods is to find a good dataset to train the network. There was no appropriate dataset that can be used in this project so a complete dataset needed to be collected, labelled and prepared for training the network. The box diagram for this module is shown in Figure 7.

After acquiring and labelling the data, the network is trained and then a test is applied. If the test result is not good enough and the accuracy is not acceptable to the solve the problem, more investigations are required for both the dataset and network's parameters and structure.

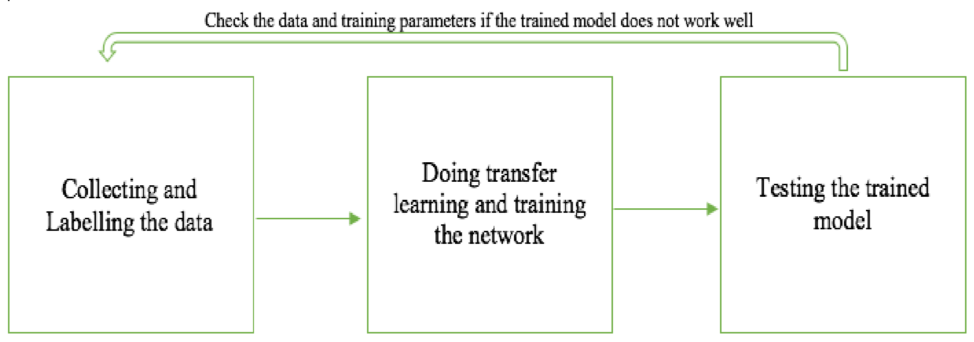

Check the data and training parameters if the trained model does not work well
Collecting and Labelling the data
Doing transfer learning and training the network
Testing the trained model

Figure 7. Box diagram for hand detector module.

2.2.1. Data Collecting and Labelling

This phase of the project went through many steps of testing and looking back into the dataset to remove noisy and inappropriate data, such as outliers, corrupted or unbalanced data. The three main steps for this phase are the following:

1. First, around 150 images were collected in one indoor place and then labelled. Some samples are shown in Figure 8. All the images were from one class (open hand). The reason for this stage was to test the SSD network performance with hand detection problem. The labelling process was done by setting a box around each hand appearing in each image in the dataset and setting the corresponding class for it. The box is called a ground-truth bounding box. Each box consists of

a start point (top-left) and an end point (bottom-right).

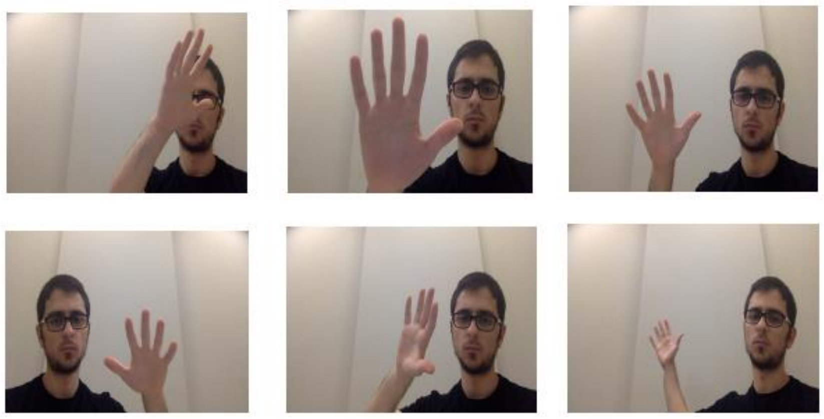


Figure 8. Samples of the dataset. All the samples are from one class.

The performance of the trained model was acceptable and promising, which means that SSD can work well with this problem.

2. More data were collected and labelled. The dataset in this stage was around ~2300 images which were collected in different environments, both outdoors and indoors. In this stage, the network had to detect six classes for each hand appearing in the image. The six classes represent the number of open fingers in the hand that appear in the image (see Figure 9).

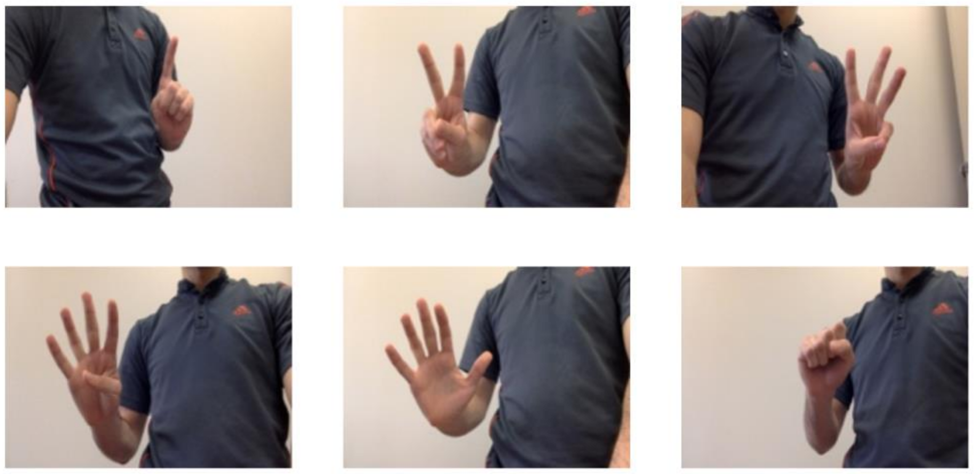


Figure 9. Samples of the six classes.

After training the network, one problem that sometimes occurred was that the network started considering the face as a hand, as shown in Figure 10. After going back to the training data, it was noticed that in many samples, a hand was located exactly in front of the face, so the network started considering in some cases the face as a hand.

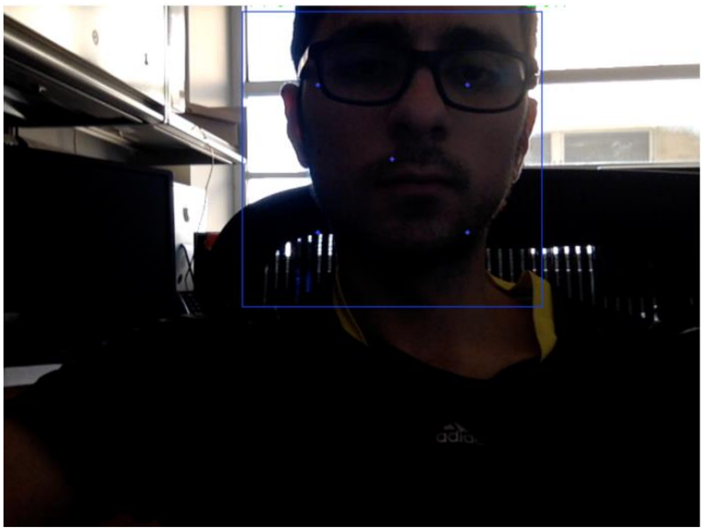


Figure 10. The network detects the face as hand from class 5.

3. More images were added to the dataset and the part of the images that caused the problem in the previous stage were removed so the network would not overfit, or generalize because the dataset was unbalanced. Most of the new added images have two hands either with same class or from different classes (see Figure 11). The dataset in this stage contains around ~3500 images. The trained model from this stage works well and the performance of the network is sufficient to detect the hands for this project.

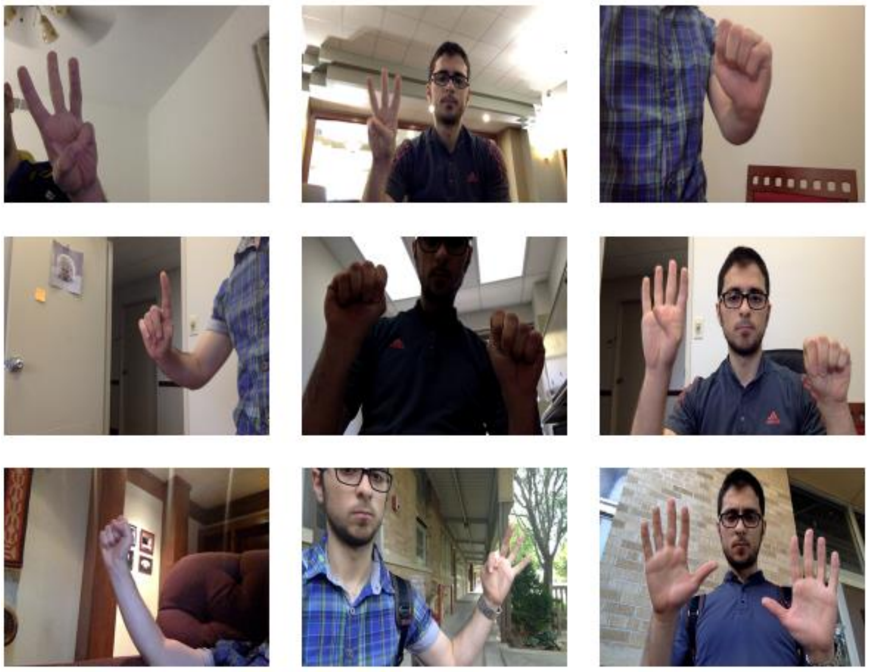
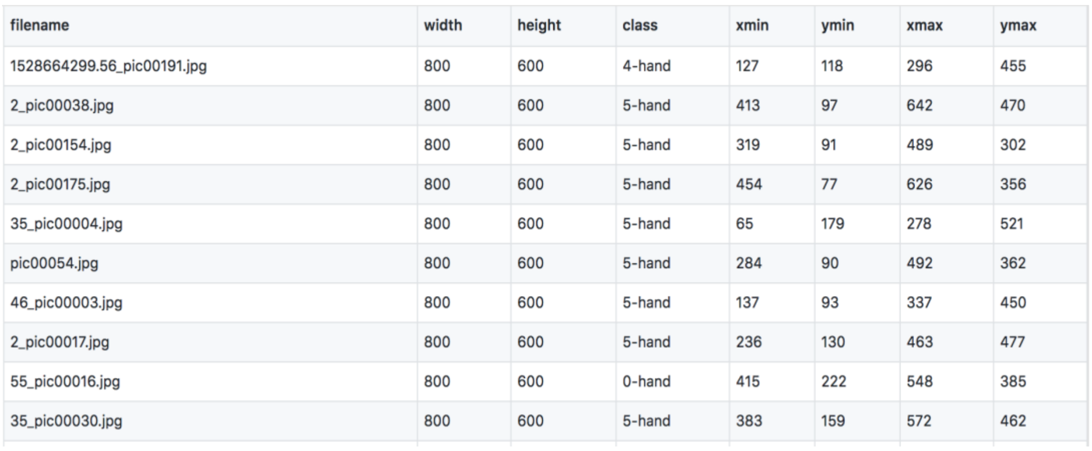


Figure 11. Samples from the final dataset.

2.2.2. Transfer Learning and network training

In general, Transfer Learning means using a model which was trained on network A to solve task A in initializing the parameters of network B to solve problem B. An example of this would be using a model trained to detect cars to initialize another network to detect people. By using transfer learning, the training process will be much quicker than initializing the parameters randomly. The reason for this is most of the time the first few layers in any network are responsible for detecting edges and recognizing some patterns. The main differences usually occur in the last few layers of the networks. A model trained to detect pets was used to do transfer learning to initialize the network for this project.

Figure 12 shows samples of the input for the training and testing dataset. xmin and ymin are the coordinates of the left-top point of the bounding-box whereas xmax and ymax are the coordinates of the right-bottom point. The width and height were resized to be 300*300.



filename	width	height	class	xmin	ymin	xmax	ymax
1528664299.56_pic00191.jpg	800	600	4-hand	127	118	296	455
2_pic00038.jpg	800	600	5-hand	413	97	642	470
2_pic00154.jpg	800	600	5-hand	319	91	489	302
2_pic00175.jpg	800	600	5-hand	454	77	626	356
35_pic00004.jpg	800	600	5-hand	65	179	278	521
pic00054.jpg	800	600	5-hand	284	90	492	362
46_pic00003.jpg	800	600	5-hand	137	93	337	450
2_pic00017.jpg	800	600	5-hand	236	130	463	477
55_pic00016.jpg	800	600	0-hand	415	222	548	385
35_pic00030.jpg	800	600	5-hand	383	159	572	462

Figure 12. Samples of the input rows for the training dataset.

The training process was done using the open source TensorFlow [23] deep learning library on the Debian [24] operating system with a Nvidia GPU GeForce GTX 1050 and 16GB RAM (Random Access Memory). The input of the network is a 300*300 image. The dataset was split randomly into a training and testing dataset by 80% and 20% accordingly. The training time was around 34 hours but the total loss graph shows that the training process could be finished after around 22 hours and the loss value kept oscillating for the rest of the time. The total loss is a combination of localization loss and confidence loss. Localization loss computes the error between the predicted boundary box and the ground truth box. Confidence loss measures the error in the class prediction. The total loss is computed as:

$$total\_loss = 1/N * (confidence\_loss + \alpha * localization\_loss)$$

where α is the weight for localization loss and N is the positive match.

Figure 13 shows the total loss. The vertical axis represents the loss value while the horizontal axis represents the time.

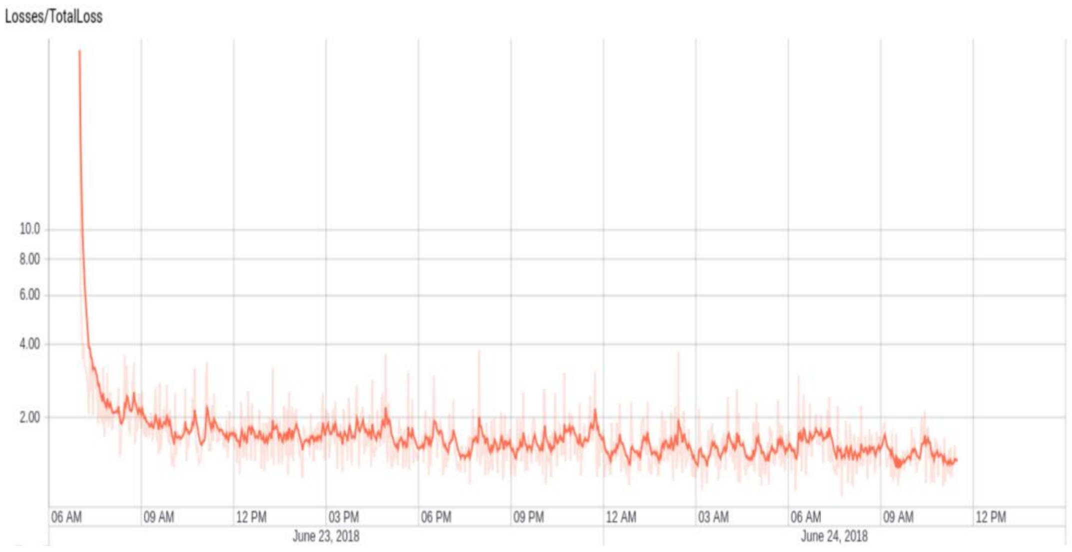


Figure 13. Total loss.

2.2.3. *Testing the trained model*

The real test is the only efficient way to test the accuracy of the model. After the training process was done and the loss was close to 1.00 and consistent with other similar investigations, two different tests were done to judge the efficiency of the model. First, the model was tested on the same machine which was used for training, the machine was equipped with a GPU. The performance was sufficient for this project and the model was able to detect all the hands in real-time without any latency. There were two problems which we have solved using some image processing solutions and they will be introduced in Chapter 4. The second test was done on another machine with an Intel Core i5 CPU and 8 GB RAM. It was tested by different people and again it worked in real-time, (see Figure 14 for samples of the results) which proved that the performance of the network is robust and sufficient, therefore ready for the next module to recognize gestures.

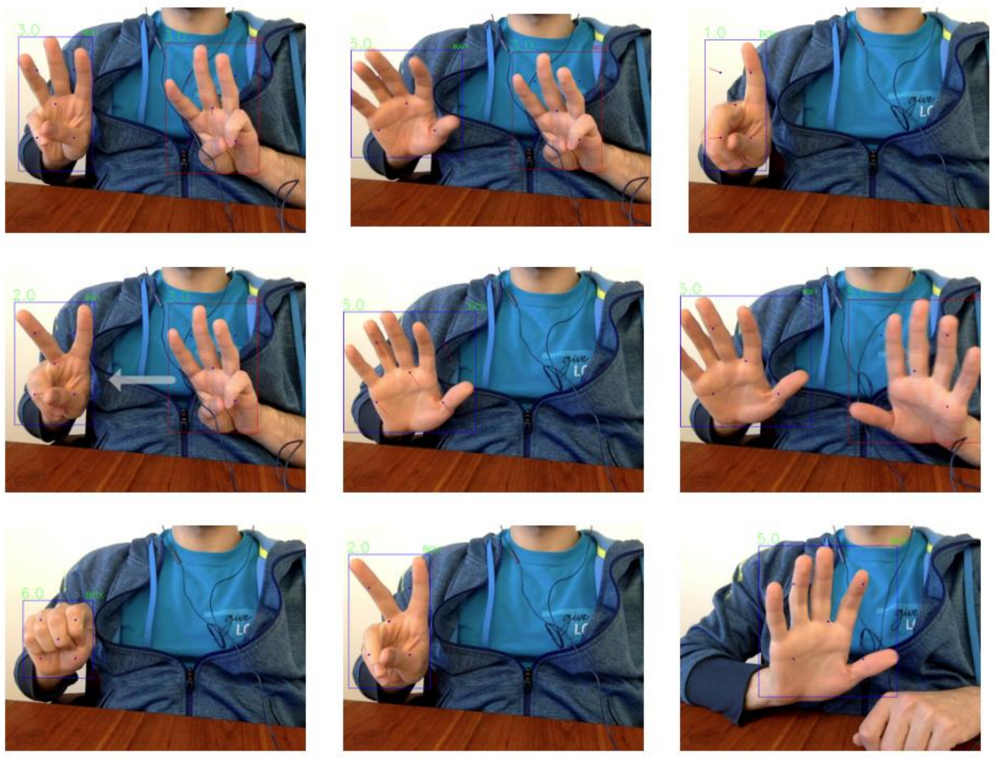


Figure 14. Samples from the output of the trained model.

Chapter 3: Gesture Recognizer

The main purpose for the gestures or motions detector module is to recognize specific gestures that are done by one or two hands and then map these gestures to drone commands. Those commands will be the input for the last module (Drone Controller) in this system. The input for this module is a list of bounding boxes around each hand that appears in the image and is generated for each time frame by the hand detector module.

As was mentioned before, for the real-time neural network detector, the accuracy may not be perfect and there will be some problems which should be fixed. Two problems were encountered during the testing phase in the previous module which were fixed in the beginning of this module using image processing methods.

3.1. Multiple boxes for same hand problem:

The SSD network should solve this problem internally using non-maxima suppression to make sure that the object is identified only once. The hand detector sometimes generates two bounding boxes for the same hand, as shown in Figure 15, which makes it impossible to track and recognize the gestures. The gestures for two hands are totally different from the gestures for one hand. The reason for this problem might be because the labelling for the data is not perfect. For example, for the same class, the author might sometimes set a slightly bigger ground-truth box around the hand and in another data sample for the same class, he might set a slightly smaller or shifted ground-truth box around the same hand class.

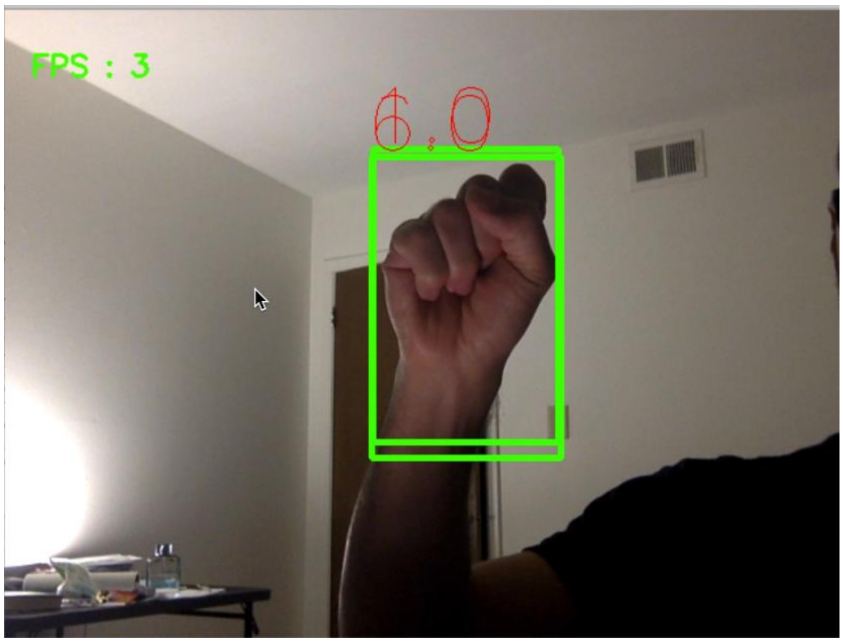


Figure 15. Multiple boxes for same hand problem.

There are three suggestions to solve this problem:

a. Going back to the whole dataset and re-labelling it accurately. This solution is unrealistic and it would take a lot of time to go back to thousands of images to re-label them. Moreover, it is extremely difficult to perfectly label all the images. Finally, even if it was done, the problem might still show up and it is a shortcoming of the detector itself, not because of a labelling error.

b. Increasing the dataset and at least doubling its size. This solution may help and adjust the weights for the network to be more sensitive to those cases. Again, it is not guaranteed and requires a lot of time to collect, label and re-train the network so this solution was not applied and was set as the last solution to be tested.

c. Using image processing techniques to solve the problem. The redundant boxes for one hand are intersected with at least 80% of their areas. Each box is specified by two points: left-top point and right-bottom point, so we can calculate the area for the box as in (1).

$$area\_box1 = (right\_box1 - left\_box1) * (bottom\_box1 - top\_box1). \quad (1)$$

Then we calculate the intersection area as shown in (2), (3), (4), (5) and (6). If it is more than 80% of the area of the box, then one of the boxes is redundant (7).

$$x = max\ (left\_box1, left\_box2). \quad (2)$$

$$y = max\ (top\_box1, top\_box2). \quad (3)$$

$$intersection\_width = min\ (right\_box1, right\_box2) - x. \quad (4)$$

$$intersection\_height = min\ (bottom\_box1, bottom\_box2) - y. \quad (5)$$

$$intersection\_area = intersection\_width * intersection\_height. \quad (6)$$

The condition of redundancy:

$$Redundant?\ \ intersection\_area >= area\_box1*0.8 \quad (7)$$

The previous solution guarantees that each hand will have just one box around it.

3.2. Tracking the hands in consecutive frames problem:

This problem happens when there is more than one hand and the system should recognize their gestures during multiple consecutive frames. The output from the previous module (hand detector) is a list of bounding boxes around each hand appearing in the current frame but the problem is that the list is not ordered. For example, if the hand detector module detects two hands and the list has two boxes in frame #1, the first box in this list points to the hand that is located in the left part in frame #1 but the second box points to the hand that is located in the right part in the same frame. In frame #2, there is no guarantee that the first element in the list is still pointing to the hand in the left part and the same for the other one. It is essential to keep tracking the hands in consecutive frames to recognize specific motions or gestures.

The solution for this problem is based on the theory that the movement in the position for one hand in two consecutive frames will be very small. So, for each new frame, a copy of the previous boxes is kept and the new list of boxes is calculated. After removing the redundant boxes, if they exist as described before, each one of the new boxes is linked to a previous one if the number of new boxes in the new frame is more than one (more than one hand). The linking process is done by comparing the intersection area between each box in the new frame with all boxes in the previous frame. The two boxes with the biggest intersection area will be linked together.

This algorithm and the previous one should be optimized so that the system still works in real-time without any delay.

3.3. The supported gestures

Now, the gesture recognition can be applied. The goal for the whole system is to control the drone. So, the operations to be supported should be first summarized and then each mapped to one specific gesture. The gestures are generated either by one hand or two hands. All the gestures were chosen arbitrarily and we tried to make them practical and easy to use.

3.3.1. Arm and Takeoff gesture

To takeoff and fly the drone, it should first be armed which means to turn on the motors. There are some conditions that should be met before the drone is able to arm and takeoff and if one of those conditions fails, the drone will reject the arm command. More details about the drone characteristics will be introduced in the third and last module in Chapter 5. So, if the user does the corresponding gesture for this operation and the drone is ready for it, it will set the correct flight mode (GUIDED mode), arm the motors and takeoff, and hover at an altitude of one meter.

The gesture corresponding to this operation is to do the following in the same sequence with just one hand (see Figure 16).:

1. Open the hand: Open means that at least four fingers should be opened.
2. Close the hand: Close means all fingers should be closed.
3. Open the hand again.

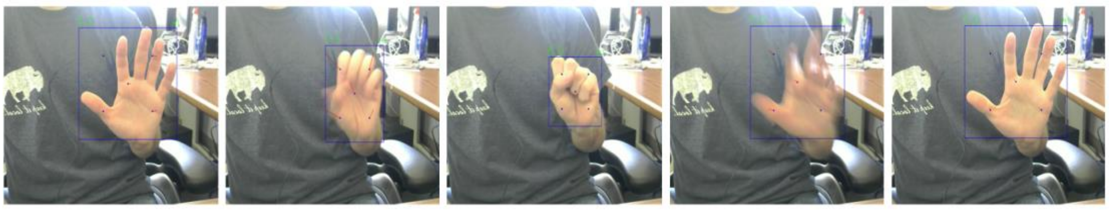


Figure 16. Arm and Takeoff gesture from left to right.

If there is a delay in performing the gesture in the same sequence mentioned above, the gesture will be ignored.

3.3.2. Controlling the drone movement gestures

As was mentioned, all the gestures were chosen arbitrarily. So, for controlling the drone movement, it was decided to make it smooth and easy to use. The author tried to imitate the way people control any ground vehicle using the steering wheel.

3.3.2.1. Moving straight forward

First, the driver should hold the steering wheel with both hands. To get the same behavior in this system, the user should close their hands in front of the camera. Whenever the user closes their hands, the movement starts and a virtual steering wheel shows up. When the user opens both hands, the movement pauses. Moreover, in any vehicle to make the movement exactly straight forward, the steering wheel should be in the center and both hands should be parallel and in the same horizontal line as shown in Figure 17.

3.3.2.2. Moving forward with right/left angle

If the user wants to move the vehicle to the right with a specific rotation angle, they should rotate the steering wheel and the left hand should be above the horizontal central line of the steering wheel but the right one goes down. For a left movement, the driver can achieve that in the opposite way of the right movement.

The user for this system can get exactly the same behavior with very similar hand movements. After they close both hands, they can move to the right with a specific rotation angle when they raise their left hand above the horizontal central line to the steering wheel and incline the right one. When they raise the left hand more, the rotation angle will be bigger to right. The opposite behavior (raising the left hand and inclining the right one) for moving forward with an angle to the left. Figure 17 shows some of the drone movement gestures. The top two left images correspond to the move straight gesture. The top right image corresponds to move straight with an angle to the right gesture. The second row corresponds to move straight with a different angle to the left gesture. The third row corresponds to the stop movement gesture.

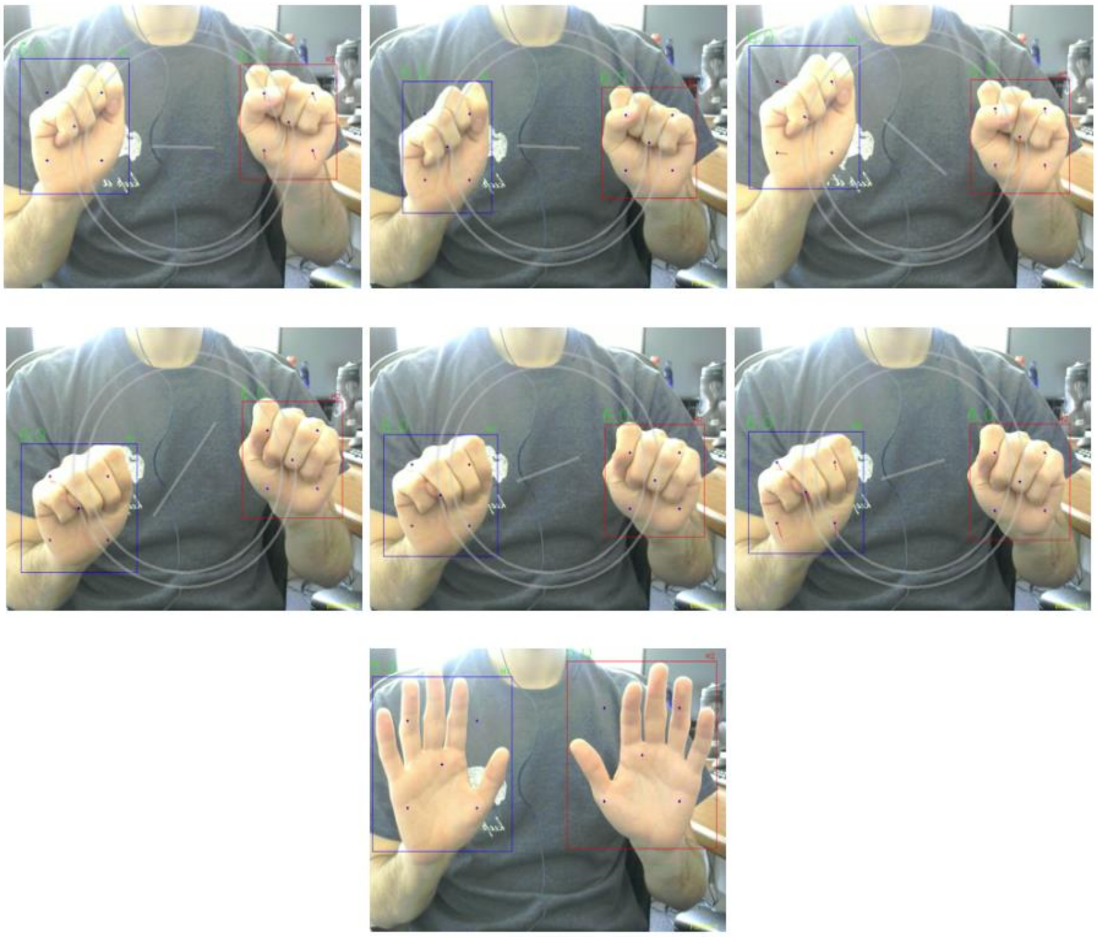


Figure 17. Drone Movement Gestures.

3.3.2.3. Moving straight backward

For the backward movement, the driver has to change the gear stick position to reverse the movement and make it backward. In this system, the user can flip the movement from forward to backward (and return back to forward from backward) using the gesture that appears in Figure 18 for three consecutive frames. A small yellow label indicates the movement direction either forward or backward, which is shown in the bottom right of each frame to help the user know the current direction.

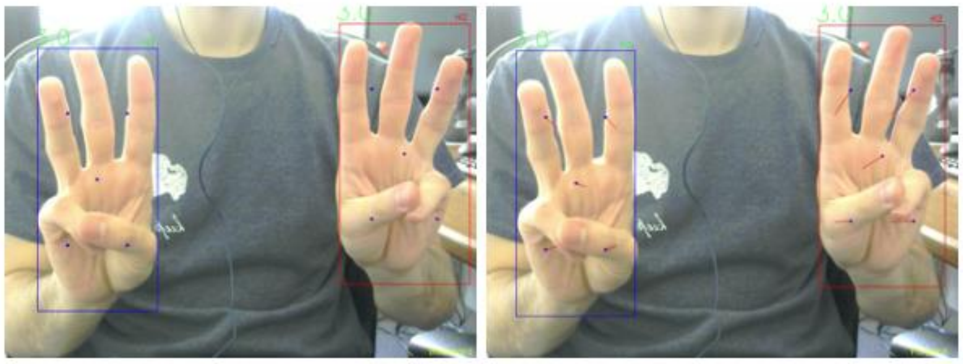


Figure 18. Flipping the movement direction gesture from front to back and vice versa.

3.3.2.4. Moving backward with right/left angle

To start moving backward with an angle to the left or right, one should first flip the movement direction as described in the previous section and, after that, all the rules for moving forward with an angle is applied to moving backward with an angle.

3.3.2.5. Moving straight right/left

Some drones can move in other ways which are not available in all vehicles like cars. The quadcopter (or copters in general) drone can go exactly right or left without going forward or backward. The same behavior is not available in fixed-wing drones. To get that lateral movement in this system, the user should close their right hand and open the left one to move the drone right. To move it to the left, they should reverse the gesture and open the right hand and close the left one. An animated arrow will show up to help the user indicate the movement’s direction whether it is right or left (see Figure 19).

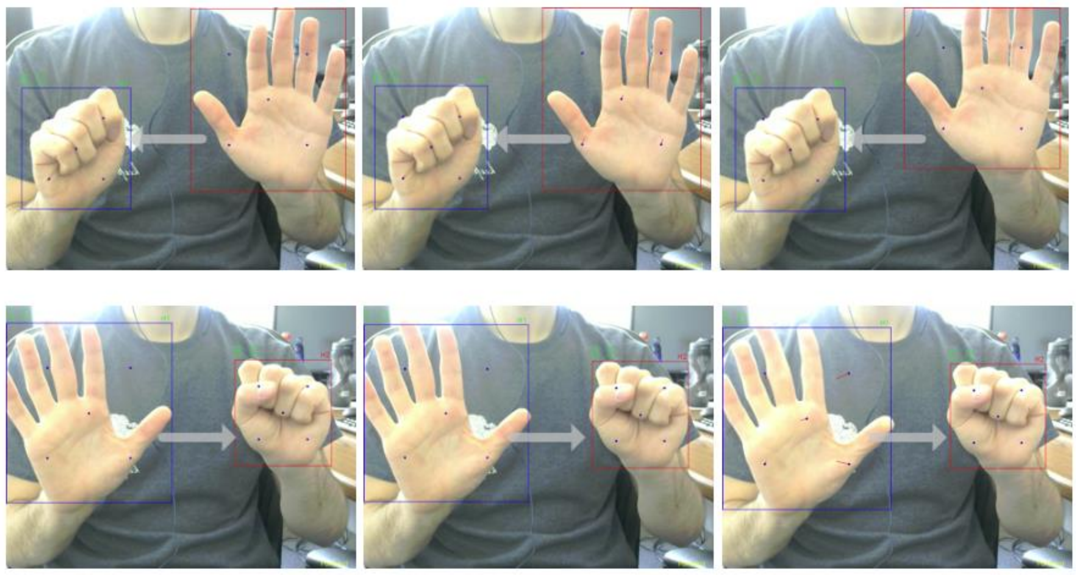


Figure 19. Straight right and left movement gestures.

3.3.2.6. Increase/Decrease the altitude

The quadcopter drone can also increase and decrease its altitude by going up or down. That movement can be achieved using two hands. To increase the altitude, close the left hand and open just one finger on the right hand. To decrease the altitude, reverse the gesture (the right hand should be closed). Again, an animated arrow will appear to help the user indicate the movement, whether it’s up or down, as shown in Figure 20. An operator should always keep their eyes on the altitude to avoid crashing the drone when decreasing its altitude.

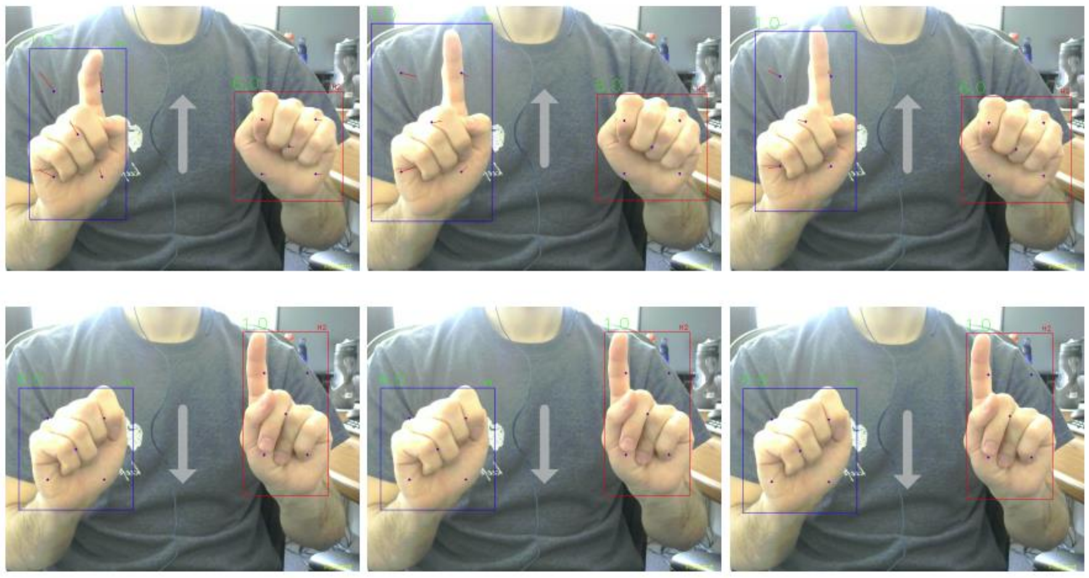


Figure 20. Up and down movement gestures.

3.3.3. *Land and disarm gesture*

The corresponding gesture to the land and disarm command is the same defined gesture for arm and takeoff. So, if the drone status is armed and the altitude is more than zero meters, that gesture will change the flight mode to Land. When the drone reaches the land, it will disarm and turn off its motors. If it is already disarmed, the gesture will make the drone arm and takeoff as described in the arm and takeoff section.

Chapter 4: Drone Controller

The first part of this chapter introduces the most important concepts in the drone area. The second part explains the module that is responsible for communicating with the drone. The communication with the drone is bidirectional which means that the commands will be sent to the drone and feedback and telemetry data will be received from the drone.

4.1. Drone Terminology

The following section will introduce some of the important definitions and terminology in the drone world.

4.1.1. Unmanned Aerial Vehicle (UAV)

A UAV is any aircraft without a human pilot onboard. There are many types of UAVs, like copters and fixed-wing aircraft (Plane). The copter itself has many types like the helicopter, tricopter, quadcopter, hexacopter…etc. The differences between them is the number of rotors. If the number of rotors is more than two, it can be called a multicopter. This system described in this thesis can work with multicopters because some of the selected movements are not available in planes, like moving backward. So, whenever the word “drone” is mentioned, it means multicopters. Figure 21 [13] shows different types of UAVs.

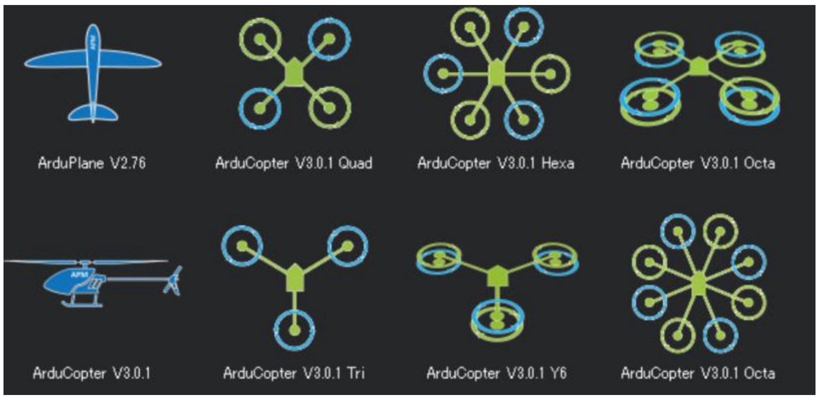


Figure 21. Different types of UAVs [13].

4.1.2. Flight Controller and Flight Stack

The Flight Controller (FC) is a hardware board that controls the motors and all other parts of the drone. It varies in complexity. The most famous one is Pixhawk The Cube [25]. Most flight controllers are configurable and employ sensors like Inertial Measurement Units (IMUs) [26]. Most of the Flight Controllers contain more than one IMU and each IMU has accelerometers and gyroscopic sensors.

The FC's Firmware or Flight Stack is the software that runs on the FC and utilizes the FC and all other resources connected to it. It is also called the Autopilot. There are many firmware and the most famous open source ones are PX4 [15] and Ardupilot [14]. They are very similar and provide most of the main functionalities like navigation and control algorithms. Both recently have started working on the implementation of many smart functions like autonomous flying and obstacle avoidance. Figure 22 [27] shows the building block system for the PX4 flight stack.

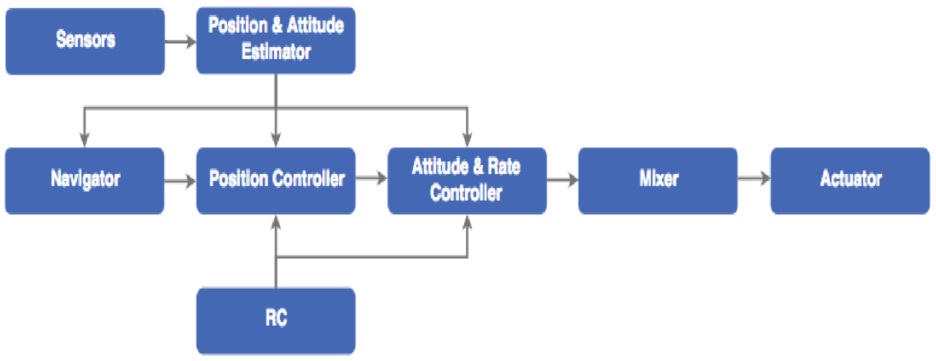


Figure 22. PX4 flight stack building blocks [27].

The main difference between the PX4 and Ardupilot flight stacks is the license. Ardupilot is General Public License (GPL) [28] which means developers who modify and then sell Ardupilot are obligated to make their updates open source unlike PX4 which is Berkeley Software Distribution (BSD) [29] which means it is not required to make the modifications open. INAV [30] is another open source autopilot which started growing recently.

4.1.3. Drone Hardware

There are some essential parts to build the drone and there are other optional ones. The main steps to assemble the drone’s essential parts are:

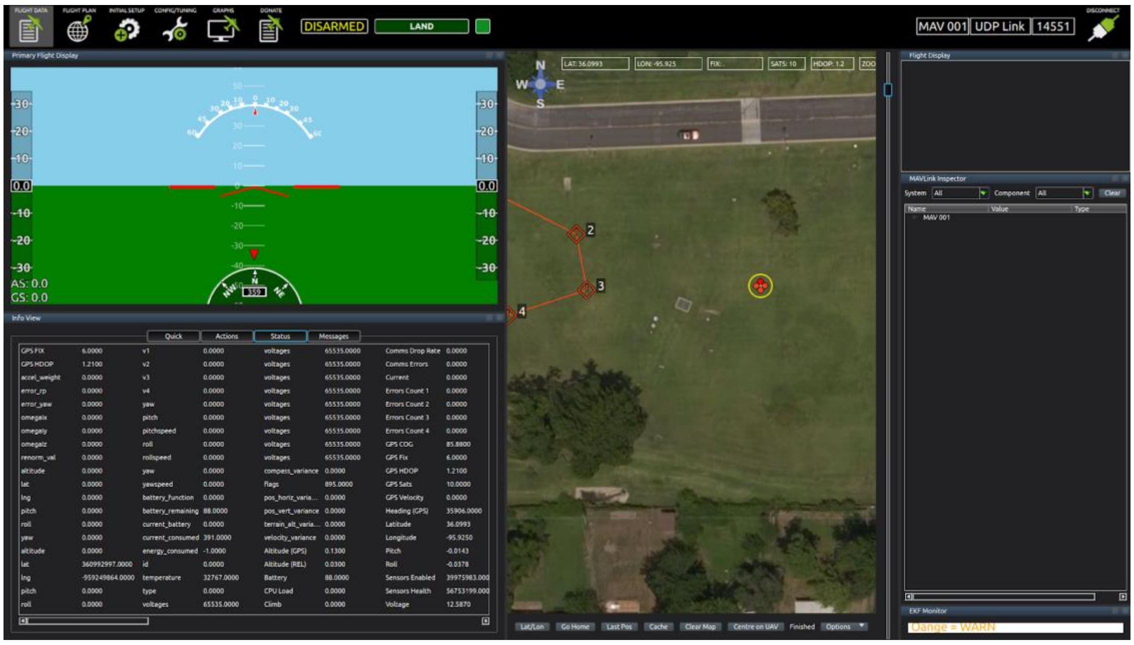
1. Choose the copter frame. The usage of the drone can help to select the appropriate frame. It is totally different if the drone is for video capturing from racing.
2. Mount the Flight Controller on the frame.
3. Connect the GPS (Global Positioning System) module and Compass module to Flight Controller.
4. Connect Electronic Speed Controllers (ESCs) and Motors with FC.
5. Set the battery and propellers.

6. Set the Radio Control (RC). It is used to control the drone manually.

There are many optional parts like the Wi-Fi module, companion computer, camera, gimbal… etc.

4.1.4. Ground Controller Station (GCS)

A GCS is any software application that runs on ground-based station which is connected to the UAV. Most of them can display real-time telemetry data about sensors, position and performance coming from the drone, as shown in the lowest left part in Figure 23. A GCS is essential to load the firmware onto FC and calibrate the drone’s parts. It is also very useful to modify configuration parameters for the FC and to check some of the problems that may happen. The most famous GCSs with GUI (Graphical User Interface) are APM Planner2 [31] for Linux and MacOS and Mission Planner [32] for Windows.



GPS FIX	6.0000	v1	0.0000	voltages	65535.0000	Comms Drop Rate	0.0000
GPS HDOP	1.2100	v2	0.0000	voltages	65535.0000	Comms Errors	0.0000
accel_weight	0.0000	v3	0.0000	voltages	65535.0000	Current	0.0000
error_rp	0.0000	v4	0.0000	voltages	65535.0000	Errors Count 1	0.0000
error_yaw	0.0000	yaw	0.0000	voltages	65535.0000	Errors Count 2	0.0000
omegaIx	0.0000	pitch	0.0000	voltages	65535.0000	Errors Count 3	0.0000
omegaIy	0.0000	pitchspeed	0.0000	voltages	65535.0000	Errors Count 4	0.0000
omegaIz	0.0000	roll	0.0000	voltages	65535.0000	GPS COG	85.8800
renorm_val	0.0000	rollspeed	0.0000	voltages	65535.0000	GPS Fix	6.0000
altitude	0.0000	yaw	0.0000	compass_variance	0.0000	GPS HDOP	1.2100
lat	0.0000	yawspeed	0.0000	flags	895.0000	GPS Sats	10.0000
lng	0.0000	battery_function	0.0000	pos_horiz_varia...	0.0000	GPS Velocity	0.0000
pitch	0.0000	battery_remaining	88.0000	pos_vert_variance	0.0000	Heading (GPS)	35906.0000
roll	0.0000	current_battery	0.0000	terrain_alt_varia...	0.0000	Latitude	36.0993
yaw	0.0000	current_consumed	391.0000	velocity_variance	0.0000	Longitude	-95.9250
altitude	0.0000	energy_consumed	-1.0000	Altitude (GPS)	0.1300	Pitch	-0.0143
lat	360992997.0000	id	0.0000	Altitude (REL)	0.0300	Roll	-0.0378
lng	-959249864.0000	temperature	32767.0000	Battery	88.0000	Sensors Enabled	39975983.000
pitch	0.0000	type	0.0000	CPU Load	0.0000	Sensors Health	56753199.000
roll	0.0000	voltages	65535.0000	Climb	0.0000	Voltage	12.5870

Figure 23. APM Planner2 GCS.

4.1.5. Calibration

After assembling the hardware and uploading the firmware onto the flight controller using any GCS software, some parts of the drone need to be calibrated. It is required to calibrate the following parts: ESCs, Sensors like the Accumulator and Compass, Radio Transmitter. Some optional hardware devices also need calibration like the PX4FLOW sensor [33] which is used to calculate the optical flow and ground velocity for the drone. It is mainly used when the drone has to work in indoor environments where the GPS cannot work. PX4FLOW is supported in both Ardupilot and PX4 flight stacks.

4.1.6. MAVLink Protocol [16]

The MAVLink (Micro Air Vehicle communication protocol) is used for communication between GCSs and UAVs. MAVLink messages are used in both the control and navigation commands that are sent from the GCS and the telemetry data coming from drone. Most of the autopilots support MAVLink. The telemetry data are the messages that the drone keeps sending, for example, containing information about the sensors, battery, location (if there is enabled GPS) … etc.

4.1.7. Ardupilot [14]

Ardupilot is an open source autopilot project, which consists of navigation system for controlling any unmanned vehicle system [14]. It is growing and evolving very fast. The Ardupilot firmware is compatible with many controller boards to control different types of unmanned vehicles. So, it is not just dedicated to a drone but it also supports ground rovers and underwater vehicles. Some of the most important features of Ardupilot are its large community where you can find support and solutions for your issues easily, the well-organized documentation and its quick continuous growth.

4.1.7.1. Copter Flight Modes [34]:

Ardupilot supports around twenty flight modes for copters. Most of the known and commonly used flight modes are available in most of the other autopilot systems, but they may appear as different names. This section introduces a brief list of the most common and recommended flight modes to use. There are two main categories for flight modes. The first one works with GPS, so the drone needs to satisfy the GPS Lock condition to be able to do takeoff and fly. If the GPS Lock is not satisfied, the takeoff fails and the drone sends a “GPS is not locked” message. The other group of flight modes do not require GPS data. The following list includes the modes that require GPS Lock:

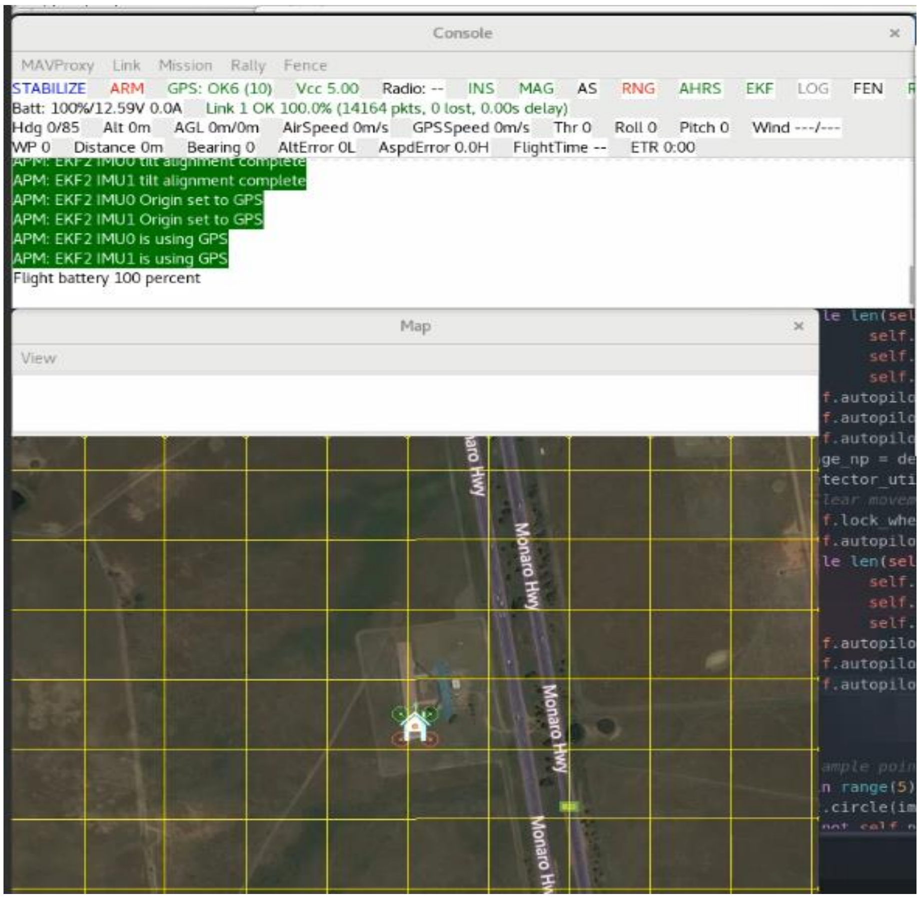
1. Loiter Mode: the copter holds its position (heading and location) and altitude automatically when the user releases the RC’s sticks. This is the recommended mode for new users.
2. RTL (Return to Launch) Mode: when the user sets RTL as a flight mode for the copter, it goes back to home location automatically after rising to a specific altitude, which is a configurable parameter.
3. Auto Mode: the user can define a list of waypoints and upload it to the drone. Each waypoint consists of latitude, longitude, and altitude. When the user sets Auto mode for the drone, it will automatically follow the waypoints one by one in the same order the user defined.

The following list of modes do not require GPS Lock:

1. Stabilize Mode: it is a Manual mode, so it is the responsibility of the user to maintain its yaw, roll and pitch angles, and altitude. It is not recommended for new learners.
2. Alt-Hold Mode: the copter holds its altitude, but the yaw, roll and pitch angles work as in Stabilize mode.
3. Land Mode: in this mode, the drone will go straight down and land. There are some parameters for this mode which the user can configure using GCS.
4. Guided Mode: This is one of the most important flight modes for a developer. It allows dynamic control of the drone in real-time. If it depends on GPS data, it requires GPS Lock. If it depends on other sensors such as vision sensors and PX4FLOW to do hovering and stabilization, there is no need for GPS lock.

4.1.7.2. SITL (Software in The Loop) [35]:

SITL is a simulator used to run the Ardupilot flight stack on any PC with any operating system, without any special hardware. SITL is not just a copter simulator. The user can also run Rover and Plane. SITL is very beneficial in testing any application or code before implementing it on a real drone and saving it from crashing. It even accepts MAVLink messages and sends telemetry data. The user can also connect GCS to it. The Figure 24 shows the map and console windows for copter SITL.

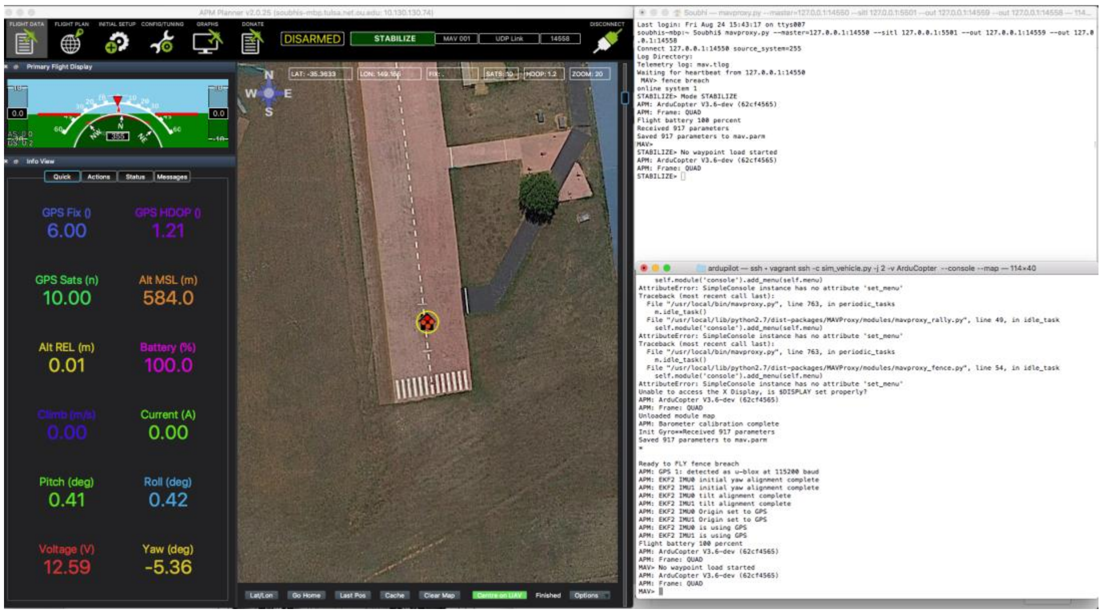


```
STABILIZE   ARM   GPS: OK6 (10)   Vcc 5.00   Radio: --   INS   MAG   AS   RNG   AHRS   EKF   LOG   FEN
Batt: 100%/12.59V 0.0A   Link 1 OK 100.0% (14164 pkts, 0 lost, 0.00s delay)
Hdg 0/85   Alt 0m   AGL 0m/0m   AirSpeed 0m/s   GPSSpeed 0m/s   Thr 0   Roll 0   Pitch 0   Wind ---/---
WP 0   Distance 0m   Bearing 0   AltError 0L   AspdError 0.0H   FlightTime --   ETR 0:00
APM: EKF2 IMU0 tilt alignment complete
APM: EKF2 IMU1 tilt alignment complete
APM: EKF2 IMU0 Origin set to GPS
APM: EKF2 IMU1 Origin set to GPS
APM: EKF2 IMU0 is using GPS
APM: EKF2 IMU1 is using GPS
Flight battery 100 percent
```

Figure 24. Map and Console in SITL Copter.

4.1.7.3. MAVProxy [36]

MAVProxy is a lightweight GCS with command line interface. It provides all the essential functionalities and supports MAVLink. It is a command line interface by default, but it can use some plugins to launch a simple GUI. The most beneficial feature in MAVProxy is the possibility to open multiple ports to forward messages for one drone so one can connect many GCSs to the same drone. The proposed system in this thesis is considered as a type of GCS so it needs first to be connected to SITL. The IP (Internet Protocol) address for the SITL simulator is 127.0.0.1 if it runs on the same computer that the GCS runs on. If they are on different computers but connected to the same network, the IP address should be the address of the computer the SITL runs on. The port is customized and it is possible to open many ports to forward messages to SITL. For example, if the following applications are running on the same computer at the same time: SITL, MAVProxy, APM Planner2, and the proposed system. MAVProxy will be connected to SITL by the default opened port. Using MAVProxy, it is possible to open two additional ports so that the other applications can communicate with the SITL simulator. The connection can be UDP (User Datagram Protocol), TCP (Transmission Control Protocol) or serial. Figure 25 shows the APM Planner2 GCS on the left connected to port 14588, MAVProxy on the top right connected to SITL and is set to forward messages to 14559, and 14558 ports with 127.0.0.1 IP Address and on the bottom right SITL command line.



```
GPS Fix ()        GPS HDOP ()
6.00              1.21
GPS Sats (n)      Alt MSL (m)
10.00             584.0
Alt REL (m)       Battery (%)
0.01              100.0
Climb (m/s)       Current (A)
0.00              0.00
Pitch (deg)       Roll (deg)
0.41              0.42
Voltage (V)       Yaw (deg)
12.59             -5.36
```

```
Soubhi — mavproxy.py --master=127.0.0.1:14550 --sitl 127.0.0.1:5501 --out 127.0.0.1:14559 --out 127.0.0.1:14558 — 114...
Last login: Fri Aug 24 15:43:17 on ttys007
soubhis-mbp:~ Soubhi$ mavproxy.py --master=127.0.0.1:14550 --sitl 127.0.0.1:5501 --out 127.0.0.1:14559 --out 127.0
.0.1:14558
Connect 127.0.0.1:14550 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from 127.0.0.1:14550
 MAV> fence breach
online system 1
STABILIZE> Mode STABILIZE
APM: ArduCopter V3.6-dev (62cf4565)
APM: Frame: QUAD
Flight battery 100 percent
Received 917 parameters
Saved 917 parameters to mav.parm
MAV>
STABILIZE> No waypoint load started
APM: ArduCopter V3.6-dev (62cf4565)
APM: Frame: QUAD
STABILIZE> 
```

```
ardupilot — ssh • vagrant ssh -c sim_vehicle.py -j 2 -v ArduCopter  --console --map — 114×40
    self.module('console').add_menu(self.menu)
AttributeError: SimpleConsole instance has no attribute 'set_menu'
Traceback (most recent call last):
  File "/usr/local/bin/mavproxy.py", line 763, in periodic_tasks
    m.idle_task()
  File "/usr/local/lib/python2.7/dist-packages/MAVProxy/modules/mavproxy_rally.py", line 49, in idle_task
    self.module('console').add_menu(self.menu)
AttributeError: SimpleConsole instance has no attribute 'set_menu'
Traceback (most recent call last):
  File "/usr/local/bin/mavproxy.py", line 763, in periodic_tasks
    m.idle_task()
  File "/usr/local/lib/python2.7/dist-packages/MAVProxy/modules/mavproxy_fence.py", line 54, in idle_task
    self.module('console').add_menu(self.menu)
AttributeError: SimpleConsole instance has no attribute 'set_menu'
Unable to access the X Display, is $DISPLAY set properly?
APM: ArduCopter V3.6-dev (62cf4565)
APM: Frame: QUAD
Unloaded module map
APM: Barometer calibration complete
Init Gyro**Received 917 parameters
Saved 917 parameters to mav.parm
*

Ready to FLY fence breach
APM: GPS 1: detected as u-blox at 115200 baud
APM: EKF2 IMU0 initial yaw alignment complete
APM: EKF2 IMU1 initial yaw alignment complete
APM: EKF2 IMU0 tilt alignment complete
APM: EKF2 IMU1 tilt alignment complete
APM: EKF2 IMU0 Origin set to GPS
APM: EKF2 IMU1 Origin set to GPS
APM: EKF2 IMU0 is using GPS
APM: EKF2 IMU1 is using GPS
Flight battery 100 percent
APM: ArduCopter V3.6-dev (62cf4565)
APM: Frame: QUAD
MAV> No waypoint load started
APM: ArduCopter V3.6-dev (62cf4565)
APM: Frame: QUAD
MAV> 
```

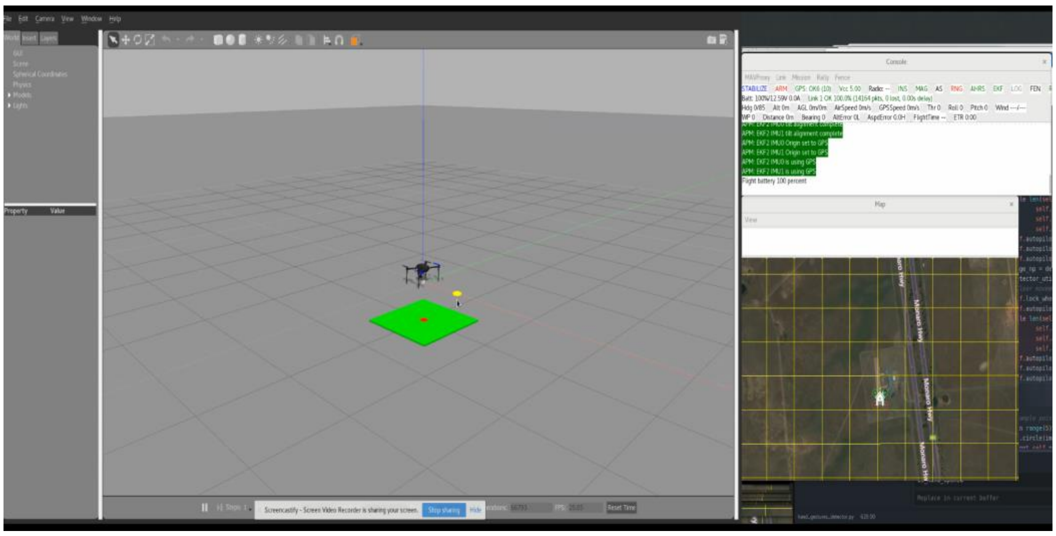
Figure 25. SITL Copter command line, MAVProxy and APM Planner2.

4.1.8. *Dronekit [37]*

Dronekit is a simple python library used to build applications for drones. It depends on MAVLink messages and is compatible with Ardupilot. It is under the Apache v2 License [38]. Some of its features are used in this module to build the drone controller module. The complete module is wrapped into a simple autopilot python module [39].

4.2. Gazebo Simulator [40]

Gazebo is a free software simulator used in the robotics field of study. It is extremely beneficial as a 3D full customized simulator. It is possible to build the whole environment and add a robot to see how it is going to act and behave before implementing it on real-world robot. One can also add a camera to a robot and work on the collision avoidance or autonomous navigation algorithm. A large number of plugins and sensors are supported. For example, it is possible to add the Realsense 3D vision sensor [41] to the drone and add full autonomous navigation to this system. Gazebo is used in this project as a complementary to SITL because it is easier to test and visualize the system using 3D simulator. There is an Ardupilot plugin and 3DR IRIS copter model [42] which is used in this project, as shown in Figure 26.


File Edit Camera View Window Help
Console
MAVProxy Link Mission Rally Fence
Map
View

```
STABILIZE  ARM  GPS: OK6 (10)  Vcc 5.00  Radio: --  INS  MAG  AS  RNG  AHRS  EKF  LOG  FEN
Batt: 100%/12.59V 0.0A   Link 1 OK 100.0% (14164 pkts, 0 lost, 0.00s delay)
Hdg 0/85  Alt 0m  AGL 0m/0m  AirSpeed 0m/s  GPSSpeed 0m/s  Thr 0  Roll 0  Pitch 0  Wind ---/---
WP 0  Distance 0m  Bearing 0  AltError 0L  AspdError 0.0H  FlightTime --  ETR 0:00
APM: EKF2 IMU0 tilt alignment complete
APM: EKF2 IMU1 tilt alignment complete
APM: EKF2 IMU0 Origin set to GPS
APM: EKF2 IMU1 Origin set to GPS
APM: EKF2 IMU0 is using GPS
APM: EKF2 IMU1 is using GPS
Flight battery 100 percent
```

Figure 26. Gazebo simulator side by side with SITL.

4.3. Drone Controller Functionalities

This module consists of an interface that wraps all the functionalities which are required from the drone in this system. For example, in this interface, there is a function called "takeoff" which takes one parameter, "target_altitude," and indicates the height that the drone should reach after taking off. Inside that function, a takeoff MAVLink packet is generated and sent to the connected drone. After that, a feedback/acknowledgment will come back from the drone to indicate whether the command succeeded or not. The available functions in this interface are:

1. Connect: to connect to a drone with a specific IP and Port.
2. Arm: to turn on (arm) the motors if the drone is not armed and but ready. If the flight mode requires some conditions like GPS Lock and they are not met, the drone will reject the command.
3. Disarm: to turn off (disarm) the motors if the drone is on the ground. If it is still in air, it will reject the command.
4. Takeoff: to takeoff to a specific altitude.
5. Change_flight_mode: to change the flight mode for the connected drone.
6. Move: to navigate the drone in a NED frame (North, East, Down). The negative values mean the opposite direction.
7. Land: to land the drone.
8. Listener_heartbeat: get general information from the incoming heartbeat packet of the drone.
9. Listener_ack: to get feedback/acknowledgment about the commands sent to the drone.

Chapter 5: Experiments and Results

After the description of the main three modules in this work, it is possible to draw a general block diagram for the proposed system (Figure 27). The input for the system is an image, so a camera is needed to be connected to the device that runs the system.

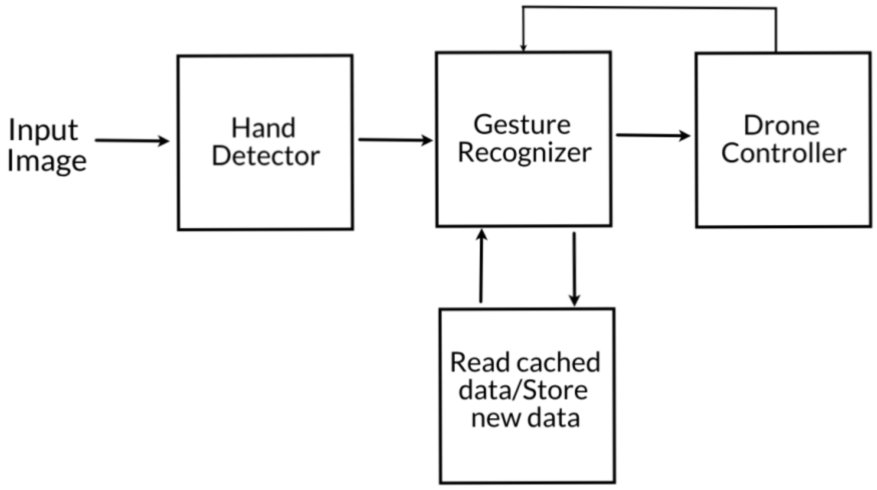


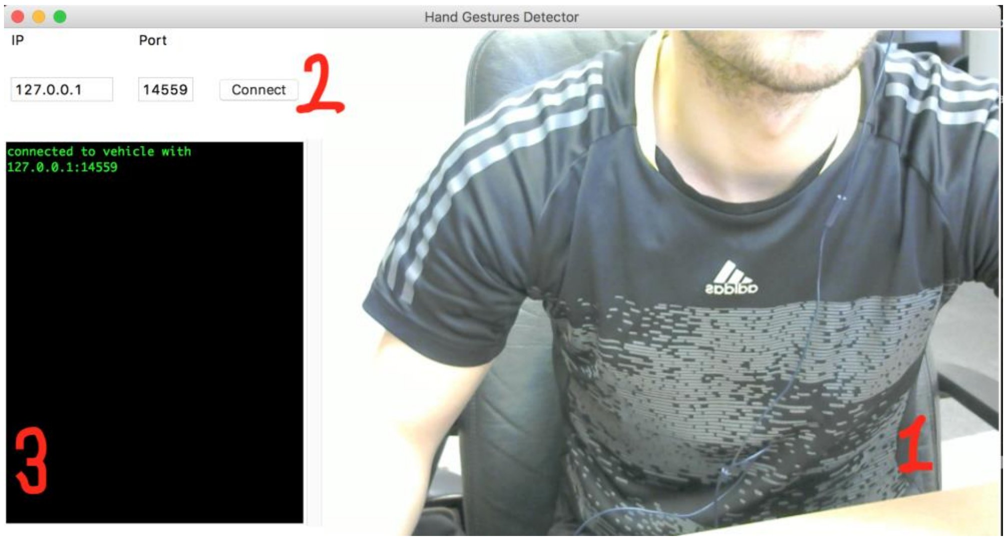
Figure 27. General block diagram for the proposed system.

5.1. System GUI (Graphical User Interface)

All the modules are implemented in a simple GUI application, as shown in Figure 28. The application consists of the following sections:

1. Live streaming from the output of the first two modules (hand detector and gesture recognizer). A thin box will be drawn around each hand which appears in the image, with a small label indicating the class of the hand. Some animated figures, like the virtual wheel and arrows, are shown when the system recognizes one of the defined gestures.
2. IP and Port inputs to connect to a specific drone. The SITL simulator should be run before trying to connect. Otherwise, the connection will fail.

3. Log: shows detailed information about what is happening in the system, such as the connection to the SITL and whether it succeeded or failed.



```
connected to vehicle with
127.0.0.1:14559
```

Figure 28. Application GUI.

5.2. Simulator Tests and Results

As mentioned earlier, the SITL and Gazebo simulators were used to test the whole system. MAVProxy was used to forward the messages to the application and any other GCS. Figure 29 shows the whole test environments:

The Gazebo simulator on the left, the SITL simulator in the middle, and the system GUI on the top right.

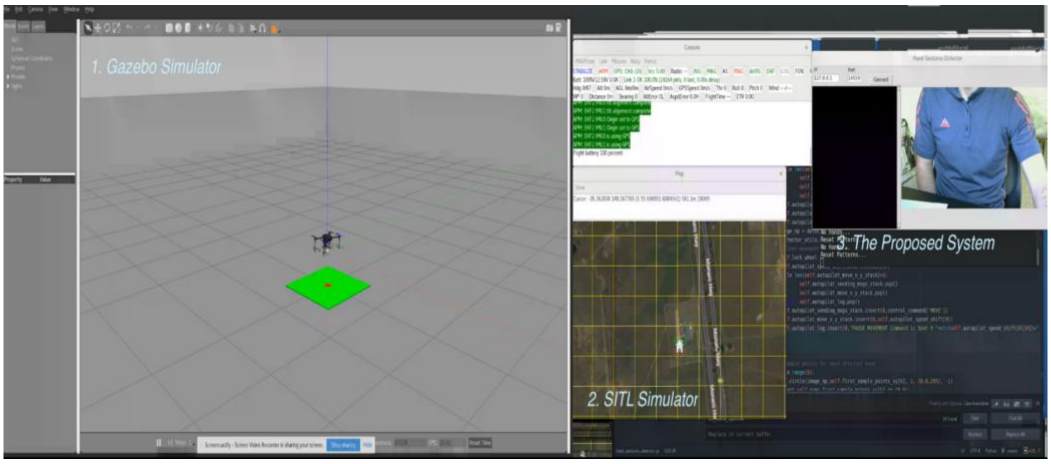


Figure 29. Test environments.

First, the system should connect to the drone or SITL by entering its IP and Port. The system can send navigation commands to the drone (here SITL) and, at the same time, it can receive and understand messages coming from drone. All the messages are logged and shown in the log area. The messages that were sent to the drone appear in white font, while the incoming ones appear in green font.

Figure 30 shows that the drone after executing the Arm and Takeoff gesture. The log area shows that the system is connected to the drone with IP: 127.0.0.1 and Port:14559, and it is in green font, which means it came from the drone controller module as a feedback and the connection succeeded. Then, an Arm message was sent, which shows in white font. After that, again many messages came from the drone to indicate that the altitude was increasing until the drone reached one-meter altitude which is the default value the author has set for the Arm and Takeoff gesture.

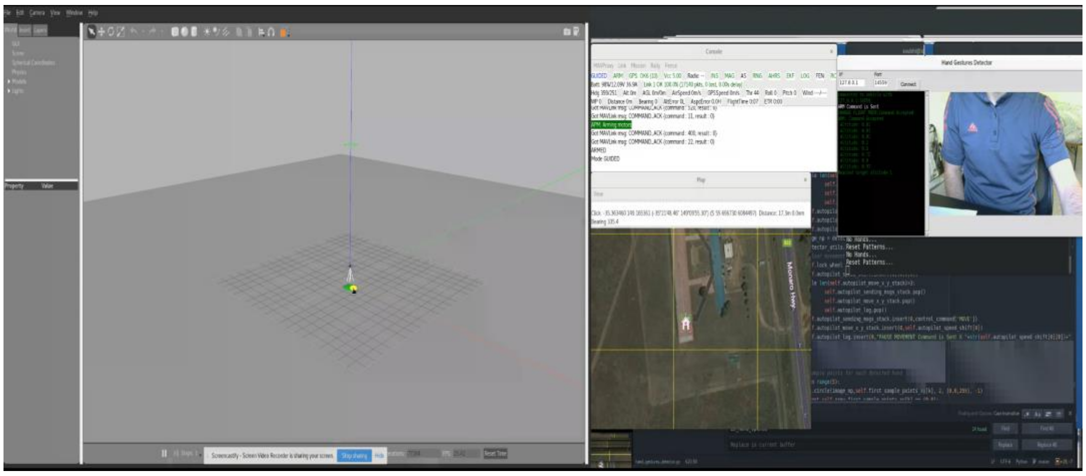


Figure 30. The system after Arm and Takeoff gesture was executed.

In Figure 31, the drone is increasing its altitude and the corresponding gesture for that command is shown in the system application GUI with animated arrow. The console window for SITL shows in the first line that the flight mode is GUIDED and in the third line the altitude is 4. So, the altitude increased from 1-meter to 4-meters.

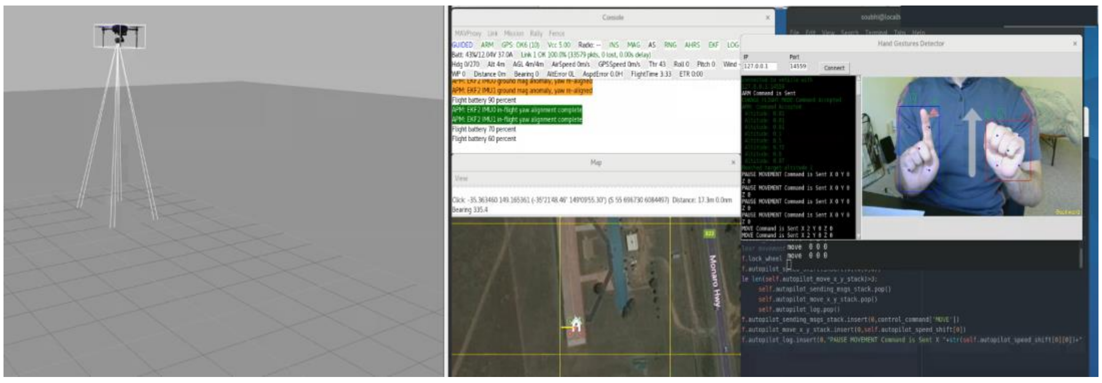


Figure 31. Moving up gesture and the drone simulator is moving up.

Chapter 6: Conclusion and Future Work

6.1. Conclusion

This thesis presented research that was done in the area of Human-Drone Interface (HDI) at the University of Oklahoma, Tulsa Campus. The main goal of this work was to build a novel and robust system to control the drone using hand gestures in real-time with high efficiency. The gestures can be performed using one or two hands. The proposed system consists of three main modules:

1. Hand Detector.
2. Gesture Recognizer.
3. Drone Controller.

A deep learning method was used in the first module. The module was trained using a dataset which was acquired and processed for this project. A Single Shot MultiBox Detect (SSD) network was used to detect the hands. The second module utilized image processing methods and it was totally dynamic, which meant it was very simple to add new gestures with no need to re-train the neural network and go back to the first module. The last module, Drone Controller, depends on Ardupilot which is one of the most popular autopilot systems. The three modules communicate with each other smoothly.

The tests for the whole system were applied using simulators. The first simulator was the Ardupilot SITL (Software in The Loop) simulator and the other one was the Gazebo Robot 3D Simulator.

This novel work utilizes hand gestures as an effective HDI method to control the movement of the drones. A deep learning neural network detector is a practical solution to detect the hands and it can substitute any additional hand detection hardware. The gestures recognition is defined using image processing, independent of deep learning detector. Consequently, defining a new gesture is an easy process without the need of retraining the deep learning model. The proposed system can operate with any drone supporting the MAVLink protocol, which makes the system compatible with most of the drones available in the market nowadays.

6.2. Future Work

The most important factor to move a product from a research area to industry is to prove its efficacy and reliability. So, the next stage of this project will be to do accurate studies on the performance and feedback from the drone's users. The following are the main points for future work:

- Trying other types of neural network structures and compare their performance to the one used in this work. One of the suggested networks is the Long Shot Term Memory (LSTM) [43] which is like a network with memory.
- Collecting more data and it should be balanced, i.e., collected from a variety of different environments and from different people.
- Moving the work from Python to C++ to speed up the processing and compare the performance accurately. Trying to parallel the processing and optimize it.
- Conducting more tests on real drones. The tests should be done using different types of drones in variety of environments, both indoor and outdoor, and by different users.

References

[1]. Federal Aviation Administration Releases Aerospace Forecast, “https://www.faa.gov/news/updates/?newsId=89870”, [Online]. [Accessed August 2018]

[2]. Alex Roney Mathew, A. Al Hajj and A. Al Abri, "Human-Computer Interaction (HCI): An overview," in IEEE International Conference on Computer Science and Automation Engineering, Shanghai, China, 2011, pp. 99-100.

[3]. ACM/IEEE International Conference on Human Robot Interaction, “http://humanrobotinteraction.org/2018/”, [Online]. [Accessed August 2018]

[4]. R. Cauchard, Jessica & E, Jane & Y. Zhai, Kevin & Landay, James, “Drone & me: an exploration into natural human-drone interaction,” in ACM Inernational Joint Conference, Osaka, Japan, 2015, 361-365, 10.1145/2750858.2805823.

[5]. Ma, Lu & Lung Cheng, Lee, “Studies of AR Drone on Gesture Control,” in 3rd International Conference on Materials Engineering, Manufacturing Technology and Control (ICMEMTC), Taiyuan, China, 2016, 10.2991/icmemtc-16.2016.350.

[6]. A. Sarkar, K. A. Patel, R. K. G. Ram and G. K. Capoor, "Gesture control of drone using a motion controller," in International Conference on Industrial Informatics and Computer Systems (CIICS), Sharjah, UAE, 2016, pp. 1-5.

[7]. Kathiravan Natarajan, Truong-Huy D. Nguyen, Mutlu Mete, “Hand Gesture Controlled Drones: An Open Source Library,” in 1st International Conference on Data Intelligence and Security (ICDIS), Texas, USA, 2018, 1803.10344

[8]. Microsoft Kinect sensor, "https://developer.microsoft.com/en-us/windows/kinect", [Online]. [Accessed August 2018].

[9]. AR Drone, “https://www.parrot.com/us/ “, [Online]. [Accessed August 2018]

[10]. Leap Motion Controller. "https://www.leapmotion.com". [Online]. [Accessed August 2018].

[11]. A. Haar, “Zur theorie der orthogonalen functionensysteme. inaugural,” Ph.D. dissertation, Dissertation, Gottingen, 1909, 1–49. Math. Annal. 69, 331–271, 1910.

[12]. Xiaogang Cheng, Qi Ge, Shipeng Xie, Guijin Tang, Haibo Li, “UAV Gesture Interaction Design for Volumetric Surveillance,Procedia Manufacturing,” in 6th

International Conference on Applied Human Factors and Ergonomics (AHFE), Las Vegas, NV, USA, 2015, Volume 3,2015,Pages 6639-6643, ISSN 2351-9789

[13]. P. Molchanov, X. Yang, S. Gupta, K. Kim, S. Tyree and J. Kautz, "Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D Convolutional Neural Networks," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 4207-4215. doi: 10.1109/CVPR.2016.456

[14]. Ardupilot Autopilot, “http://ardupilot.org/”, [Online]. [Accessed September 2018].

[15]. PX4 Autopilot, “http://px4.io/”, [Online]. [Accessed September 2018].

[16]. MAVLink Protocol “https://mavlink.io/en/messages/”, [Online]. [Accessed August 2018].

[17]. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in NIPS’12 Proceedings of the .25th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 2012, pages1097–1105.

[18]. K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in ICLR, San Diego, CA, USA, 2015, arXiv:1409.1556.

[19]. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, arXiv:1512.03385.

[20]. Redmon, Joseph & Divvala, Santosh & Girshick, Ross & Farhadi, Ali, “You Only Look Once: Unified, Real-Time Object Detection,” in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, doi: 10.1109/CVPR.2016.91.

[21]. Liu, Wei & Anguelov, Dragomir & Erhan, Dumitru & Szegedy, Christian & Reed, Scott, “SSD: Single Shot MultiBox Detector,” in the 14th European Conference on Computer Vision (ECCV), Amsterdam, Netherlands, 2016, doi: 10.1007/978-3-319-46448-0_2.

[22]. Everingham, M. and Eslami, S. M. A. and Van~Gool, L. and Williams, C. K. I. and Winn, J. and Zisserman, A, “The Pascal Visual Object Classes Challenge: A Retrospective,” in International Journal of Computer Vision (IJCV), Hingham, MA, USA, 2015, doi: 10.1007/s11263-014-0733-5.

[23]. TensorFlow, “https://www.tensorflow.org/ “, [Online]. [Accessed August 2018].

[24]. Debian Operating System, “https://www.debian.org/”, [Online]. [Accessed August 2018].

[25]. Pixhawk, “https://pixhawk.org/”, [Online]. [Accessed September 2018].

[26]. Inertial Measurement Unit (IMU), ”https://en.wikipedia.org/wiki/Inertial_measurement_unit”, [Online]. [Accessed September 2018].

[27]. PX4 Flight Stack, “https://dev.px4.io/en/concept/architecture.html”, [Online]. [Accessed September 2018].

[28]. GPL License, “https://www.gnu.org/licenses/gpl-3.0.en.html”, [Online]. [Accessed September 2018].

[29]. BSD License, “https://opensource.org/licenses/BSD-3-Clause”, [Online]. [Accessed September 2018].

[30]. INAV Autopilot, “https://github.com/iNavFlight/inav/wiki”, [Online]. [Accessed September 2018].

[31]. APM Planner2 GCS, “http://ardupilot.org/planner2/”, [Online]. [Accessed September 2018].

[32]. Mission Planner GCS, “http://ardupilot.org/planner/”, [Online]. [Accessed September 2018].

[33]. PX4FLOW Smart Camera, “https://docs.px4.io/en/sensor/px4flow.html”, [Online]. [Accessed September 2018].

[34]. Ardupilot Copter Flight Modes, “http://ardupilot.org/copter/docs/flight-modes.html”, [Online]. [Accessed September 2018].

[35]. SITL Simulator (Software in The Loop), “http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html”, [Online]. [Accessed September 2018].

[36]. MAVProxy GCS, “http://ardupilot.github.io/MAVProxy/html/index.html”, [Online]. [Accessed September 2018].

[37]. Dronekit, “http://dronekit.io/”, [Online]. [Accessed September 2018].

[38]. Apache License Version 2.0, “https://www.apache.org/licenses/LICENSE-2.0” [Online]. [Accessed September 2018].

[39]. Autopilot Module, “github.com/SubhiH/Autopilot/blob/master/autopilot.py”, [Online]. [Accessed September 2018].

[40]. Gazebo Robot Simulator, “http://gazebosim.org/”, [Online]. [Accessed September 2018].

[41]. Intel Realsense Sensor, “www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html”, [Online]. [Accessed September 2018].

[42]. Ardupilot Gazebo Plugin & Model, “www.github.com/swiftgust/ardupilot_gazebo”, [Online]. [Accessed September 2018].

[43]. Hochreiter, Sepp & Schmidhuber, Jürgen, “Long Short-term Memory,” in Neural computation. doi: 10.1162/neco.1997.9.8.1735.

Appendix A – Drone Controller

```
from dronekit import connect, VehicleMode
from pymavlink import mavutil
import time
import mock


FLIGHT_MODES = ['STABILIZE', 'LAND', 'OF_LOITER', 'RTL', 'DRIFT',
                'FLIP', 'AUTOTUNE', 'BRAKE','GUIDED_NOGPS', 'AVOID_ADSB',
                'POSITION', 'SPORT', 'FLOWHOLD', 'POSHOLD','AUTO', 'GUIDED',
                 'ACRO', 'SMART_RTL', 'ALT_HOLD', 'LOITER', 'CIRCLE', 'THROW']

ACK_RESULT_TYPE = {0:'Command Accepted',
                1:'Command Rejected',
                2:'Access Denied',
                3:'Command Not Supported',
                4:'Command Failed'}


MAVLINK_COMMAND_ID = {'ARM/DISARM':400,
                  'TAKEOFF':22,
                  'CHANGE_FLIGHT_MODE':11}

is_waiting_for_ack = False
ack_command_id = None
ack_command_result = None
is_connected = False
is_armed = False
feed_back_stack = []
class autopilot:
    """autupilot class has many functions to connect and control the drone.
    Attributes:
        is_waiting_for_ack (boolean): boolean variable to manage the acknowledgment from the vehicle`.
        ack_command_id (int): id of the command to wait for its acknowledgment`.
        ack_command_result (int): id of the result type ACK_RESULT_TYPE`.
        is_connected (boolean): flag to check if the vehicle is connected or not`.
        is_armed (boolean): flag to check if the vehicle is armed or not`.
    """
```

```
def __init__(self):
    self.autopilot = None

def connect(self,ip,port):
    """Starts a connection with a vehicle.
    Args:
        ip (str): the ip for the vehicle like `127.0.0.1`.
        port (int): the port for the vehicle like 14450
    """
    global is_connected
    global feed_back_stack
    self.autopilot = connect(str(ip)+":"+str(port), wait_ready=True)
    if not self.autopilot is None:
        is_connected = True
        feed_back_stack.insert(0,'connected to vehicle with '+str(ip)+':'+str(port))

        '''
        Receives COMMAND_ACK mavlink packets
        '''
        @self.autopilot.on_message('COMMAND_ACK')
        def listener_ack(self, name, message):
            global is_waiting_for_ack
            global ack_command_id
            global ack_command_result
            if is_waiting_for_ack:
                if message.command == ack_command_id:
                    ack_command_result = message.result
                    is_waiting_for_ack = False
                else:
                    print 'message: %s' % message,message.command,ack_command_id


        '''
        Receives HEARTBEAT mavlink packets
        '''
        @self.autopilot.on_message('HEARTBEAT')
        def listener_heartbeat(self, name, message):
            global is_armed
            if (message.base_mode & 0b10000000)==128:
                is_armed = True
            else:
                is_armed = False


    else:
        self.is_connected = False


def takeoff(self,altitude):
    """Sends takeoff command to connected vehicle.
    Takeooff will fail in any of these situations:
        1.  There is no connection with a vehicle.
        2.  The vehicle is disarmed.
        3.  The vehicle is already above the ground.
    Args:
        altitude (str): the target altitude.
    """
    global is_connected
    global is_armed
    global is_waiting_for_ack
    global ack_command_id
    global ack_command_result
    global feed_back_stack
    if is_connected and is_armed:
        tick = 0
        is_waiting_for_ack = True
        ack_command_id = MAVLINK_COMMAND_ID['TAKEOFF']
        self.autopilot.simple_takeoff(altitude)
        while is_waiting_for_ack:
            tick=1
            time.sleep(0.5)
            if tick>5:
                feed_back_stack.insert(0,'TAKEOFF Time Out ')
                print 'TAKEOFF Time Out!'
                return
            print 'TAKEOFF: ',ACK_RESULT_TYPE[ack_command_result]

            if ack_command_result==0:
                while True:
                    feed_back_stack.insert(0," Altitude: "+str(self.autopilot.location.global_relative_frame.alt))
                    print " Altitude: ", self.autopilot.location.global_relative_frame.alt
                    #Break and return from function just below target altitude.
                    if self.autopilot.location.global_relative_frame.alt>=altitude*0.95:
                        feed_back_stack.insert(0,'Reached target altitude '+str(altitude))
                        print "Reached target altitude"
                        break
                    time.sleep(1)
            elif ack_command_result==4 and self.autopilot.location.global_relative_frame.alt>1:
                feed_back_stack.insert(0,'The Vehicle is already above the ground!, use MOVE command...
')
                print 'The Vehicle is already above the ground!, use MOVE command... '
                return
        else:
            feed_back_stack.insert(0,'Vehicle is NOT armed ')
            print 'Vehicle is NOT armed!'


    def change_flight_mode(self,flight_mode_name):
        """Changes the flight mode for the connected Vehicle.
        change_flight_mode will fail of there is no connection with a vehicle.
        Args:
            flight_mode_name (str): the name of flight_mode.
        """
        global is_connected
        global is_armed
        global is_waiting_for_ack
        global ack_command_id
        global ack_command_result
        global feed_back_stack
        if is_connected:
            if flight_mode_name.upper() in FLIGHT_MODES:
                tick = 0
                is_waiting_for_ack = True
                ack_command_id = MAVLINK_COMMAND_ID['CHANGE_FLIGHT_MODE']
                self.autopilot.mode = VehicleMode(flight_mode_name.upper())
                while is_waiting_for_ack:
                    tick=1
                    time.sleep(0.5)
                    if tick>5:
                        print 'CHANGE_FLIGHT_MODE Time Out!'
                        feed_back_stack.insert(0,'CHANGE_FLIGHT_MODE Time Out ')
                        return
                print 'CHANGE_FLIGHT_MODE:',ACK_RESULT_TYPE[ack_command_result]

feed_back_stack.insert(0,'CHANGE_FLIGHT_MODE:'+str(ACK_RESULT_TYPE[ack_command_result]
))
            else:
                feed_back_stack.insert(0,'The available flight modes:'+str(FLIGHT_MODES))
                print 'The available flight modes: ',FLIGHT_MODES
        else:
            feed_back_stack.insert(0,'There is no connection with any vehicle!')
            print 'There is no connection with any vehicle!'



    def arm(self):
        """arms/turns on the motors.
        arm will fail of there is no connection with a vehicle.
        """
        global is_connected
        global is_armed
        global is_waiting_for_ack
        global ack_command_id
        global ack_command_result
        global feed_back_stack
        if self.autopilot is None:
            print "There is NO connection with any vehicle!"
            feed_back_stack.insert(0,'There is NO connection with any vehicle!')
            return

        if is_armed:
            print 'The vehicle is already armed!'
            feed_back_stack.insert(0,'The vehicle is already armed!')
    else:
        tick = 0
        is_waiting_for_ack = True
        ack_command_id = MAVLINK_COMMAND_ID['ARM/DISARM']
        self.autopilot.armed = True
        tick=0
        while is_waiting_for_ack:
            tick=1
            time.sleep(0.5)
            if tick>5:
                print 'Time Out!'
                feed_back_stack.insert(0,'Arm Time Out!')
                return
        if ack_command_result==0:
            is_armed = True
        print 'ARM: ',ACK_RESULT_TYPE[ack_command_result]
        feed_back_stack.insert(0,'ARM: '+str(ACK_RESULT_TYPE[ack_command_result]))

def disarm(self):
    """disarms/turns off the motors.
    disarm will fail of there is no connection with a vehicle.
    """
    global is_armed
    global ack_command_id
    if not is_armed:
        print 'The vehicle is already disarmed!'
    else:
        ack_command_id = MAVLINK_COMMAND_ID['ARM/DISARM']
        self.autopilot.armed = False
        tick=0
        while is_waiting_for_ack:
            tick=1
            time.sleep(0.5)
            if tick>5:
                print 'DISARM Time Out!'
                return
        if ack_command_result==0:
            is_armed = False
        print 'DISARM: ',ACK_RESULT_TYPE[ack_command_result]
        #send mavlink packet


def move(self,velocity_x, velocity_y, velocity_z,duration):
    """
    Move vehicle in direction based on specified velocity vectors.
    """
    msg = self.autopilot.message_factory.set_position_target_local_ned_encode(
        0,       # time_boot_ms (not used)
        0, 0,    # target system, target component
        mavutil.mavlink.MAV_FRAME_LOCAL_NED, # frame
        0b0000111111000111, # type_mask (only speeds enabled)
        0, 0, 0, # x, y, z positions (not used)
        velocity_x, velocity_y, velocity_z, # x, y, z velocity in m/s
        0, 0, 0, # x, y, z acceleration (not supported yet, ignored in GCS_Mavlink)
        0, 0)    # yaw, yaw_rate (not supported yet, ignored in GCS_Mavlink)


    # send command to vehicle on 1 Hz cycle
    for x in range(0, duration):
        self.autopilot.send_mavlink(msg)
        time.sleep(1)

def land(self):
    self.change_flight_mode('LAND')

def pop_from_feedback_stack(self):
    global feed_back_stack
    if len(feed_back_stack)>0:
        return feed_back_stack.pop()
    else:
        return None
```

Glossary

1. FAA: Federal Aviation Administration.
2. UAV: Unmanned Aerial Vehicle.
3. HCI: Human-Computer Interface
4. HRI: Human-Robot Interface.
5. HDI: Human-Drone Interface.
6. GUI: Graphical User Interface.
7. ROS: Robotic Operating System.
8. 3D-CNN: Recurrent Three-dimensional Convolutional Neural Network.
9. CTC: Connectionist Temporal Classification.
10. MAVLink: Micro Air Vehicle Communication protocol.
11. RGB: Red, Green and Blue.
12. HSV: Hue, Saturation and Value.
13. HSL: Hue, Saturation and Lightness.
14. YOLO: You Only Look Once.
15. SSD: Single Shot MultiBox Detector.
16. FC: Flight Controller.
17. IMU: Inertial Measurement Unit.
18. GPL: General Public License.
19. BSD: Berkeley Software Distribution.
20. RC: Radio Control.
21. ESC: Electrical Speed Controller.
22. GPS: Global Positioning System.
23. RTL: Return to Lunch.

24. SITL: Software in The Loop.

25. UDP: User Datagram Protocol.

26. TCP: Transmission Control Protocol.

27. LSTM: Long Short-Term Memory.