Department of Electrical Engineering
University Of Evry Val d'Essone , France
Poznan university of technology, Poland
University Of Paris Saclay, France

Master's Thesis

Deployment of drone demonstators - Automatic take-off and landing of a drone on a mobile robot.

Submitted in fulfillment of
the requirements for the award of the degree of

Masters Electronique, Energie Electrique, Automatique :
Smart Aerospace and Autonomous Systems

Submitted by

SUSHIL KUMAR SHARMA

Under the supervision of:

Prof.Yasmina Bestaoui SEBBANE
Prof.Naima AITOUFROUKH

In:

IBISC Laboratory

POLITECHNIKA POZNAŃSKA
Poznan University of Technology

université
PARIS-SACLAY

September 2017

UNIVERSITY OF PARIS SACLAY

Faculty of Engineering Sciences
Department of Electrical Engineering

Master's Thesis

by SUSHIL KUMAR SHARMA

Abstract

The aim of this work is the control of a quadrotor with the goal of spatial stabilization. Stability is of great importance when mounting additional devices such as HD camera, transmitter and dipole antenna in the drone. The control approach searches for the autonomy of the robot taking into account internal sensors. The vision-based method is chosen for landing target detection.

By computer vision, we estimate the relative position of the quadrotor and then further control and its automatic landing. For the automatic landing of the system consisting of node.js, which allows our autonomous system to use some specific modules. An OpenCV algorithm is developed to estimate a the relative position and orientation of a quadrotor using the landpad marker that has the black and white circle called the Oriental Roundel. Finally, some tests are performed on the simulator in which an automatic landing of the frame is achieved. An automatic take-off and landing algorithm, we have implemented node.js to make a mission successful in a short time. The copter node programming libraries must manage the drone. The drone is able to track a mobile robot independently at a constant speed and land on it.

Last but not least, there is another scenario that we use the robot operating system (ROS) with artificial neural networks (ANNs) to control the drone and to stabilize the launch and landing around a fixed point. When the drone lifts, it shows a boustrophedon path with a synthetic data system

Acknowledgement

First and foremost I would like to express my sincerely grateful to my departmental supervisor **Prof.Yasmina Bestaoui SEBBANE** and **Prof.Naima AITOUFROUKH**. Their constant guidance and advice played the vital role in making the execution of the report.Their always gave me their precious suggestion that was crucial in making this report as flawless as possible; Their continuous encouragements and enthusiasm were of utmost important especially at times,I faced difficulties during my internship.

I also an acknowledge the financial support if IBISC laboratory. I am grateful to hem for funding this project without which the completion would otherwise be impossible.
As my master thesis was part of a bigger project, I am equally thankful to all the Aerospace PhD scholars in particular; Prof Elmehdi Zareb, Prof Redouane Ayad, Mr.The Hung Pham.

I would further like to thank my family and relatives for their moral support. Finally, I would like to thank all the people around me who have shown me support and love in pursuit of my studies these are in particular: Jibin, Rayene, Massi and Hoàng Anh Pham all my classmates at Poznan Polytechnic University and at the University of Paris Saclay and everyone else I am forever grateful.

"Any effort becomes successful when there is the effect of synergy the concept that two and two make more than four"

SUSHIL KUMAR SHARMA
University of Paris Saclay
September 2017

Contents

List of Figures

List of Tables

Chapter 1

General Introduction

1.1 State of the Art

Over the past decade, technological advances and numerous applications have aroused increasing interest in aerial robotics. Small vehicles unmanned aerial vehicles have obvious commercial applications in inspection Structures such as bridges, dams or high-tension lines, exploration Hazardous environments such as burning forests or radioactive areas, Military reconnaissance missions, etc [1].

The drones can also be used indoor applications such as inspection of mines or large buildings, the search for survivors in a dangerous environment and confined as a exploded building or a building affected by an earthquake [1][2]. The difficulty of navigating autonomous way in these very congested environments then occurs. Functions Obstacle avoidance, take-off and automatic landing in environments unknown are obviously necessary to ensure the safety of the craft and the success of the mission. At the same time, many studies have focused on the study of behaviour insects. In particular, attention was paid to the insect's responses to Barriers, the techniques used to follow a corridor or their strategies landing. It is known that insects such as the bee or the fly are very sensitive to movements created by their own displacement with respect to the environment Or by the movement of objects with respect to them. It is these perceived movements by the facets of their eyes that enable them to navigate effectively in their middle. As a result of these studies, behavioural control strategies Insects were born for the control of robots.

Since 2003, unmanned aerial vehicle has developed a vertical take-off and landing vehicle with four rotors. The attitude control, navigation based on GPS and The visual servoing on a known target has already been tested and validated on the prototype.

Unfortunately, the inertial measurements alone are not sufficient for a reliable estimation of the vehicle speed. On the other hand, in an urban or inner environment, GPS is not available, which later makes it impossible to measure the proximity of obstacles. Finally, the vision control techniques often assume that the environment is known or partially known to extract an image of the velocity. Since the Quadcapter is used, there are many tutorials and guides on how to acquire, construct, control, measure and use this platform, For example, [2]-[3]. Even within the same type of structure, there are many ways, especially when referring to open-source autopilot, as specified in cite c. Despite the variety of options, the goal of all of them is the same - fly with a quadcopter and be able to easily control the 6 DOF with only four variables.

The Institute of Systems and Robotics, Research and Rescue Project of the Superior Technical Institute [2] It is one of the applications where Vicon systems can not be used. In this kind of environment there is a need for another system that is capable of providing a good estimate of the (relative) position and orientation of the quadcopter. There are some options, related to Augmented Reality (RA) that can be is there a good solution, especially since these types of systems require a much simpler structure and calibration: a regular calibrated camera (and a single), an AR software and a printed marker landing pad, It could be printed on the top of the RAPOSA robot of the same project. There are currently these are some very good solutions for AR processing, namely ArUco [4] (described in Section 3.2.1). These systems are designed to apply computer vision techniques calculate the position and orientation of a marker board with respect to the camera. This type of solution is very interesting because it is performed in real time and can therefore be used in an on-board processing environment. Although the sensor systems set here represent the absolute pose of the quadcopter[6].

On the other hand, the field of robotics has grown in recent years and currently has a very important role, with a growing investment for its application potential. A special field of robotics they are air robots, mainly unmanned aerial vehicles (UAVs). A UAV that is commonly used is the AR.Drone 2.0. This drone is widespread because it is versatile and has a relatively simple model (assuming a series of simplistic assumptions and valid [3]-[6].

One of the problems of this type of vehicles is the autonomy. The energy required to perform the flight is stored in batteries that are heavy and thus limit the autonomy of the robot. Even with fully charged batteries and in a controlled environment, the quadrotor can fly only a few minutes (The quadrotor used in this work can fly up to 20 minutes). If you have a little more weight (like a camera or some kind of gauge) this time will be gradually reduced. It is obviously the need to charge the vehicle batteries very often.

Today this burden has to be done with direct people interaction, that is, someone must manually land the quadrotor.

1.2 Objective and Contribution

The aim of this work is to make the drone an autonomous landing on a mobile platform (turtlebot). It is also a prerequisite for the drone to perform this task without relying on a highly calibrated environment using only on-board sensors to achieve this goal. Specially it uses ardrone 2.0, which is equipped with a regular camera (bottom) and a board processing unit. To identify the runway, a marker board or landing pad (Oriental roundel) is used. The image captured by the camera is processed and the markers must be recognized and identified. Computer-controlled visual algorithms are used to estimate the quadrant's pose. This position estimate is used to control the rotation of the quadrotor and to perform the necessary maneuvers to achieve the desired goal: to make the quadrotor autonomously land on the mobile robot. In the meantime, the modification of AR.Drone2.0 is with some additional devices that allow to use a video or images for image analysis.

The contribution of the work making autonomous landing algorithm on a moving robot using image processing algorithms, Finally, make a video-related to work that will help the next generation. If we compare a different work and my work, I would say that it is really affordable because the price AR.Drone2.0 is cheap as compare other luxurious drone like phantom drone, bebop, etc. so we decided to modify Ardrone 2.0 to do any type of task. As we can see Chapter 7 At the end of this work the following steps were carried out:

- Implementation of computer-controlled visual algorithms and reliability analysis
- Autonomous landing algorithm
- Fully autonomous concept implementation, without external devices
- Modify AR.Drone with highly standard camera with image analysis.
- Record a high definition video with additional device like Blackmagic device

1.3 Thesis organization

This thesis work is organized as follows:

Chapter 1: State of the Art

This chapter starts with an introduction of prosight device which consists three main components (camera, receiver, transmitter) hardware and balckmagic device for recording video application software related to transmission data through Ar drone.

Chapter 2: Quadrotor

This chapter starts with the robotic platform drones hardware and the available sensors, we first state the basic flight properties of a quadrotor and then describe the Parrot AR.Drone and the available sensors in particular.The control algorithm of quadrotor and the hardware and software resources.

Chapter 3: Turtlebot

This chapter starts with an introduction to the hardware and software directly related to TurtleBot and continues with a short presentation of the vast open source library ROS, which holds much of the base functionality of the TurtleBot.

Chapter 4: Research Methodology

This chapter starts with the robotic platform drones hardware and the available sensors, we first state the basic flight properties of a quadrotor and then describe the Parrot AR.Drone and the available sensors in particular.The control algorithm of quadrotor and the hardware and software resources.

Chapter 5: Experimentation

This chapter starts with experimental results and simulation which is conducted by node.js and Artificial Neural Networks using robot operating system.

Chapter 6: Transmission data through Ardrone 2.0

This chapter starts with an introduction of prosight device which consists three main components (camera, receiver, transmitter) hardware and balckmagic device for recording video application software related to transmission data through Ar drone.

Chapter 7: Conclusion and future works

This chapter starts with concluded and the anticipated future work is presented. The overall comments on the approaches used in this thesis are also presented. Apart from that, comments on the techniques can be improved further are also presented.

Chapter 2

Quadrotor Platform

2.1 Introduction

This chapter starts with the robotic platform drones hardware and the available sensors, we first state the basic flight properties of a quadrotor and then describe the Parrot AR.Drone and the available sensors in particular.The control algorithm of Quadrotor and the resources which include software and hardware resources.

The quadrotor is a more specific term used to refer to a drone that is controlled by four rotors. It is also known as quadcopter or a quadrotor helicopter. The rotors on the quadrotor each consist of a motor and a propeller. In addition, these UAVs are always controlled remotely instead of being controlled by a pre- programmed, onboard computer. Quadcopters resemble helicopters but balance themselves by the movement of the blades and not by the use of a tail rotor. Furthermore, There are many different types of quadcopters on the market today, all designed to offer varying degrees of recreational uses. In this project, we performed to use algorithm implement a different type of Quadrotor.

2.2 Robotic UAV'S Platform

The different kind of robotic flying platform such as ArDrone 2.0, Bebop 2 and Mini-drone. so that the explaination of the Flying platform dicussed are as follows

2.2.1 Parrot minidrone - Airborne Cargo

The Parrot MiniDrone airborne is the type of Drone which is the main purpose to play as a toy. It is very easy to fly and save with the help of software application is called Freeflight. Nevertheless, In this project, we want to configure mini-drone with coding using flying automatic take-off and landing algorithm.

FIGURE 2.1: Parrot MiniDrone Airborne Cargo Mars

2.2.2 Parrot Bebop 2

The Parrot Bebop is the type of drone, is an aircraft without a human pilot aboard. Bebop Drone is known as unmanned aerial vehicle (UAV) which include a ground controller or RC and a system of communication between the two. Basically the aim to use bebop for that report to verify the algorithm which has automatic take off and landing in precisely fully autonomous. On the other hand, it has a fascinating camera which is really good for image processing and computer vision based algorithm.

FIGURE 2.2: Parrot Bebop 2

2.2.3 Parrot AR.Drone 2.0

The Parrot AR.Drone was initially meant to serve as a toy for augmented reality games. The quadcopter was supposed to be controlled by a human operator with a smartphone or a similar device. In spite of the original design as a high-tech toy, the drone quickly caught attention of universities and research institutions, and today is used in several research projects in the fields of Robotics, Artificial Intelligence, and Computer Vision in contrast to many other available remote-controlled aerial vehicles, the drone is robust and easy to use and fly.In our thesis, we will exploit some of the possibilities it opens up. First, we will take a look at the quadcopter's mechanics, then we will see what kind of hardware and software it has as well as all of its onboard sensors.

FIGURE 2.3: Parrot AR.Drone 2.0

2.3 Hardware Explanation

2.3.1 Basic Quadrotor Mechanics

A quadrocopter is a helicopter, which is lifted and maneuvered by four rotors. It can be maneuvered in three-dimensional space solely by adjusting the individual engine speeds (see Figure 3.3): while all four rotors contribute to the upwards thrust, two opposite ones are rotating clockwise (rotors 2 and 3) while the other two (rotors 1 and 4) are rotating counter-clockwise, canceling out their respective torques. Ignoring mechanical inaccuracies and external influences, running all engines at equal speed - precisely nullifying gravity - allows a quadrocopter to stay in the air without moving[10]. The following actions can be taken to maneuver the quadrocopter:

- vertical acceleration is achieved by increasing or decreasing the speed of all four rotors equally,
- yaw rotation can be achieved by increasing the speed of engines 1 and 4, while decreasing the speed of engines 2 and 3 (or vice-versa) - resulting in an overall clockwise (or counter-clockwise) torque, without changing overall upwards thrust or balance,
- horizontal movement can be achieved by increasing the speed of one engine while decreasing the speed of the opposing one, resulting in a change of the roll or pitch angle, and thereby inducing horizontal acceleration.

The fine tuning of the relative engine speeds is very sensitive to small changes, making it difficult to control a quadrocopter without advanced controlling routines and accurate sensors.

2.3.2 The Unmanned Aerial Vehicle

The Parrot AR.Drone has dimensions of 52.5 cm × 51.5 cm with, and 45 cm × 29 cm without the hull. It has four rotors with a 20 cm diameter, fastened to a robust carbon-fiber skeleton cross providing stability. A removable styrofoam hull protects the drone and particularly the rotors during indoor-flights, allowing the drone to survive minor and not-so-minor crashes such as flying into various types of room furniture, doors, and walls – making it well suited for experimental flying and development. An alternative outdoor-hull - missing the rotor protection and hence offering less protection against collisions - is also provided and allows for better maneuverability and higher speeds. The drone weighs 380 g with the outdoor hull, and 420 g with the indoor hull. Although not officially supported, in our tests the drone was able to fly with an additional payload of up to 120 g using the indoor hull - stability, maneuverability, and battery however suffered significantly, making the drone hardly controllable with that kind of additional weight.

FIGURE 2.4: The Parrot AR.Drone 2.0 with indoor hull(left)and outdoor hull(right)

The drone is equipped with two cameras (one directed forward and one directed downward), an ultrasound altimeter, a 3-axis accelerometer (measuring acceleration), a 2-axis gyroscope (measuring pitch and roll angle) and a one- axis yaw precision gyroscope. The onboard controller is composed of an ARM9 468MHz processor with 128Mb DDR Ram, on which a BusyBox based GNU/Linux distribution is running. It has a USB service port and is controlled via wireless LAN.

2.3.2.1 Cameras

The AR.Drone has two onboard cameras, one pointing forward and one pointing downward. The camera pointing forward runs at 18 fps with a resolution of 640 $\times$ 480 pixels, covering a field of view of 73.5° $\times$ 58.5°. Due to the used fisheye lens, the image is subject to significant radial distortion. Furthermore, rapid drone movements produce strong motion blur, as well as linear distortion due to the camera's rolling shutter (the time between capturing the first and the last line is approximately 40 ms). The camera pointing downwards runs at 60 fps with a resolution of 176 $\times$ 144 pixels, covering a field of view of only 47.5° $\times$ 36.5°, but is afflicted only by negligible radial distortion, motion blur or rolling shutter effects. Both cameras are subject to an automatic brightness and contrast adjustment.

2.3.2.2 Gyroscopes and Altimeter

The measured roll and pitch angles are, with a deviation of only up to 0.5°, surprisingly accurate and not subject to drift over time. The yaw measurements, however, drift significantly over time (with up to 60°per minute, differing from drone to drone - much lower values have also been reported [9]). Furthermore,an ultrasound-based altimeter with a maximal range of 6m is installed on the drone.

2.4 Dynamic Modelling of Quadrotor

The Dynamic Model is developed by first defining the various frames, kinematics and then deriving a dynamic model in mathematics form [11].

2.4.1 Frames

Body and NED frames are used for mathematical modeling of dynamics. The Body frame is attached to the body with the origin at the CG of the quadrotor and the X, Y and Z axes are presented in Fig. 2.3. The NED frame is also attached to the body with the origin at the CG of quadrotor; X, Y and Z axes of NED point towards geographical North, East, and Down respectively.

The Body and NED frames are associated with each other through Euler angles – Roll (ϕ), Pitch (θ) and Yaw (ψ). Rotation of ψ about Z_{NED} , followed by rotation of θ by resulting Y and then rotation of ϕ about resulting X takes from NED to Body Frame.

The transformation matrix from Body to NED frame is given in Eq 2.1

$$C_{NED,B} = C_3\psi C_2\theta C_3\phi \tag{2.1}$$

The individual transformation matrices in Eq 2.1 are elaborated in Eqs 2.2-2.4

$$C_1(\Phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\Phi & -sin\Phi \\ 0 & sin\Phi & cos\Phi \end{bmatrix} \tag{2.2}$$

$$C_2(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \tag{2.3}$$

$$C_3(\psi) = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

The Euler Angles are collectively expressed by $\eta_2 = (\Phi\theta\psi)^T$

Thus, the transformation matrix from Body to NED frame is expressed as follows.

$$C_{NED,B}\left(\eta_2\right) = \begin{pmatrix} cos\psi cos\theta & -sin\psi cos\phi + cos\psi sin\theta sin\phi & sin\psi sin\phi + cos\psi sin\theta cos\phi \\ sin\psi cos\theta & cos\psi cos\phi + sin\psi sin\theta sin\phi & -cos\psi sin\phi + sin\psi sin\theta cos\phi \\ -sin\theta & cos\theta sin\phi & cos\theta cos\phi \end{pmatrix} \tag{2.5}$$

2.4.2 Kinematics Model

The position is denoted by η_1 and given in Eq 2.6

$$\eta_1 = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{2.6}$$

The angular velocity in body frame is related to the Euler rates and given by the equation 2.7

$$\omega = \begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} \dot{\phi} - \dot{\psi} sin\theta \\ \dot{\theta} cos\phi + \dot{\psi} sin\phi cos\theta \\ -\dot{\theta} sin\phi + \dot{\psi} cos\phi cos\theta \end{pmatrix} \tag{2.7}$$

The linear velocity in body frame is related to velocity in body frame by Eq 2.8

$$\nu = \begin{pmatrix} \upsilon \\ \nu \\ \omega \end{pmatrix} = C_{NED,B^T} \begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix} \tag{2.8}$$

2.4.3 Dynamic Model

The Dynamic Model is developed about the CG of the quadrotor. The quad is assumed to be a rigid body to simplify the mathematical modeling. Since the quad has 4 rotors, it generates torques in a roll, pitch and yaw and a vertical force along its Z_{body}

Let the lift generated by the 4 rotors be denoted by f_1 , f_2 , f_3 and f_4. Thus, the total lift generated along the Z_{body} is given by Eq 2.9

$$F_z^b = \sum_{i=1}^{4} f_i \tag{2.9}$$

The torques are given in Eq 2.10

$$\begin{aligned} \tau_\phi &= l(f_2 - f_4) \\ \tau_\theta &= l(f_3 - f_1) \\ \tau_\psi &= (Q_1 + Q_2 + Q_3 + Q_4) \end{aligned} \tag{2.10}$$

Propeller thrust and torque are generally proportional to the square of the angular speed of the rotor. Also, the relation between Q i and the angular speed is complex. Thus, the vertical force and the torques are related to the angular speed of the rotor and given in Eq 2.11

$$\begin{pmatrix} F_z^b \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix} = \begin{pmatrix} \rho & \rho & \rho & \rho \\ 0 & -l_\rho & 0 & l_\rho \\ -l_\rho & 0 & l_\rho & 0 \\ \kappa_\alpha & -\kappa_\alpha & \kappa_\alpha & -\kappa_\alpha \end{pmatrix} \begin{pmatrix} \omega_{r1} \\ \omega_{r2} \\ \omega_{r3} \\ \omega_{r4} \end{pmatrix} \tag{2.11}$$

The 6DOF Equations of Motion can be written in simplified form through the Eq 2.12

$$\dot{X} = \begin{pmatrix} \dot{\Phi} \\ \dot{\theta} \\ \dot{\psi} \\ \ddot{\Phi} \\ \ddot{\theta} \\ \ddot{\psi} \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \begin{pmatrix} \dot{\Phi} \\ \dot{\theta} \\ \dot{\psi} \\ a_1\dot{\theta}\dot{\psi} + a_2\dot{\theta}\Omega_r + b_1U_1 \\ a_3\dot{\phi}\dot{\psi} - a_4\dot{\phi}\Omega_r + b_2U_2 \\ a_5\dot{\theta}\dot{\phi} + b_3U_3 \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \frac{sin\psi sin\phi + cos\psi sin\theta cos\phi}{m}U_4 \\ -\frac{-cos\psi sin\phi + sin\psi sin\theta cos\phi}{m}U_4 \\ g - \frac{cos\theta cos\phi}{m}U_4 \end{pmatrix} \tag{2.12}$$

where

$$\begin{cases} U_1 = b(-\Omega_2^2 + \Omega_4^2) \\ U_2 = b(\Omega_1^2 - \Omega_3^2) \\ U_3 = b(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \\ U_4 = b(\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \\ \Omega_r = -\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4 \end{cases} \tag{2.13}$$

and

$$\begin{cases} a_1 = \frac{I_{yy}-I_{zz}}{I_{xx}}, a_2 = \frac{J_r}{I_{xx}} \\ a_3 = \frac{I_{zz}-I_{xx}}{I_{yy}}, a_4 = \frac{J_r}{I_{yy}}, a_5 = \frac{I_{xx}-I_{yy}}{I_{zz}} \\ b_1 = \frac{l}{I_{xx}}, b_2 = \frac{l}{I_{yy}}, b_3 = \frac{l}{I_{zz}} \end{cases} \tag{2.14}$$

2.5 Control Algorithm

Proportional Integral Derivative is an algorithm that consists, as its name suggests, of three basic coefficients: proportional, integral and derivative which is varied to get optimal response.

PID is a control loop feedback mechanism that continuously computes an error value as the difference between a measured value and the desired value, trying to minimize the error over time (see Figure 2.5).

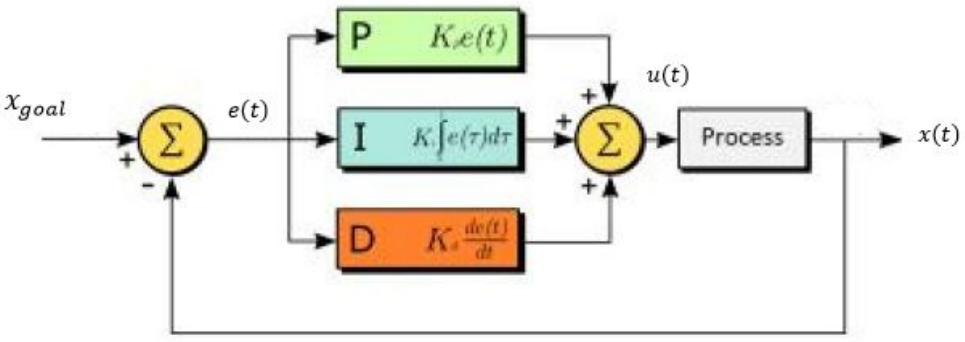

FIGURE 2.5: The general block diagram of PID controller

2.6 PID Formulation

This is the general formula of the PID controller

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{d}{dt} e(t) \tag{2.15}$$

where K_p K_i and K_d are all non-negative constants for the proportional, integral and derivative terms, respectively, e is the error, which is computed as the ideal output minus the measurable output. In figure 2.5 PID controller diagram output, t is the actual time and is the variable of integration, which takes values from time 0 to time t.

2.6.1 Proportional Term

In the quadcopter field, we can significantly simplify the Proportional (P) function and say that it is the value that handles the stability and the control. P is the most important value and indicates the correction level which needs to be applied. The larger the value of P, the harder the drone will try to stabilize the controlling board, but if we exceed this value, the drone will turn too sensible and will provoke oscillations. We have to find the adequate value for P.

- If the P value is very low, it will be very difficult to control the drone because it will be easy over-correct the maneuvers, which will make impossible to maintain it stable.
- If the P value is right, it will be easy the stability and it will throttle adequately.
- If the P value is too high, the drone will oscillate very fast. Also, it will gain height easily (like 'jumping') and it will be difficult to maintain.

2.6.2 Integral Term

The Integral term (I) indicates the loop velocity of the proportional action. If we keep I equal 0, the drone's movement will be very robotic and abrupt. The I term function is to make more progressive the returning movement to stability that P dictates.

- If the I value is too low, the drone will tend to raise the nose during the changing direction movement and will divert it.
- If the I value is right, it will maintain the angle more precisely.
- If the I value is very high.

2.6.3 Derivative Term

The Derivative term (D) makes the movement smoother and faster by changing the applied strength to correct the position error.

- A very low value of D will make the drone 'lazy'.
- A low value of D will make smoother the reactions.
- A high value of D will make the reactions more nervous.
- A very high value of D will provoke fast oscillations.

2.7 Resources

We make the division of the resources that we are going to use to develop the project between hardware, software and other resources. For each kind of resource, there is a little information about their use. There are main two resources which utilize during the project such as Hardware and Software resources. firstly Parrot ArDrone 2.0 which is used for to make the complete mission using the bottom camera to make detection and tracking, Secondly the main part is Unmanned ground vehicle (UAV's) which is mobile robot to following the trajectory manually via Joystick; for the software resources the main utilize of Operating system Ubuntu 14.04 and Robot operating system (ROS), node.js, C++, Python,OpenCV and Artificial neural networks.

Chapter 3

Mobile Platform - Turtlebot

3.1 Introduction

This chapter starts with the mobile robot control behavior in robot operating system. Firstly, explains the hardware and internal sensors controlling with different teleoperation such as Keyboard and Joystick. Turtlebot is a mobile robot platform to get familiar with ROS and robots in general. It is an open source platform so a lot of applications are already available. The turtlebot is composed by a kobuki robot as a base, a laptop with ROS and a Xbox Kinect as a sensor. With this simple setup, the turtlebot is able to handle vision, localization, communication and mobility.

3.2 System Overview

The turtlebot can also be modified for other purposes. For example, there is the turtlebot arm, this is a robotic arm that can be placed on the turtlebot for intervention operations. The turtlebot is not only used for research purposes but also for recreation. A well-programmed turtlebot can help you in your house and execute simple tasks. Note that the turtlebot is mainly used indoor and not resistant to outdoor activities.

FIGURE 3.1: Turtlebot 2

3.3 Kobuki Base

iClebo Kobuki is the perfect base for the turtlebot. It is a mobile base. It has sensors, motors,and power sources, however by itself, it cannot do anything. This base has replaced the previous iRobot Create base that was used in the first turtlebot. The iRobot Create was based on the Roomba vacuum cleaner that was also developed by iRobot.

The iClebo Kobuki is nothing on his own it is a base for customization. Therefore it is a low-cost mobile research base that provides power supplies for an external computer as well as additional sensors and actuators. The expected operation time is around 3 hours with a small battery and 7 hours with a big battery.

FIGURE 3.2: Kobuki Base

3.4 Kinect

The Microsoft Kinect was developed for the Xbox for gaming applications. But this high accurate depth sensor can be used for much more. One of those applications is in the turtlebot. The turtlebot uses the Kinect sensor as the robot main sensor. This leads to aversatile usage of the turtlebot.

The overview of Kinect sensor is shown in Figure 3.3. The middle hole is the "RGB camera" which is a camera that gathers color images at 30 frames per second. The left hole and the right one are "3D depth sensors", the sensor on the left side is an infrared projector. On the right side, there is an infrared camera which is used to compute how far the objects are from the camera. It has four microphones which are built into the Kinect and are used to record audio or even for speech recognition. The last one is a motorized tilt which allows the Kinect to move in a vertical way which means up and down. The sensor can tilt between -27 and 27. The field of view of the Kinect is 57.8.The range of the sensor is between 0.6 meters and 4 meters.

FIGURE 3.3: Kinect

3.5 Robot Computer

The robot computer enclosed with the TurtleBot is an ASUS PC which is a lightweight netbook. The main task of the robot computer is to collect data and send it to a desktop computer or record it to a hard drive. The robot computer also interacts with the hardware in the TurtleBot. Due to the computational power in the enclosed robot computer, some calculations can also advantageously be performed on this computer, instead of letting the desktop computer do all the work.

3.6 Desktop Computer

In this project, an HP EliteBook 8460p laptop will be used as the desktop computer. The computer has an Intel Ci7 (2.8 GHz) CPU, 8 GB internal memory and an Intel® HD Graphics 3000 integrated graphics card. The main task of the desktop computer is to do computations on data provided by the robot computer. Another task is to visualize the result of the algorithms, a task that demands a powerful computer in order to run smoothly.

3.7 General mathematical modeling of the robot

The important part of a mobile robot is its steering system. This will help the robot to navigate in the environment. We will use the differential drive model to reduce the complexity, cost, and size of the robot. A differential –drive robot consists of two main wheels mounted on a common axis controlled by separate motors. A differential drive system or steering system is a nonholonomic system, which means it has constraints on the pose change. A car is an example of a nonholonomic system, as it cannot change its position without changing its pose. Let's look at how our robot works and how we model the robot in terms of its mathematics.

In the term of robot kinematics which is the study of the mathematics of motion without considering the forces that affect motion. It mainly deals with the geometric relationships that govern the system. Robot dynamics is the study of motion in which all the forces are modeled.

A mobile robot or vehicle has six degrees of freedom (DOF) expressed by the pose $(x, y, z, \phi, \theta, \psi)$. It consists of position (x, y, z) and attitude (roll, pitch, and yaw). Roll refers to the sidewise rotation, pitch refers to forward and backward and yaw refers to the direction in which the robot moves in the x-y plane (called heading and orientation). The differential-drive robot moves from the x-y plane in the plane, so the 2D pose consists mainly of x, y, and , where is the head of the robot that points in the forward direction of the robot. In figure 3.4 the pose of the robot in x,y, and θ in the global coordinate system.

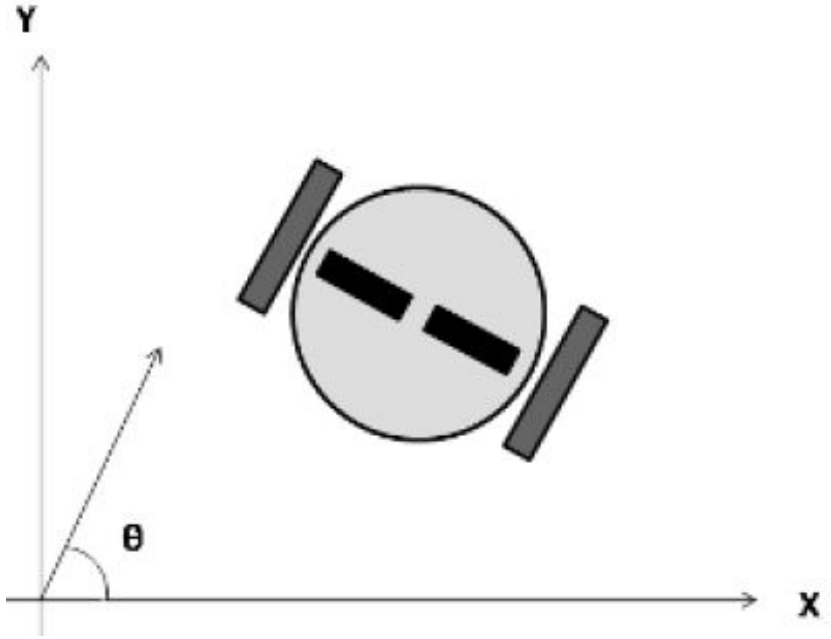

FIGURE 3.4: Modelling of mobile robot

3.8 Robot Operating System (ROS)

ROS constitutes the base of the TurtleBot system and defines the means of external communication when building specialized applications [12]. On the ROS home page http://wiki.ros.org/ one can read the following:

"ROS provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.

ROS is built up by stacks which in turn are built up by packages. A stack delivers the functionality of a whole system or major parts of a system, e.g.

- the turtleBot stack, which provides code for the TurtleBot
- the ros_comm stack, which provides internal communication in ROS and
- the perception_pcl stack, integrate Point Cloud Library into ROS.

The packages, new and modified, of this project, would be suitable to add together to a stack. A stack typically contains everything from a handful up to a couple of dozens of packages.

A ros package often holds the functionality of a smaller, more delimited task than a stack, e.g. the navigation stack contains various packages that do localization and route planning. Every package contains at least one node. A node typically solves one, or a few, separate tasks, e.g.

- the robot_pose_ekf node does EKF filtering,
- the pcd_viewer node provides a GUI for rendering point clouds and
- the image_proc node can rectify RGB and depth images.

Nodes are started up using the rosrun or roslaunch command. The first command starts one single node whereas the latter can start multiple nodes at the same time using the specified launch file. The launch file also has the neat property that values of dynamic parameters may be supplied to the nodes. This results in an accessible way of troubleshooting and trimming without re-compiling the C++ code.

An ROS-message is a data structure that can be sent between nodes. Messages can hold any information and ROS comes with useful predefined messages. User-defined messages are also allowed and the ROS framework is prepared so that defining user-specific messages is easily done.

The communication between nodes is handled either by a topic or a service. A topic implements an asynchronous communication pattern. A message is posted on the topic and the posting node continues its work. The nodes that listen to the topic, the subscribers, get a callback and can then process the information in the posted message. A service implements a synchronous communication model and any service call is thus blocking for the calling node and requires the immediate attention of the answering node. What defines a service in ROS is the input message type, the service function to be run and the output message type. To define a new service one simply specifies the message types to use in a text file and a corresponding service class is automatically generated.

The ROS framework also has a neat command named rosbag that can record and play a bag file. A bag file is a file that contains the messages published on all the topics specified when calling the rosbag record command and it can, therefore, replay a whole sequence of events, e.g. a drive was done by the robot.

3.9 Turtlebot getting started

First of all the necessary software have to be installed. The most important software packages are the main turtlebot packages, gazebo and rviz to simulate the turtlebot or visualize measurements [12].

After installing all the software it is a matter of connecting the turtlebot laptop with your own computer using ROS. It is possible to only use the turtlebot laptop but it is recommended to connect your own computer to limit the processing effort of the turtlebot laptop. In Figure 3.5 most common way to connect two computers is through Wi-Fi.

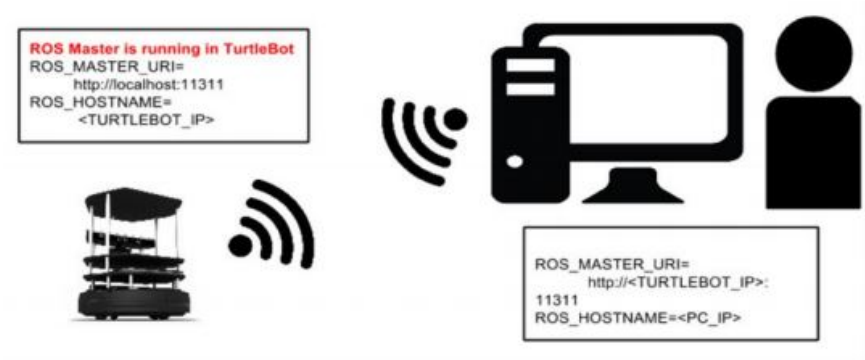

FIGURE 3.5: Connecting Environment

SSH is used to access the turtlebot from the remote PC. Ssh is a command to log in to a remote service. Now the ROS master can be started with the command roscore. If everything is configured properly the topics, nodes, and service can be found by the main PC even when the topics, nodes, services are in the turtlebot PC [12].

```
► ssh turtlebot@ [ip_of_turtlebot]
```

3.10 Turtlebot Configuration

Once the workspace is configured and a roscore is running it is possible to bring up the turtlebot. This bring-up starts the basic nodes for the turtlebot. There are different bring-ups available but in this case, the most simple one is used (minimal).

```
► roslaunch turtlebot_bringup minimal.launch
```

In figure 3.6 shows the all published nodes and topics which use the different approaches such as the level of the battery and command velocity, we need the position and orientation angle of the moving robot so in figure 3.7 the red circle shows the topics name which work on the position of the robot.

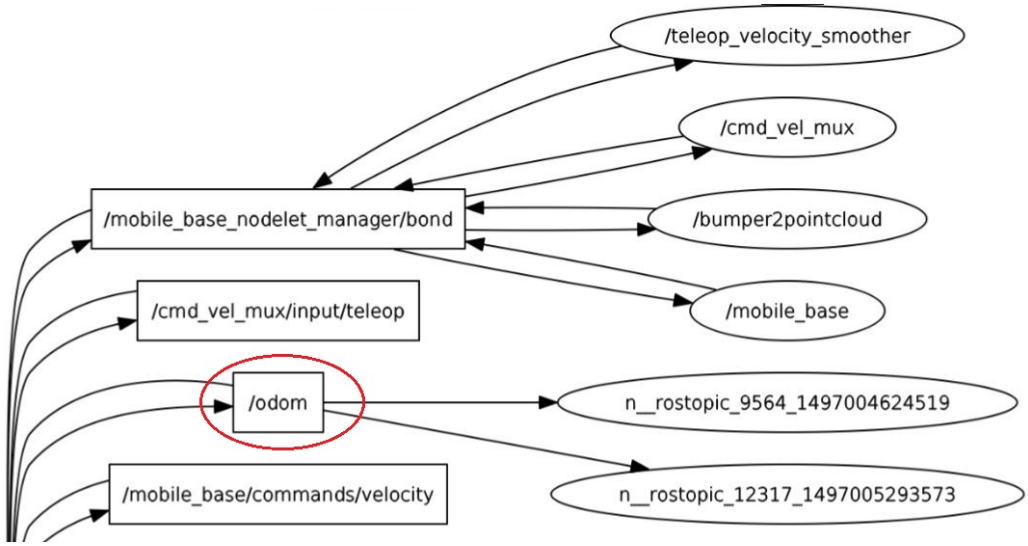

FIGURE 3.6: Zoom view for published topics and nodes

After bringing-up the turtlebot. It is ready to use. The first thing to do to test if everything is working correctly is bringing up the turtlebot dashboard. If everything is working correctly a dashboard application will show the battery status of the laptop and the turtlebot and other useful information about warnings and errors. If there is something wrong with the turtlebot the problem will be pointed in the dashboard application.

3.11 Teleoperation

Teleoperation allows to manually control Turtlebot. There are the two primary ways to control turtlebot.

3.11.1 Keyboard

In teleoperation, keyboard is the primary device to control turtlebot. After launch turtlebot_bringing we have to launch file to access the keyboard.

To use the keyboard for teleoperation, run the following on the workstation:

```
► roslaunch turtlebot_teleop keyboard_teleop.launch
```

After launching the keyboard_teleop we would see this terminal to show control key letters.

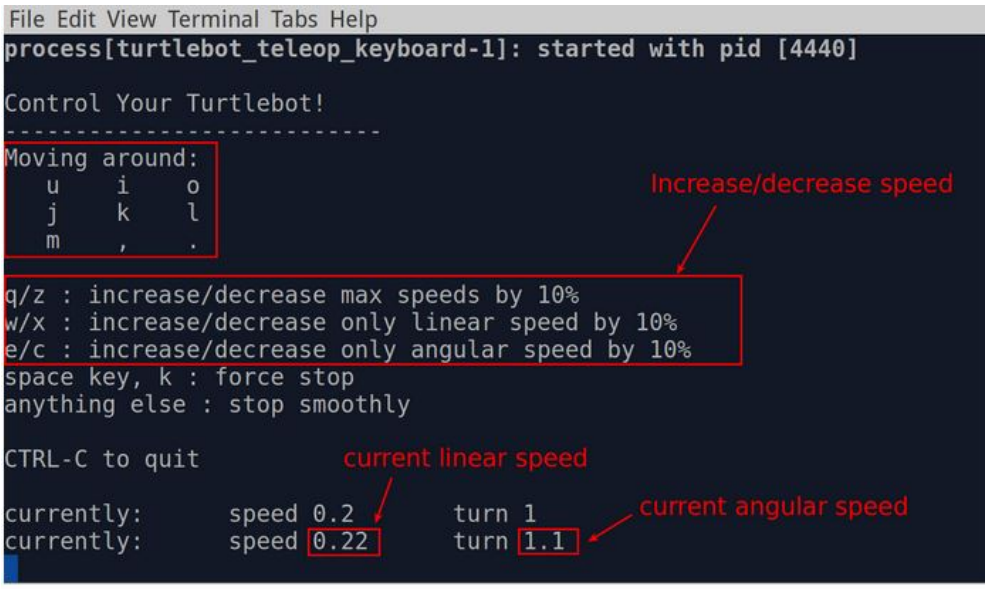

FIGURE 3.7: Control Keys

3.11.2 Freedom 2.4 Cordless Joystick

This is a cordless joystick, of which Logitech claims it keeps working up until 20 feet from your PC. The stick itself not only can be moved in the X- and Y-axis but also rotated.

It is really easy to use because joystick is cordless then move from one place to another without a barrier. Fortunately, turtlebot works with the Logitech joystick but there's a "gotcha" – they use a "deadman" button which must be held down at all times while operating the joystick, otherwise, nothing will happen. So that we have to update the deadman axis button. The first thing to figure out is which buttons correspond to which number.

```
► jstest /dev/input/js0
```

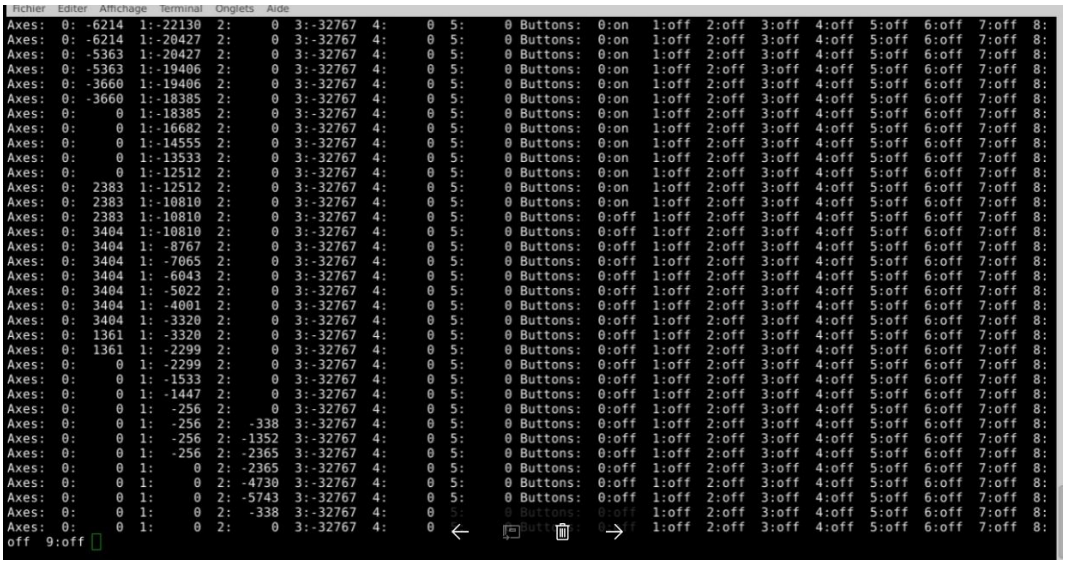

FIGURE 3.8: Screenshot of deadman axis during configuration the button

Now, we have selected which button we want to be our deadman axis use of above command line, firstly make a custom launch file which is located that file gedit myjoystick.launch

we would see that file which we have to change the velocity of dead axis trigger button

```
<!--
  Driver for the logitech rumblepad2 joystick.

  Use the D pad while pressing and holding the left trigger button (5) to control
    .
-->
<launch>
  <!--  smooths inputs from cmd_vel_mux/input/teleop_raw to cmd_vel_mux/input/
    teleop -->
  <include file="$(find turtlebot_teleop)/launch/includes/velocity_smoother.
    launch.xml"/>

  <node pkg="turtlebot_teleop" type="turtlebot_teleop_joy" name="
    turtlebot_teleop_joystick">
    <param name="scale_angular" value="1.5"/>
    <param name="scale_linear" value="0.5"/>
    <remap from="turtlebot_teleop_joystick/cmd_vel" to="teleop_velocity_smoother/
    raw_cmd_vel"/>
  </node>

  <node pkg="joy" type="joy_node" name="joystick"/>

</launch>
```

Start teleoperation with our new launch file command line

```
► roslaunch turtlebot_teleop myjoystick.launch
```

In the teleoperation we choose to use joystick because there is cordless device which is really easy to use as compare to keyboard.

FIGURE 3.9: Joystick

3.12 Odometry

Odometry is the use of data from moving sensors to estimate change in position over time. Odometry is used by some robots, either with legs or with wheels, to estimate their position relative to a departure point.

The publisher node and topics how many node and topics work, In figure 3.7 the odom topics use for find the position of turtlebot so if we need position record odom topic that shows rotation, position and orientation.

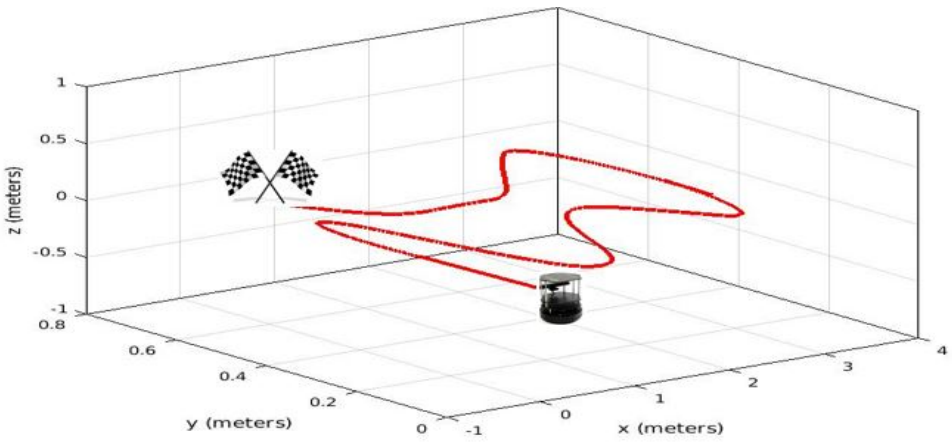

FIGURE 3.10: Odometry of turtlebot

In order to improve the odometry of the turtlebot, a gyroscope is added to the base of the turtlebot. The robot pose ekf node uses the spin and Odom data to compute and publish the most accurate combined Odom. The robot pose ekf is an Extended Kalman Filter (EKF) used measurements observed over time that contain noise and other inaccuracies.

The kalman filter generates values that are closer to the actual values of the measurements than the corresponding calculated values. As a result, the odometry of the turtlebot is more accurate.

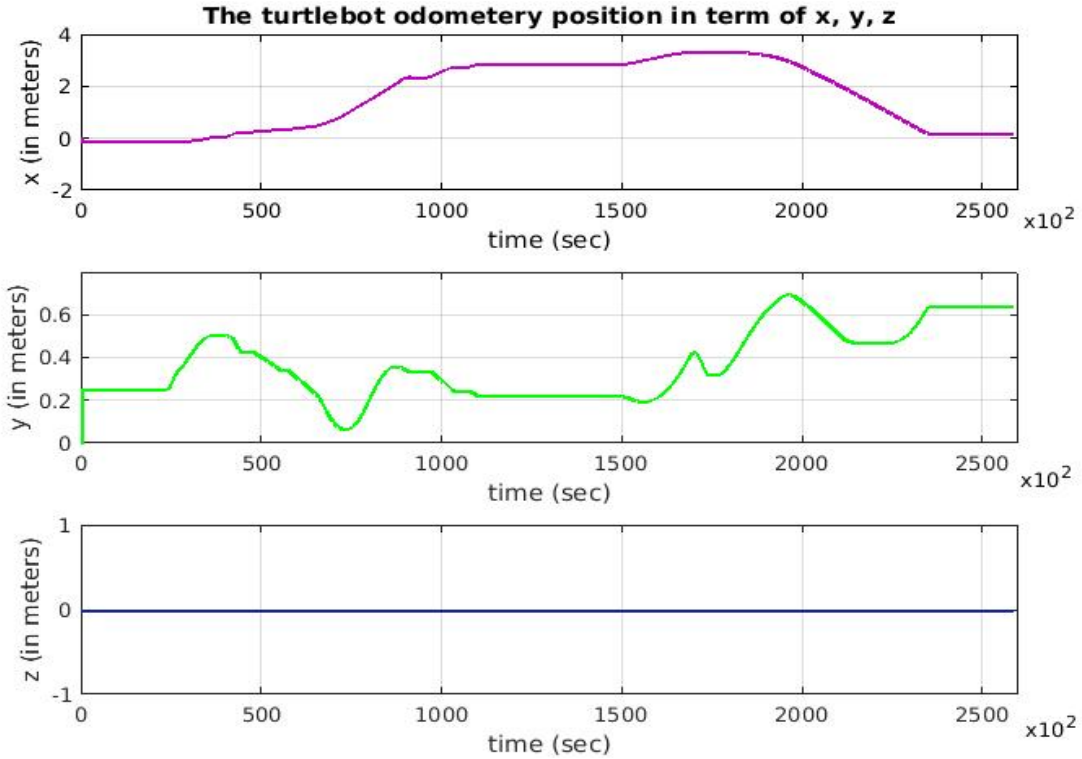

FIGURE 3.11: Position of turtlebot

Chapter 4

Research Methodology

4.1 Introduction

The main idea of this project to handle the automatic take-off and landing on a moving target without a human pilot. Firstly we have to choose the programming platform because there are different approaches, for instance, C++, Node.js, Python. First thing, why we need autonomous; Autonomous means that object will move automatically to reach given target without any human interface,

1. Adding new autonomous functions, such as autonomous landing for flying drones,helps the users with routine maneuvers and so allows using them with a lower risk of breaks caused by loss of control in critical operational phases.
2. Short Battery Life that's why we need some time to take automatic landing.
3. Autonomous landing which belongs to the most dangerous phases of the flight, for example especially the windy conditions are difficult for manual landing and often crashes and damages.

In order to accomplish this objective, six main tasks were formulated:

- Implement Image detection through image processing
- Develop control Algorithms
- Implement autonomous landing on a moving target
- Conduct testing

- Implement prosight device on a normal Ardrone 2.0
- Refine system performing

However, even before these steps could be taken, It is necessary to choose programming platform which has easy and robustness result with in less time. Firstly I had literature survey, I read a lot of paper publication which has idea to do programming in Node javascript

The node copter programming platform is done by German guys especially for the drone stuff, it is new programming to know automatic take-off and landing. So I get the idea from node copter to do the task. The Node javascript has different kind of libraries which are able to fly drone and tracking ball or face detection. The libraries called node ar-drone ,ardrone-autonomy for mission and Node-opencv.

4.2 Project Setup

Before starting it is necessary to understand how to communicate all device and connectivity as well. In figure 4.1 we see the connection between equipment .

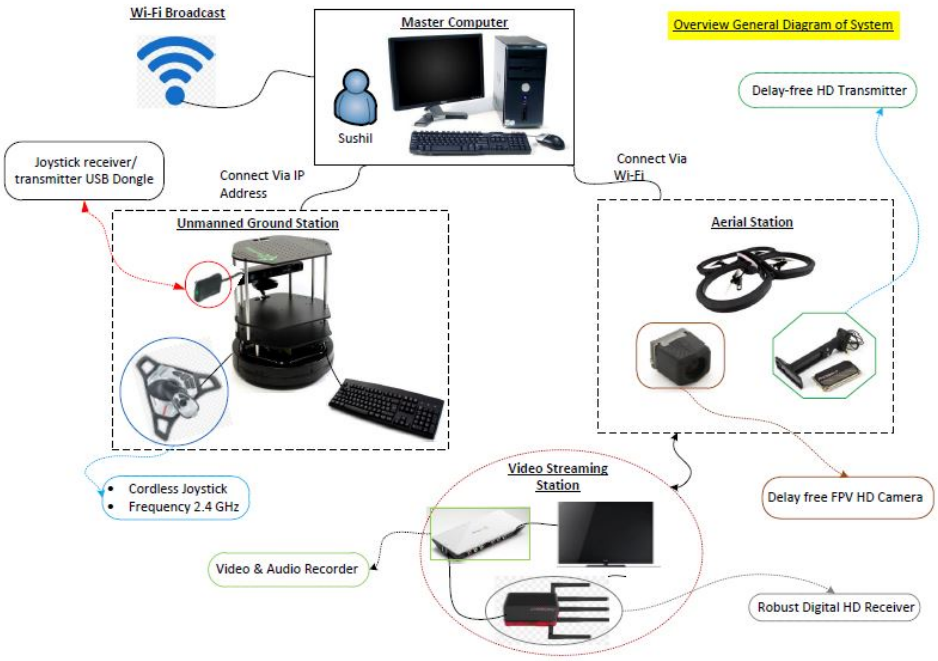

FIGURE 4.1: General overview project setup

Figure 4.1 shows the hardware configuration of the project for connecting devices to the device Connectivity. First the main computer must be connected via Wifi Broadcast

Secondly, we must connect Turtlebot with its own notebook to go with, We use turtlebot notebook IP address as a salve and our master computer as a king. On the other hand, the Ardrone 2.0 is also connected via Wi-Fi called Aerial Station. In the air station where we installed the Prosight unit gave us good video quality See Chapter 5 for more details.

4.3 Marker Board

A marker was designed, for the image processing recognition block especially for this purpose. The board has a large black round circle rectangle in which a white in the circle and the shape of the terminator as in Figure 4.2. The main attraction of the circular size and terminator markers is that the monocular camera of a drone can be seen from the variety of distances, From 5cm to 3cm. The marker board or landing pad is called Oriental Roundel

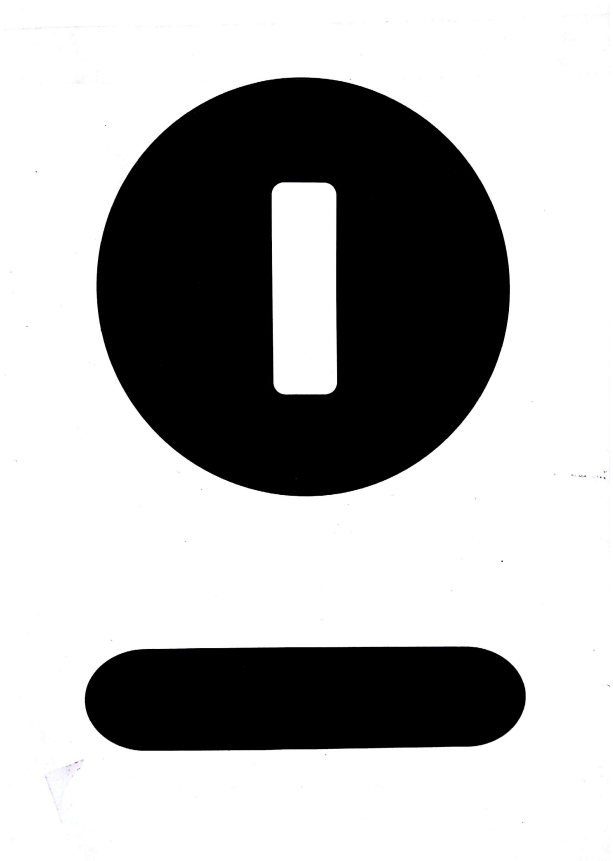

FIGURE 4.2: Marker Oriental roundel

4.4 Image Processing Algorithm

In order to navigate autonomously, a quad rotor must be able to gather data and learn about the characteristics of the environment. In our case the image marker is quite simple to use and detect because there is only black colour and white.

Once the image processing techniques are selected, proper pre-processing can be conducted. When an image is captured from the drone camera. The OpenCV library reads it in as a two dimensional array of RGB values which means that the color of every pixel in the image is specified by a combination of red, green and blue. However, in order to do color-based thresholding later in the analysis step, it is generally easier to convert to the HSV convention.

For the Image processing we are using 'Hough Transform' Algorithm. The Hough Circle Transform is detect the circle or edge detection

In order to detect circles in images, you'll need to make use of the cv2.HoughCirclesfunction. It's definitely not the easiest function to use, but with a little explanation.

cv2.HoughCircles(image, method, dp, minDist)

- **Image:** 8-bit, single channel image. If working with a color image, convert to grayscale first.
- **Method:** Defines the method to detect circles in images. Currently, the only implemented method is cv2.HOUGH_GRADIENT,
- **minDist:** Minimum distance between the center (x, y) coordinates of detected circles. If the minDist is too small, multiple circles in the same neighborhood as the original may be (falsely) detected. If the minDist is too large, then some circles may not be detected at all.
- **param1:** Gradient value used to handle edge detection
- **param2:** Accumulator threshold value for the CV2.hough_gradient method. The smaller threshold is, the more circles will be detected (including false circles). The larger threshold is, the more circles will potentially be returned
- **minRadius:** Minimum size of the radius (in pixels).
- **maxRadius:** Maximum size of the radius (in pixels).

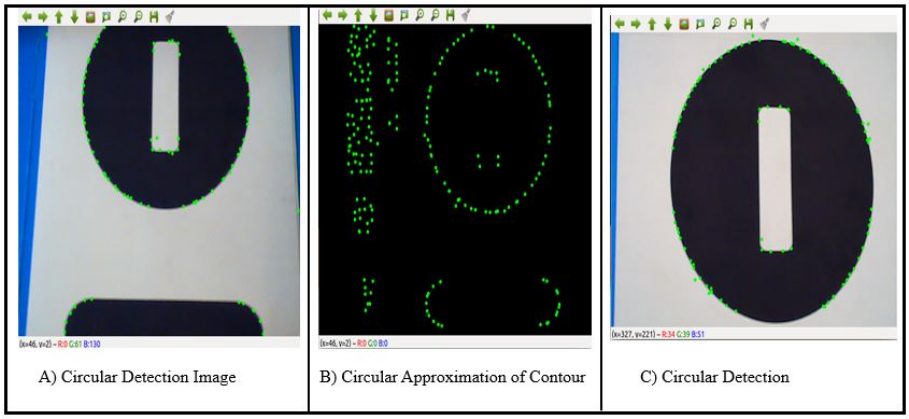

FIGURE 4.3: Detection Circle

However, after testing the OpenCV algorithm in general then in term of node js to implement the node OpenCV which is libraries to detect the face. The concept of face detection in node js we got the idea of tag detection to change the property of oriental roundel which is exist in node js. Secondly the image of tags the circle we have to know about the radius of circle and the equation of circle we need three parameters to define a circle.

$$C : (X_{center}, Y_{center}, r) \tag{4.1}$$

where (X_{center}, Y_{center})define the center position and r is the radius, which allows us to completely define a circle.

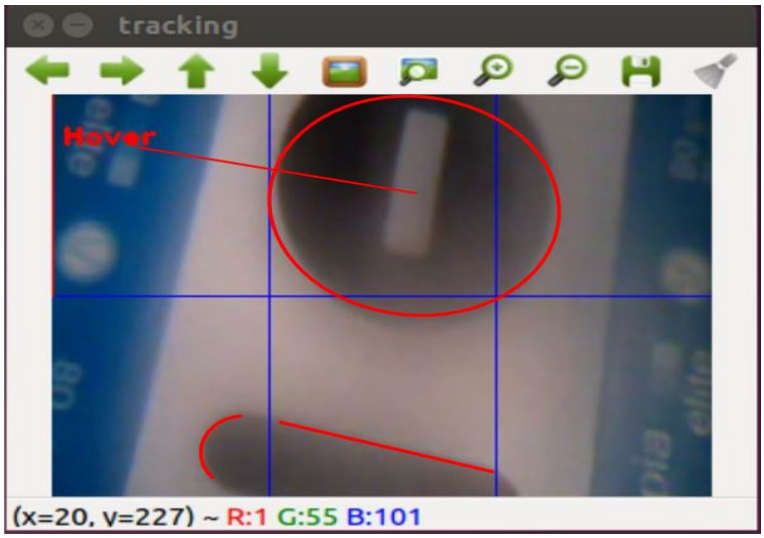

FIGURE 4.4: Detection Circle

In figure 4.4 the drone detect oriental roundel marker using bottom camera of ardrone 2.0 for detection and tracking. Firstly the tracking we faced problem to track turtlebot because of PID control so that we calculate the velocity of drone and turtlebot to get the same velocity. Another problem we faced to detect the marker use bottom camera as we know the drone bottom camera is not good as compare the front camera which has 720p. Furthermore the landing of drone bottom camera works complexity to find the coordinates of marker.

4.5 Landing Description

For autonomous landings, the drone must reliably recognize the land point. There are methods for visual navigation (eg, with a landing Pattern or a particular circular pattern [7]), however, require better visual sensors (Cameras) and usually do not support the horizontal alignment [5].

We decided on the landing pattern, which consisted of the shape of the circle and the terminator this makes it very easy to be prepared by an inexperienced user. In addition to this simple adjustment of the landing pattern, the other advantage is that such pattern also defines the horizontal orientation. To pre-process the image, use directly the HSV filter, the image erosion and the image dilation (this Basic model processing algorithms are available and described OpenCV library [8]). Then to identify circularly in the image, although circular recognition The algorithm is used and the center of the torque is computed to set the landing target Point. The drone is equipped with four independent proportional-integral derivative (PID) which is supplied by the distance to the destination. The entrance For each PID controller, the coordinate part of the target distance is in its specificity address and output is a control value for the low-level AD.Drone interface functions (pitch, roll, yaw and vertical speed).

The figure 4.5 shows the schema from the situation, which is used to calculate the distances. The Drone camera is located at point D just above the point V, T is the target point, C1 and C2 are the edges of the range visible through the camera and x is the height measured by the ultrasonic sensor. First, we use x to approximate the real height v, since the angle is usually small and the ultrasonic sensor has a small mistake anyway. Furthermore, The camera does not provide the other measurements directly; Enter an image as shown in the Figure 4.5 (top left).

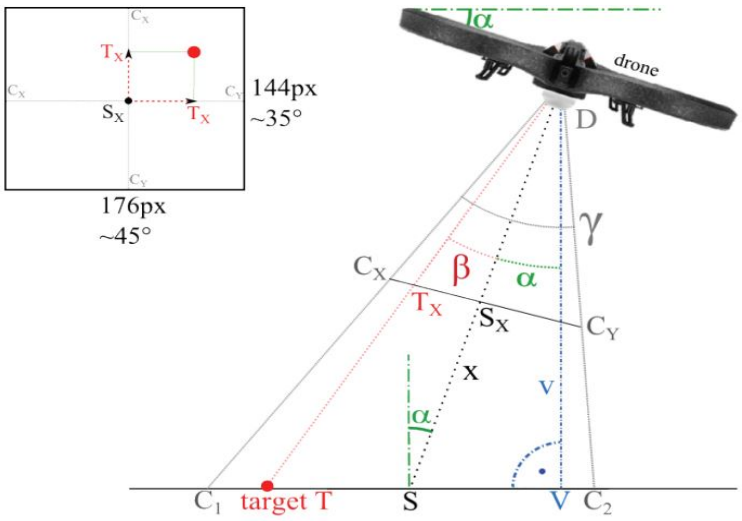

FIGURE 4.5: The principle of landing

We need to compute the distance between T and V, which can be done using the formula:

$$dist\ (\mathrm{T,V}) = \mathrm{v.tan}(\alpha + \beta)$$

The angle α is known from the sensors – it is either pitch for the forward/backward distance or roll for the left/right distance. The angle β proportionally corresponds to the distance between T and S. We measure this distance in pixels from the camera picture. This is not the real distance of points but the distance of their projections in the camera picture. Notice that we already know the pixel distance between CY and SX and the angle γ from the camera properties (± 72 pixels and $\pm 17.5°$ for the longitudinal axis, ± 88 pixels and $\pm 22.5°$ for the lateral axis). Hence we can compute the angle β using the formula:

$$\beta = arctan(tan(\gamma/2).pixel\_dist(TX, SX)/pixel\_dist(CX, SX)) \tag{4.2}$$

In addition, the flow chart of drone landing it is easy understandable to know how drone landing in term of image processing, To be precise the drone works node.js platform which has programming which gives us freedom to fly drone automatic take-off and landing.

The figure 4.6 shows clearly the way of landing in flow chart firstly drone take-off and implement image processing which has opencv libraries to detect the circle and terminator shape which has a lines, whenimage processing is done then it goes to second step which has if algorithm to able to detection then find the image coordinate of image or otherwise drone would hovering and searching the platform.

Secondly when drone find the coordinate it is ready to go down for landing if velocity of turtlebot and drone are same otherwise drone move upward and do again same thing. In addition drone move down for landing to check threshold value if everything is going well. The drone will be land on the moving target.

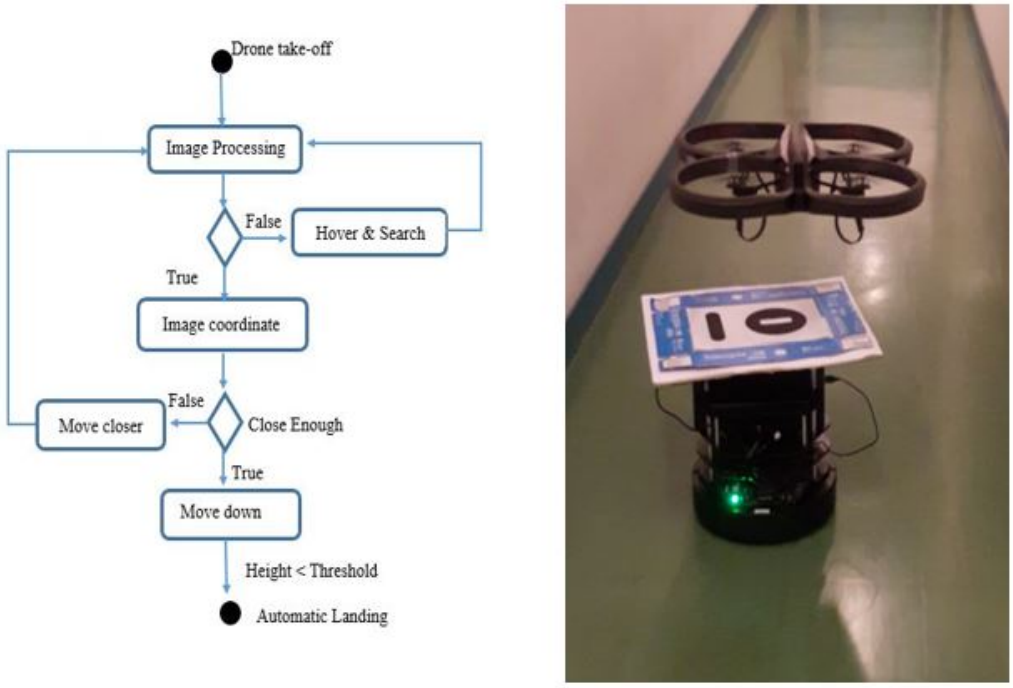

FIGURE 4.6: Landing structure and the drone follow marker in real experiment

4.6 Scenario 1

Automatic take-off and landing on a moving platform (Turtlebot)

In these scenarios, the drone goes to automatic tracking and tag detection. If it recognizes the image of the marker and calculates the coordinate of the image according to the landing principle (see Figure 4.5) and then lands on it. NodeJS is a popular JavaScript server platform based on plugins running locally. Many developers know or have heard of it and it is a great platform for experiments For the automatic landing, we chose the platform js node, which is really easy to use And less time to carry out. The programming of the nodecopter has libraries to perform the Ardrone 2.0, that is, node-ar-drone, drone-autonomy, and node-OpenCV. We used these libraries to fly the automatic takeoffs with one Minimum time (in a second) apart from the fact that we have to call the picture marker oriental round. The mission is take off drone and go forward 1 meter, To the right 1 meter, backward 1 meter and 1meter left and hover then the earth doing A square

In Figure 4.7 follow square with a robust stabilization then complete the mission with short moment time and land it safe.

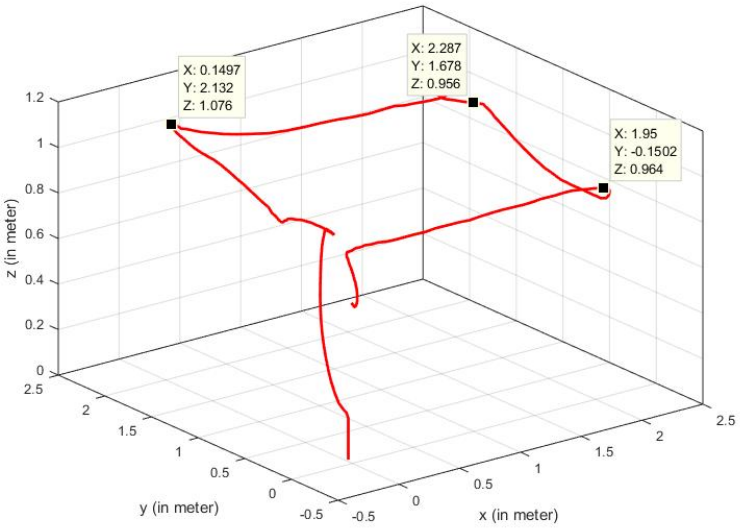

FIGURE 4.7: The drone following square trajectory

In general, the actual position of the drone is when the drone goes to the mission and according to the commands.

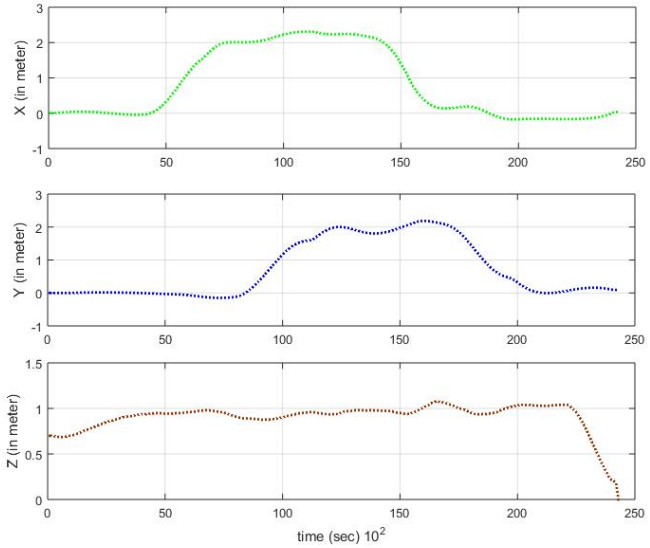

FIGURE 4.8: The actual behaviour of drone in term of position x,y and z

Before the start, the synchronization of the drone and turtlebot we need to know the mission of how it works and the landing on the moving target. The speed of Drone and turtlebot is pretty much the same at the same time to communicate between the drone and turtlebot .This graphic shows the starting go behind in the turtlebot as a

robot. Therefore, the speed of the drone and the robot is equal because the turtlebot is controlled via joystick. As we have explained before marking recognition, which we put in the turtlebot, as we can see in the Turtlebot in the graphic. In the marking we used the image processing for the detection and tracking with the algorithm this means hough transformation and PID algorithm for control and tracking.

4.7 Artificial Neural Networks

An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another. 'Neural' is an adjective for neuron and 'Network ' denotes a graph like structure.

Artificial Neural Networks (ANN), also called neurocomputing, connectionism, or parallel distributed processing (PDP), provide an alternative approach to be applied to problems where the algorithmic and symbolic approaches are not well suited. Artificial Neural Networks are inspired by our present knowledge of biological nervous systems, although they do not try to be realistic in every detail (the area of ANN is not concerned with biological modelling, a different field).

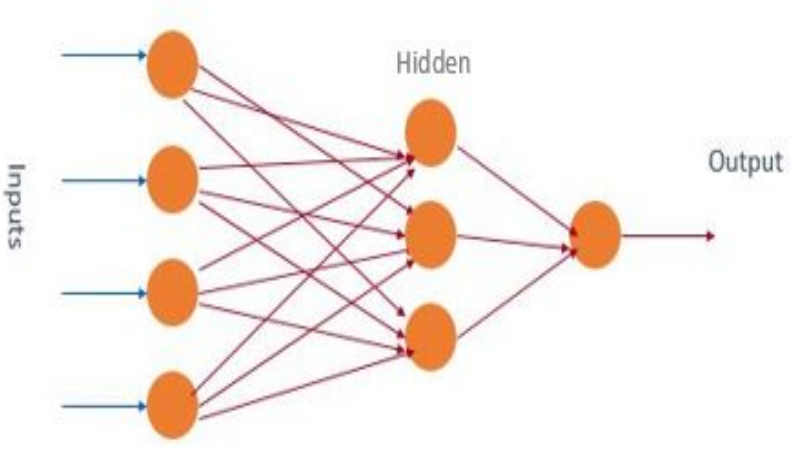

FIGURE 4.9: General idea

- **Input Layer:** The activity of the input units represents the raw information that is fed into the network.
- **Hidden Layer:** The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units.

- **Output Layer:** The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

The McCullogh-Pitts model

The McCulloch-Pitts model was an extremely simple artificial neuron. The inputs could be either a zero or a one. And the output was a zero or a one. And each input could be either excitatory or inhibitory. Now the whole point was to sum the inputs. If an input is one, and is excitatory in nature, it added one. If it was one, and was inhibitory, it subtracted one from the sum.

This is done for all inputs, and a final sum is calculated.Now, if this final sum is less than some value (which you decide, say T), then the output is zero. Otherwise, the output is a one. Here is a graphical representation of the McCulloch-Pitts model

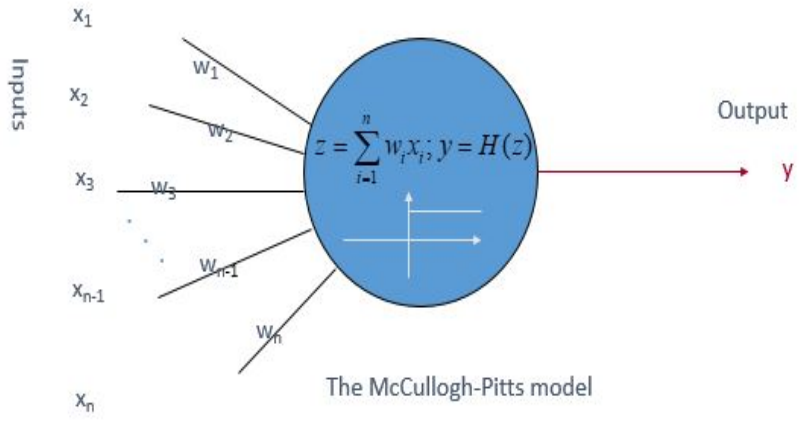

FIGURE 4.10: THe McCullogh-Pitts model

Nonlinear generalization of the McCullogh-Pitts neuron:

$$y = f(x, w)$$

y is the neuron's output, x is the vector of inputs, and w is the vector of synaptic weights.

4.8 Scenario 2

Automatic take off and follow boustrophedon trajectory detect a tag which is on the fixed platform using Artificial Neural Networks

In these scenarios, the automatic drone take off and follow Boustrophedon Trajectory recognizes a label that is on the fixed platform. For the task, we choose robot operating system (ROS)[16] and python programming to go to boustrophedon path.
Why do we use Boustrophedon to the task, because there are two methods to search for one, The first is boustrophedon path, which runs in one direction on a line, for example from left to right and in the opposite direction on the following line For example, from right to left. The second is the helix, which goes on next Circle after reducing the radius of the circle. For this task, we experimented in the pavilion simulation, which was actually saved comparison with the experiment in real-time since it is the automatic landing at a fixed point.

During this experiment, for example, we had problems

- The drone is not stable due to the PID controller, so we have changed a certain value of the PID controller. In the simulation, the drone lands nearby target.
- The boustrophedon path works well in the simulation, but in the real environment it takes more time to complete the mission

The gazebo simulator uses a different marker that is slightly black and white Labels are called alvar_tags that already exist in ros which is why we have chosen gazebo simulation.
The main subject of the differently captured mark is that the camera is capable see at least one marker from a distance, 5 cm (landed drone) Up to 3 cm. The marker has a smaller marker that is near the reference point because it helps the drones camera to see more than one reference point, even if it is the drone is very weak.

FIGURE 4.11: Marker using ANN

4.9 Planning of work

The working methodology defines how to manage the time during the project task. Before the start of this project, we had ensured that the economic resources to get the drone and all the necessary materials, as well as check if the goals are achievable. In addition, as we would like to add weight to the drone like HD camera, HD transmitter and antenna. We have made sure that the drone can deal with it.

This section illustrates the schedule of the project and describes the tasks performed to carry out the project (see Figure 4.12). The project began on 1 March 2017 and ends with the last defense on 5 September 2017.

Now we will describe the tasks we have planned to make our project, starting with the preliminary tests to ensure that the project is possible to finish the review that everything works as we planned and the final details before Deliver the report. In Figure 4.12 the pie chart shows the utilization of weeks, so the first week of the project is to review the literature to see what kind of problem they see in the same type of project. After review, we have a solution to change.

In the circular diagram, the Boustrophrdon path section shows the utilization of 3 weeks because the artificial neural network (ANN) is the huge subject of using it so that it can realize the path of Boustrophrdon take time with Python on the ROS operating system platform.

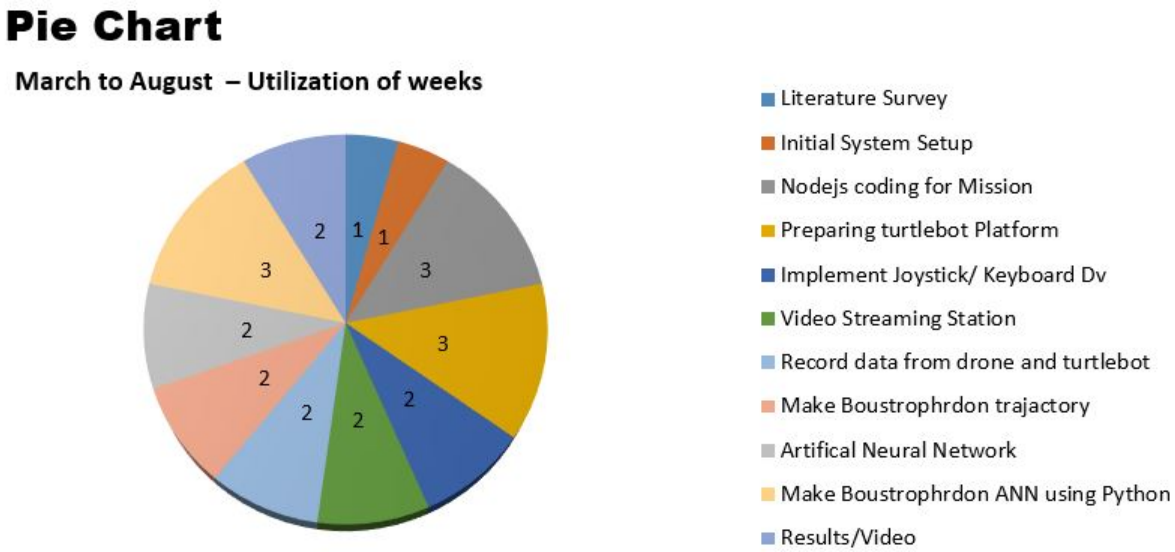

FIGURE 4.12: Pie chart of the weeks of different type of tasks

Figure 4.13 shows the Gantt diagram, which shows the dependency between the tasks and the project duration and also shows the project planning. The last test was to verify that everything worked as expected. We prepare everything for the delivery of the project, the robustness of the code and we check all documents to avoid mistakes and to prepare the final presentation.

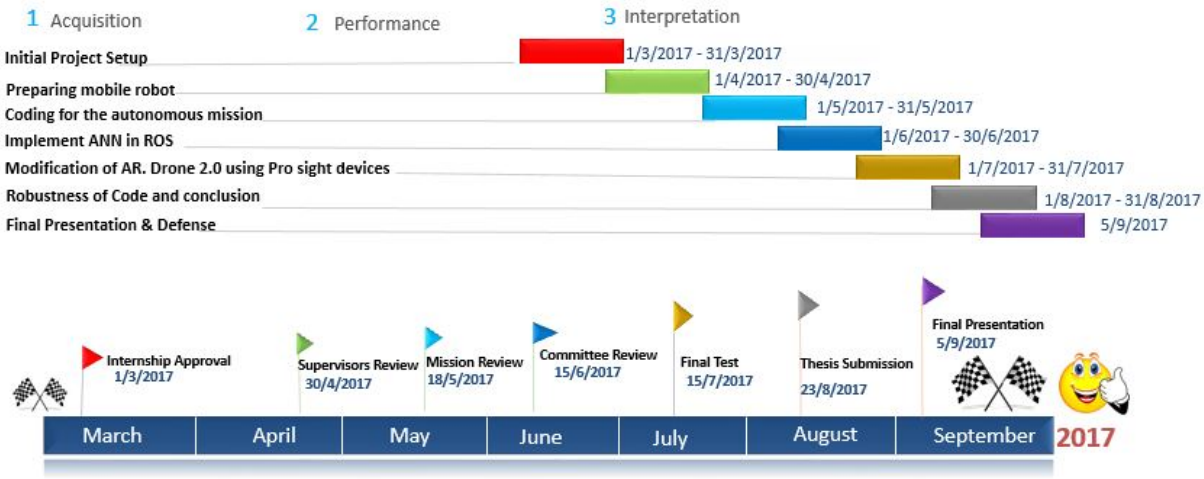

FIGURE 4.13: Gantt diagram of the planning project

Chapter 5

Experimentation

5.1 Introduction

In this chapter we will describe the experimental results and the simulation of node.js and artificial neural networks with a robot operating system. Once you have been informed about the development of the project, it is time to carry out experiments with our tool and review the results obtained to evaluate the quality of the project. In this chapter, Taking into account the constraints we had during the development of the experiments, we will present the three scenario experiments we have done to finalize the results.

5.2 Setup of the Experiment

The configuration of the experiment, which has different scenarios to follow the mobile robot. However, the main target of the experiment is the automatic launch and landing on a mobile platform, In node.js is allowed to make mission for the autonomous airborne landing.

5.2.1 Scenario 1

In figure 5.1 shows the synchronization between the unmanned aerial vehicle (UAV) and the unmanned ground vehicle (UGV) in a suitable manner. The scenario was drone following the turtlebot and landing on moving platform using node.js [15], to be more clear we provide a video showing our system in action at

https://www.youtube.com/watch?v=bDb7Vs93J0M

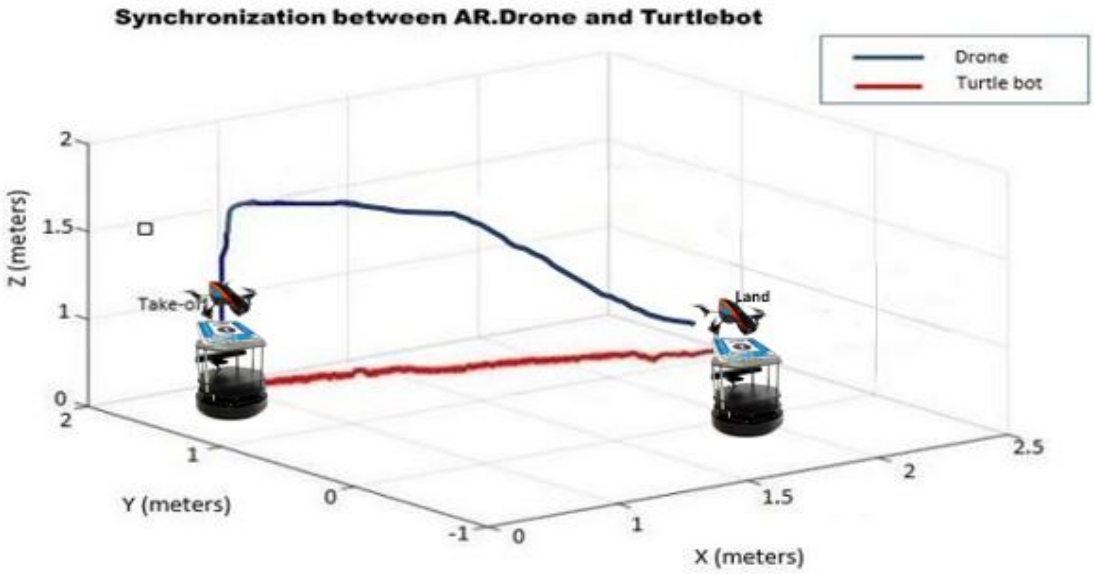

FIGURE 5.1: The synchronization between ArDrone and turtlebot

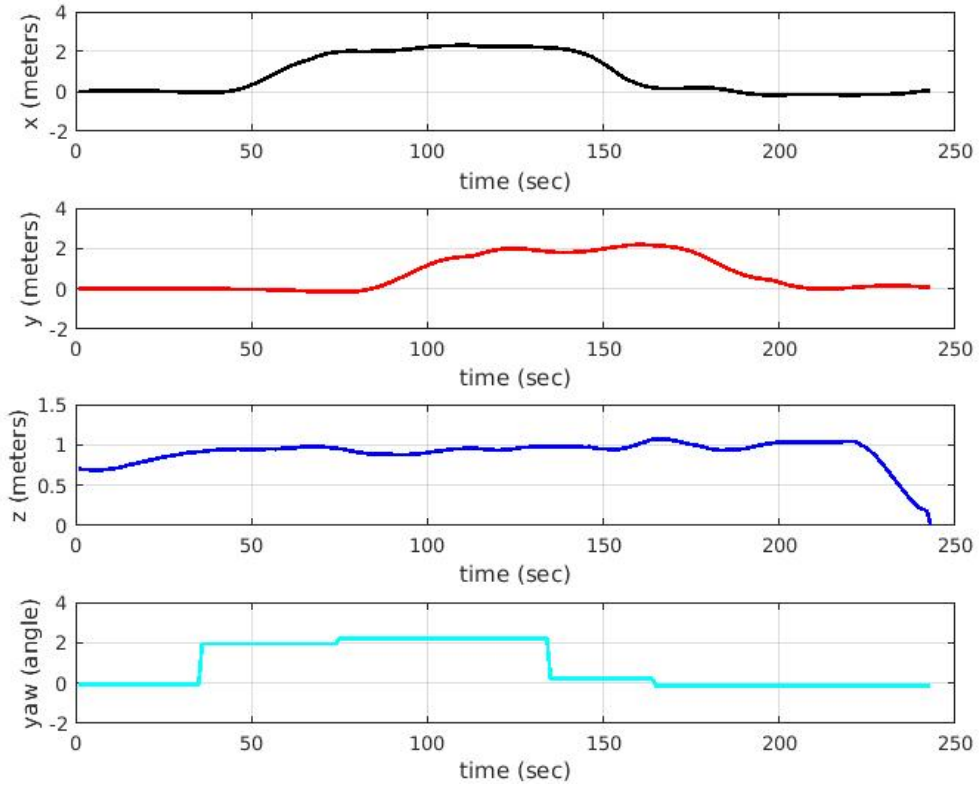

FIGURE 5.2: The position of Adrone follow moving robot

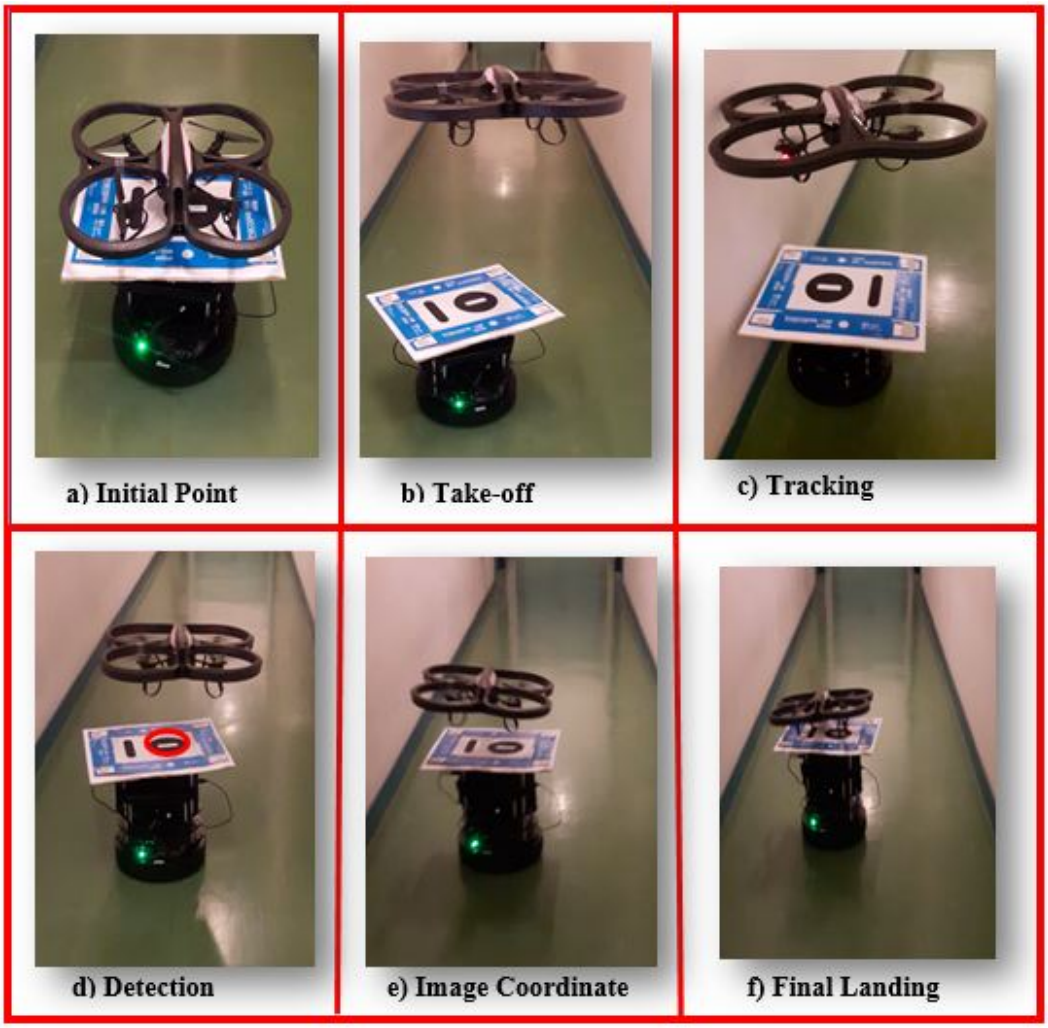

FIGURE 5.3: Flow chart communication between ArDrone and turtlebot

Table 5.1 shows the synchronization between Drone and Turtlebot. The first section explains the manually controlled turtlebot speed via joystick so that the drone control is successfully landed on the mobile platform

Velocity (m/s)	**UAV land on a UGV**
1	Successful landing
2	Successful landing
3	Successful landing
4	Successful landing
5	Unsuccessful landing

TABLE 5.1: Landing on the different velocities of mobile robot

5.2.2 Scenario 2

In this scenario, we need to use a simple path called boustrophedon. First, the drone follows a boustrophedon path, which has the path from right to left and from left to right in alternating lines. When the drone reached the last point to land in a turtlebot, we decided to move a robot a bit to confuse the drone in it. But the intelligent algorithm is able to recognize tag and land. As you can see in the graphic, the red line shows the drone and blue is a robot. After the end of the journey, the drone can move slowly, meanwhile we are dramatically increasing the speed of the robot, but our intelligent system is able to follow the robot and land on it

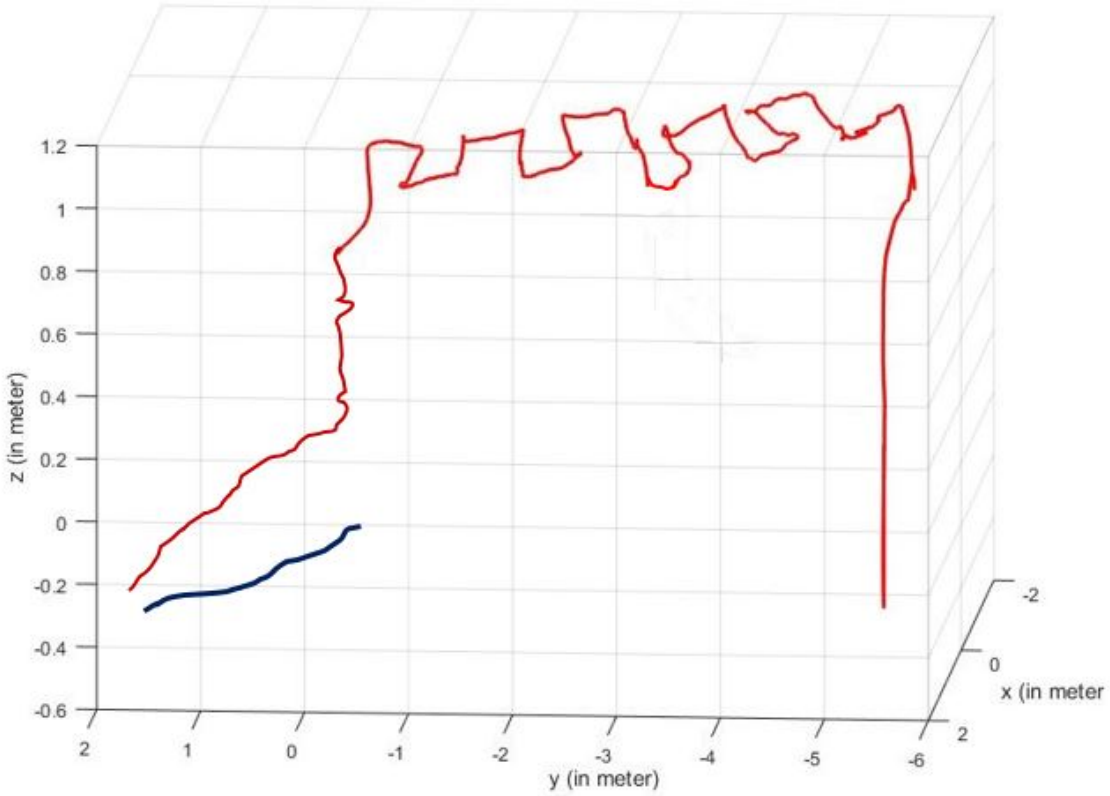

FIGURE 5.4: Boustrophedon path follows drone and land on turtlebot

5.2.3 Scenario 3

The automatic start and follow the boustrophedon path will recognize a label that is on the fixed platform and end up on it with artificial neural networks, in this type of task We have installed libraries that have alvar _tag [16] detection in ros. So far the drone would work on the simulator in a gazebo and it works pretty well but in the real atmosphere, the drone has problems with the bottom camera , if the bottom camera

works properly, we get more accurate results. In part of the simulation we would explain many things.

On the other hand, the drone is actually far from the fixed point and then follows the trajectory as soon as the drone is reached near the fixed point or when the drone recognizes the label and the information about the label of the central coordinate to the control of the neural network Full screen can then land at the fixed point. The drone does not get the full image coordinate that would land near a fixed point but sometimes the controller has a problem to get the information, we can see the results of the simulation.

Before starting the scenario, we explain the starting point of the drone and the fixed point. If the drone was far from the target, how to see the results of the simulation.(see figure 5.5)

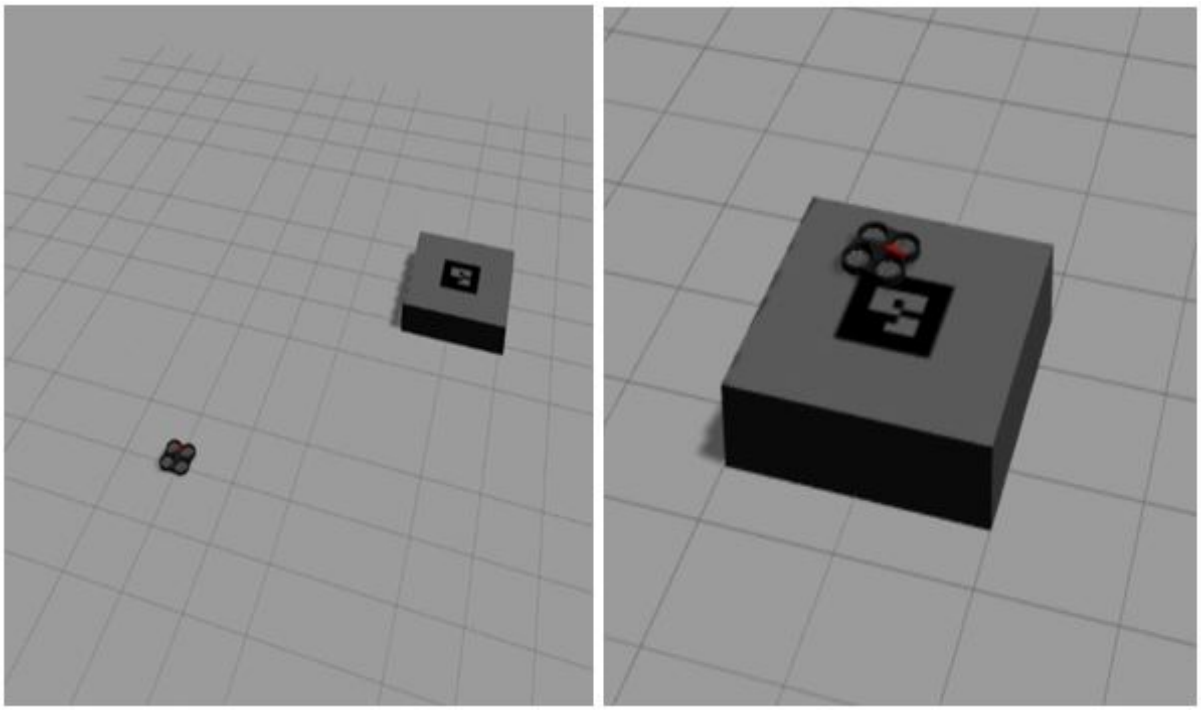

FIGURE 5.5: Initial point UAV far from the target and final point on a target

In figure 5.5 shows the image on the left, the drone is very far from the target Follow the special route and land on the fixed destination. The drone of the image on the right followed the path and landed on the target without obstacles. For the explanation of the faith the results of the gazebo simulator you could see the section of the simulation for each main point we discussed about the state of the drone will land. Before we start the simulation, we capture the data from the predicted Pose _ardrone Or the position of the drone with respect to x, y and z. The position shows how the sum goes to the end

point to the end point. At the same time, the rosbag to able to record the Euler angles and the position as well.

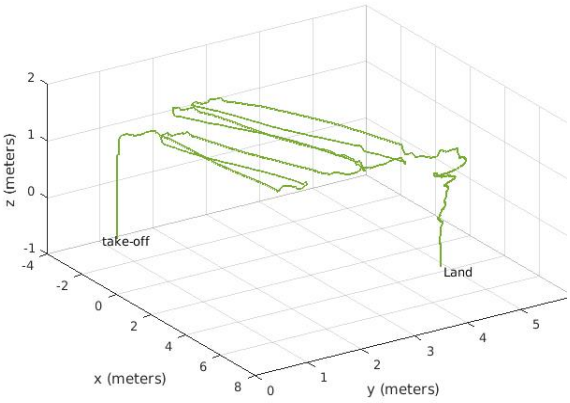

FIGURE 5.6: 3D view of Boustrophedon Trajectory using ANN

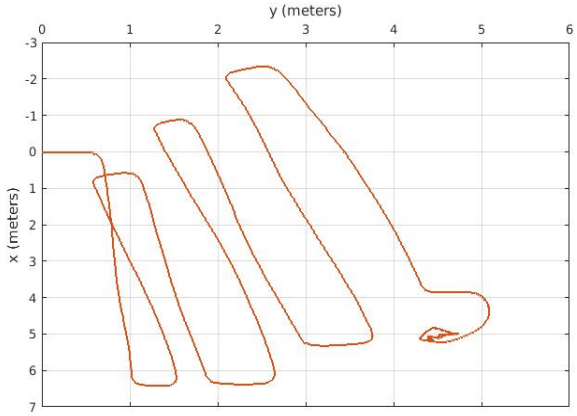

FIGURE 5.7: 2D view of Boustrophedon Trajectory using ANN

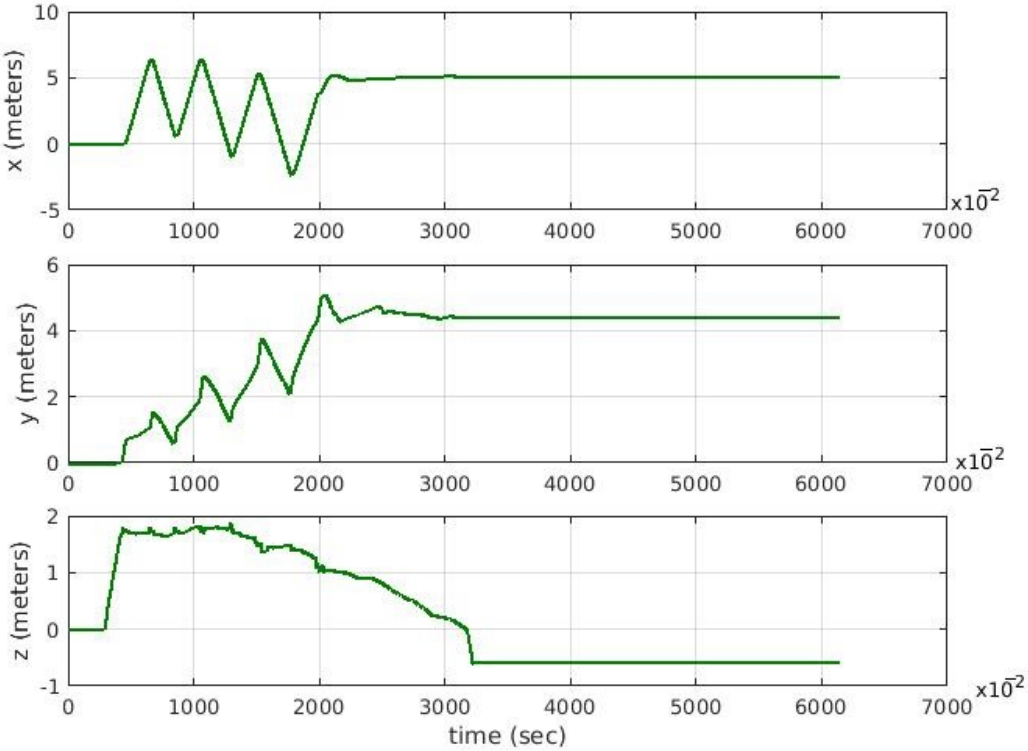

FIGURE 5.8: The position of drone when followed Boustrophedon path

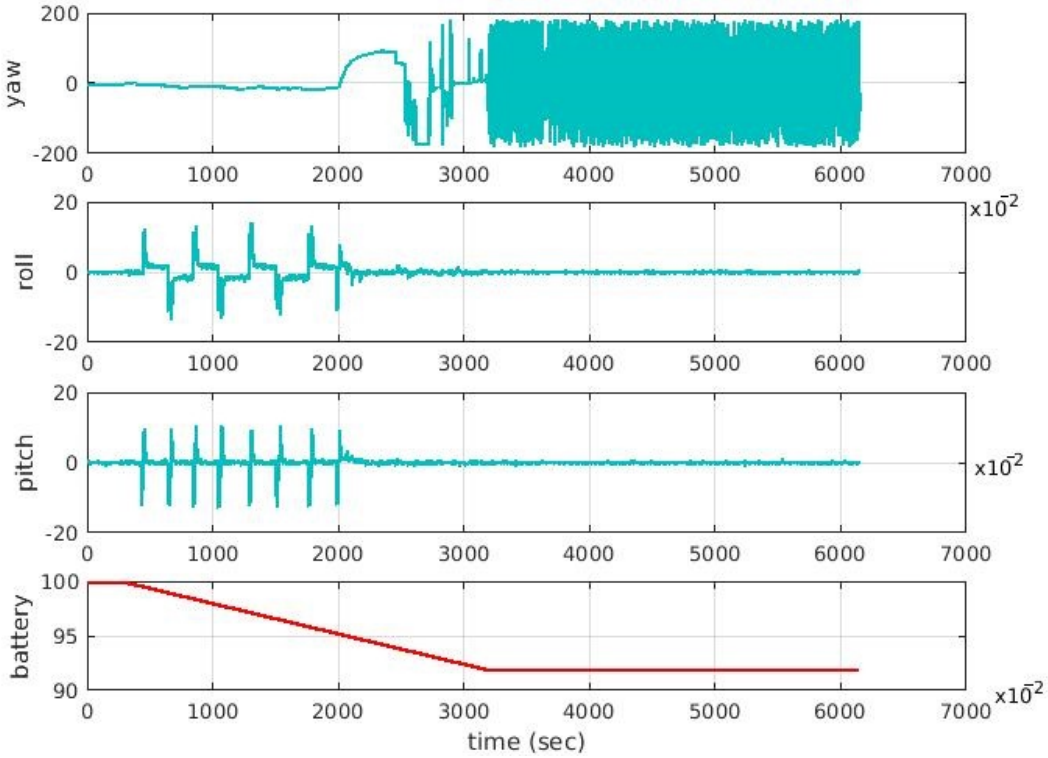

FIGURE 5.9: The Euler angles of drone when followed Boustrophedon path

5.3 Simulation

Simulation of the results of the section A drone flying over the target and sending the image through bottom camera using NN controller (see figure 5.10) , Section C the drone land on the target, but not exactly the center of the image because an image has not sent any registration information using NN controller so drone land nearby the image. Similarly Section B has the same problem because the first drone gets the image and then lost the information about the image. Section D clearly shows that the drone at the fixed point is reached without obstacles.

Bottom part of drone as you can see the image information of the drone using gazebo simulator but the section B there is no image so far because the drone is landed on the ground, simultaneously the section D exotic landed at the spot Still image because that Cordinates image to center point of the image to ready land on it.

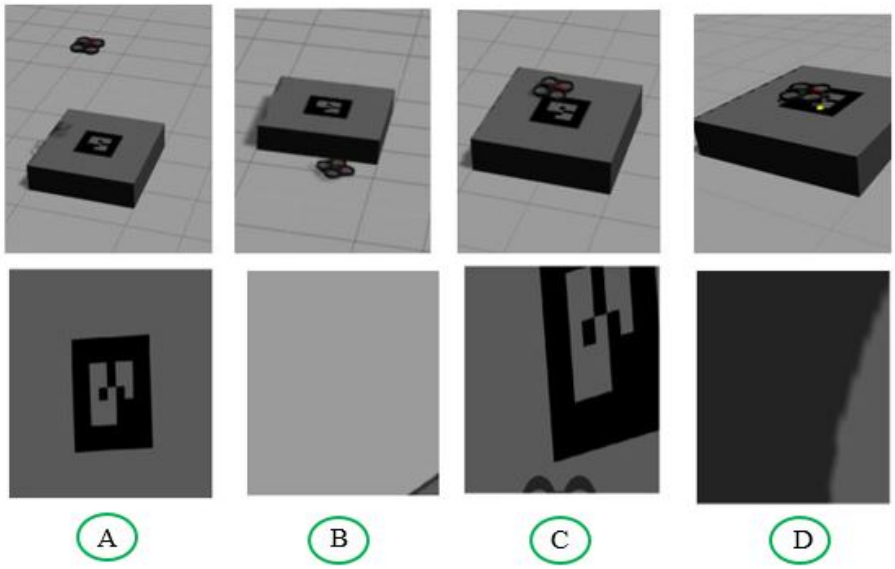

FIGURE 5.10: The simulation results using gazebo simulator (ANN)

The screen shot of the terminal indicates when the drone has reached the fixed point, the images send the coordinate information, which gives information every second. As soon as the drone receives the coordinate of the image for the landing, The drone gets the image, but not the drone coordinate would land anywhere, as you can see section B or the bottom of the image on a ground so there is no tag image. The neural network controller must control the drone, but it depends on it When pre-processing the image to detect the ground pad at a fixed point.

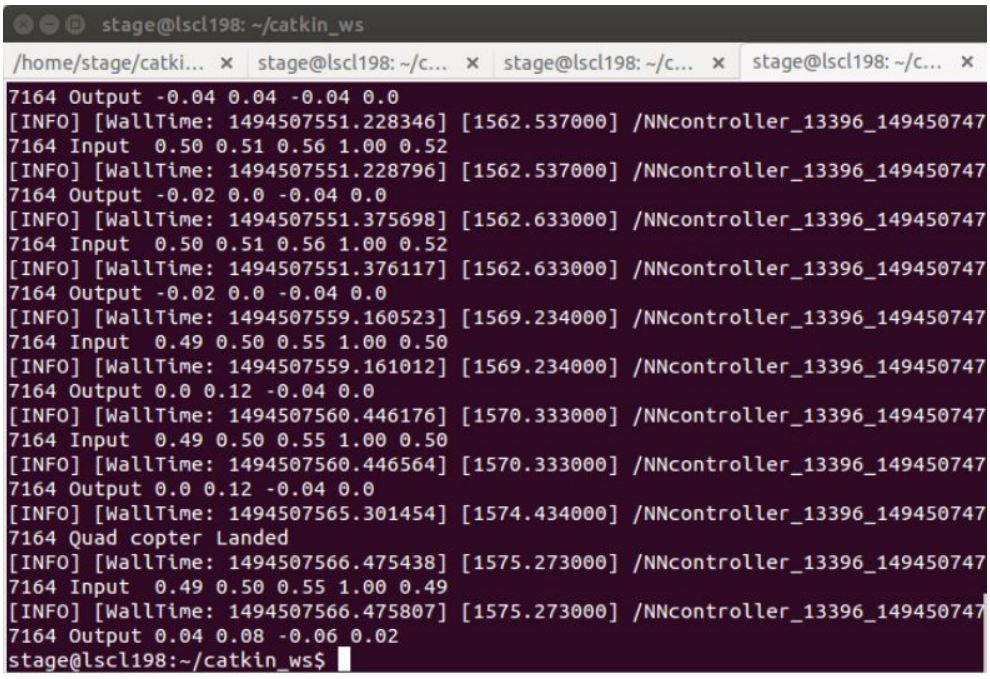

FIGURE 5.11: Terminal shows a NN controller

5.4 Modification of AR.Drone 2.0

Ar.Drone 2.0 on a mobile platform, we have modified the normal AR.Drone 2.0 in addition some special devices, which enable the recording of a video with high resolution. Figure 5.12 shows how to install the device in the drone, in the results we get the good stability of the drone and the best video streaming. We gave the normal drone a different name like IBISCdrone. Chapter 6 is the best way to get the most information about the Modification and transmission data through the normal Ar drone 2.0.

FIGURE 5.12: IBISC Inspection drone

Chapter 6

Transmission data through ArDrone 2.0

6.1 Introduction

In this Chapter, we are performing to get a live video-streaming from a drone to a Laptop/PC so that we can use this video for processing for different applications. The application can be used commercially, defense or even entertainment purpose.We are introducing two components which we use to make this possible. They are Connex Prosight HD vision pack and Blackmagic design intensity shuttle.

CONNEX ProSight is all about transforming FPV drones into a completely new kind of immersive experience for beginners and experts alike.The Connex ProSight HD Vision Pack is a true game changer. It delivers unparalleled vision performance with the delay-free wireless transmission. Its superior image quality combined with simple installation, smooth configuration, and improved multi-pilot flying experience [13].

6.2 System Overview

The figure 6.1 shown below is the basic working model of this process. The Connex pro sight HD video pack is used for capturing and transmitting video to the station.The transmitter mounted on the drone transmit the video captured by the HD camera.The receiver at the station receives this signal and convert it to video and provide and output through HDMI out. We can get a direct FPV vision using the glasses or if we connect to a monitor. But we can't connect directly to a Laptop.In that case, we need a video recorder. Therefore we use Blackmagic Design Intensity Shuttle [14] that acts as a video

recorder and more over it converts the HDMI input into HDMI, USB, and TV outs. So it can even act a hub i.e., at a time we can get an output to our PC and to the FPV glass and even to the TV screen.

FIGURE 6.1: General Diagram

6.3 How it works

Amimon's CONNEX provides a high-end, high-performance wireless HD connection that can operate in challenging unmanned air or ground platforms under harsh conditions with zero latency, such as UAV/UGV. The small and lightweight CONNEX system transmits commercial, industrial, inspection and monitoring video in real time to its Ground Unit, which can be located up to 1,000 meters away.

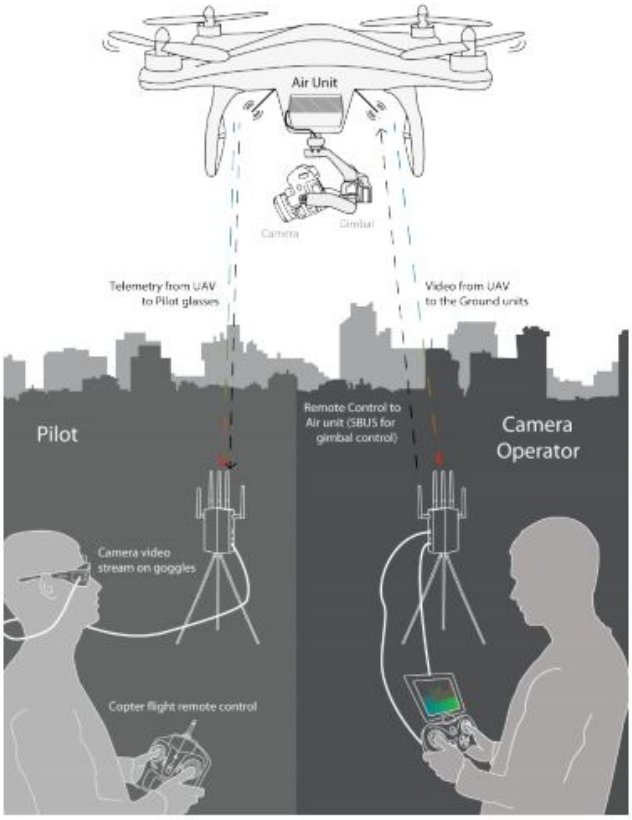

FIGURE 6.2: how Prosight device works

- **Air unit:** The air unit is connected to an aircraft in order to capture video from the aircraft camera and to transmit it to up to four receivers simultaneously (multicast), thus creating a wireless video link and it is known as a transmitter.
- **Ground Unit:** The ground unit connects to various types of monitors, video goggles or a portable video monitor via the HDMI port. This enables the pilot and/or camera operator to monitor the video transmitted and it is known as a receiver.
- **Pilot:** The pilot can view the video on a monitor or wear video goggles connected to the receiver. Flight control (telemetry) information from the aircraft is overlaid on the video. The pilot uses a remote flight controller to control the aircraft.
- **Camera Operator:** The camera operator can hold a portable or PC video monitor on which the video can be viewed. The camera operator could also use a gimbal remote control to control the aircraft camera's gimbal through the S.BUS port of the air unit.

6.4 Key Features

- True full HD 1080P at 60fps
- Up to 1,000-meter range (LOS)
- Zero latency, real-time video
- Extremely resilient 5GHz digital link
- Automatic Frequency Selection (AFS) that fully complies with regulations and automatically selects the best free frequency available
- Encrypted and secure

6.5 Components of Prosight device

6.5.1 Robust Digital HD Receiver

The CONNEX Prosight ground receiver unit provides a robust digital link over the 5GHz band. Its HDMI interface, compatible with all popular HDMI goggles screens in the market.

6.5.2 Delay-free HD Transmitter

The CONNEX Prosight transmitter supports delay-free uncompressed all- digital HD transmission. It is Optimized for multi-pilot FPV Racing and provides fully immersive flight experience.

6.5.3 Delay-free FPV HD Camera

The CONNEX Prosight HD camera is optimized for both indoor outdoor flights, featuring advanced High-Dynamic-Range sensor, allowing 720P delay-free crystal clear video.

		Transmitter	Receiver	Camera
Dimensions		70x36.5x7.2 mm	75x115x17 mm	28x20x27 mm
Weight		32 gram	134 gram	13 gram
Input Voltage		8v-16v	8v-26v	5v from Transmitter
Power Connector		4 Pins	DC Jack	MIPI Connector
Power Consumption		4.1 watt	3.5 watt	1 watt
Antennas Connector		2 external MMCX Conn.	5 SMA Conn.	NA
Video Interface		MIPI	HDMI (Type-A)	MIPI
SW Upgrade		Micro USB	Micro USB	Through Transmitter

FIGURE 6.3: Technical specification of device

6.6 Implementation of Prosight device on a drone

As prosight is an additional component, it is counted as a payload outside of the weight of the drone itself. The greater the payload, lesser will be the maneuverability of the drone. The Connex prosight Components (camera, transmitter, and receiver) are mounted on the Parrot Ardrone 2.0 in a fashion such that the weight distribution allows the center of gravity to remain unchanged. In figures 6.4 and 6.5 shows the arrangement of components to achieve stability criterion.

FIGURE 6.4: Prosight device mounted on Ar drone 2.0

6.7 Placement Guidelines – Receiver

The following describes the mandatory requirements for optional placement of the receiver and its antennas.

- **Place the receiver as high as Possible:** Place the receiver on a tripod, pole or table so that it is as high as can be. A height of 2 meters is optimal.
- **Avoid Interference:** Place the receiver cable antennas as far away as possible from other transceiver devices, especially from a transmitter in the 5 GHz band.
- **Place the Receiver Antennas facing the transmitter**: Place the receiver so that its antennas are facing upwards in the general direction in which the aircraft will be flying. The five receiver antennas can be placed as if the receiver and its antennas are an open hand with the palm facing the aircraft, as shown below:

FIGURE 6.5: Receiver Antennas Facing the Aircraft like an Open Palm

6.8 Check Stability of HD drone

Mounting prosight components provision the capture of high-quality video and images. And this allows us to embellish quadrotor as an HD drone.

FIGURE 6.6: Flying HD drone at 1m attitude

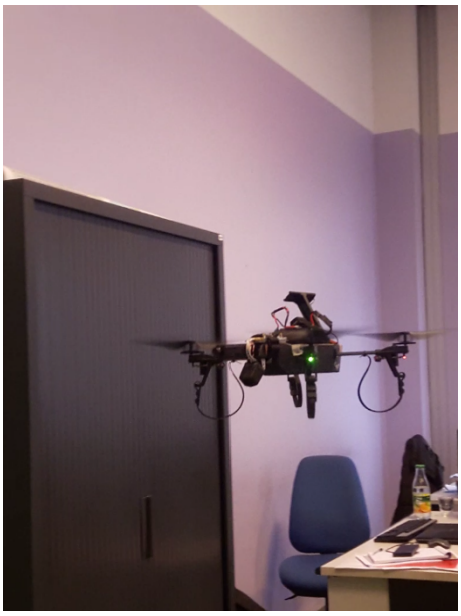

FIGURE 6.7: Flying HD drone at 3m attitude

In Figures 6.6 and 6.7 show the arrangement of prosight components. Their consequent weight distribution makes it quite stable in hovering and in fight mode. As shown in figure 6.6 the drone reaches a height of 1 meters without instabilities. Simultaneously figure 6.7 the drone is seen to rise steadily to an altitude of 3 meters. The drone stable is some point which ensures that the transmission signal range, the drone is seen to attend

different altitudes. The Connex HD camera gives us the live streaming video as shown in figure 6.8

FIGURE 6.8: The live streaming video using HD drone

6.9 Blackmagic Design video hardware

The Blackmagic is a company which design video hardware to record audio/video in high quality even Ultra HD. Blackmagic Design's Desktop video [14] provides a software which works in conjunction with Ultra studio and Decklink Intensity. The Desktop video software includes drivers, plugins, and application like Blackmagic Desktop video utility and media express.

The computer requires at least 4GB of RAM, 64-bit version of Windows with the latest service pack installed; Windows 7,8 and 10 are supported. Now connect video hardware with USB 3.0, firstly connect a superspeed USB 3.0 cable between the unit and a dedicated USB

3.0 port on your computer.The identification of USB 3.0 show four pins for the USB 1.x/2.0 and the plastic insert is in the USB 3.0 standard blue color known as Pantone 300C.

There are mainly three applications which we are used.

1. Blackmagic Desktop Video Utility
2. Blackmagic Media Express
3. Blackmagic Disk Speed Test

6.9.1 Blackmagic Desktop Video Utility:

It provides a central location for configuring hardware settings, plus a real time status display showing the video connected to hardware's inputs and outputs. The home page of Blackmagic Desktop Video Utility displays connected hardware and provides an overview of all video activity on hardware's input and output connectors. Sending a video signal to the input will be automatically detected and the format will be displayed under the video input icon.

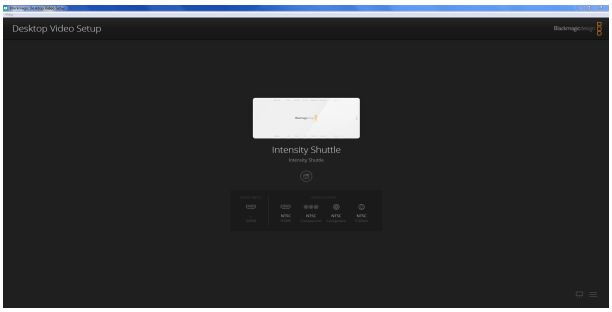

FIGURE 6.9: Blackmagic Desktop Video Utility home page

6.9.2 Blackmagic Media Express:

Blackmagic Media Express is included with every Ultrastudio, Decklink, and intensity as well as every ATEM switcher, Blackmagic camera, H.264 Pro Recorder, Teranex processor, and universal video hub. Media Express is a great tool when you don't need the complexity of NLE software but simply want to capture, Playback and output clips to tape.

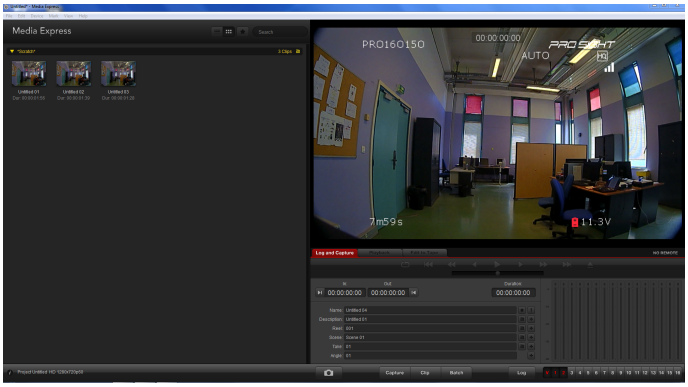

FIGURE 6.10: Blackmagic Media Express

In figure 6.10 display the video live streaming and to record clips, The captured clips are added to the media list on the left side of media express. There is a capture button to start and stop recording.

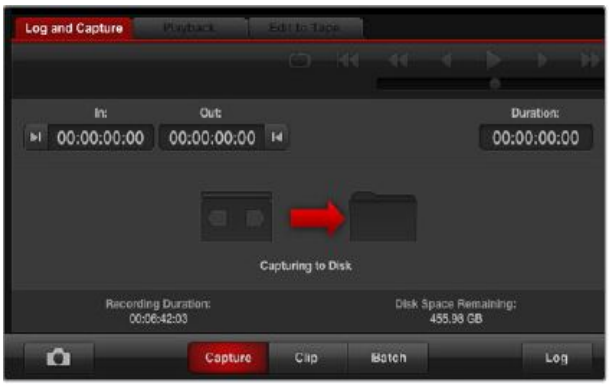

FIGURE 6.11: Capture the video

6.9.3 Blackmagic Disk Speed Test:

It measures the read and write performance of storage media in video frame sizes. In figure 6.12, the panel shows common video formats and displays a check mark or cross to indicate if disk performance is adequate. Performing several test cycles reveal any video formats for which the disk performance might be marginal. If a video format exhibits a check mark switching between a cross, it indicates that the disk storage cannot reliably support the video format. Moreover, it also shows how fast the device executes as indicated by the number in green, recorded in frames per second (fps).

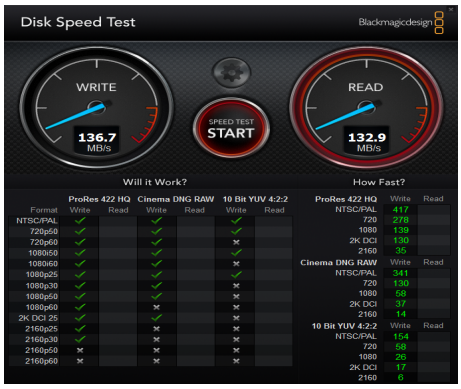

FIGURE 6.12: Blackmagic Disk speed Test

6.10 Video Streaming Station

The video streaming station is ready to record every moment through the drone. Before flying drone check all the connection should be tight otherwise it might be possible to lose signal .In figure 6.13 allow seeing the live video using both devices which are Black magic and Prosight.

FIGURE 6.13: Video Streamming Station

On the other hand, As shown in figure 6.14 show the drone flew outdoor with stable condition and gives us information about ground and air unit video signal strength, transmitter power voltage level and recorded serial number which would help to find the record in the system.

FIGURE 6.14: On screen display view

6.11 Results for Transmission distance Outdoor

Table 6.1 illustrates two area; Agricultural and Industrial to test the range of transmission distances outdoor. For the company, the column shows the value of the transceiver is more than the own experience, It depends on your way how to modify your device (for instance the position of the receiver should we open like a palm figure 6.5) then you would get better results. Test flight of HD drone indoor and outdoor helped to analyze the range of prosight device.

Area	**For Company**	**Own Experience**
Agriculture area (Open ground)	1000 meter	600 meter
Industrial area (Many Obstacles like Boundaries)	700 meter	300 meter

TABLE 6.1: Transmission distance Outdoor

Chapter 7

Conclusion

This chapter evaluates and interprets the results of the previous chapter. It summarizes the contributions and gives an outlook to further development.

At first, we described the different type of robotic platform that served as a testbed for our experiments. Afterward, we discussed the quad copter which from the perspective of control theory and discussed several possible alternatives how to model the system. Then we selected the PID controller as a suitable controller for our purposes. We showed that this controller performs reasonably well in practice on AR.Drone 2.0 when the proportional, derivative and integral constants of the PID controller are set with care. Subsequently, we studied the field of object tracking extensively. Firstly, we described the image processing algorithm which explored the color detection, edge detection etc. Finally a very efficient algorithm for edge detection- Hough transform

Before the Hough transform, we tried a different kind of methods. The first method was based on the mathematical trigonometric function which took a long time to detection because of drone monocular camera. Second the image processing some algorithm is very compact to detection circle and edge. Finally, in OpenCV node module has relied on the variable of upstream data to edge detection. On the other hand, this project was done several tasks, for instance, a method for autonomous landing an Unnamed aerial vehicle was developed as long as it is possible to use an Oriental roundel marker board utilize within different scenarios. In the real environment experimentation the work has been done automatic take-off and landing on a moving target but we already did a different scenario and it did really authenticate. Nevertheless, There was different task using the artificial neural network in ros platform using Gazebo simulation. In both task the strategy is once the target position has been reached, we detect the land pad again, now using the downward-facing camera. To correct possible drift while approaching the land

marker, the pose of the quadrotor with respect to the land marker is estimated and used to correct quadrotor position.

For the transmission data through the drone, we used a different kind of devices such as camera, transmitter, and receiver. This device mounted on a drone to take the high-quality video which would work for image processing. Beforeimage processing application to know the stability of drone because of the law of aerodynamics so we mounted the components the drone was stable in different altitudes such as 1m, 3m, and 15m. As far the drone was flying without the frame so gives the error for initial point. In the ANN, the gazebo simulator also includes a model for a quad copter. In the simulator, a successful landing was taken, thus producing the desired achievement without any obstacles. To sum up, the following tasks were developed.

- Autonomous tracking algorithm
- Autonomous approach landing algorithm
- Autonomous following search path algorithm (Boustrophedon path)
- Maked a drone for the high definition streaming video

7.1 Scope of the study

The scope of thesis has to commercial and military surveillance application, now a days the amazon company are making those kind of drone they would able to fly autonomous navigating path and landing on it. For instance the parcel package send via drone to delivered the package with in short time.

On the other hand, drone system specifically, US technology appears to have a long way to go. Some drones including the global hawk and scan eagle, are capable of automatic take-off and landing, and they could follow a pre- programmed flight path for extended periods of time provided that poor weather conditions or lack of power do not require continuous operator intervention. For the military reconnaissance they have another application for example imagine Humvees equipped with small landing pads for armed quad copters sent to attack targets, then autonomously return and land safely, all while the car is in motion, no less.

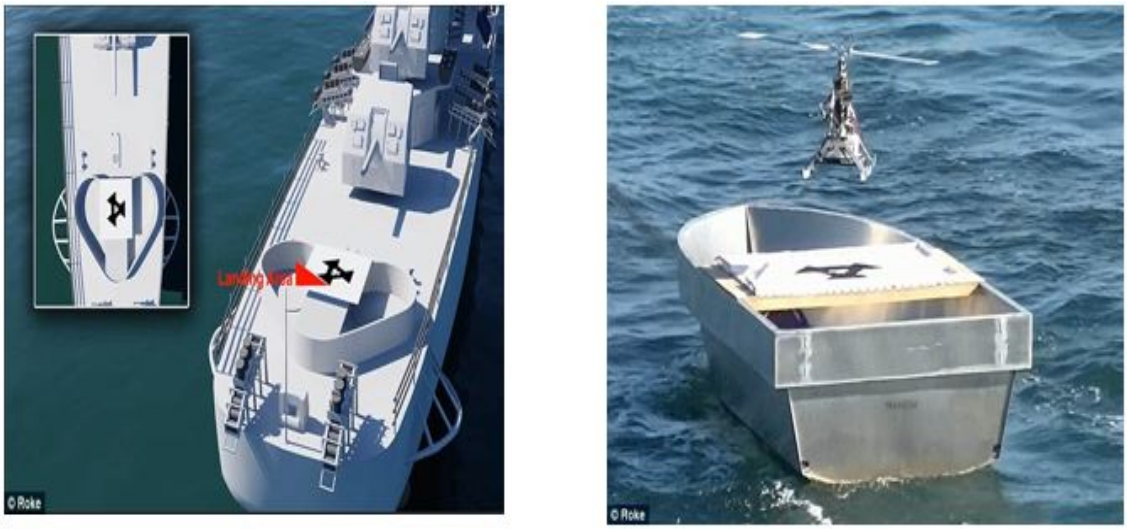

FIGURE 7.1: General overview project setup (source from Internet)

Appendix A

Node.js coding for autonomous take-off and landing on a mobile robot

A.1 PID Controller

```
% % PID Controller
module.exports = PID;
function PID(kp, ki, kd) {
    this.configure(kp, ki, kd);
    this.reset();
}

PID.prototype.configure = function(kp,ki,kd) {
    this._kp = kp;
    this._ki = ki;
    this._kd = kd;
}

PID.prototype.reset = function() {
    this._last_time = 0;
    this._last_error = Infinity;
    this._error_sum = 0;
}

PID.prototype.getCommand = function(e) {
    // Compute dt in seconds
    var time = Date.now();
    var dt = (time - this._last_time) / 1000

    var de = 0;
    if (this._last_time != 0) {
        // Compute de (error derivation)
        if (this._last_error < Infinity) {
```

```
            de = (e - this._last_error) / dt;
        }

        // Integrate error
        this._error_sum += e * dt;
    }

    // Update our trackers
    this._last_time = time;
    this._last_error = e;

    // Compute commands
    var command = this._kp * e
                + this._ki * this._error_sum
                + this._kd * de;

    return command;
}
```

A.2 Extended kalman Filter (EKF)

```
%% Extended Kalman Filter (EKF)
var sylvester    = require('sylvester');
var util         = require('util');

var Matrix       = sylvester.Matrix;
var Vector       = sylvester.Vector;

EKF.DELTA_T = 1 / 15; // In demo mode, 15 navdata per second

module.exports = EKF;
function EKF(options) {

  options = options || {};

  this._options     = options;
  this._delta_t     = options.delta_t || EKF.DELTA_T;

  this.reset();
}

EKF.prototype.state = function() {
    return this._state;
}

EKF.prototype.confidence = function() {
    return this._sigma;
}

EKF.prototype.reset = function() {
  this._state       = this._options.state   || {x: 0, y: 0, yaw: 0};
  this._sigma       = Matrix.I(3);
```

```
  this._q           = Matrix.Diagonal([0.0003, 0.0003, 0.0001]);
  this._r           = Matrix.Diagonal([0.3, 0.3, 0.3]);
  this._last_yaw    = null;
}

EKF.prototype.predict = function(data) {
    var pitch = data.demo.rotation.pitch.toRad()
      , roll  = data.demo.rotation.roll.toRad()
      , yaw   = normAngle(data.demo.rotation.yaw.toRad())
      , vx    = data.demo.velocity.x / 1000 //We want m/s instead of mm/s
      , vy    = data.demo.velocity.y / 1000
      , dt    = this._delta_t
    ;

    // We are not interested by the absolute yaw, but the yaw motion,
    // so we need at least a prior value to get started.
    if (this._last_yaw == null) {
        this._last_yaw = yaw;
        return;
    }

    // Compute the odometry by integrating the motion over delta_t
    var o = {dx: vx * dt, dy: vy * dt, dyaw: yaw - this._last_yaw};
    this._last_yaw  = yaw;

    // Update the state estimate
    var state = this._state;
    state.x   = state.x + o.dx * Math.cos(state.yaw) - o.dy * Math.sin(state.yaw)
    ;
    state.y   = state.y + o.dx * Math.sin(state.yaw) + o.dy * Math.cos(state.yaw)
    ;
    state.yaw = state.yaw + o.dyaw;

    // Normalize the yaw value
    state.yaw = Math.atan2(Math.sin(state.yaw),Math.cos(state.yaw));

    // Compute the G term (due to the Taylor approximation to linearize the
    function).
    var G = $M(
           [[1, 0, -1 * Math.sin(state.yaw) * o.dx - Math.cos(state.yaw) * o.dy],
            [0, 1,  Math.cos(state.yaw) * o.dx - Math.sin(state.yaw) * o.dy],
            [0, 0, 1]]
            );

    // Compute the new sigma
    this._sigma = G.multiply(this._sigma).multiply(G.transpose()).add(this._q);
}
 /*
  * measure.x:   x-position of marker in drone's xy-coordinate system (
    independant of roll, pitch)
  * measure.y:   y-position of marker in drone's xy-coordinate system (
    independant of roll, pitch)
  * measure.yaw: yaw rotation of marker, in drone's xy-coordinate system (
    independant of roll, pitch)
  *
```

```
  * pose.x:   x-position of marker in world-coordinate system
  * pose.y:   y-position of marker in world-coordinate system
  * pose.yaw: yaw-rotation of marker in world-coordinate system
  */
EKF.prototype.correct = function(measure, pose) {
    // Compute expected measurement given our current state and the marker pose
    var state  = this._state;
    var psi    = state.yaw;
    this._s    = {x: state.x, y: state.y, yaw: state.yaw};

    // Normalized the measure yaw
    measure.yaw = normAngle(measure.yaw);
    this._m = {x: measure.x, y: measure.y, yaw: measure.yaw};

    var z1 = Math.cos(psi) * (pose.x - state.x) + Math.sin(psi) * (pose.y - state
    .y);
    var z2 = -1 * Math.sin(psi) * (pose.x - state.x) + Math.cos(psi) * (pose.y -
    state.y);
    var z3 = pose.yaw - psi;
    this._z = {x: z1, y: z2, yaw: z3};

    // Compute the error
    var e1 = measure.x - z1;
    var e2 = measure.y - z2;
    var e3 = measure.yaw - z3;
    this._e = {x: e1, y: e2, yaw: e3};

    // Compute the H term
    var H = $M([[ -Math.cos(psi), -Math.sin(psi), Math.sin(psi) * (state.x - pose
    .x) - Math.cos(psi) * (state.y - pose.y)],
                [  Math.sin(psi), -Math.cos(psi), Math.cos(psi) * (state.x - pose
    .x) + Math.sin(psi) * (state.y - pose.y)],
                [  0, 0, -1]]);

    // Compute the Kalman Gain
    var Ht = H.transpose();
    var K = this._sigma.multiply(Ht).multiply(H.multiply(this._sigma).multiply(Ht
    ).add(this._r).inverse())

    // Correct the pose estimate
    var err = $V([e1, e2, e3]);
    var c = K . multiply(err);
    state.x = state.x + c.e(1);
    state.y = state.y + c.e(2);

//  TODO - This does not work, need more investigation.
//  In the meanwhile, we don't correct yaw based on observation.
//  state.yaw = state.yaw + c.e(3);

    this._sigma = Matrix.I(3).subtract(K.multiply(H)).multiply(this._sigma);
};

function normAngle(rad) {
    while (rad >  Math.PI) { rad -= 2 * Math.PI;}
    while (rad < -Math.PI) { rad += 2 * Math.PI;}
```

```
    return rad;
}

/** Converts numeric degrees to radians */
if (typeof(Number.prototype.toRad) === "undefined") {
  Number.prototype.toRad = function() {
    return this * Math.PI / 180;
  }
}

/** Converts radians to numeric dregrees */
if (typeof(Number.prototype.toDeg) === "undefined") {
  Number.prototype.toDeg = function() {
    return this * 180 / Math.PI;
  }
}
```

A.3 Controller

```
%% Controller
var EventEmitter = require('events').EventEmitter;
var Timers  = require('timers');
var util    = require('util');
var PID     = require('./PID');
var EKF     = require('./EKF');
var Camera  = require('./Camera');

EPS_LIN      = 0.1; // We are ok with 10 cm horizontal precision
EPS_ALT      = 0.1; // We are ok with 10 cm altitude precision
EPS_ANG      = 0.1; // We are ok with 0.1 rad precision (5 deg)
STABLE_DELAY = 200; // Time in ms to wait before declaring the drone on target

module.exports = Controller;
util.inherits(Controller, EventEmitter);
function Controller(client, options) {
    EventEmitter.call(this);

    options = options || {};

    // A ardrone client to pilot the drone
    this._client  = client;

    // The position of a roundel tag to detect
    this._tag     = options.tag || {x: 0, y: 0, yaw: 0};

    // Configure the four PID required to control the drone
    this._pid_x   = new PID(0.5, 0, 0.35);
    this._pid_y   = new PID(0.5, 0, 0.35);
    this._pid_z   = new PID(0.8, 0, 0.35);
    this._pid_yaw = new PID(1.0, 0, 0.30);

    // kalman filter is used for the drone state estimation
```

```
    this._ekf     = new EKF(options);

    // Used to process images and backproject them
    this._camera  = new Camera();

    // Control will only work if enabled
    this._enabled = false;

    // Ensure that we don't enter the processing loop twice
    this._busy = false;

    // The curretn target goal and an optional callback to trigger
    // when goal is reached
    this._goal     = null;
    this._callback = null;

    // The last known state
    this._state   = null,

    // The last time we have reached the goal (all control commands = 0)
    this._last_ok = 0;

    // Register the listener on navdata for our control loop
    var self = this;
    client.on('navdata', function(d) {
        if (!this._busy && d.demo) {
            this._busy = true;
            self._processNavdata(d);
            self._control(d);
            this._busy = false;
        }
    });
}

/*
 * Enable auto-pilot. The controller will attempt to bring
 * the drone (and maintain it) to the goal.
 */
Controller.prototype.enable = function() {
    this._pid_x.reset();
    this._pid_y.reset();
    this._pid_z.reset();
    this._pid_yaw.reset();
    this._enabled = true;
};

/*
 * Disable auto-pilot. The controller will stop all actions
 * and send a stop command to the drone.
 */
Controller.prototype.disable = function() {
    this._enabled = false;
    this._client.stop();
}
```

```
/*
 * Return the drone state (x,y,z,yaw) as estimated
 * by the Kalman Filter.
 */
Controller.prototype.state = function() {
    return this._state;
}

/*
 * Sets the goal to the current state and attempt to hover on top.
 */
Controller.prototype.hover = function() {
    this._go({x: this._state.x, y: this._state.y, z: this._state.z, yaw: this.
    _state.yaw});
}

/*
 * Reset the kalman filter to its base state (default is x:0, y:0, yaw:0).
 *
 * This is especially usefull to set mark the drone position as the starting
    position
 * after takeoff. We must disable, to ensure that the zeroing does not trigger a
    sudden move
 * of the drone.
 */
Controller.prototype.zero = function() {
    this.disable();
    this._ekf.reset();
}

/*
 * Move forward (direction faced by the front camera) by the given
 * distance (in meters).
 */
Controller.prototype.forward = function(distance, callback) {
   // Our starting position
   var state = this.state();

   // Remap our target position in the world coordinates
   var gx = state.x + Math.cos(state.yaw) * distance;
   var gy = state.y + Math.sin(state.yaw) * distance;

   // Assign the new goal
   this._go({x: gx, y: gy, z: state.z, yaw: state.yaw}, callback);
}

/*
 * Move backward by the given distance (in meters).
 */
Controller.prototype.backward = function(distance, callback) {
    return this.forward(-distance, callback);
}

/*
 * Move right (front being the direction faced by the front camera) by the given
```

```
 * distance (in meters).
 */
Controller.prototype.right = function(distance, callback) {
   // Our starting position
   var state = this.state();

   // Remap our target position in the world coordinates
   var gx = state.x - Math.sin(state.yaw) * distance;
   var gy = state.y + Math.cos(state.yaw) * distance;

   // Assign the new goal
   this._go({x: gx, y: gy, z: state.z, yaw: state.yaw}, callback);
}

/*
 * Move left by the given distance (in meters).
 */
Controller.prototype.left = function(distance, callback) {
    return this.right(-distance, callback);
}

/*
 * Turn clockwise of the given angle. Note that this does not
 * force a clockwise motion, if the angle is > 180 then the drone
 * will turn in the other direction, taking the shortest path.
 */
Controller.prototype.cw = function(angle, callback) {
    var state = this.state();
    var yaw   = state.yaw.toDeg() + angle;

    return this._go({x: state.x, y: state.y, z: state.z, yaw: yaw.toRad()},
    callback);
}

/*
 * Turn counter clockwise of the given angle (in degrees)
 */
Controller.prototype.ccw = function(angle, callback) {
    return this.cw(-angle, callback);
}

/*
 * Climb ups by the given distance (in meters).
 */
Controller.prototype.up = function(distance, callback) {
    var state = this.state();
    return this._go({x: state.x, y: state.y, z: state.z + distance, yaw: state.
    yaw}, callback);
}

/*
 * Lower itself by the given distance (in meters).
 */
Controller.prototype.down = function(distance, callback) {
    return this.up(-distance, callback);
```

```
}

/*
 * Go to the target altitude
 */
Controller.prototype.altitude = function(altitude, callback) {
    var state = this.state();
    return this._go({x: state.x, y: state.y, z: altitude, yaw: state.yaw},
    callback);
}

/*
 * Go to the target yaw (argument in degree)
 */
Controller.prototype.yaw = function(yaw, callback) {
    var state = this.state();
    return this._go({x: state.x, y: state.y, z: state.z, yaw: yaw.toRad()},
    callback);
}

/*
 * Sets a new goal and enable the controller. When the goal
 * is reached, the callback is called with the current state.
 *
 * x,y,z in meters
 * yaw in degrees
 */
Controller.prototype.go = function(goal, callback) {
    if (goal.yaw != undefined) {
        goal.yaw = goal.yaw.toRad();
    }

    return this._go(goal, callback);
}

Controller.prototype._go = function(goal, callback) {
    // Since we are going to modify goal settings, we
    // disable the controller, just in case.
    this.disable();

    // If no goal given, assume an empty goal
    goal = goal || {};

    // Normalize the yaw, to make sure we don't spin 360deg for
    // nothing :-)
    if (goal.yaw != undefined) {
        var yaw = goal.yaw;
        goal.yaw = Math.atan2(Math.sin(yaw),Math.cos(yaw));
    }

    // Make sure we don't attempt to go too low
    if (goal.z != undefined) {
        goal.z = Math.max(goal.z, 0.5);
    }
```

```
    // Update our goal
    this._goal = goal;
    this._goal.reached = false;

    // Keep track of the callback to trigger when we reach the goal
    this._callback = callback;

    // (Re)-Enable the controller
    this.enable();
}

Controller.prototype._processNavdata = function(d) {
    // EKF prediction step
    this._ekf.predict(d);

    // If a tag is detected by the bottom camera, we attempt a correction step
    // This require prior configuration of the client to detect the oriented
    // roundel and to enable the vision detect in navdata.
    // TODO: Add documentation about this
    if (d.visionDetect && d.visionDetect.nbDetected > 0) {
        // Fetch detected tag position, size and orientation
        var xc = d.visionDetect.xc[0]
          , yc = d.visionDetect.yc[0]
          , wc = d.visionDetect.width[0]
          , hc = d.visionDetect.height[0]
          , yaw = d.visionDetect.orientationAngle[0]
          , dist = d.visionDetect.dist[0] / 100 // Need meters
          ;

        // Compute measure tag position (relative to drone) by
        // back-projecting the pixel position p(x,y) to the drone
        // coordinate system P(X,Y).
        // TODO: Should we use dist or the measure altitude ?
        var camera = this._camera.p2m(xc + wc/2, yc + hc/2, dist);

        // We convert this to the controller coordinate system
        var measured = {x: -1 * camera.y, y: camera.x};

        // Rotation is provided by the drone, we convert to radians
        measured.yaw = yaw.toRad();

        // Execute the EKS correction step
        this._ekf.correct(measured, this._tag);
    }

    // Keep a local copy of the state
    this._state = this._ekf.state();
    this._state.z = d.demo.altitude;
    this._state.vx = d.demo.velocity.x / 1000 //We want m/s instead of mm/s
    this._state.vy = d.demo.velocity.y / 1000
}

Controller.prototype._control = function(d) {
    // Do not control if not enabled
    if (!this._enabled) return;
```

```
// Do not control if no known state or no goal defines
if (this._goal == null || this._state ==  null) return;

// Compute error between current state and goal
var ex   = (this._goal.x != undefined)   ? this._goal.x   - this._state.x   :
 0
  , ey   = (this._goal.y != undefined)   ? this._goal.y   - this._state.y   :
 0
  , ez   = (this._goal.z != undefined)   ? this._goal.z   - this._state.z   :
 0
  , eyaw = (this._goal.yaw != undefined) ? this._goal.yaw - this._state.yaw :
 0
  ;

// Normalize eyaw within [-180, 180]
while(eyaw < -Math.PI) eyaw += (2 * Math.PI);
while(eyaw >  Math.PI) eyaw -= (2 * Math.PI);

// Check if we are within the target area
if ((Math.abs(ex) < EPS_LIN) && (Math.abs(ey) < EPS_LIN) && (Math.abs(ez) <
EPS_ALT) && (Math.abs(eyaw) < EPS_ANG)) {
    // Have we been here before ?
    if (!this._goal.reached && this._last_ok != 0) {
        // And for long enough ?
        if ((Date.now() - this._last_ok) > STABLE_DELAY) {
            // Mark the goal has reached
            this._goal.reached = true;

            // We schedule the callback in the near future. This is to make
            // sure we finish all our work before the callback is called.
            if (this._callback != null) {
                setTimeout(this._callback, 10);
                this._callback = null;
            }

            // Emit a state reached
            this.emit('goalReached', this._state);
        }
    } else {
        this._last_ok = Date.now();
    }
} else {
    // If we just left the goal, we notify
    if (this._last_ok != 0) {
        // Reset last ok since we are in motion
        this._last_ok = 0;
        this._goal.reached = false;
        this.emit('goalLeft', this._state);
    }
}

// Get Raw command from PID
var ux = this._pid_x.getCommand(ex);
var uy = this._pid_y.getCommand(ey);
```

```
    var uz = this._pid_z.getCommand(ez);
    var uyaw = this._pid_yaw.getCommand(eyaw);

    // Ceil commands and map them to drone orientation
    var yaw  = this._state.yaw;
    var cx   = within(Math.cos(yaw) * ux + Math.sin(yaw) * uy, -1, 1);
    var cy   = within(-Math.sin(yaw) * ux + Math.cos(yaw) * uy, -1, 1);
    var cz   = within(uz, -1, 1);
    var cyaw = within(uyaw, -1, 1);

    // Emit the control data for auditing
    this.emit('controlData', {
        state:   this._state,
        goal:    this._goal,
        error:   {ex: ex, ey: ey, ez: ez, eyaw: eyaw},
        control: {ux: ux, uy: uy, uz: uz, uyaw: uyaw},
        last_ok: this._last_ok,
        tag:     (d.visionDetect && d.visionDetect.nbDetected > 0) ? 1 : 0
    });

    // Send commands to drone
    if (Math.abs(cx) > 0.01) this._client.front(cx);
    if (Math.abs(cy) > 0.01) this._client.right(cy);
    if (Math.abs(cz) > 0.01) this._client.up(cz);
    if (Math.abs(cyaw) > 0.01) this._client.clockwise(cyaw);

}

function within(x, min, max) {
    if (x < min) {
        return min;
    } else if (x > max) {
        return max;
    } else {
        return x;
    }
}
```

A.4 Camera

```
%% Camera
var sylvester    = require('sylvester');
var util         = require('util');


// TODO: Extend to support roll/pitch in back projection
// TODO: Add support for front-facing camera
// TODO: Make image aspect ratio configurable

// AR Drone 2.0 Bottom Camera Intrinsic Matrix
// https://github.com/tum-vision/ardrone_autonomy/blob/master/calibrations/
    ardrone2_bottom/cal.ymli
```

```
var K_BOTTOM = $M([[686.994766, 0, 329.323208],
                   [0, 688.195055, 159.323007],
                   [0, 0, 1]]);

module.exports = Camera;
function Camera(options) {
    this._options = options || {};
    this._k = this._options.k || K_BOTTOM;

    // We need to compute the inverse of K to back-project 2D to 3D
    this._invK = this._k.inverse();
}

/*
 * Given (x,y) pixel coordinates (e.g. obtained from tag detection)
 * Returns a (X,Y) coordinate in drone space.
 */
Camera.prototype.p2m = function(x, y, altitude) {
    // From the SDK Documentation:
    // X and Y coordinates of detected tag or oriented roundel #i inside the
    picture,
    // with (0; 0) being the top-left corner, and (1000; 1000) the right-bottom
    corner regardless
    // the picture resolution or the source camera.
    //
    // But our camera intrinsic is built for 640 x 360 pixel grid, so we must do
    some mapping.
    var xratio = 640 / 1000;
    var yratio = 360 / 1000;

    // Perform a simple back projection, we assume the drone is flat (no roll/
    pitch)
    // for the moment. We ignore the drone translation and yaw since we want X,Y
    in the
    // drone coordinate system.
    var p = $V([x * xratio, y * yratio, 1]);
    var P = this._invK.multiply(p).multiply(altitude);

    // X,Y are expressed in meters, in the drone coordinate system.
    // Which is:
    //          <--- front-facing camera
    //        |
    //       / \------- X
    //       \_/
    //        |
    //        |
    //        Y
    return {x: P.e(1), y: P.e(2)};
}
```

A.5 Mission

```
%% Mission

var async = require('async')
  , fs    = require('fs')
  ;

module.exports = Mission;
function Mission(client, controller, options) {

    options = options || {};

    this._options   = options;
    this._client    = client;
    this._control   = controller;

    this._steps     = [];
}

Mission.prototype.client = function() {
    return this._client;
}

Mission.prototype.control = function() {
    return this._control;
}

Mission.prototype.run = function(callback) {
    async.waterfall(this._steps, callback);
}

Mission.prototype.log = function(path) {
    var dataStream = fs.createWriteStream(path);
    var ekf = this._control._ekf;

    this._control.on('controlData', function(d) {
        var log = (d.state.x + "," +
                   d.state.y + "," +
                   d.state.z + "," +
                   d.state.yaw + "," +
                   d.state.vx + "," +
                   d.state.vy + "," +
                   d.goal.x + "," +
                   d.goal.y + "," +
                   d.goal.z + "," +
                   d.goal.yaw + "," +
                   d.error.ex + "," +
                   d.error.ey + "," +
                   d.error.ez + "," +
                   d.error.eyaw + "," +
                   d.control.ux + "," +
                   d.control.uy + "," +
                   d.control.uz + "," +
                   d.control.uyaw + "," +
                   d.last_ok + "," +
```

```
                    d.tag);

        if (d.tag > 0) {
            log = log + "," +
                  ekf._s.x + "," +
                  ekf._s.y + "," +
                  ekf._s.yaw.toDeg() + "," +
                  ekf._m.x + "," +
                  ekf._m.y + "," +
                  ekf._m.yaw.toDeg() + "," +
                  ekf._z.x + "," +
                  ekf._z.y + "," +
                  ekf._z.yaw.toDeg() + "," +
                  ekf._e.x + "," +
                  ekf._e.y + "," +
                  ekf._e.yaw.toDeg()
        } else {
            log = log + ",0,0,0,0,0,0"
        }

        log = log + "\n";

        dataStream.write(log);
    });
}

Mission.prototype.takeoff = function() {
    var self = this;
    this._steps.push(function(cb) {
        self._client.takeoff(cb);
    });

    return this;
}

Mission.prototype.land = function() {
    var self = this;
    this._steps.push(function(cb) {
        self._client.land(cb);
    });

    return this;
}

Mission.prototype.hover = function(delay) {
    var self = this;
    this._steps.push(function(cb) {
        self._control.hover();
        setTimeout(cb, delay);
    });

    return this;
}

Mission.prototype.wait = function(delay) {
```

```
    this._steps.push(function(cb) {
        setTimeout(cb, delay);
    });

    return this;
}

Mission.prototype.task = function(task) {
    this._steps.push(function(cb) {
        task(cb);
    });

    return this;
}

Mission.prototype.taskSync = function(task) {
    this._steps.push(function(cb) {
        task();
        cb();
    });

    return this;
}

Mission.prototype.zero = function() {
    var self = this;
    this._steps.push(function(cb) {
        self._control.zero();
        cb();
    });

    return this;
}

Mission.prototype.go = function(goal) {
    var self = this;
    this._steps.push(function(cb) {
        self._control.go(goal, cb);
    });

    return this;
}

Mission.prototype.forward = function(distance) {
    var self = this;
    this._steps.push(function(cb) {
        self._control.forward(distance, cb);
    });

    return this;
}

Mission.prototype.backward = function(distance) {
    var self = this;
    this._steps.push(function(cb) {
```

```
        self._control.backward(distance, cb);
    });

    return this;
}

Mission.prototype.left = function(distance) {
    var self = this;
    this._steps.push(function(cb) {
        self._control.left(distance, cb);
    });

    return this;
}

Mission.prototype.right = function(distance) {
    var self = this;
    this._steps.push(function(cb) {
        self._control.right(distance, cb);
    });

    return this;
}

Mission.prototype.up = function(distance) {
    var self = this;
    this._steps.push(function(cb) {
        self._control.up(distance, cb);
    });

    return this;
}

Mission.prototype.down = function(distance) {
    var self = this;
    this._steps.push(function(cb) {
        self._control.down(distance, cb);
    });

    return this;
}

Mission.prototype.cw = function(angle) {
    var self = this;
    this._steps.push(function(cb) {
        self._control.cw(angle, cb);
    });

    return this;
}

Mission.prototype.ccw = function(angle) {
    var self = this;
    this._steps.push(function(cb) {
        self._control.ccw(angle, cb);
```

```
    });

    return this;
}

Mission.prototype.altitude = function(altitude) {
    var self = this;
    this._steps.push(function(cb) {
        self._control.altitude(altitude, cb);
    });

    return this;
}

Mission.prototype.yaw = function(angle) {
    var self = this;
    this._steps.push(function(cb) {
        self._control.yaw(angle,cb);
    });

    return this;
}
```

A.6 StateEstimator

```
%% StateEstimator
var EventEmitter = require('events').EventEmitter;
var util         = require('util');

module.exports = StateEstimator;
util.inherits(StateEstimator, EventEmitter);
function StateEstimator(client, options) {
  EventEmitter.call(this);

  options = options || {};

  this._options           = options;
  this._client            = client;
  this._delta_t           = options.delta_t || StateEstimator.DELTA_T;
  this._state             = {roll: 0, pitch: 0, yaw: 0, x: 0, y: 0, z: 0};
  this._mode              = options.mode || "yaw";

  if (this._client == null) throw new Error("This won't work if you don't pass a
    proper ardrone client.");

  console.log('State estimator initialized in %s mode.', this._mode);

  this._bind();
}

StateEstimator.DELTA_T = 1 / 15; // In demo mode, 15 navdata per second
```

```
StateEstimator.prototype.state = function() {
    return this._state;
}

StateEstimator.prototype._bind = function() {
    var self = this;
    this._client.on('navdata', function(data) {
        self._processNavData(data);
    });
}

StateEstimator.prototype._processNavData = function(data) {
    var pitch = data.demo.rotation.pitch.toRad()
      , roll  = data.demo.rotation.roll.toRad()
      , yaw   = data.demo.rotation.yaw.toRad()
      , mag   = data.magneto.heading.fusionUnwrapped.toRad()
      , vx    = data.demo.velocity.x / 1000 //We want m/s instead of mm/s
      , vy    = data.demo.velocity.y / 1000
      , vz    = data.demo.velocity.z / 1000
      , alt   = data.demo.altitude
      , dt    = this._delta_t;
      ;

    var phi = (this._mode == "magneto" && mag != null) ? mag : yaw;

    this._state.x = this._state.x + dt * (vx * Math.cos(phi) - vy * Math.sin(phi)
    );
    this._state.y = this._state.y + dt * (vx * Math.sin(phi) + vy * Math.cos(phi)
    );
    this._state.z = alt;
    this._state.roll = roll;
    this._state.pitch = pitch;
    this._state.yaw = yaw;

    this.emit('state', this._state);
};

/** Converts numeric degrees to radians */
if (typeof(Number.prototype.toRad) === "undefined") {
  Number.prototype.toRad = function() {
    return this * Math.PI / 180;
  }
}
```

A.7 First Scenario

```
%% First Scenario
var df = require('dateformat')
var autonomy =require('ardrone-autonomy');
var mission = autonomy.createMission();
```

```
//mission.client().config('general:navdata_demo', true);
//mission.client().config('video:video_channel', 1);
//mission.client().config('detect:detect_type', 12);

mission.log("mission-" + df(new Date(), "yyyy-mm-dd_hh-MM-ss") + ".txt");

mission.takeoff()
       .altitude(2)  // Climb to          .altitude = 2 meter
       .wait(200)
       .forward(1)
       .left(1)
       .backward(1)
       .right(1)
       .wait(200)
       .land();


mission.run(function (err,result) {
     if (err) {
       //mission.client().stop();
       mission.client().land();
    } else {
         process.exit(0);
     }
 });
```

A.8 Second Scenario BT

```
%% main script
var df = require('dateformat')
var autonomy = require('ardrone-autonomy');
var mission  = autonomy.createMission();

mission.log("mission-" + df(new Date(), "yyyy-mm-dd_hh-MM-ss") + ".txt");


mission.takeoff()
       .zero()       // Sets the current state as the reference
       .altitude(2)  // Climb to altitude = 1 meter
       .wait(200)
       .forward(0.5)
       .right(1)
       .forward(0.5)
       .left(1)
       .forward(0.5)
       .right(1)
       .forward(0.5)
       .left(1)
       .forward(0.5)
       .right(1)
       .forward(0.5)
```

```
        .left(1)
        .forward(0.5)
        .right(1)
        .forward(0.5)
        .left(1)
        .forward(0.5)
        .right(1)
        .forward(0.5)
        .left(1)
        .forward(0.5)
        .hover(100)  // Hover in place for 1 second
        .wait(150)
        .land();


mission.run(function (err, result) {
    if (err) {
        console.trace("Oops, something bad happened: %s", err.message);
        mission.client().stop();
        mission.client().land();
    } else {
        console.log("Mission success!");
        process.exit(0);
    }
});
```

A.9 Third Scenario

```
%% main script
var df = require('dateformat')
var autonomy = require('ardrone-autonomy');
var mission = autonomy.createMission();

mission.log("mission-" + df(new Date(), "yyyy-mm-dd_hh-MM-ss") + ".txt");


mission.takeoff()
        .zero()
        .altitude(0.5)
        .wait(250000)
        .forward(1)
        .land();

mission.run(function (err, result) {
    if (err) {
        console.trace("Oops, something bad happened: %s", err.message);
        mission.client().stop();
        mission.client().land();
    } else {
        console.log("Mission success!");
        process.exit(0);
    }
```

```
});
```

A.10 Opencv

```
var cv = require('../lib/opencv');

var lowThresh = 0;
var highThresh = 100;
var nIters = 2;
var maxArea = 2500;

var GREEN = [0, 255, 0]; // B, G, R
var WHITE = [255, 255, 255]; // B, G, R
var RED   = [0, 0, 255]; // B, G, R

cv.readImage('./files/stuff.png', function(err, im) {
  if (err) throw err;
  var width = im.width();
  var height = im.height();
  if (width < 1 || height < 1) throw new Error('Image has no size');

  var big = new cv.Matrix(height, width);
  var all = new cv.Matrix(height, width);

  im.convertGrayscale();
  var im_canny = im.copy();

  im_canny.canny(lowThresh, highThresh);
  im_canny.dilate(nIters);

  var contours = im_canny.findContours();
  const lineType = 8;
  const maxLevel = 0;
  const thickness = 1;

  for(i = 0; i < contours.size(); i++) {
    if(contours.area(i) > maxArea) {
      var moments = contours.moments(i);
      var cgx = Math.round(moments.m10 / moments.m00);
      var cgy = Math.round(moments.m01 / moments.m00);
      big.drawContour(contours, i, GREEN, thickness, lineType, maxLevel, [0, 0]);
      big.line([cgx - 5, cgy], [cgx + 5, cgy], RED);
      big.line([cgx, cgy - 5], [cgx, cgy + 5], RED);
    }
  }

  all.drawAllContours(contours, WHITE);

  big.save('./tmp/big.png');
  all.save('./tmp/all.png');
  console.log('Image saved to ./tmp/big.png && ./tmp/all.png');
});
```

```
% % Detects circles
var cv = require('../lib/opencv');

var lowThresh = 0;
var highThresh = 100;
var nIters = 2;
var maxArea = 2500;

var GREEN = [0, 255, 0]; // B, G, R
var WHITE = [255, 255, 255]; // B, G, R
var RED   = [0, 0, 255]; // B, G, R

cv.readImage('./files/stuff.png', function(err, im) {
  if (err) throw err;
  var width = im.width();
  var height = im.height();
  if (width < 1 || height < 1) throw new Error('Image has no size');

  var big = new cv.Matrix(height, width);
  var all = new cv.Matrix(height, width);

  im.convertGrayscale();
  var im_canny = im.copy();

  im_canny.canny(lowThresh, highThresh);
  im_canny.dilate(nIters);

  var contours = im_canny.findContours();
  const lineType = 8;
  const maxLevel = 0;
  const thickness = 1;

  for(i = 0; i < contours.size(); i++) {
    if(contours.area(i) > maxArea) {
      var moments = contours.moments(i);
      var cgx = Math.round(moments.m10 / moments.m00);
      var cgy = Math.round(moments.m01 / moments.m00);
      big.drawContour(contours, i, GREEN, thickness, lineType, maxLevel, [0, 0]);
      big.line([cgx - 5, cgy], [cgx + 5, cgy], RED);
      big.line([cgx, cgy - 5], [cgx, cgy + 5], RED);
    }
  }

  all.drawAllContours(contours, WHITE);

  big.save('./tmp/big.png');
  all.save('./tmp/all.png');
  console.log('Image saved to ./tmp/big.png && ./tmp/all.png');
});
```

A.11 Bebop

```
% % Bebop main script
var bebop = require('node-bebop');

var drone = bebop.createClient();

drone.connect(function() {
  drone.takeOff();

  setTimeout(function() {
    drone.land();
  }, 5000);
});
```

Appendix B

Artificial Neural Network

B.1 autonomous_landingNeural.py

```
% % main script of autonomous landing
import roslib
roslib.load_manifest("ardrone_control")

import rospkg
import rospy
import time
import message_filters
import numpy as np
from std_msgs.msg import Empty
from sensor_msgs.msg import Range
from geometry_msgs.msg import Twist
from visualization_msgs.msg import Marker
from drone_controller import BasicDroneController

def land():
        controller.SendLand()
        rospy.loginfo(rospy.get_caller_id() + " Quad copter Landed")
        time.sleep(1)
        return

def sigmoid(num):
        return 1.0/(1+np.exp(-1*num))

def normalize(data,MAX):
        return (data+1)/MAX

def callback(data):
        if(controller.commandTimer==None):
                controller.StartSendCommand()
        x1 = normalize(data.pose.position.x,MAX)
        x2 = normalize(data.pose.position.y,MAX)
        x3 = normalize(data.pose.position.z,MAX)
        x4 = normalize(data.pose.orientation.x,MAX)
```

```
        x5 = normalize(data.pose.orientation.y,MAX)

        if(x3<=0.55):
                land()
                rospy.signal_shutdown("Landed")

        X=[1,x1,x2,x3,x4,x5]

        Y=sigmoid(np.dot(X,w1))
        Z=sigmoid(np.dot(Y,w2))
        Z=np.around(Z, decimals=2)
        rospy.loginfo(rospy.get_caller_id() + " Input  %s %s %s %s %s", format(x1
    , '.2f'),format(x2, '.2f'),format(x3, '.2f'),format(x4, '.2f'),format(x5, '.2
    f'))
        control(Z)

def load_weights(file):
        fp=np.load(file)
        w1=fp['arr_0']
        w2=fp['arr_1']
        #print w1
        #print w2
        return [w1,w2]

def control(Z):
        cntrl = Twist()
        x=2*Z[0]-1
        y=2*Z[1]-1
        z=2*Z[2]-1
        angz=2*Z[3]-1
        rospy.loginfo(rospy.get_caller_id() + " Output %s %s %s %s", x,y,z,angz)
        controller.SetCommand(y,x,angz,z)

def limit_reading(data,MAX):
        if(data>MAX):
                data=MAX
        return data
def listener():
    # In ROS, nodes are uniquely named. If two nodes with the same
    # node are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.

        rospy.Subscriber("/visualization_marker", Marker, callback)
        rospy.spin()

if __name__ == '__main__':
        # get an instance of RosPack with the default search paths
        rospack = rospkg.RosPack()

        path = rospack.get_path('ar2landing_neural')
        rospy.init_node('NNcontroller', anonymous=True)
        MAX = 2
        [w1,w2]=load_weights(path+"/data/weights.npz")
```

```
        controller = BasicDroneController()
        listener()
```

B.2 autonomous_search.py

```
%autonomous search fixed point
import roslib
# ardrone_tutorials has a good ardrone classs for controlling drone.
    load_manifest import that python module
roslib.load_manifest("ardrone_control")

import rospy
from geometry_msgs.msg import Twist
from std_msgs.msg import Empty
from visualization_msgs.msg import Marker
from drone_controller import BasicDroneController
import time



def detect_tag(data):
        # This will set a flag and stop executing autonomous search
        controller.SetCommand(0,0,0,0)
        controller.StopSendCommand()
        global detect
        detect = 1

def sleep(sec):
        while(sec and not rospy.is_shutdown()):
                if(detect==1):
                        break
                time.sleep(1)
                sec -= 1

def reset_drone(pub1):
        controller.SetCommand(0,0,0,0)
        time.sleep(1)

def search_pattern():
        controller.SetCommand(0,0,0,1.5)
        sleep(5)
        controller.SetCommand(0,0,0,0)
        sleep(1)
        while not rospy.is_shutdown() and detect == 0:
                controller.SetCommand(0,1,0,0)
                sleep(1)
                controller.SetCommand(-1,0,0,0)
                sleep(9)
                controller.SetCommand(0,1,0,0)
                sleep(1)
                controller.SetCommand(1,0,0,0)
                sleep(9)
```

```
if __name__ == '__main__':
        rospy.init_node('autonomous_search', anonymous=True)
        pub1 = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        controller = BasicDroneController()
        controller.StartSendCommand()
        time.sleep(1)
        detect = 0
        controller.SendTakeoff()
        controller.StartSendCommand()
        pub2 = rospy.Subscriber("/visualization_marker", Marker, detect_tag)
        search_pattern()
        reset_drone(pub1)
        rospy.spin()
```

B.3 train.py

```
% %main script
import rospkg
import time
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(num):
        return 1.0/(1+np.exp(-1*num))
def sigmoid_h(num):
        return np.tanh((num/2)/2+0.5)

# get an instance of RosPack with the default search paths
rospack = rospkg.RosPack()

# get the file path for rospy_tutorials
path = rospack.get_path('ar2landing_neural')

data = np.loadtxt(path+"/data/data.txt")
# X = 4x3 mxn
# Y = 4x1 oxp




Y_n = 4 # No of outputs
X = data[:,:-1*Y_n]
Y = data[:,-1*Y_n:]
X = X / 1#X.max(axis=0)
Y = Y / 1#Y.max(axis=0)
X = np.insert(X,0,1,axis=1)
[m, n] = X.shape
[o, p] = Y.shape
# -----------------
print "Data",data
print "X:",X
print "X:",X.shape
```

```
print "Y:",Y.shape

#-----------------

w1 = 2*np.random.random((n,4)) - 1
w2 = 2*np.random.random((4,p)) - 1

#-----------------

#print "w1",w1.shape
#print "w2",w2.shape

iteration = 100000

error =[float("inf")]
plt.ion()
plt.show()
for i in range(iteration):
        l1 = sigmoid(np.dot(X,w1))
        l2 = sigmoid(np.dot(l1,w2))

        l2_error = Y - l2
        l2_delta = l2_error * l2 * (1-l2) *.01

        l1_error = np.dot(l2_delta,w2.T)
        tmp = l1*(1-l1)
        l1_delta = l1_error*tmp*.01
        #---------------------

        w2 += np.dot(l1.T,l2_delta)
        w1 += np.dot(X.T,l1_delta)
        tmp = np.sum(l2_error**2,axis=0)
        tmp = np.sum(tmp,axis=0)
        error.append(tmp)
        #---------------------
        if(i%200==0):
                plt.scatter(i, tmp)
                plt.draw()
                print error[i+1]
                #if(abs(error[i]-error[i+1])<0.001):
                #       break


np.set_printoptions(precision=2)

np.savez(path+"/data/weights.npz",w1,w2)
```

B.4 train_data_collector.py

```
import rospkg
import rospy
import message_filters
```

```
import numpy as np
from visualization_msgs.msg import Marker
from geometry_msgs.msg import TwistStamped

def callback(data,twist_data):
        MAX = 2
        if (twist_data.twist.linear.x !=0 or twist_data.twist.linear.z !=0 or
    twist_data.twist.angular.z !=0 or twist_data.twist.linear.y !=0 ):
                x1 = normalize(data.pose.position.x,MAX)
                x2 = normalize(data.pose.position.y,MAX)
                x3 = normalize(data.pose.position.z,MAX)
                x4 = normalize(data.pose.orientation.x,MAX)
                x5 = normalize(data.pose.orientation.y,MAX)
                x6 = normalize(twist_data.twist.linear.x,MAX)
                x7 = normalize(twist_data.twist.linear.y,MAX)
                x8 = normalize(twist_data.twist.linear.z,MAX)
                x9 = normalize(twist_data.twist.angular.z,MAX)
                msg = str(format(x1, '.3f'))+" "+str(format(x2, '.3f'))+" "+str(
    format(x3, '.3f'))+" "+str(format(x4, '.3f'))+" "+str(format(x5, '.3f'))+" "+
    str(format(x6, '.3f'))+" "+str(format(x7, '.3f'))+" "+str(format(x8, '.3f'))
    +" "+str(format(x9, '.3f'))+"\n"

                rospy.loginfo(rospy.get_caller_id() + "I heard %s", msg)
                f.write(msg)

def normalize(data,MAX):
        return (data+1)/MAX

def listener():
    # In ROS, nodes are uniquely named. If two nodes with the same
    # node are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.

        rospy.init_node('NNDataCollector', anonymous=True)
        filter0 = message_filters.Subscriber('/visualization_marker', Marker)
        filter1 = message_filters.Subscriber('/cmd_vel_header', TwistStamped)
        ts = message_filters.ApproximateTimeSynchronizer([filter0, filter1],10,1)
        ts.registerCallback(callback)

        rospy.spin()

if __name__ == '__main__':
        # get an instance of RosPack with the default search paths
        rospack = rospkg.RosPack()

        # get the file path for rospy_tutorials
        path = rospack.get_path('ar2landing_neural')
        f = open(path+'/data/data.txt','a+')
        listener()
```

B.5 CMakeLists

```
cmake_minimum_required(VERSION 2.8.3)
project(ar2landing_neural)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  geometry_msgs
  rospy
  std_msgs
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)


## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
# catkin_python_setup()

################################################
## Declare ROS messages, services and actions ##
################################################

## To declare and build messages, services or actions from within this
## package, follow these steps:
## * Let MSG_DEP_SET be the set of packages whose message types you use in
##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
## * In the file package.xml:
##   * add a build_depend tag for "message_generation"
##   * add a build_depend and a run_depend tag for each package in MSG_DEP_SET
##   * If MSG_DEP_SET isn't empty the following dependency has been pulled in
##     but can be declared for certainty nonetheless:
##     * add a run_depend tag for "message_runtime"
## * In this file (CMakeLists.txt):
##   * add "message_generation" and every package in MSG_DEP_SET to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * add "message_runtime" and every package in MSG_DEP_SET to
##     catkin_package(CATKIN_DEPENDS ...)
##   * uncomment the add_*_files sections below as needed
##     and list every .msg/.srv/.action file to be processed
##   * uncomment the generate_messages entry below
##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )
```

```
## Generate services in the 'srv' folder
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )

## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   geometry_msgs#   std_msgs
# )

################################################
## Declare ROS dynamic reconfigure parameters ##
################################################

## To declare and build dynamic reconfigure parameters within this
## package, follow these steps:
## * In the file package.xml:
##   * add a build_depend and a run_depend tag for "dynamic_reconfigure"
## * In this file (CMakeLists.txt):
##   * add "dynamic_reconfigure" to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * uncomment the "generate_dynamic_reconfigure_options" section below
##     and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the 'cfg' folder
# generate_dynamic_reconfigure_options(
#   cfg/DynReconf1.cfg
#   cfg/DynReconf2.cfg
# )

###################################
## catkin specific configuration ##
###################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if you package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also
    need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES ar2landing_neural
#  CATKIN_DEPENDS geometry_msgs rospy std_msgs
#  DEPENDS system_lib
```

```
)

############
## Build ##
############

## Specify additional locations of header files
## Your package locations should be listed before other locations
# include_directories(include)
include_directories(
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(ar2landing_neural
#   src/${PROJECT_NAME}/ar2landing_neural.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(ar2landing_neural ${${PROJECT_NAME}_EXPORTED_TARGETS} ${
    catkin_EXPORTED_TARGETS})

## Declare a C++ executable
# add_executable(ar2landing_neural_node src/ar2landing_neural_node.cpp)

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(ar2landing_neural_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${
    catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against
# target_link_libraries(ar2landing_neural_node
#   ${catkin_LIBRARIES}
# )

##############
## Install ##
##############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# install(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables and/or libraries for installation
# install(TARGETS ar2landing_neural ar2landing_neural_node
#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
```

```
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

##############
## Testing ##
##############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_ar2landing_neural.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

Bibliography

[1] B.Hèrissè Asservissement et Navigation Autonome d'un drone à l'aide de capteurs visuels embarquès et d' exploitation de donnèes mutlicapteurs thèse de doctoorat, univ de Nice, 2010.

[2] Hoffman, G.M, Waslander, S.L, Tomlin, C.J Quadcopter Helicopter Trajectory Tracking Control, AIAA.

[3] Hyon Lim, Jaemann Park, Daewon Lee, and H. J. Kim. Build your own quadrotor: Open-source projects on unmanned aerial vehicles. Robotics Automation Magazine, IEEE , 19(3):33–45, Septem- ber 2012. ISSN 1070-9932. doi: 10.1109/MRA.2012.2205629

[4] Aplicaciones de la Visión Artificial from Universidad de Córdoba. ArUco: a minimal library for Aug- mented Reality applications based on OpenCV, April 2013. URL http://www.uco.es/investiga/ grupos/ava/node/26 . Consulted on October 2013.

[5] R. Barták, A. Hraško, D. Obdržálek, On Autonomous Landing of AR.Drone. Hands-on Experience. Proceedings of FLAIRS 28 (2014)

[6] Tiago gomes carreira, Quadcopter automatic landing on a docking station October 2013.

[7] S. Lange, N. Sünderhauf, P. Protzel, Vision Based Onboard Approach for Landing and Position Control of an Autonomous Multirotor UAV in GPS-Denied Environments.International Conference on Advanced Robotics, pp. 1-6 (2009)

[8] OpenCV software library. Accessed May 14, 2014.http://opencv.org/

[9] T. Krajn ık, V. Von asek, D. Fiser, and J. Faigl. AR-drone as a platform for robotic research and education. In Proc. of the Communications in Computer and Information Science (CCIS), 2011.

[10] Jakob Julian Engel Autonomous Camera-Based Navigation of a Quadrocopter December 15, 2011

[11] Yasmina Bestoui Sebbane Quadcopter Modeling October 2016

[12] Learn turtlebot with robot operating system http://learn.turtlebot.com/ http://learn.turtlebot.com/2015/02/01/7/

[13] https://www.mikrocontroller.com/images/Amimon20CONNEX20User20Guide_EN.pdf

[14] http://documents.blackmagicdesign.com/DesktopVideo/Desktop_Video_Manual_2015-03-31.pdf

[15] https://github.com/felixge/node-ar-drone

[16] https://github.com/krishnan793/ar2landing_neural