TASK MANAGEMENT, SYSTEM SELF TEST AND DIAGNOSTICS OF WAREHOUSE ROBOTIC NETWORK

Group Members

AHMAD SHAYAN ABID 150326
SUFIAN SAEED 150428
ABDULLAH ABBASI 150290

BE ELECTRICAL (2015-2019)

Project Supervisor

MUMAJJED UL MUDASSIR

Assistant Professor

DEPARTMENT OF ELECTRICAL ENGINEERING

FACULTY OF ENGINEERING

AIR UNIVERSITY, ISLAMABAD

TASK MANAGEMENT, SYSTEM SELF TEST AND DIAGNOSTICS OF WAREHOUSE ROBOTIC NETWORK

Final Year Project

(SESSION 2015-2019)

DEPARTMENT OF ELECTRICAL ENGINEERING

TASK MANAGEMENT, SYSTEM SELF TEST AND DIAGNOSTICS OF WAREHOUSE ROBOTIC NETWORK

Submitted by:

AHMAD SHAYAN ABID	**150326**
SUFIAN SAEED	**150428**
ABDULLAH ABBASI	**150290**

Project Supervisor

Mumajjed ul Mudassir

Assistant Professor

Head of Department

DR SHERYAR SALEEM

Assistant Professor

Acknowledgement

First of all, I would like to thank my project supervisor, Sir Mumajjed Mudassir for being such a tremendous mentor and guide throughout this outstanding journey. His illuminating views, constructive criticism and consistent feedback made us sail through numerous obstacles that came our way. I want to express my sincere gratitude to Sir Adnan Jafar for being a glimmer of hope in times of dismay, his unconditional engagement and encouragement has sustained us this far. His friendly advice and insightful comments have brought out the zeal and valour within us. I would like to pay my regards to Sir Shuja for providing access to his lab equipment and most importantly our Department Head Dr Sheryar Saleem for his strong impression and belief in every student. This has been a collective effort so how can I forget my beloved group members and the sleepless nights we put into shaping this project to perfection. Lastly, I would like to thank my parents for their spiritual support and my friends for expecting nothing less than excellence from my side.

Abstract

The project aims to develop a wireless network of swarm UAVs with some role and task (performing in different regions/sectors) in the ware house doing a predefined job. A system which shall monitor the location of UAVs on the map along with their functionality statuses of hardware modules (Battery, Camera and IMU sensors etc). The UAVs have another mode in which when the monitoring station gives command to a particular UAV, it leaves its default zone and move to the zone where the UAV is instructed to go. We give a specific task related to package finding and tracking in a ware house and swarm UAVs have to find and locate it. The swarm UAVs are equipped with modified object detection algorithm. This system of swarm was implemented using Robot Operating System (ROS) Framework. In this Project 3 quadcopter agents have been used which give feedback of the tasks they are performing. Image processing has been used on the robots for object detection, localization and tracking.

Table of Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Purpose

Experts say that around 85% of a company's inventory in the warehouse is stationary and many companies carry excess inventory because they don't know what they have or where it's located. Solving that problem could allow companies to cut their inventory carrying costs by billions of dollars. So to minimize these issues we need a network of robots which are low of cost, time saving, and can ease our work.

Figure 1.1: Warehouse environment

1.2 Simulation

A swarm of UAVs [1] could solve such problems. We tested our drones [22] in software based simulation before going towards real-time experiments. Softwares such as Gazebo and Rviz were a great help for us. Gazebo [32] is a 3D simulator that can

efficiently simulate complex environments. For monitoring sensor data and state information we used Rviz.

1.3 Packages

Our project is based on Robot Operating System (ROS) [15]. Software in ROS is organized in packages. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module.

1.4 Packages used in Task Management

The following are some useful packages used in Task Management:

• Ardrone Autonomy
• TUM Ardrone
• TUM Simulator
• Aruco ros
• Marker Pose Detection
• OpenCv
• Aruco mapping
• Ardrone Moveit

1.4.1 Ardrone_Autonomy

Ardrone Autonomy [12] is a ROS driver for ardrones. It is based on the official SDK of ardrone 2.0. This package was developed by Autonomy Lab Simon Fraser University.

Installation:

Open a terminal and enter the following commands:

$ apt-get install ros-indigo-ardrone-autonomy

$ cd /catkin_ws/src

$ git clone https://github.com/AutonomyLab/ardrone_autonomy.git -b indigo-devel

$ cd ∼/catkin_ws

$ rosdep install –from-paths src -i

$ catkin_make

Running:

To run ardrone autonomy in PC. Simply enter the following in a separate terminal:

$ rosrun ardrone_autonomy ardrone_driver

Now the information sent to the drone is called publishing and the information received from the drone is called subscribing. This information is carried in topic. The information received from the drone is published to the ”ardrone/navdata” topic. Fig1.2 shows the basic terminal interface that opens when ever ardrone autonomy [12] is run on PC(ground station).

Fig1.3 shows a typical ROS topic list in a terminal.

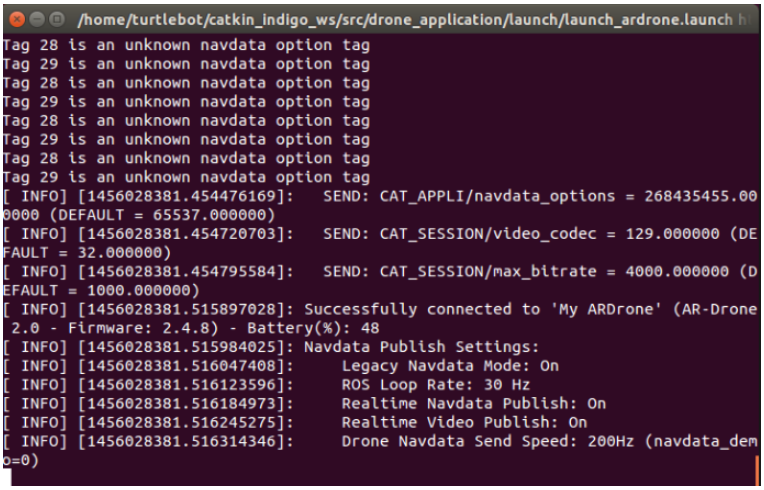

Figure 1.2: Ardrone Autonomy terminal

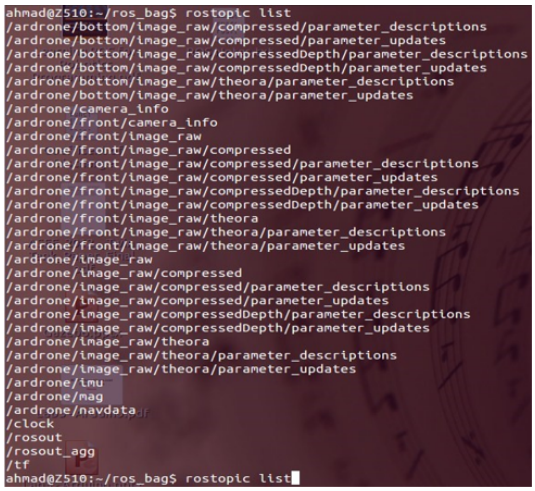

Figure 1.3: ROS-topic list

1.4.2 TUM_Ardrone

This is a package that is basically used to control ardrone using PC. Further-more it also contains information of the following :

• Scale-Aware Navigation

• Camera-Based Navigation

• Accurate Figure Flying with a Quadcopter

The nodes that TUM ardrone package [13] contains are:

• drone_state-estimation

• drone_autopilot

• drone_gui

Installation:

Open a terminal and enter the following commands to download and install TUM Ardrone package:

$ cd catkin_ws/src

$ git clone git://github.com/tum-vision/tum_ardrone.git tum_ardrone

$ cd..

$ catkin_make

Running:

Now to run tum ardrone first of all we will connect the drone with our PC via wifi. Then we will connect the drone using ardrone autonomy driver. For that enter the following command in a separate terminal:

$ rosrun ardrone_autonomy ardrone_driver

Now to control the drone we will use tum ardrone package [13] (as a controller). To run tum ardrone enter the following commads in a separate terminal:

$ cd

$ cd catkin_ws

$ roslaunch tum_ardrone tum_ardrone.launch

After all, the terminal must look like Fig1.4

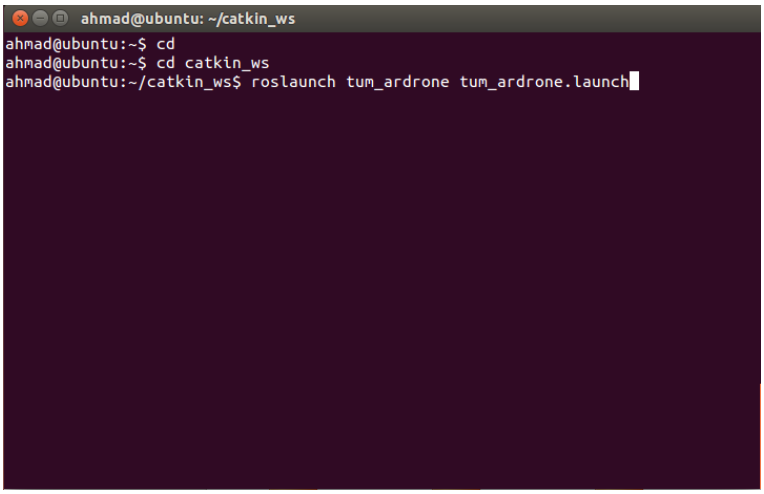

Figure 1.4: TUM Ardrone terminal

After the above a new window will apper which will look like Fig1.5

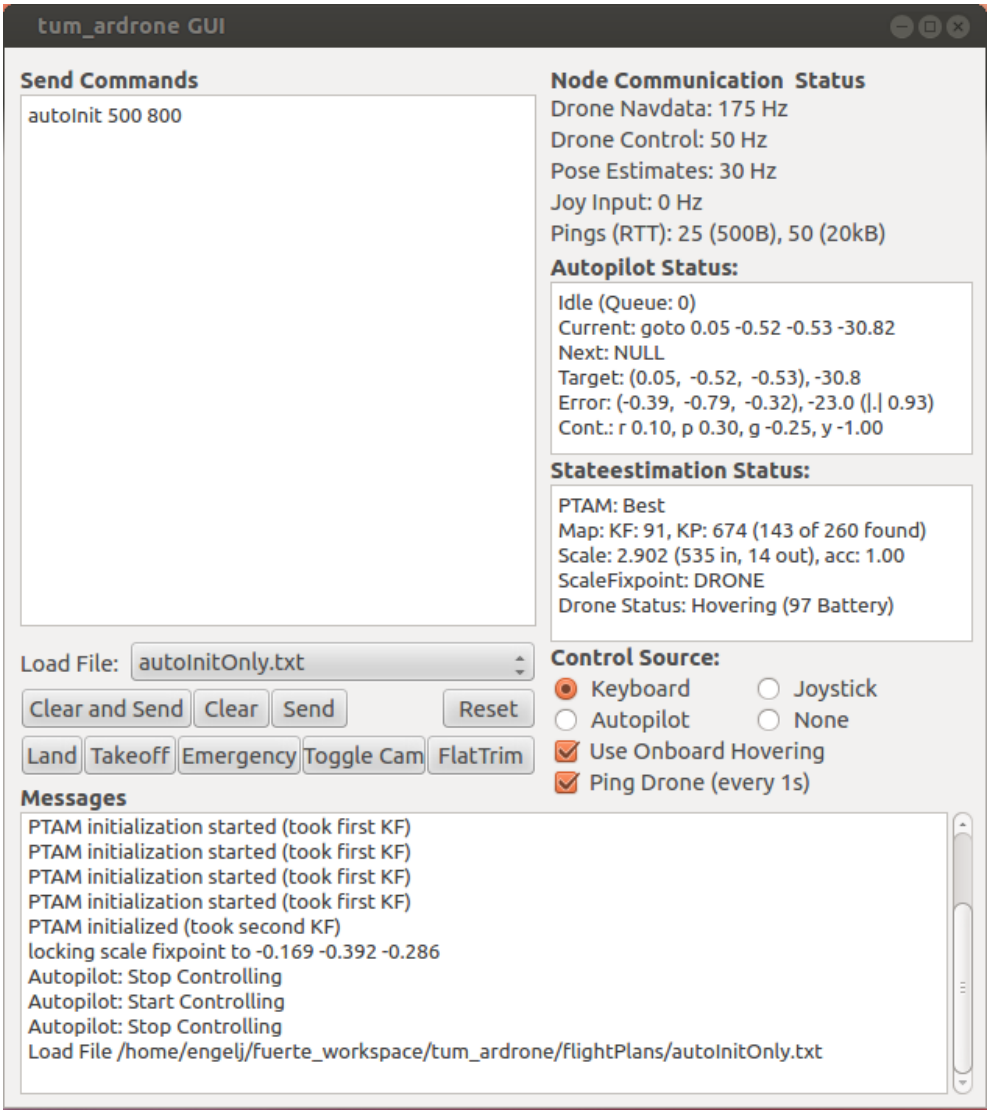

Figure 1.5: TUM Ardrone GUI

Fig1.6 shows the keys to fly ardrone using keyboard.

- q,a: fly up & down.
- i,j,k,l: fly horizontally.
- u,o: rotate yaw.
- F1: toggle emergency
- s: takeoff
- d: land

Figure 1.6: TUM Ardrone keyboard

Another window will appear which would look like Fig1.7

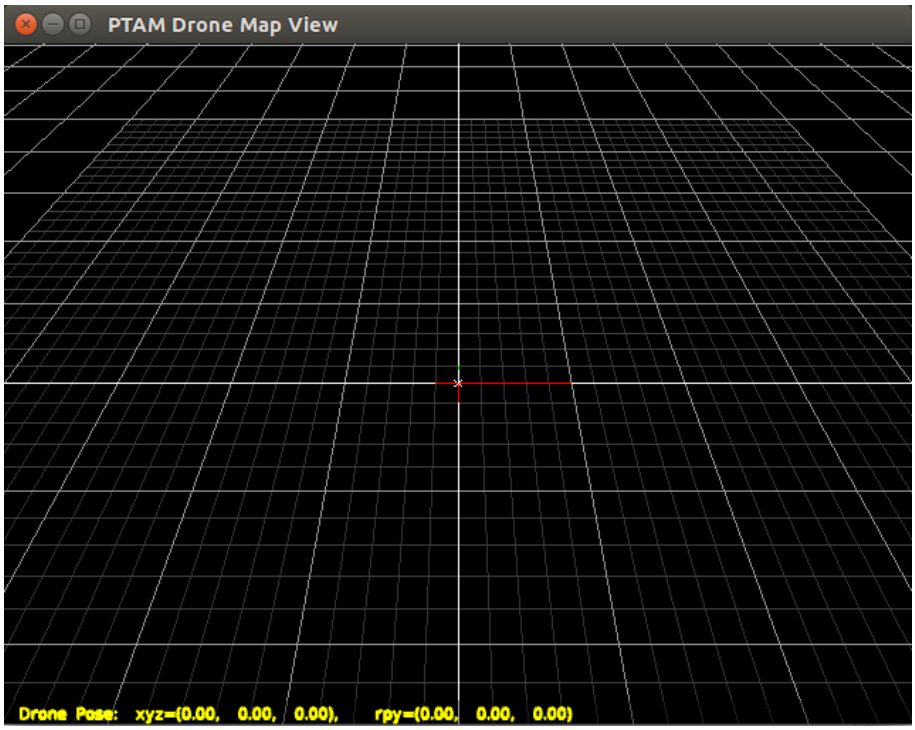

Figure 1.7: PTAM

And a third one as Fig1.8

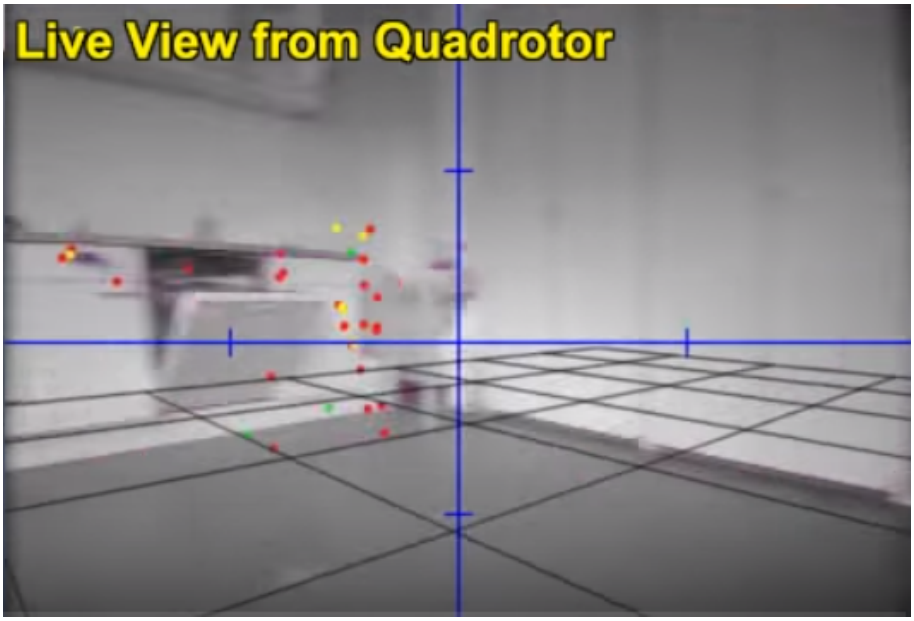

Figure 1.8: TUM Ardrone video

1.4.3 TUM_Simulator

This is the package that contains Gazebo simulator [32] and all of its other worlds for Parrot Ardrone 2.0. It has been written by Hongrong Huang and Juergen Sturm at technical University of MUNICH [25] .

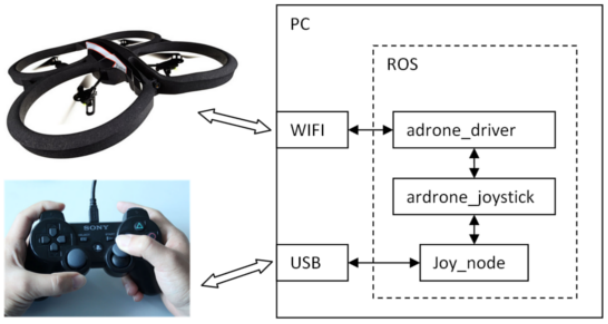

Figure 1.9: Real structure

The quadcopter connects with the user using Wifi and the quadcopter in hand can be further controlled via joystick(Fig1.9). A simulated picture of it can be seen in gazebo [25] . It appears as a game in it.(Fig1.10)

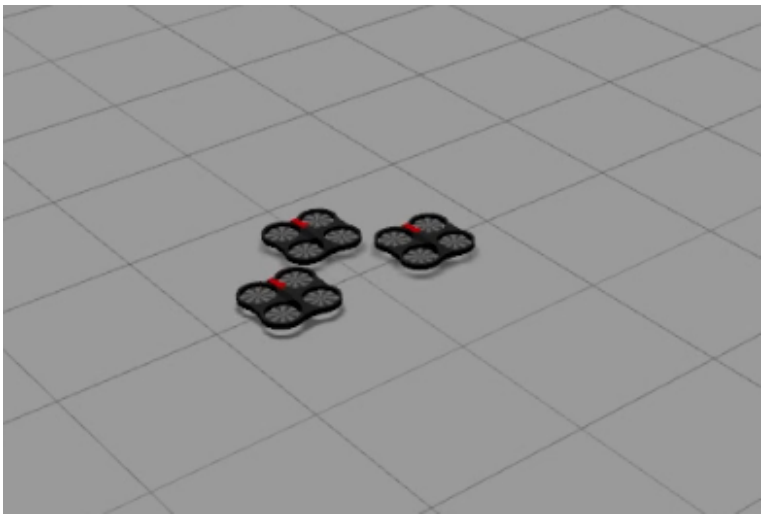

Figure 1.10: Simulated world in gazebo

Installation:

For Tum Simulator to run our computer must have ROS and gazebo simulator installed. Now to install Tum simulator enter the following commands in a separate terminal:

$ cd catkin_ws/src

$ git clone https://github.com/tum-vision/tum_simulator.git

$ cd catkin_make

The terminal must look like Fig1.11

Running:

Now to run Tum simulator for viewing ardrone in the world, enter the following command in a separate termial.

$ cd catkin_ws

$ roslaunch cvg_sim_gazebo ardrone_testworld.launch

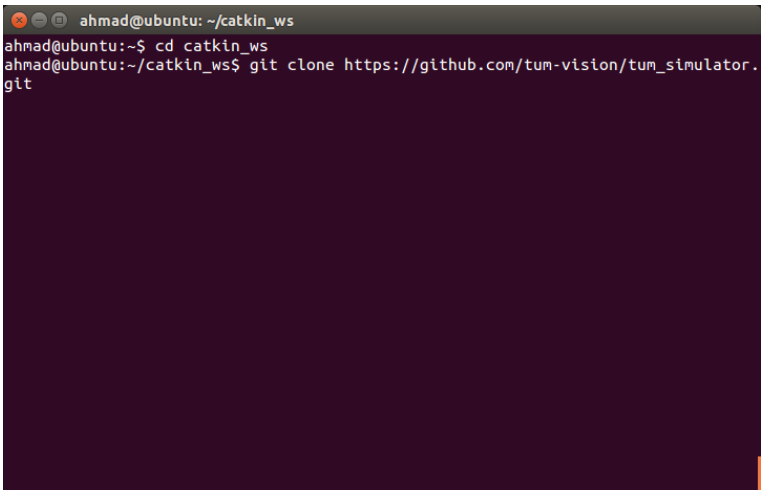

Figure 1.11: Tum Simulator terminal

Finally after some time (around 5-10 mins) a window of Gazebo [32] will appear which would have ardrone inside a world of gazebo and a white house in it. The window will look like Fig1.12

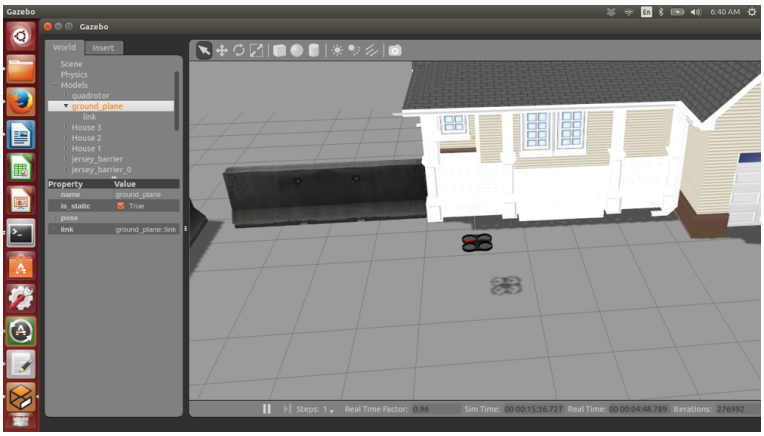

Figure 1.12: Ardrone in Gazebo world

1.5 System Design

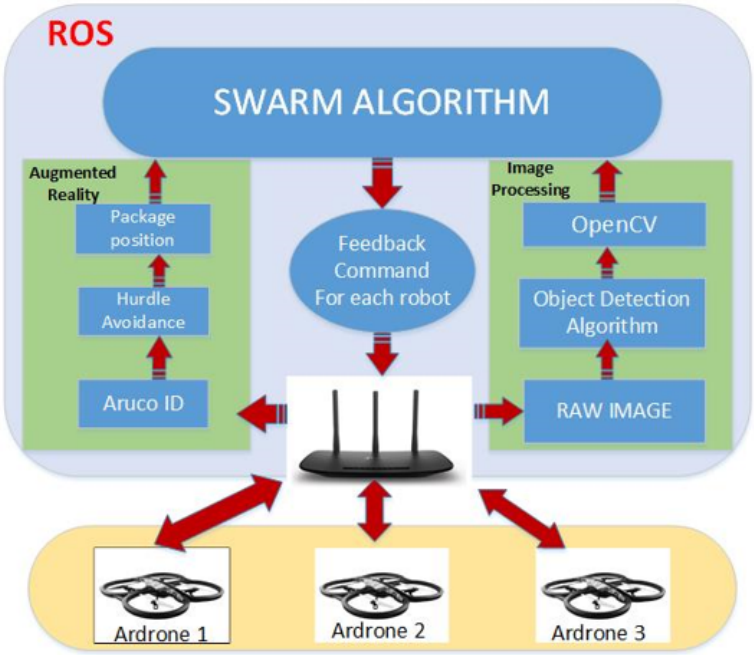

Figure 1.13: System Design

As shown in Fig1.13 Task Management revolves around ROS [] . ROS [33] acts as a main Operating system for the network. The Ardrones send image raw information to ROS via wifi router. The information is further sent to two blocks. The first block is of image processing. Here Raw Image is sent to Object Detection Algorithm which implies its algorithms to check whether its the required image or not. If the image is correct then it is further sent to Open Source Computer Vision(Open CV). Open CV [18] in turn converts the image to RGB format for further evaluation so that it could be made understandable to Swarm Algorithm. Now the swarm algorithm decides that what to do next and sends commands to individual drone it self.

In the second part the information is sent to Augmented Reality Block [28] where it is checked whether the image is the required Aruco ID or not. If it is the required one then it is further sent to Hurdle Avoidance block where it recognizes that if it a hurdle or an object. If it is not a hurdle then it is further sent to Package position block where the position of package is checked. Every Aruco ID [28] carried it own package information

which is checked by this block. Certainly after the package is localized, the information is sent to Swarm Algorithm which then makes decision that what to do next with the position of drones.

The Swarm Algorithm [2] controls the position of each drone by sending twist or empty commands through Wifi router.

1.6 Swarming

For demonstration purposes we have used a group of three Ar.drones [25] in our project, though it can be expanded as per desire of company. The concept of swarming is derived from insects or flock of birds Fig2.1. Swarming [2] allows us to build large numbers of low-cost expendable agents that can be used to overwhelm an adversary. We have used Decentralized swarming algorithm which means that every robot is intelligent enough to make its own decisions. Unlike master slave algorithm, in decentralized system if any of the agent fails or runs out of battery, others will continue the mission as none of them depend on each other.

Figure 1.14: A swarm of birds

Chapter 2

Robot Operating System

2.1 INTRODUCTION to ROS

Robot Operating System is a robotics middleware. Though ROS is not an operating system, it still provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of the commonly used functionality, and package management as well (Fig2.1).

Figure 2.1: ROS as an Operating System

2.2 Objectives of ROS

A frequently asked question over many years in ROS related seminars is the comparison of ROS with some other related software platforms such as OpenRTM, OPRos, Player, Orocos, Microsoft Robot software. Now each of them have their own purpose of use. The basic goals of ROS is to Collaborate each others work together. Such that if one company is expert in Image processing and other is professional in making quality batteries, then they both can work with each other and result in making a useful Robot that is perfect in every aspect. Hence ROS is perfect in code reuse in robotics network and development rather that individual researching.

For instance, ROS has the following characteristics:

- Distributed process: It is programmed in the form of minimum units so that it could be made executable (nodes) and each every process would run independently and the exchange of data could take place systematically.
- Package management: Multiple processes having same purposes are assembled together to create a package that can be made useful and developed by other developers in the near future and also redistributed and shared within themselves.

- Public repository: Every package is open-source meaning it is free of any restrictions for common public and every developer can use a repository such as Github and for specific professional functions a license could be obtained.
- API: While developing a program in ROS, ROS is designed to simply call an API and insert it easily in the code being used. As shown in later chapters it can be seen that ROS programming is not much different from other languages such as C++, Python, assembly etc.

- Supporting various programming languages: The ROS program provides its user with a large variety of library2 for the support of various programming languages. So the Library can be imported and converted in different Lua, and Ruby. In other words, you can develop a ROS program using a preferred programming language.

So such characteristics of ROS have allowed the users for the establishment of an environment where it is possible to collaborate on ROS development on a global scale. The reuse of code in ROS is becoming common and it is the ultimate goal of ROS.

2.3 Components of ROS

As it is shown in Table2.1, ROS consists of a client library to support various programming languages, a hardware interface is used for hardware control, communication for the data transmit and receive, the Robotics Application Framework to help create various Robotics Applications, the Robotics Application which is a service application based on the Robotics Application Framework, Simulation tools which can control the robot in a virtual space.

Client Layer	roscpp	rospy	roslisp	rosjava	roslibjs		
Robotics Application	MoveIt!	navigatioin	executive smach	descartes	rospeex		
	teleop pkgs	rocon	mapviz	people	ar track		
Robotics Application Framework	dynamic reconfigure	robot localization	robot pose ekf	Industrial core	robot web tools	ros realtime	mavros
	tf	robot state publisher	robot model	ros control	calibration	octomap mapping	
	vision opencv	image pipeline	laser pipeline	perception pcl	laser filters	ecto	
Communication Layer	common msgs	rosbag	actionlib	pluginlib	rostopic	rosservice	
	rosnode	roslaunch	rosparam	rosmaster	rosout	ros console	
Hardware Interface Layer	camera drivers	GPS/IMU drivers	joystick drivers	range finder drivers	3d sensor drivers	diagnostics	
	audio common	force/torque sensor drivers	power supply drivers	rosserial	ethercat drivers	ros canopen	
Software Development Tools	RViz	rqt	wstool	rospack	catkin	rosdep	
Simulation	gazebo ros pkgs	stage ros					

Table 2.1: Components of ROS

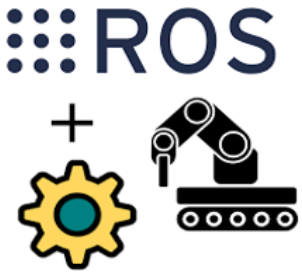

Figure 2.2: ROS in Industry

2.4 History of ROS

ROS was started in May 2007 by borrowing the early open-source robotic software frameworks including switchyard, which was developed by Dr. Morgan Quigley by the Stanford Artificial Intelligence Laboratory. In November 2007, U.S. robot company Willow Garage developed ROS even more. Willow Garage is a well-known company in the field for personal robots. It is also famous for developing and supporting the Point Cloud Library, which is widely used for 3-Dimensional devices such as Kinect and the image processing open source library such as OpenCV. Willow Garage started to develop ROS in November of 2007, and on January 22, 2010, ROS 1.0 just came out into the world. The official version that is known to us nowadays was released on March 2, 2010 known as ROS Box Turtle. Afterwards, C Turtle, Diamondback and many other versions were released in order of alphabets like Ubuntu and Android. ROS is based on the BSD 3-Clause License and Apache License 2.0, which means that everybody is allowed to modify, reuse, and redistribute the data. Not only this ROS also provided a large number of the latest software and participated actively in the education and academics, becoming known first through the robotics academic society. There is now ROSDay and ROSCon conferences which are held for developers and users, and also various community gatherings known for the name of ROS Meetup. In addition to this, development of robotic platforms that can apply ROS are also accelerating day by day. Some of the examples are PR2 which is Personal Robot and the TurtleBot, and many applications have been introduced through these platforms, resulting in ROS as the dominating robot software platform.

Figure 2.3: Open Robotics Logo

2.5 ROS Versions

ROS Releases and Conferences:

Fig2.4 shows animated versions in a single figure.

- Dec 08, 2017 - Release of ROS 2.0
- Sep 21, 2017 - ROSCon2017 (Canada)
- May 23, 2017 - Release of Lunar Loggerhead
- May 16, 2017 - Changed name from OSRF to Open Robotics
- Oct 8, 2016 - ROSCon2016 (South Korea)
- Oct 3, 2015 - ROSCon2015 (Germany)
- May 23, 2015 - Release of Jade Turtle
- Sep 12, 2014 - ROSCon2014 Conference (U.S.)
- Jul 22, 2014 - Release of Indigo Igloo
- Jun 6, 2014 - ROS Kong 2014 Conference (Hong Kong)
- Sep 4, 2013 - Release of Hydro Medusa
- May 11, 2013 - ROSCon2013 Conference (Germany)
- Feb 11, 2013 - Open Source Robotics Foundation
- Dec 31, 2012 - Release of Groovy Galapagos
- May 19, 2012 - ROSCon2012 Conference (U.S.)
- Apr 23, 2012 - Release of Fuertes
- Mar 2, 2011 - Release of Diamondback
- Aug 2, 2010 - Release of C Turtle
- Mar 2, 2010 - Release of Box Turtle
- Jan 22, 2010 - Release of ROS 1.0
- Nov 1, 2007 - Willow Garage starts development under the name ROS
- May 1, 2007 - Switchyard Project, Stanford AI LAB, Stanford University
- 2000 - Player/Stage Project, University of Southern California (USC)

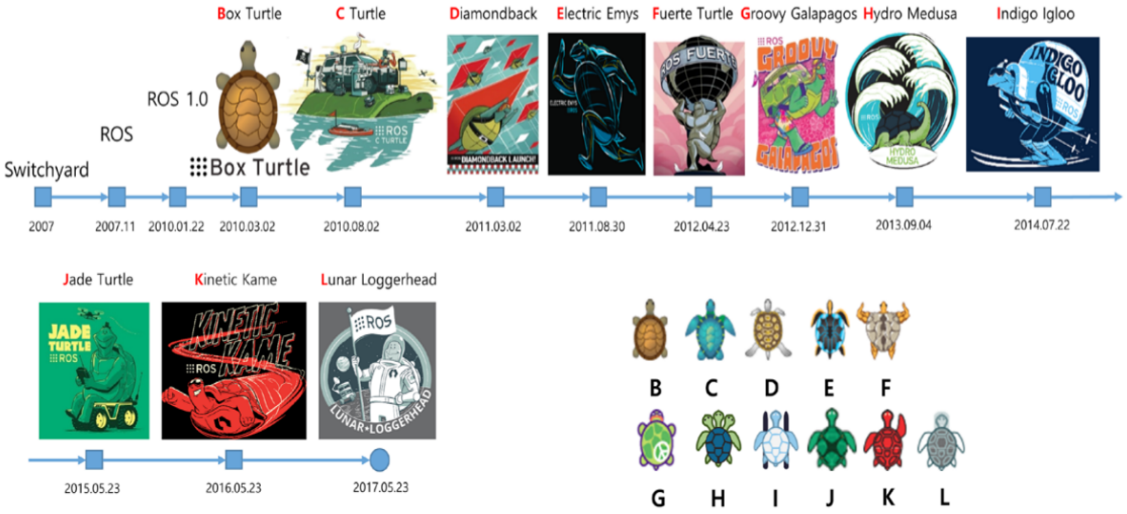

Figure 2.4: ROS versions

2.6 Selecting a Version

As ROS is a meta-operating system, we will need to choose an OS so that we could use it. ROS supports Debian, Ubuntu, OSx, Linux Mint, Fedora, Gentoo, openSUSE, Arch Linux, and Windows, but among all of these the most popular operating systems are Debian, Ubuntu and Linux. Ubuntu LTS is one of the most commonly used ROS version.

For information Ubuntu for the kinetic version of ROS can be found at the corresponding information page. As it can be seen from the following that Ubuntu versions, 14.04 Trusty is the Ubuntu 'T' version and is being released in alphabetical order. We should always go for a stable version. The latest version of ROS are still in progress. If the package we are using is not converted for the latest ROS version, we should wait or the stable version to be released.

- Ubuntu 18.04 Bionic Beave (LTS)
- Ubuntu 17.10 Artful Aardvark
- Ubuntu 17.04 Zesty Zapus
- Ubuntu 16.04 Xenial Xerus (LTS)

- Ubuntu 15.10 Wily Werewolf
- Ubuntu 15.04 Vivid Vervet
- Ubuntu 14.10 Utopic Unicorn
- Ubuntu 14.04 Trusty Tahr (LTS)
- Ubuntu 13.10 Saucy Salamander
- Ubuntu 13.04 Raring Ringtail
- Ubuntu 12.10 Quantal Quetzal
- Ubuntu 12.04 Precise Pangolin (LTS)
- Operating System: Ubuntu 16.04 Xenial Xerus (LTS)
- Linux Mint 18.x or Debian Jessie
- ROS: ROS Kinetic Kame

2.7 ROS Command List

In ROS, we enter commands in a Shell to perform tasks such as using file systems, editing, building, package management, etc. For using ROS properly we should get familiarize with all the commands. Following are some useful commands:

ROS Shell Commands:

Command	Importance	Command Explanation	Description
roscd	★★★	ros+cd(changes directory)	Move to the directory of the designated ROS package
rosls	★☆☆	ros+ls(lists files)	Check file list of ROS package
rosed	★☆☆	ros+ed(editor)	Edit file of ROS package
roscp	★☆☆	ros+cp(copies files)	Copy file of ROS package
rospd	☆☆☆	ros+pushd	Add directory to the ROS directory index
rosd	☆☆☆	ros+directory	Check the ROS directory index

Table 2.2: ROS shell commands

ROS Execution Commands:

Command	Importance	Command Explanation	Description
roscore	★★★	ros+core	master(ROS name service) + rosout(record log) + parameter server(manage parameter)
rosrun	★★★	ros+run	Run node
roslaunch	★★★	ros+launch	Launch multiple nodes and configure options
rosclean	★★☆	ros+clean	Examine or delete ROS log file

Table 2.3: ROS Execution commands

ROS Information Commands:

Command	Importance	Command Explanation	Description
rostopic	★★★	ros+topic	Check ROS topic information
rosservice	★★★	ros+service	Check ROS service information
rosnode	★★★	ros+node	Check ROS node information
rosparam	★★★	ros+param(parameter)	Check and edit ROS parameter information
rosbag	★★★	ros+bag	Record and play ROS message
rosmsg	★★☆	ros+msg	Check ROS message information

Table 2.4: ROS Information Commands

ROS Catkin Commands:

Command	Importance	Description
catkin_create_pkg	★★★	Automatic creation of package
catkin_make	★★★	Build based on catkin build system
catkin_eclipse	★★☆	Modify package created by catkin build system so that it can be used in Eclipse
catkin_prepare_release	★★☆	Cleanup log and tag version during release
catkin_generate_changelog	★★☆	Create or update 'CHANGELOG.rst' file during release
catkin_init_workspace	★★☆	Initialize workspace of the catkin build system
catkin_find	★☆☆	Search catkin

Table 2.5: ROS Catkin commands

ROS Shell Commands:

Command	Importance	Command Explanation	Description
roscd	★★★	ros+cd(changes directory)	Move to the directory of the designated ROS package
rosls	★☆☆	ros+ls(lists files)	Check file list of ROS package
rosed	★☆☆	ros+ed(editor)	Edit file of ROS package
roscp	★☆☆	ros+cp(copies files)	Copy file of ROS package
rospd	☆☆☆	ros+pushd	Add directory to the ROS directory index
rosd	☆☆☆	ros+directory	Check the ROS directory index

Table 2.6: ROS Shell Commands

Chapter 3

SIMULATORS

3.1 Gazebo

3.1.1 Why Gazebo?

Robot simulation is an essential tool in any robotic Industry. So a great and well designed simulator software is a basic requirement in it. To test the algorithm, train AI system and perform regression testing is a need nowadays. In such situation Gazeboo offers the ability to accurately simulate a robot in complex indoor and outdoor environment.

Figure 3.1: Gazebo icon

Gazebo follows many physics laws including gravity, air resistance, fluif dynamics and liquid flow etc. Gazebo provides us with high-graphics and graphical interfaces. All out of it Gazebo is an open source and free software having a vibrant community ran by experienced developers and researchers.

In gazebo the robot models are uploaded in the form of SDF files.

3.1.2 What is SDF?

SDF is an XML format of a robot in gazebo(see Fig3.2). It is used in simulators. Originally SDF were only part of basic gazebo but years later today SDF has been opened for the general public use which means that anybody and can use and modify it according to his/her need.

We modified certain SDF files to add three more Ardrones in Gazebo world so that

Figure 3.2: SDF robot

we could test our swarm UAVs in simulation before hand. Fig3.3 shows the simulated Ardrones in Gazebo world.

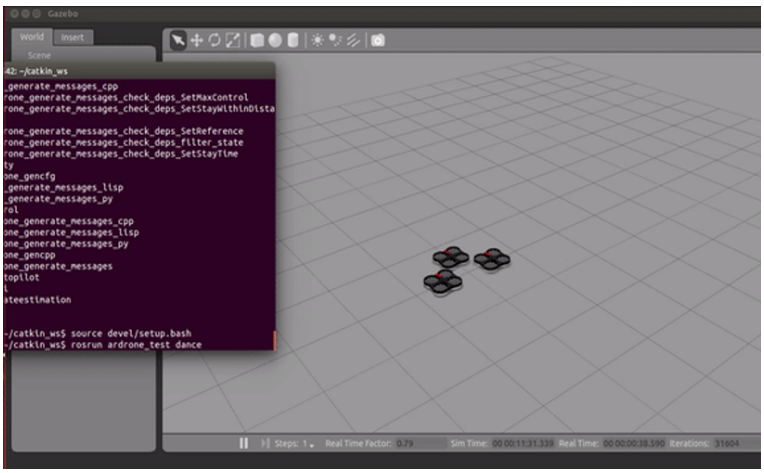

Figure 3.3: Three Ardrones in Gazebo world

3.1.3 Gazebo Simulator supported version

The main version of gazebo that is present in ROS is selected at the beginning of the ROS installation(v7.0) and is kept during the whole life of the ROS distribution. ROS Indigo uses an older version of Gazebo which is v2.0. Currently NASA (Fig3.4) and boston Dyamics are widely using ROS.

Figure 3.4: NASA-GM Robonaut2

3.2 RVIZ:

Rviz (ROS visualization) is a 3Dimensional visualizing tool for displaying the sensor data and state information from the ROS.

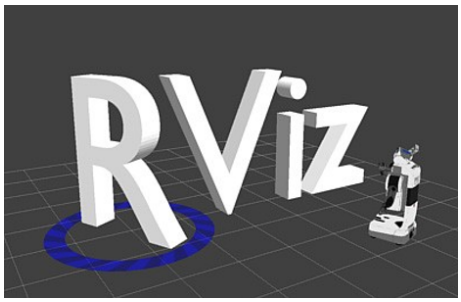

Figure 3.5: ROS

Using rviz, we can visualize a virtual model of the robot. We can also display live

representations of sensor values that are coming over the ROS Topics including camera data, infrared distance measurements, sonar data, and other physics formulas etc.

A detailed description about Rviz is covered in the later chapters.

Installation:

$ sudo apt-get update

$ sudo apt-get install ros-indigo-rviz

Usage:

To run Rviz run the following command in terminal:

$ rosrun rviz rviz

Robot Visualization

As an example a visualized robot in RVIZ is shown in Fig3.6

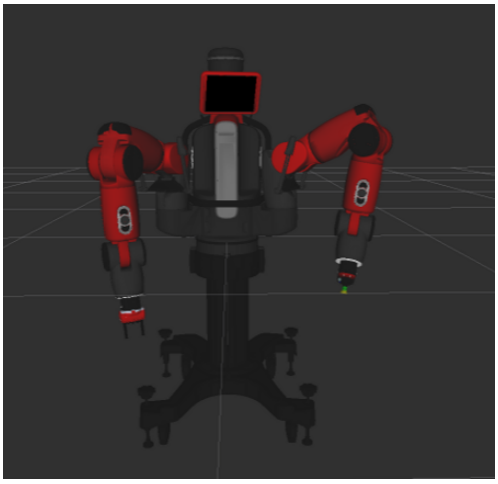

Figure 3.6: RVIZ Model

Rviz display

For a simulated ardrone in RVIZ, the picture may look Fig3.7

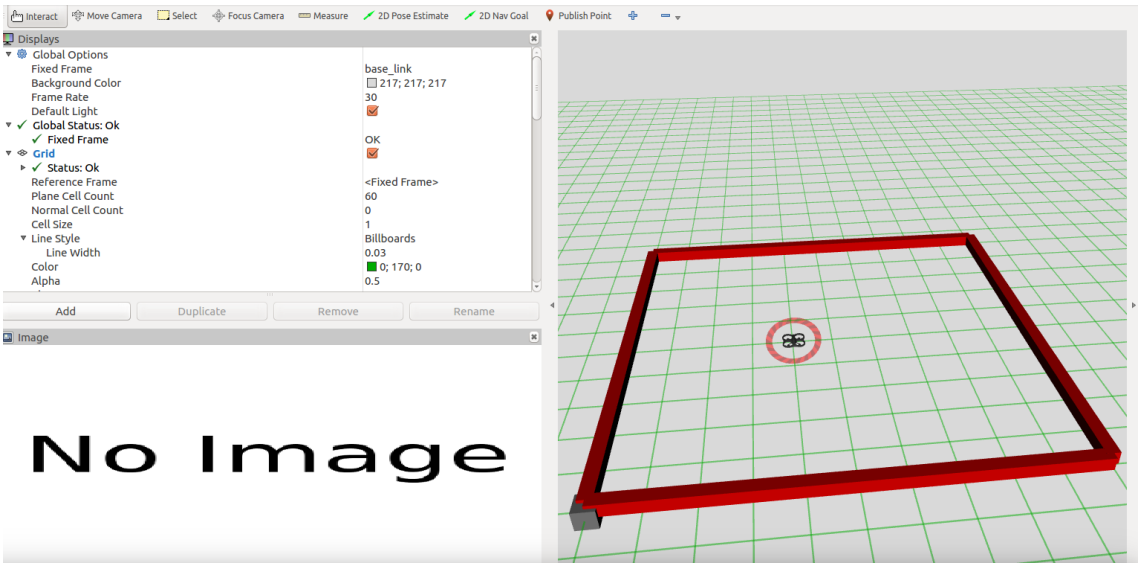

Figure 3.7: Ardrone in RVIZ

Chapter 4

Swarm Intelligence

4.1 Introduction

As, Swarm Intelligence is discussed previously in project introduction, now here we will discuss in detail the definition of swarming and from where the concept of swarming was taken. Step by step we will see how swarming is implemented on drones. So first we will start with the definition of Swarm.

4.2 Definition

It is the collective behavior of systems, natural or artificial to be decentralized and self-organized. The inspiration often comes from nature especially biological studies of insects, ants and other fields in nature which may include bird flocking ,animal herding, bacterial growth fish schooling where swarm behaviors occurs.

Figure 4.1: Biological swarms in the nature

As there is no centralized control structure dictating the agents how to behave. The agents simply follow some simple rules such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents.

4.3 Swarm Robotics

The application of swarm principles to robots emerges swarm robotics. Swarm robotics is currently one of the most important application areas for swarm intelligence. Swarm robotics is an approach to the coordination of multiple robots as a system which consists of large numbers of mostly simple physical robots. The advantage of swarm is that it provides the possibility of enhanced task performance, high reliability (fault tolerance), low unit complexity and decreased cost over traditional robotic systems. They can accomplish some tasks that would be impossible for a single robot to achieve.

Figure 4.2: Swarm of drones

For example Swarm bots are a collection of mobile robots able to self-assemble and to self-organize in order to solve problems that cannot be solved by a single robot. These robots combine the power of swarm intelligence with the flexibility of self-reconfiguration as aggregate swarm-bots can dynamically change their structure to match environmental variations. Swarm robots can be applied to many fields, such as flexible manufacturing systems, spacecraft's, inspection/maintenance, construction, agriculture and medicine work.

4.4 Software Implementation

As we have implemented this swarm quadcopters system algorithm using ROS platform. In this project, we have modified two package first is ardrone autonomy and second is tum ardrone packages to be reusable for multiple quadcopters using their IP addresses. Ardrone autonomy is a package used as driver between ROS and the quadcopter. Tum ardrone package is used as odometry system for estimating robots position from its IMU sensor. This package is also used to autopilot system that handles robot's moving to any position. This node subscribes robot pose and image data. Then publish image to tum ardrone package to display live video from drone's front camera.

4.5 Software Simulation

To visualize the results of project we choose a gazebo simulator in which we can observe the behavior of our drone according to written scripts. In gazebo we need a environment in which we have the structure of ardrone so for that we used the package named as tum simulator which created a environment which was required.

4.6 Tum Simulator

This package contains the implementation of a gazebo simulator for the Ardrone 2.0 and has been written by Hongrong Huang and Juergen Sturm of the Computer Vision Group at the Technical University of Munich.This package is based on the ROS package tu-darmstadt-ros-pkg by Johannes Meyer and Stefan Kohlbrecher and the Ardrone simulator which is provided by Matthias Nieuwen-huisen. Fig3.8 shows the behavior of ardrone in gazebo simulator using tum simulator package.We modified this package for multiple ardrones to see the behavior of swarm of ardrones on 3D simulator. We made necessary changes in tum simulator internal files to make a world of three ardrones.

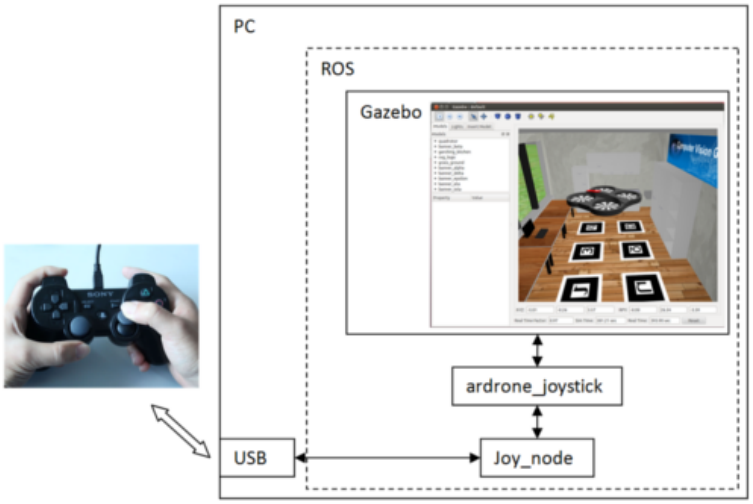

Figure 4.3: TUM Simulator Working

Fig4.4 shows the behavior of ardrone in gazebo simulator using tum simulator package. We modified this package for multiple ardrones to see the behavior of swarm of ardrones

on 3D simulator. We make necessary changes in tum simulator internal files to make a world of three ardrones.

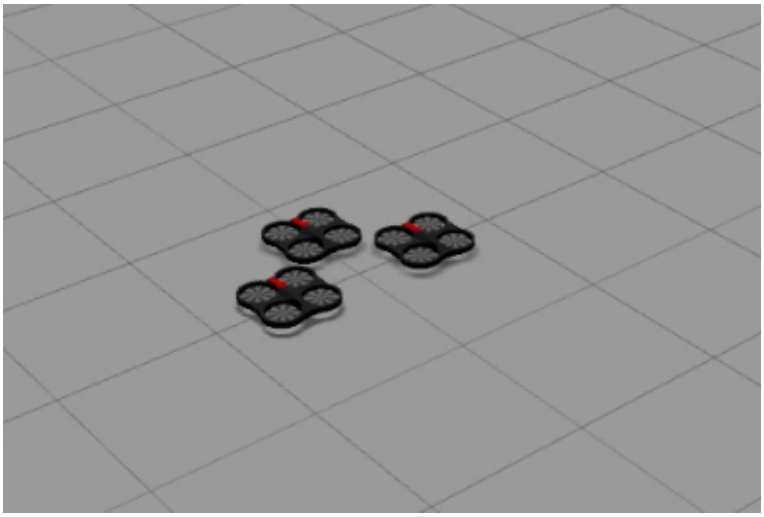

Figure 4.4: Multiple Ardrones in gazebo simulator

4.7 Ardrone autonomy

It is ROS package made for ardrone to act a driver between ROS and the drone and make both accessible for each other. It also send and receive information from ardrone .This driver is based on official AR-Drone SDK version 2.0.1. ardrone autonomy is a fork of AR-Drone Brown driver. This package is developed in Autonomy Lab of Simon Fraser University by Mani Monajjemi and other Contributors . We modified this package to connect multiple ardrones to one ground station through router. By modifying this package for multiple ardrone, we are able to fly multiple drones from one ground station.

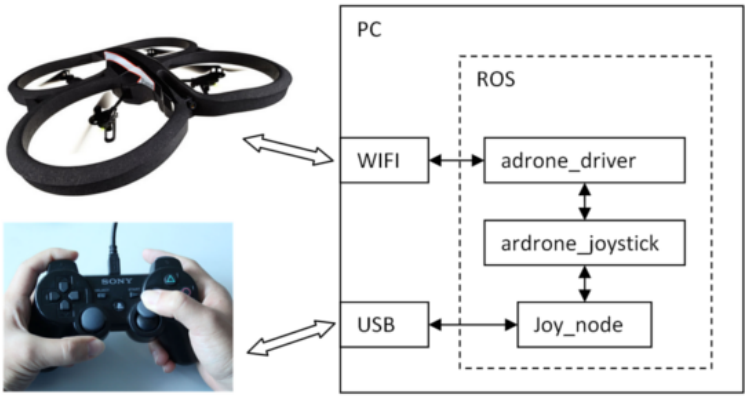

Figure 4.5: Ardrone Autonomy package working

4.8 TUM Ardrone

This is a main package to control the drone using ardrone autonomy package commands. This package is used to provide graphical user interface to control the drone and watch the live streaming from the front cameras of our drones. We modified this package for multiple ardrones to send controlling commands to multiple ardrones and receive information of camera and sensors from multiple ardrone.

4.9 Block Diagram

The working of swarm behaviour is completely described in a block diagram as shown in Fig4.6. Three ardrones are connected to ROS platform via router. In ROS, ardrone_autonomy package works as a driver between three ardrones and ROS. Using this package we can subscribe infromation from three ardrones which will be given further to swarm of multiple quadcopters. In swarm of multiple quadcopter information of each ardrone is seperated by their name which helps in publishing new commands according to the given information. After differentiating three ardrones information it will further given to tum_ardrone package.

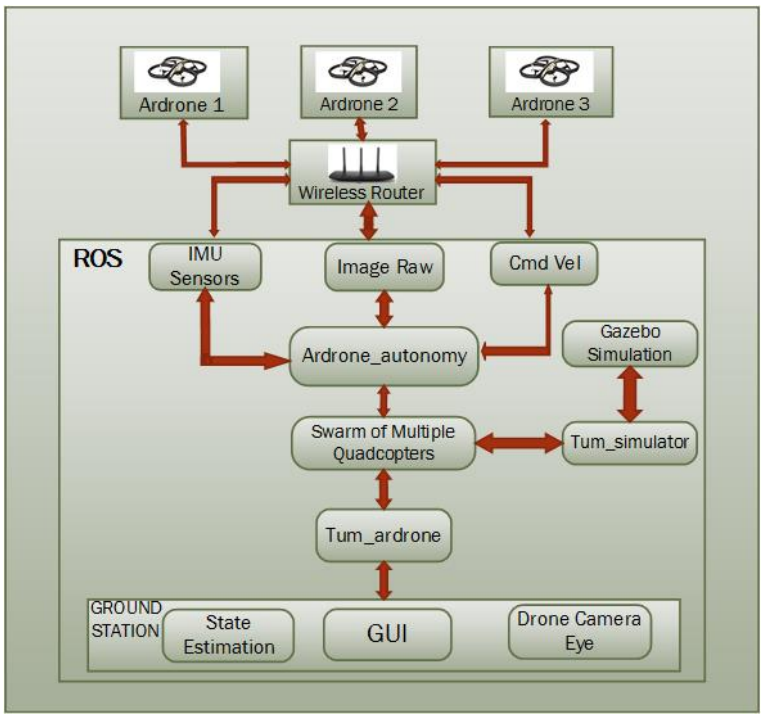

Figure 4.6: Block Diagram of Swarm

Tum_ardrone package provide us GUI (Graphical User Interface), State Estimation and Drone camera Eye. Using GUI we can send commands and behavior to ardrones and also see the battery percentage and commands which are executing. These commands send back to ardrone_autonomy pacakge as well as to tum_simulator pacakge to see the behaviour of ardrones in gazebo simulator.

4.10 Implementation

First of all we create a workspace where we compile our codes. To create a catkin workspace we need to execute the following commands on terminal:

$ $mkdir - p\ /catkin\_ws/src$

$ $cd\ /catkin\_ws/src$

$ $catkin\_init\_workspace$

$ $catkin\_make$

Then we download the tum_simulator package in catkin_ws to create a environment of

ardrone in gazebo simulator. For this we execute the following commands on terminal:

$ *cd /catkin_ws/src*

$ *sudo git clone https : //github.com/occomco/tum_simulator.git*

$ *cd..*

$ *catkin_make*

$ *source devel/setup.bash*

Now run a simulator by executing a launch file in cvg_sim_gazebo package by following comands:

$ *cd /catkin_ws*

$ *roslaunch cvg_sim_gazebo ardrone_testworld.launch*

After executing the commands gazebo simulator will open on a screen in which ardrone world have been created as shown in Fig4.7.

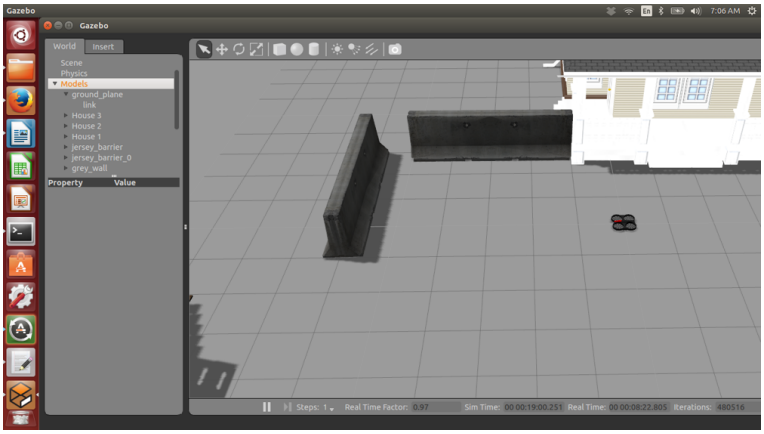

Figure 4.7: Gazebo Simulator for ardrone

4.10.1 Manual Flight

To fly a ardrone using keyboard we write a following code and make it executable to run like a node.

Create a file and write the following python code:

```
#!/usr/bin/env python
from __future__ import print_function
import rospy
```

```
from geometry_msgs import msg
from std_msgs.msg import Empty, Float64
from ardrone_autonomy.msg import Navdata
import actionlib
import sys
doc_msg = """
Control Your AR Drone!
---------------------------
Moving around:
        i
   j    k    l

   y - up
   h - down
   m - hover
   u - counterclockwise
   o - clockwise
t : takeoff
g : land

CTRL-C to quit
"""
moveBindings = {
    # forward and back
    'i': (1.0, 0.0, 0.0, 0.0, 0.0, 0.0),
    'k': (-1.0, 0.0, 0.0, 0.0, 0.0, 0.0),
    # left and right
    'j': (0.0, 1.0, 0.0, 0.0, 0.0, 0.0),
    'l': (0.0, -1.0, 0.0, 0.0, 0.0, 0.0),
```

```
    # up and down
    'y': (0.0, 0.0, 1.0, 0.0, 0.0, 0.0),
    'h': (0.0, 0.0, -1.0, 0.0, 0.0, 0.0),
    # counterclockwise and clockwise
    'u': (0.0, 0.0, 0.0, 0.0, 0.0, 1.0),
    'o': (0.0, 0.0, 0.0, 0.0, 0.0, -1.0),
    # stop
    'm': ((0.0, 0.0, 0.0, 0.0, 0.0, 0.0))}
def getKey():
    """Get key from keyboard.
    Returns:
        string: The key pressed
    """
    tty.setraw(sys.stdin.fileno())
    rlist, _, _ = select.select([sys.stdin], [], [], 0.1)
    if rlist:
        key = sys.stdin.read(1)
    else:
        key = ''
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key
def check_battery(data):
    """Check the battery and lands if low."""
    if data.batteryPercent < 15:
        land_pub.publish()
        twist = msg.Twist()
        import time
        while True:
            state['last_time'] = time.time()
```

```
            key = getKey()
            if key in moveBindings.keys():
                xyz = moveBindings[key]
            elif key == 't':
                # take off
                take_off_pub.publish()
            elif key == 'g':
                # land
                land_pub.publish()
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
```

After writing a code save it with a name ardrone_teleop_key.py

Now to make it executable we need to do the following steps:

Place the code file "ardrone_teleop_key.py" into the script folder in catkin_ws/src

Open the terminal and execute the following commands:

$ $cd\ /catkin\_ws/src/script$

$ $chmod\ +x\ ardrone\_teleop\_key.py$

To make sure your code is node type "ls" in terminal, if your code colour is green, then run code by executing the following commands:

Open a new terminal.

$ $cd\ /catkin\_ws$

$ $source\ devel/setup.bash$

$ $rosrun\ script\ ardrone\_teleop\_key.py$

Now we can fly the drone by keyboard in gazebo simulator.

To fly a single drone in real world we need to do following steps:

Open terminal and execute the following commands:

$ $cd\ /catkin\_ws$

$ $sudo\ git\ clone\ https://github.com/AutonomyLab/ardrone\_autonomy.git$

$ $cd..$

$ $catkin\_make$

$ *source devel/setup.bash*

Now connect the drone to laptop over wifi channel and execute the following commands:

$ *cd /catkin_ws*

$ *source devel/setup.bash*

$ *roslaunch ardrone_autonomy ardrone.launch*

$ *rosrun script ardrone_teleop_key.py*

Now we can fly ardrone in real world through keyboard.

4.10.2 Autonomous Flight

To make ardrone fly autonomously we need to write a script to fly our drone accordingly. Following is the code to fly a ardrone autonomously in python:

```
#!/usr/bin/env python
import rospy
import time
from geometry_msgs.msg import Twist
from std_msgs.msg import String
from std_msgs.msg import Empty
from ardrone_autonomy.msg import Navdata
from drone_status import DroneStatus
COMMAND_PERIOD = 1000
class AutonomousFlight():
    def __init__(self):
        self.status = ""
        rospy.init_node('forward', anonymous=False)
        self.rate = rospy.Rate(10)
        self.pubTakeoff = rospy.Publisher
("ardrone/takeoff",Empty, queue_size=10)
        self.pubLand = rospy.Publisher
("ardrone/land",Empty, queue_size=10)
```

```
            self.pubCommand = rospy.Publisher
('cmd_vel',Twist, queue_size=10)
            self.command = Twist()
            #self.commandTimer = rospy.Timer
(rospy.Duration(COMMAND_PERIOD/1000.0),self.SendCommand)
            self.state_change_time = rospy.Time.now()
            rospy.on_shutdown(self.SendLand)
        def SendTakeOff(self):
            self.pubTakeoff.publish(Empty())
            self.rate.sleep()
        def SendLand(self):
            self.pubLand.publish(Empty())
        def SetCommand
            (self, linear_x, linear_y, linear_z, angular_x,
             angular_y, angular_z):
            self.command.linear.x = linear_x
            self.command.linear.y = linear_y
            self.command.linear.z = linear_z
            self.command.angular.x = angular_x
            self.command.angular.y = angular_y
            self.command.angular.z = angular_z
            self.pubCommand.publish(self.command)
            self.rate.sleep()
if __name__ == '__main__':
        try:
            i = 0
            uav = AutonomousFlight()
            while not rospy.is_shutdown():
                uav.SendTakeOff()
```

```
            if i <= 30 :
                uav.SetCommand(0,0,1,0,0,0)
                i+=1
            elif i<=60 :
                uav.SetCommand(1,0,0,0,0,0)
                i+=1
            else:
                uav.SetCommand(0,0,0,0,0,0)
    except rospy.ROSInterruptException:
        pass
```

Save the code in a file named forward.py and repeat the steps as described in the manual Flight section to make a code to run as a node. We can see the behavior of this code in gazebo simulator as well as in real world by repeat the same steps as described in section manual Flight.

4.11 Multiple Drones

To make a swarm of three drones which fly together through one ground station we need to modify the tum_simulator for multiple ardrones, so that we can see the behavior of three ardrones in gazebo simulator for this you need to do following steps:
Navigate to tum_simulator package which was already compiled in a catkin_workspace. Change the Eleven files inside the tum_simulator package to allow multiple ardrones in gazebo simulator. The path of the eleven code files are given bellow.

1. cvg_sim_gazebo/launch/testworld_with_three_ardrone.launch
2. tum_simulator/cvg_sim_gazebo/urdf/sensors/generic_camera.urdf.xacro
3. tum_simulator/cvg_sim_gazebo_plugins/include/hector_quadrotor_controller/quadrotor_simple_controller.h
4. tum_simulator/cvg_sim_gazebo_plugins/src/the gazebo_ros_gps.cpp

5. tum_simulator/cvg_sim_gazebo_plugins/src/gazebo_ros_imu.cpp
6. tum_simulator/cvg_sim_gazebo_plugins/src/gazebo_ros_magnetic.cpp
7. tum_simulator/cvg_sim_gazebo_plugins/src/quadrotor_simple_controller.cpp
8. tum_simulator/cvg_sim_gazebo_plugins/src/quadrotor_state_controller.cpp
9. tum_simulator/cvg_sim_gazebo_plugins/urdf/quadrotor_sensors.urdf.xacro
10. tum_simulator/message_to_tf/src/message_to_tf.cpp
11. tum_simulator/cvg_sim_gazebo_plugins/include/hector_quadrotor_controller/quadrotor_state_controller.h

After successfully changed the files with new code then open a terminal and execute the following commands to compile the codes.

$ $cd\ /catkin\_ws$

$ $catkin\_make$

$ $source\ devel/setup.bash$

To launch a gazebo with multiple ardrones we need to execute the following command in terminal:

$ $roslaunchcvg\_sim\_gazebotestworld\_with\_three\_ardrone.launch$

After executing the command gazebo simulator will open and we see the world with three ardrones as shown in Fig4.8. Now to make them fly we need to download the tum_ardrone package and modify it to fly it in a gazebo simulator.

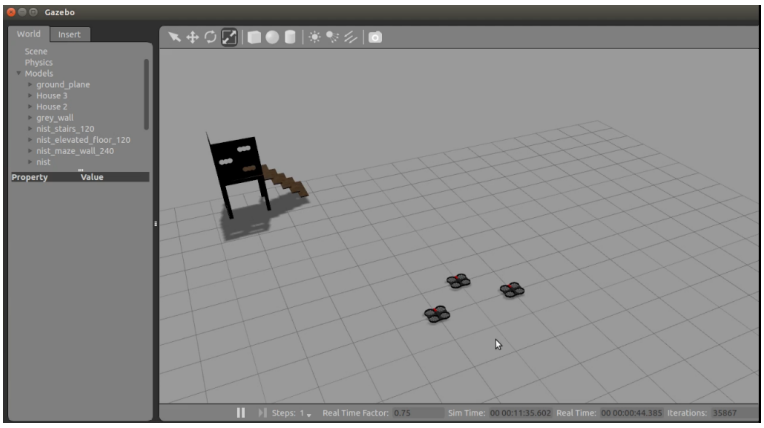

Figure 4.8: Gazebo Simulator with multiple ardrone

To make this happen you need to execute the following commands:

$ $cd\ /catkin\_ws$

$ $git\ clone\ https://github.com/tum-vision/tum\_ardrone$

$ $cd\ ..$

$ $catkin\_make$

Now we navigate to following path and modify the code for multiple ardrone:

1. tum_ardrone/src/UINode/RosThread.cpp
2. tum_ardrone/src/UINode/RosThread.h
3. tum_ardrone/src/stateestimation/EstimationNode.cpp
4. tum_ardrone/src/stateestimation/EstimationNode.h
5. tum_ardrone/src/autopilot/ControlNode.cpp
6. tum_ardrone/src/autopilot/ControlNode.h

After successfully change the code proceed towards compilation. Open a terminal and execute the following command to run tum_ardrone package:

$ $roslaunchtum\_ardronetum\_ardrone.launch$

Now three windows open in which one is graphical user interface window, second is stateestimation window and third is carmea window.

From GUI we can control multiple ardrones and see the behavior on the gazebo simulator.

4.12 Real TIme Implementation of Swarm

To Implement swarm in real time we need to connect three ardrone to wifi router and ground station will be connected with router.
To connect ardrone to router we need to perform folllowing steps for each ardrone in which we assign a new ip to ardrone and give then SSID of the router to connect with it.

1. Open the terminal.
2. Connect the laptop to drone as we do in mobile app.
3. Run the following command in terminal step by step.

   $ $telnet$192.168.1.1

   $ $vi\ /data/wifi.sh$

   A screen will open in vi mode
4. press i and paste the following code

   killall udhcpd

   ifconfig ath0 down

   iwconfig ath0 mode managed essid dronenet

   ifconfig ath0 192.168.1.11 netmask 255.255.255.0 up
5. In line 3 change ”dronenet” to the SSID you chose when

   setting up the router.
6. The IP in line 4 should be unique to each drone, so change it

   between different drones.

7. To save and exit vi press ”esc” to exit insert mode and then ”:wq”
8. write following commands to make it executable.

   $ $chmod\ +x\ /data/wifi.sh$

   $ $exit$
9. While the previous sections should only be done once, this step should be done whenever you want to make the AR-Drone connect to your wireless (for example every time you turn on the drone).
10. Open a terminal again.
11. Execute the following command to remotly run wifi.sh on the drone:

    $ $echo$ "$./data/wifi.sh$" | $telnet$192.168.1.1
12. Your computer should now disconnect from the drones network. This is due to the drone’s network cofigurations being changed.
13. check that you have connection to the drone using ping:

    $ $ping$192.168.1.10
14. Where the IP address is the same as the one configured in wifi.sh of this drone

Now to connect three ardrones with ROS we need to make a launch file for multiple ardrone_autonomy.
Navigate to ardrone_autonomy/launch and make a launch file with name ardrone_group.launch and write the following code:

```
<?xml version="1.0"?>
<!--Launch file for multiple AR Drones -->
<launch>
  <group ns="ardrone1">
    <param name="tf_prefix" value="ardrone1" />
```

```
    <include file="$(find
ardrone_autonomy)/launch/ardrone.launch">
      <arg name="ip" value="192.168.1.10" />
      <arg name="freq" value="7" />
    </include>
  </group>

  <!--Second drone-->
  <group ns="ardrone2">
    <param name="tf_prefix" value="ardrone2" />
    <include file="$(find
ardrone_autonomy)/launch/ardrone.launch">
      <arg name="ip" value="192.168.1.11" />
      <arg name="freq" value="8" />
    </include>
  </group>

        <!--Third drone-->
  <group ns="ardrone3">
    <param name="tf_prefix" value="ardrone3" />
    <include file="$(find
ardrone_autonomy)/launch/ardrone.launch">
      <arg name="ip" value="192.168.1.12" />
      <arg name="freq" value="10" />
    </include>
  </group>
</launch>
```

Now we are able to fly three ardrones and make a swarm of them. To make them fly autonomously we need to write a script and make trajectory for them. Following is a

code to fly them on a planned trajectory.

```
#include <ros/ros.h>
#include <std_msgs/Empty.h>
#include <geometry_msgs/Twist.h>
        geometry_msgs::Twist twist_msg;
        geometry_msgs::Twist twist_msg_rot;
        geometry_msgs::Twist twist_msg_left;
        geometry_msgs::Twist twist_msg_right;
        geometry_msgs::Twist twist_msg_neg;
        geometry_msgs::Twist twist_msg_hover;
        geometry_msgs::Twist twist_msg_up;
        std_msgs::Empty emp_msg;
int main(int argc, char** argv)
{       ROS_INFO("Flying␣ARdrone");
        ros::init(argc, argv,"ARDrone_test");
      ros::NodeHandle node;
      ros::Rate loop_rate(50);
        ros::Publisher pub_empty1;
        ros::Publisher pub_empty2;
        ros::Publisher pub_empty3;
        pub_empty1 = node.advertise<std_msgs::Empty>
("/ardrone1/takeoff", 1);
        pub_empty2 = node.advertise<std_msgs::Empty>
("/ardrone2/takeoff", 1);
        pub_empty3 = node.advertise<std_msgs::Empty>
("/ardrone3/takeoff", 1);
while (ros::ok())
{double time_start=(double)ros::Time::now().toSec();
while ((double)ros::Time::now().toSec()< time_start+5.0)
```

```
            pub_empty1.publish(emp_msg);
            pub_empty2.publish(emp_msg);
            pub_empty3.publish(emp_msg);
            ros::spinOnce();
            loop_rate.sleep();}
ROS_INFO("ARdrone is launched");
break;}
ROS_INFO("ARdrone Test Back and Forth Starting");
            ros::init(argc, argv,"ARDrone_test");
            ros::Publisher pub_empty_land_1;
            ros::Publisher pub_empty_land_2;
            ros::Publisher pub_empty_land_3;
            ros::Publisher pub_twist_1;
            ros::Publisher pub_twist_2;
            ros::Publisher pub_twist_3;
            ros::Publisher pub_empty_takeoff_1;
            ros::Publisher pub_empty_takeoff_2;
            ros::Publisher pub_empty_takeoff_3;
            ros::Publisher pub_empty_reset_1;
            ros::Publisher pub_empty_reset_2;
            ros::Publisher pub_empty_reset_3;
            double start_time;
            twist_msg_hover.linear.x=0.0;
            twist_msg_hover.linear.y=0.0;
            twist_msg_hover.linear.z=0.0;
            twist_msg_hover.angular.x=0.0;
            twist_msg_hover.angular.y=0.0;
            twist_msg_hover.angular.z=0.0;
//rotate message
```

```
        twist_msg_rot.linear.x=0.0;
        twist_msg_rot.linear.y=0.0;
        twist_msg_rot.linear.z=0.0;
        twist_msg_rot.angular.x=0.0;
        twist_msg_rot.angular.y=0.0;
        twist_msg_rot.angular.z=0.7;
//up message
        twist_msg_up.linear.x=0.0;
        twist_msg_up.linear.y=0.0;
        twist_msg_up.linear.z=0.0;
        twist_msg_up.angular.x=0.0;
        twist_msg_up.angular.y=0.0;
        twist_msg_up.angular.z=0.0;
//command message
        float takeoff_time=2.0;
        float fly_time=5.0;
        float land_time=2.0;
        float kill_time =1.0;
        float new_time=4.0;
//forward
        twist_msg.linear.x=1.0;
        twist_msg.linear.y=0.0;
        twist_msg.linear.z=0.0;
        twist_msg.angular.x=0.0;
        twist_msg.angular.y=0.0;
        twist_msg.angular.z=0.0;
//Backword
        twist_msg_neg.linear.x=-twist_msg.linear.x;
        twist_msg_neg.linear.y=-twist_msg.linear.y;
```

```
        twist_msg_neg.linear.z=-twist_msg.linear.z;
        twist_msg_neg.angular.x=-twist_msg.angular.x;
        twist_msg_neg.angular.y=-twist_msg.angular.y;
        twist_msg_neg.angular.z=-twist_msg.angular.z;
//left
        twist_msg_left.linear.x=0.0;
        twist_msg_left.linear.y=1.0;
        twist_msg_left.linear.z=0.0;
        twist_msg_left.angular.x=0.0;
        twist_msg_left.angular.y=0.0;
        twist_msg_left.angular.z=0.0;
//Right
        twist_msg_right.linear.x=0.0;
        twist_msg_right.linear.y=-1.0;
        twist_msg_right.linear.z=0.0;
        twist_msg_right.angular.x=0.0;
        twist_msg_right.angular.y=0.0;
        twist_msg_right.angular.z=0.0;
    pub_twist_1 = node.advertise<geometry_msgs::Twist
("/ardrone_1/cmd_vel", 1);
        pub_empty_takeoff_1 = node.advertise<std_msgs::Empty
("/ardrone_1/takeoff", 1);
        pub_empty_land_1 = node.advertise<std_msgs::Empty
("/ardrone_1/land", 1);
        pub_empty_reset_1 = node.advertise<std_msgs::Empty
("/ardrone_1/reset", 1);
        pub_twist_2 = node.advertise<geometry_msgs::Twist
("/ardrone_2/cmd_vel", 1);
        pub_empty_takeoff_2 = node.advertise<std_msgs::Empty
```

```
("/ardrone_2/takeoff", 1);
        pub_empty_land_2 = node.advertise<std_msgs::Empty
("/ardrone_2/land", 1);
        pub_empty_reset_2 = node.advertise<std_msgs::Empty
("/ardrone_2/reset", 1);
        pub_twist_3 = node.advertise<geometry_msgs::Twist
("/ardrone_3/cmd_vel", 1);
        pub_empty_takeoff_3 = node.advertise<std_msgs::Empty
("/ardrone_3/takeoff", 1);
        pub_empty_land_3 = node.advertise<std_msgs::Empty
("/ardrone_3/land", 1);
        pub_empty_reset_3 = node.advertise<std_msgs::Empty
("/ardrone_3/reset", 1);
        start_time =(double)ros::Time::now().toSec();
        ROS_INFO("Starting_loop");
while (ros::ok()) {
        while ((double)ros::Time::now().toSec()
< start_time+takeoff_time){
        pub_empty_takeoff_1.publish(emp_msg);
        pub_empty_takeoff_2.publish(emp_msg);
        pub_empty_takeoff_3.publish(emp_msg);
        pub_twist_1.publish(twist_msg_hover);
        pub_twist_2.publish(twist_msg_hover);
        pub_twist_3.publish(twist_msg_hover);
                ROS_INFO("Taking␣off");
                ros::spinOnce();
                loop_rate.sleep();}
while  ((double)ros::Time::now().toSec()>
start_time+takeoff_time+fly_time+new_time){
```

```
            pub_twist_1.publish(twist_msg_hover);
            pub_twist_2.publish(twist_msg_hover);
            pub_twist_3.publish(twist_msg_hover);
            pub_empty_land_1.publish(emp_msg);
            pub_empty_land_2.publish(emp_msg);
            pub_empty_land_3.publish(emp_msg);
            ROS_INFO("Landing");
if ((double)ros::Time::now().toSec()>
takeoff_time+start_time+fly_time+new_time+land_time+kill_time)
{           ROS_INFO("Closing Node");
            exit(0);            }
            ros::spinOnce();
            loop_rate.sleep();}
            while ( (double)ros::Time::now().toSec()>
start_time+takeoff_time && (double)ros::Time::now().toSec()
< start_time+takeoff_time+fly_time){
            if((double)ros::Time::now().toSec()
< start_time+takeoff_time+fly_time/2){
            pub_twist_1.publish(twist_msg);
            pub_twist_2.publish(twist_msg);
            pub_twist_3.publish(twist_msg);
            ROS_INFO("forward");}
if((double)ros::Time::now().toSec()>
start_time+takeoff_time+fly_time/2){
            pub_twist_1.publish(twist_msg_rot);
            pub_twist_2.publish(twist_msg_rot);
            pub_twist_3.publish(twist_msg_rot);
            ROS_INFO("left rotate");}
                        ros::spinOnce();
```

```
                loop_rate.sleep();}
        while ((double)ros::Time::now().toSec()>
start_time+takeoff_time && (double)ros::Time::now().toSec()
< start_time+takeoff_time+fly_time+new_time){
        if((double)ros::Time::now().toSec()
< start_time+takeoff_time+fly_time+new_time/2){
        pub_twist_1.publish(twist_msg);
        pub_twist_2.publish(twist_msg);
        pub_twist_3.publish(twist_msg);
        ROS_INFO("forward");}
        if((double)ros::Time::now().toSec()>
start_time+takeoff_time+fly_time+new_time/2){
        pub_twist_1.publish(twist_msg_rot);
        pub_twist_2.publish(twist_msg_rot);
        pub_twist_3.publish(twist_msg_rot);
        ROS_INFO("left␣Rotate");}
        ros::spinOnce();
        loop_rate.sleep();}
        ros::spinOnce();
        loop_rate.sleep();}}
```

4.13 Experiment Results

We conduct a experiment with three ardrones to moves autonomously in a same direction as shown in Fig4.9 ardrones are in land state, then they take off autonmously and make a behaviour accoding to the written script as shown in Fig4.10 and Fig4.11.

Figure 4.9: Ardrone in land state

Figure 4.10: Swarm of ardrone

Figure 4.11: Multiple Ardrone moving in same direction

Chapter 5

Object Detection and Tracking

5.1 Introduction

There are multiple tasks and ideas that can be implemented to drones as they are becoming more and more popular these days. One of the idea is described in our work. We created a software in C++ in ROS that control the parrot drone movement according to the movement of target object. As it is presented on the diagram below, we need: Ardrone Autonomy Package (drone driver) and cv_bridge in OpenCV library, which converts ros image messages to OpenCV objects. The objects can be modified in OpenCV library.

The main purpose was to write a program that converts ros image messages taken from the camera into openCV images, analyzes them and depending on the movement of the target object the drone movement. The target was achieved and the drone is being controlled by the object in front of the camera.

. Vision based control strategy was presented for tracking and following objects using an Unmanned Aerial Vehicle. We have developed an image based visual servoing method that uses only a forward looking camera for tracking and following objects from a multi-rotor UAV, without any dependence on GPS systems. Our proposed method tracks a user specified object continuously while maintaining a fixed distance from the object and also simultaneously keeping it in the center of the image plane. The algorithm is validated using a Parrot AR Drone 2.0 in outdoor conditions while tracking and following people, occlusions and also fast moving objects; showing the robustness of the proposed systems against perturbations and illumination changes. Our experiments show that the system is able to track a great variety of objects present in suburban areas, among others: people, windows, AC machines, cars and plants.

5.2 Computer Vision

As a scientific discipline, computer vision is concerned with the theory and technology for building artificial systems that obtain information from images or multi-dimensional data. Computer vision is a vast field attempting to tackle several difficult problems. It basically tries to describe the world from images and reconstruct its properties, such as shape, illumination, and color distributions. It seeks to automate tasks that the human visual system can do. ”Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images Despite the fact that humans are able to reconstruct the world from images almost effortlessly, it is a very challenging task for computers. It has wide ranging applications such as machine inspection, medical imaging, surveillance, 3D modelling or object recognition and tracking, just to name a few. The application mentioned last – object recognition and tracking – will be studied in the following sections and finally applied in this report.

5.3 OpenCV

OpenCV which stands for open source computer vision is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license. As this this library provides different algorithms for object detect and recognition. We have selected TLD tracking algorithm to achieved Object detection and tracking of the target object.

5.4 TLD Tracking

TLD stands for Tracking, learning and detection. This tracker decomposes the long term tracking task into three components — (short term) tracking, learning, and detection. From the author’s paper, “The tracker follows the object from frame to frame. The

detector localizes all appearances that have been observed so far and corrects the tracker if necessary. The learning estimates detector's errors and updates it to avoid these errors in the future." This output of this tracker tends to jump around a bit. For example, if you are tracking a pedestrian and there are other pedestrians in the scene, this tracker can sometimes temporarily track a different pedestrian than the one you intended to track. On the positive side, this track appears to track an object over a larger scale, motion, and occlusion. If you have a video sequence where the object is hidden behind another object, this tracker may be a good choice.

5.5 Working

As we are using ardrone autonomy package as a driver between ROS and our drone. Using this package we get raw images from the front camera of the drone.

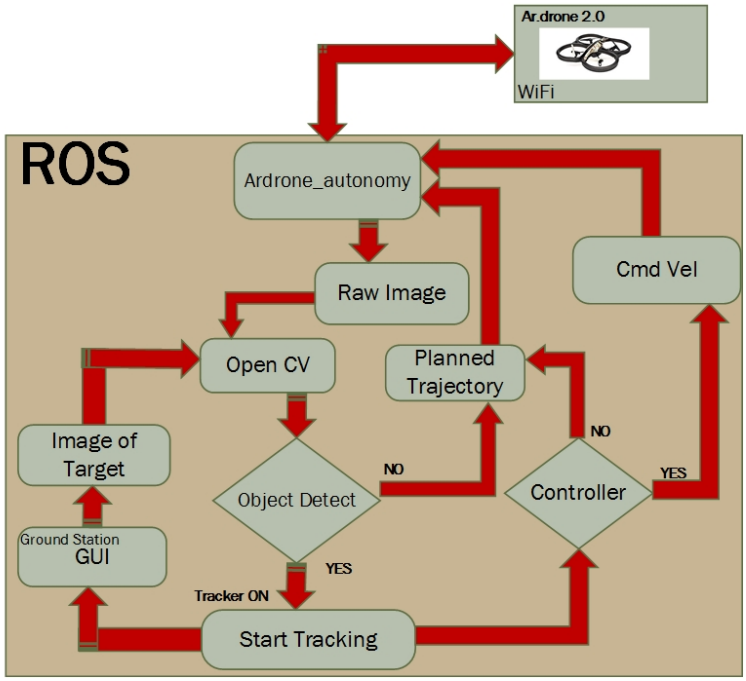

Figure 5.1: Working Algorithm of Object detection and tracking

By using the raw images we get from front camera we can see live streaming on our ground station. For object detection we have used openCV library in which we have used TLD algorithm. TLD comes with graphical user interface in which we can select

the target object by creating a bounding box across it. After saving the target object it detects the target image when it comes in a frame of front camera. We Combined the controller of ardrone with TLD algorithm, which send commands according to the object movement which results in tracking it. So when the controller is on, it starts moving in a direction of object by maintaining a fixed distance from the object. The working of this algorithm is described in a fig5.1

5.6 Implementation

Firstly, we need a external libraries to be installed. They was installed by runnig the following commands on the terminal:

$ $sudo\ apt-get\ install\ libncurses5$

$ $sudo\ apt-get\ install\ ncurses-bin$

$ $sudo\ apt-ge\ install\ ncurses-dev$

After successfully installing the external libraries we need to deinstalled the environment behaviours of the old version of the stack. This was done by executing the following commands in the terminal:

$ $sed\ -i'/IBVS\_WORKSPACE/d'\ /.bashrc$

$ $sed\ -i'/IBVS\_STACK/d'\ /.bashrc$

Now we have to create a new workspace for Object detection and tracking. This was achieved by implementing the following commands:

$ $cd\ /workspace/ros$

Now inside workspace we created a folder where our all files are complied and executed.

$ $mkdir\ –p\ /workspace/ros/cvg\_ardrone2\_ibvs$

$ $cd\ ..$

Then we initialized our workspace by executing the folllowing comand:

$ $source\ /opt/ros/indigo/setup.bash$

$ $mkdir\ src$

$ $cd\ ..$

$ $catkin\_init\_workspace$

$ *cd ..*

$ *catkin_make*

$ *source devel/setup.bash*

Now download the package for Object detection and tracking by executing the following commands:

```
$ git clone –b master https://github.com/Vision4UAV/cvg_ardrone2_ibvs.git ./src/quadrotor_stack
```

Install the workspace now:

$ *./src/stack/installation/installers/installWS.sh*

$ *./installation/installers/installStack.sh*

$ *source /.bashrc*

$ *cd IBVS_STACK*

$ *git submodule update – –init*

$ *cd IBVS_STACK*

$ *source setup.sh*

$ *cd IBVS_WORKSPACE*

$ *catkin_make*

$ *cd IBVS_WORKSPACE/src*

$ *rm CMakeLists.txt*

$ *cp /opt/ros/indigo/share/catkin/cmake/toplevel.cmake CMakeLists.txt*

$ *cd ..*

$ *source devel/setup.bash*

5.7 Camera Calibaration

By subscribing the raw images from AR drone using Ardrone_autonomy package we implemented image processing on the front camera of our drone. For this purpose we need to calibrate the front camera of the drone. This was done by implementing the following steps:

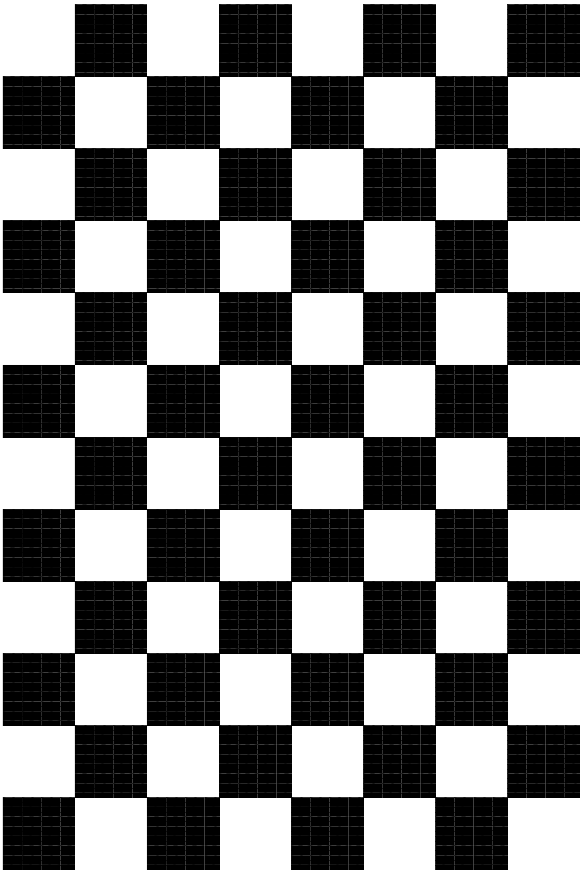

Figure 5.2: Calibration Pattern

First, we need a image pattern and then measure the size of the sides of the squares on it. We also need to count the number of inner corners of the check board pattern (the corners in the border which are shared by only 1 squares are not to be counted). We take the fig5.2.It comes from the PTAM library (it is the pattern to be used if you want to calibrate your camera to be used with PTAM; which uses a really specific camera model).Then, we used the camera calibration package that is already in ROS, because the GUI it has is really user friendly. To do so follow these steps:

Run the sh script on terminal

```
$ {IMAV_2013_Repository}/launch_dir/ardrone_camera_calibration_
frontcam_launcher.sh
```

```
$ {IMAV_2013_Repository}/launch_dir/ardrone_camera_calibration_
bottomcam_launcher.sh
```

Then click commit so that the ROS camera calibration file is saved on ~/.ros/camera_info

There will be two files generated with named ardrone_bottom.yaml and ardrone_front.yaml.

Ardrone_bottom.yaml:

```
image_width: 640
image_height: 360
camera_name: ardrone_bottom
camera_matrix:
  rows: 3
  cols: 3
  data: [667.35847908272, 0, 330.449976314247, 0,
668.587872507243, 179.014729860623, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.00951101387422823, 0.0172421866821302,
0.00677571945374994, 0.00135614726225701, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
```

```
  data: [667.782287597656, 0, 330.612895204358, 0, 0,
668.265930175781, 180.031499195671, 0, 0, 0, 1, 0]
```

Ardrone_front.yaml

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [560.015315069345, 0, 316.816016259763, 0,
556.390028885253, 192.220533455274, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.498374164117516,
0.249276393718262, -0.00264348985883458, 0.00156516603140277, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [456.138549804688, 0, 316.764636943721, 0, 0,
524.119567871094, 192.59496666532, 0, 0, 0, 1, 0]
```

Save these two files in a folder under .ros/camera_info.

5.8 Launch Script

In order to execute the stack, we have decided to run each node in a separate tab of a terminal window. It is done by writing a shell scripts that will execute on a new terminal in which each node running in its tab. Multiple tabs open with a seperate terminal in which each launch file executes, where each tab usually runs only one tab. Only one tab will close properly if we close a terminal with multiple tabs (the one that is currently selected).

The easiest way to do this fast, and cleanly is to first, press ‘control+c‘ on every tab (navigating with ‘control+repag‘ and ‘control+avpag‘), second, use the shortcut ‘ctrl+shift+w‘ to close first all terminal tabs and, third, ‘ctrl+shift+q‘ to close the last terminal tab (which closes the window too) including all tabs.

The launch scripts have to be called using the following sintax in the shell terminal:

$ $cd\ IBVS\_STACK/launchers$

$ $./parrot\_IBVSController\_launcher\_Release.sh$

5.9 Interaction with UI

Simplified instructions to work with the controller:

1. Open Window ”OpenTLD GUI”:
2. It is important to get familiarize with this window. It allows us to start/stop the tracker, toggle learning, change the tracker’s target.
3. If we want to start the tracker, press F5 to update the image shown on the OpenTLD GUI, you have to do this sometimes because no image is shown on it, or because you want to track something that is not on the currently shown image.
4. Select the target by drawing a bounding box around it in the image.
5. press ”Enter”

6. To change the tracker's target, click "reset" once, redo the "start the tracker" instructions.
7. To toggle learning, You have to be tracking an object. Then click the toggle tracking button.
8. This will change from "learning and tracking mode" to "tracking only mode".
9. To know exactly whether the tracker is currently learning or not; then go to the terminal tab which is running the OpenTLD nodes. This information is printed there.
10. To interact with the rest of the software the "IBVSCntInterf" terminal tab is used.
11. There is a switch case instruction which acts upon pressed keys.

The important keys are:

```
't': take off
'y': land
'h': enter hovering mode
'm': enter move mode
```

In move mode:

```
arrow\_left: move/tilt left
arrow\_right: move/tilt right
arrow\_up: move/tilt forwards
arrow\_down: move/tilt backwards
'q': move upwards
'a': move downwards
'z': yaw left/counterclockwise
'x': yaw right/clockwise
```

```
's': stop / send "zero" commands in every command.
```

Start the controller:

```
'o': start the controller
'i': stop the controller
'u': reset controller
```

Start the kalman filter (only used for data logging):

```
'l': start the kalman filter
'k': stop the kalman filter
'j': reset kalman filter, reinitialize estiamte to (0,0,0)
```

5.10 Experiment Results

We conduct a experiment with one ardrone in which first we freeze the picture in TLD interface as shown in Fig5.3 , then we select the object in the picture by drawing a bounding box and press enter to save the object and start starking and controlling mode as shown in Fig5.4. After that we send commands to the ardrone to take off and start detecting and tracking the object as shown in the Fig5.5 and Fig5.6.

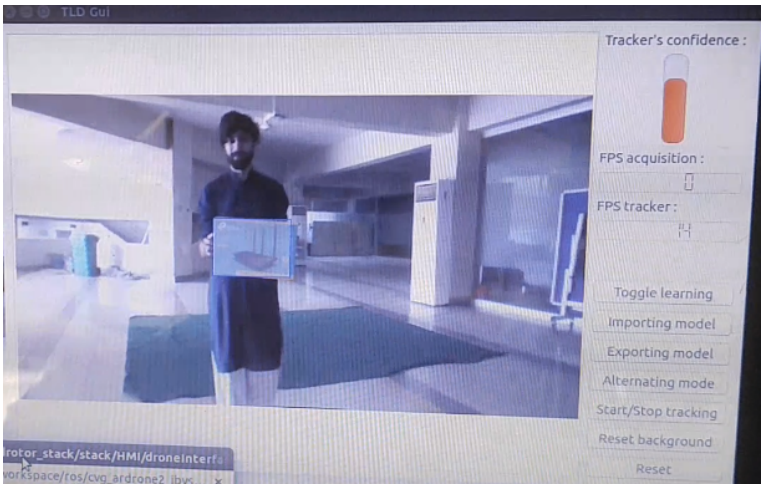

Figure 5.3: Freeze image in GUI

Figure 5.4: Tracking and Controlling mode

Figure 5.5: Ardrone tracking the object

Figure 5.6: Ardrone moving in the direction of object

Chapter 6

AUGUMENTED REALITY AND ITS IMPLEMENTATIONS

In this chapter we are going to discuss:-

- Augumented Reality.
- Introduction to Aruco.
- Aruco Markers.
- Package Localization.
- Path Planning.
- Hurdle Avoidance.
- Working.
- Aruco Block Diagram.
- Steps for Aruco execution.
- Proposed algorithm.
- Practical Results.

In this chapter we will focus on ArUco [21] , that how we used Aruco Markers [21] in our project for Pagkage Localization, Path Planning and Hurdle Avoidance for a warehouse environment using drones.

6.1 Augmented Reality(AR)

Augmented reality (AR) [28] is a type of interactive that takes the capabilities of computer generated display, sound, text and effects to enhance the user's real-world experience. The objects that reside in the real world are "Augmented" by computer generated perceptual information. The value of augmented reality (AR) is that it brings the component of digital world into person's perception of real world.

Augmented reality [28] is also transforming, where content may be accessed by scanning or viewing an image with a mobile device or by bringing immersive, marker less augmented reality experiences. Augmented reality adds digital images and data to amplify views of the real world, giving users more information about their environments. There are various uses of augmented reality like training, work and consumer applications in various industries including public safety, healthcare, tourism, gas and oil, and marketing. Here the application of Augmented reality that we used in our project is Aruco [21]. Fig6.1 shows the example of Augmented Reality using drones camera.

Figure 6.1: Augmented Reality

6.2 Introduction to Aruco

ArUco [21]is a minimal library for Augmented Reality [28] applications based on OpenCV. It's an open source library used for the detection of these markers. We can estimate the

position of camera with respect to the markers, if camera is calibrated. The library is written in C++, but the library can be used without programming with their tools. There are several markers, each belonging to the same dictionary and every marker is different from the other to avoid confusion. Aruco markers are easy to detect and they yields up to 1024 different patterns. So just we need is to download the dictionary, print the markers on in a piece of paper.

Each marker is a vector of 4 points (representing the corners in the image), a unique id, its size (in meters), and the translation and rotation that relates the center of the marker and the camera location. The Marker detector will look for squares and will analyze the binary code inside. For the code extracted, it will compare the particular marker against all the markers in the available dictionaries. Fig6.2 shows some examples of Aruco Markers.

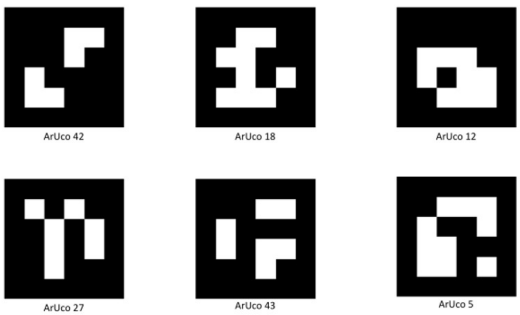

Figure 6.2: Aruco Markers

6.3 Aruco Markers

An Aruco Marker [21] is a square marker composed of external black border region and inner binary region which encodes binary pattern. The black border is responsible of fast detection in the image and the binary region allows the identification of marker. The marker size determines the internal matrix. For example the 4*4 marker size is composed of 16 bits and that of 5*5 composed of 32 bits. The more bits mean the more words in the dictionary as well as smaller chance of confusion. However, more bits means that more resolution is required for correct detection. Fig6.3 shows an example

of Aruco Marker.

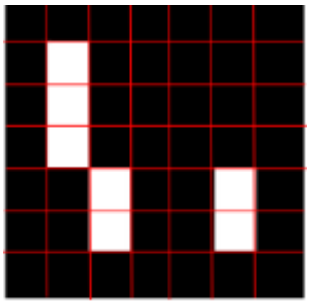

Figure 6.3: Aruco Marker

6.4 Package Localization

Package localization can be done by camera pose estimation, if the camera is calibrated. For package localization in a warehouse environment we have placed Aruco markers [21] on every package. The quad copters will go in search of the packages autonomously using its front and bottom camera. The marker detection is comprised in two main steps:

- In first step the image is analyzed in order to find square shapes that are candidates to be markers.
- Secondly it is necessary to determine if they are actually markers by analyzing their inner codification. This step starts by extracting the marker bits of each marker.

It will identify the marker by its Markers ID. As soon as the marker is detected the other thing is camera pose [30] . To perform camera pose estimation we need to know the calibration parameters of the camera of our drone. We calibrated the camera using Aruco module. When we estimate the pose with ArUco markers, we can estimate the pose of each marker individually.

When the marker is detected by Ardrone front or bottom camera, it will send the message to the ground station that the required package is found and by camera pose

estimation [30], it will send the coordinates of the marker with respect to the drone's camera to the ground station. Hence we can localize our required package with the help of these coordinates. Fig6.4 shows some Aruco Markers placed on the packages.

Figure 6.4: Aruco Packages

6.5 Path Planning

The path planner [25] generates deliberative motion references for the aerial robot given planning references and the information of the situation awareness. Planning references might include pose references, acceleration references and velocity references (among others) with or without time constraints. Path Planning generates heading motion references that must be followed by the aerial robot.

Path Planning using these Aruco markers can be done by placing these markers in the path of robot which the quad copter is going to follow. As quad copter detects the marker by its camera, firstly it will analyze the marker and secondly it will identify the Marker ID. Soon after identifying Marker ID, there is a command given to the quad copter against that Marker ID e.g. (Marker ID is 21 and command for quad copter against Marker ID.21 is to turn left). Hence the quad copter reacts accordingly against each Marker ID's which are placed in its path. Fig6.5 shows the ardones 2.0 planning its path through Aruco.

Figure 6.5: Path Planning

6.6 Hurdle Avoidance

Hurdle Avoidance can be done by using these Arucos [21]. The Aruco markers [21] are placed on every obstacle which can come across in their path. As any of the quad copter encounter with an obstacle it will detect the Aruco that is placed on the respective obstacle. After detecting the Aruco, it will send the command back to the quad copter against that Marker ID e.g. (If Aruco detects the Marker which is commanded to turn left, the quad copter reacts accordingly). In this way the hurdle can be avoided. Fig6.6 shows that drone is avoiding hurdle.

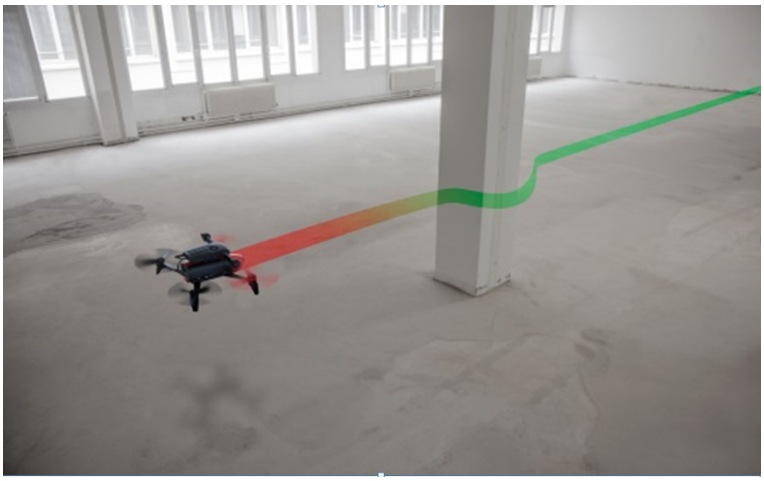

Figure 6.6: Hurdle Avoidance

6.7 Working

The working of the Augmented Reality is related to the warehouse environment. Like the three quad copters will go and search for the required package autonomously. The Aruco Markers are placed in every obstacle and the package. If any of the quad copter encounters with an obstacle, it will detect the Aruco Marker [30] placed on that obstacle. After detecting the Marker it will send the Aruco Marker ID to the ground station and with respect to that Marker a command is given to the quad copter to move left, right, rotate etc. The quad copter reacts accordingly with the given commands and can avoid hurdles.

If any of the quad copter encounters with the required package that we are looking for, it will similarly detect the Aruco Marker placed on that package. After detecting the Marker it will send the Aruco Marker ID to the ground station. In parallel to the Marker ID it will also send the camera position to the ground station in Translational and Rotational axis with reference to the position from where they takeoff. By estimating the position of ardrone's camera, we can simply estimate the location of our required package. Hence for the entire warehouse network we can easily localize our package and can avoid the hurdles which come in the path of quad copters.

6.8 Aruco Block Diagram

Fig6.7 shows the complete block diagram of Aruco that how the aruco process works.

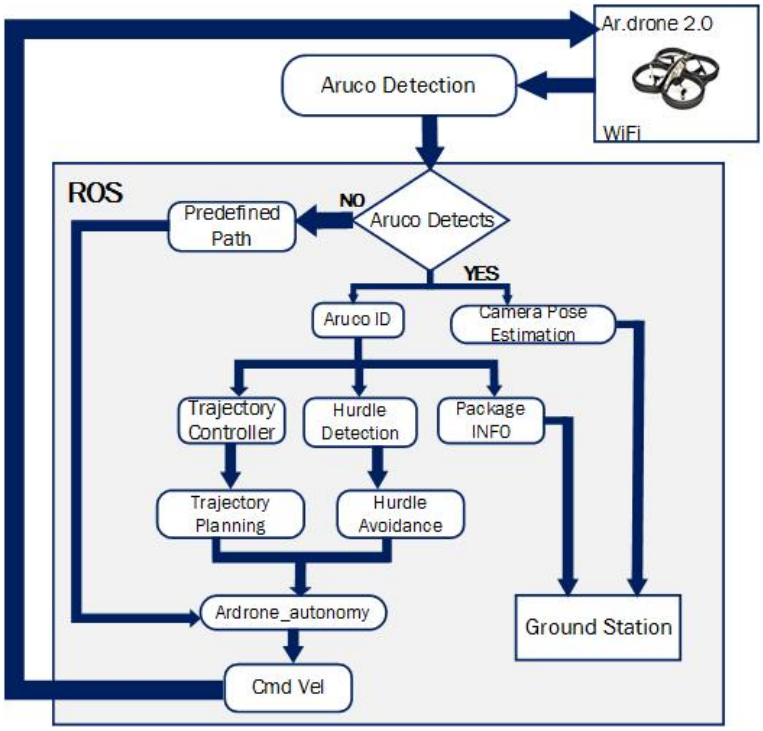

Figure 6.7: Aruco Block Diagram

6.9 Steps for Aruco Execution

6.9.1 Install Ros Indigo

“Before installing ROS and GAZEBO, You should install UBUNTU 14.0 in your machine”

1. Install Git

$ sudo apt-get install git

2. Install ROS you can follow the instruction how to install of ROS:

$ sudo apt-get update
$ sudo apt-get install ros-indigo-desktop-full
$ sudo rosdep init
$ rosdep update
$ echo ”source /opt/ros/indigo/setup.bash” » /.bashrc
$ source ˜/.bashrc
$ sudo apt-get install python-rosinstall

To check and verify write command

$ roscore

6.9.2 Install Gazebo

$ sudo apt-get install ros-indigo-simulators

6.9.3 Install TumSimulator

-> You can follow the instruction steps

$ mkdir -p ˜/tum_simulator_ws/src

$ cd ~/tum_simulator_ws/src

$ catkin_init_workspace

$ sudo git clone https://github.com/AutonomyLab/ardrone_autonomy.git

$ sudo git clone https://github.com/occomco/tum_simulator.git

$ cd ..

$ rosdep install –from-paths src –ignore-src –rosdistro indigo -y

$ catkin_make

$ source devel/setup.bash

6.9.4 Making a catkin workspace

$ source /opt/ros/indigo/setup.bash

$ mkdir -p ~/catkin_ws/src

$ cd ~/catkin_ws/

$ catkin_make

$ source devel/setup.bash

-> Run a simulation by executing a launch file in cvg_sim_gazebo package:

$ roslaunch cvg_sim_gazebo ardrone_testworld.launch

** catkin_build link (if catkin_make does not work, then follow this link to install dependencies for catkin_make)

$ sudo sh -c ’echo ”deb http://packages.ros.org/ros/ubuntu ‘lsb_release -sc‘ main” > /etc/apt/sources.list.d/ros-latest.list’

$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

$ sudo apt-get update

$ sudo apt-get install python-catkin-tools

6.9.5 Installing Aruco Package

$ cd ~/catkin_ws/src

$ git clone https://github.com/eYSIP-2017/eYSIP-2017_Navigation-in-Indoor-Environments-using-drone.git

-> " Installing pre-req packages for ArUco marker detection "

$ cd ~/catkin_ws/src

$ git clone https://github.com/simubhangu/pal_vision_segmentation -b hydro-devel

$ git clone https://github.com/simubhangu/marker_pose_detection

$ git clone https://github.com/pal-robotics/aruco_ros

6.9.6 Installing Tum_ardrone for controlling ardrone in Gazebo

$ cd ~/catkin_ws/src

$ git clone https://github.com/tum-vision/tum_ardrone.git

6.9.7 Necessary Changes to be made

-> Navigate manually to catkin_ws/src/aruco_ros/aruco/CMakelists.txt

-> Head over to line 6,col 21

-> Convert 'OpenCV 3 REQUIRED' to 'OpenCV 2 REQsUIRED'

-> Save the file

-> "Building all the packages "

-> Open a new terminal and enter the following commands to build all packages

$ cd ~/catkin_ws

$ catkin_make

-> Navigate manually to 'catkin/src/eYSIP-2017_Navigation-in-Indoor-Environments-using-drone/aruco_models'

-> Copy 'marker0' and 'marker1' folders

-> Navigate manually to home

-> Press ‘ctrl+h’ to view hidden files in home

-> Navigate manually to ‘.gazebo/models’

-> Press ‘ctrl+v’ to paste marker0 and marker1 in the folder

-> Navigate manually to ‘catkin/src/ eYSIP-2017_Navigation-in-Indoor-Environments-using-drone/camera_calibration_files’

-> Copy all files in the folder

-> Navigate manually to home

-> Navigate manually to ‘.ros’

-> Create a folder named ‘camera_info’

-> Press ‘ctrl+v’ to paste the calibration files

-> Navigate manually to ‘catkin_ws/src/eYSIP-2017_Navigation-in-Indoor-Environments-using-drone/launch/single_aruco.launch’

-> Head over to line 7, col 7

-> Modify the calibration_file destination to your actual calibration folder that you created in ‘.ros/camera_info’

-> Navigate manually to ‘catkin_ws/src/ eYSIP-2017_Navigation-in-Indoor-Environments-using-drone/worlds/ small_world_with_aruco.sdf

-> Head over to lin 269,col 20 and modify the path

-> Head over to lin 402,col 20 and modify the path

-> Head over to lin 455,col 20 and modify the path

-> Navigate manually to ‘catkin_ws/src/marker_pose_detection/launch/ viewpoint_estimation_genius_f100.launch

-> Head over to lin 17, col 35 and change ”/usb_cam/image_raw” to “/ardrone/front/image_raw”

-> Head over to lin 20, col 26 and modify the calibration path to /home/ros/.ros/camera_info/ardrone_front.ini

-> Open a new terminal and enter the following commands to install aruco libraries

$ sudo apt-get install ros-indigo-aruco

6.9.8 Running the Gazebo World with Aruco Markers

$ cd ~/catkin_ws

$ source devel/setup.bash

$ roslaunch drone_application simulator.launch

-> After the world appears, open a new terminal and enter following commands

$ cd ~/catkin_ws

$ source devel/setup.bash

$ roslaunch tum_ardrone tum_ardrone.launch

-> Open a new terminal and execute the following commands

$ cd ~/catkin_ws

$ source devel/setup.bash

$ roslaunch drone_application single_aruco.launch

-> Fly the drone to the front of any marker and you can see its pose in single_aruco.launch window

6.9.9 Takeoff, Land and other Commands with Aruco Markers

-> Navigate manually to ‘catkin_ws/src/marker_pose_detection/src/viewpoint_estimation_lib.cpp’CC -> Include the following line to the code

```
#include <std\_msgs/Empty.h>
```

-> headover to lin 228,col 1 and paste the following code

```
//=======================================================
```

```
//=================Aruco Detection and Commands=========================
//======================================================

geometry_msgs::Twist up;
geometry_msgs::Twist down;
geometry_msgs::Twist left;
geometry_msgs::Twist right;
geometry_msgs::Twist forward;
geometry_msgs::Twist backward;
geometry_msgs::Twist rotate_left;
geometry_msgs::Twist rotate_right;
geometry_msgs::Twist hover;
std_msgs::Empty emp_msg;

ROS_INFO("executing mission");

ros::NodeHandle node;
ros::Rate loop_rate(50);

ros::Publisher pub_empty_takeoff;
ros::Publisher pub_empty_land;
ros::Publisher pub_twist;
ros::Publisher pub_empty_reset;
double start_time;

up.linear.x=0.0;
up.linear.y=0.0;
up.linear.z=0.3;
up.angular.x=0.0;
```

```
up.angular.y=0.0;
up.angular.z=0.0;

down.linear.x=0.0;
down.linear.y=0.0;
down.linear.z=-0.3;
down.angular.x=0.0;
down.angular.y=0.0;
down.angular.z=0.0;

left.linear.x=0.0;
left.linear.y=0.5;
left.linear.z=0.0;
left.angular.x=0.0;
left.angular.y=0.0;
left.angular.z=0.0;

right.linear.x=0.0;
right.linear.y=-0.5;
right.linear.z=0.0;
right.angular.x=0.0;
right.angular.y=0.0;
right.angular.z=0.0;

forward.linear.x=0.5;
forward.linear.y=0.0;
forward.linear.z=0.0;
forward.angular.x=0.0;
forward.angular.y=0.0;
```

```
forward.angular.z=0.0;

backward.linear.x=-0.5;
backward.linear.y=0.0;
backward.linear.z=0.0;
backward.angular.x=0.0;
backward.angular.y=0.0;
backward.angular.z=0.0;

rotate_left.linear.x=0.0;
rotate_left.linear.y=0.0;
rotate_left.linear.z=0.0;
rotate_left.angular.x=0.0;
rotate_left.angular.y=0.0;
rotate_left.angular.z=0.5;

rotate_right.linear.x=0.0;
rotate_right.linear.y=0.0;
rotate_right.linear.z=0.0;
rotate_right.angular.x=0.0;
rotate_right.angular.y=0.0;
rotate_right.angular.z=-0.5;

hover.linear.x=0.0;
hover.linear.y=0.0;
hover.linear.z=0.0;
hover.angular.x=0.0;
hover.angular.y=0.0;
hover.angular.z=0.0;
```

```
float time=1;
float takeland_time=3.0;

if (current_marker_id==0)
{
for(int p=1; p<2; p++)
{
pub_empty_takeoff = node.advertise<std_msgs::Empty>("/ardrone/takeoff", 1);
start_time =(double)ros::Time::now().toSec();

while ((double)ros::Time::now().toSec()< start_time+takeland_time)
{
pub_empty_takeoff.publish(emp_msg);
ROS_INFO("Taking off");
}
}
}
if (current_marker_id==1)
{
for(int p=1; p<2; p++)
{
pub_twist = node.advertise<geometry_msgs::Twist>("/cmd_vel", 1);
start_time =(double)ros::Time::now().toSec();

while ((double)ros::Time::now().toSec()< start_time+time)
{
pub_twist.publish(up);
ROS_INFO("up");
```

```
}
}
}
if (current_marker_id==2)
{
for(int p=1; p<2; p++)
{
pub_twist = node.advertise<geometry_msgs::Twist>("/cmd_vel", 1);
start_time =(double)ros::Time::now().toSec();

while ((double)ros::Time::now().toSec()< start_time+time)
{
pub_twist.publish(down);
ROS_INFO("down");
}
}
}
if (current_marker_id==3)
{
for(int p=1; p<2; p++)
{
pub_twist = node.advertise<geometry_msgs::Twist>("/cmd_vel", 1);
start_time =(double)ros::Time::now().toSec();

while ((double)ros::Time::now().toSec()< start_time+time)
{
pub_twist.publish(left);
ROS_INFO("left");
}
```

```
}
}
if (current_marker_id==4)
{
for(int p=1; p<2; p++)
{

pub_twist = node.advertise<geometry_msgs::Twist>("/cmd_vel", 1);
start_time =(double)ros::Time::now().toSec();

while ((double)ros::Time::now().toSec()< start_time+time)
{
pub_twist.publish(right);
ROS_INFO("right");
}
}
}
if (current_marker_id==5)
{
for(int p=1; p<2; p++)
{
pub_twist = node.advertise<geometry_msgs::Twist>("/cmd_vel", 1);
start_time =(double)ros::Time::now().toSec();

while ((double)ros::Time::now().toSec()< start_time+time)
{
pub_twist.publish(forward);
ROS_INFO("forward");
}
```

```
}
}
if (current_marker_id==6)
{
for(int p=1; p<2; p++)
{
pub_twist = node.advertise<geometry_msgs::Twist>("/cmd_vel", 1);
start_time =(double)ros::Time::now().toSec();

while ((double)ros::Time::now().toSec()< start_time+time)
{
pub_twist.publish(backward);
ROS_INFO("backward");
}
}
}
if (current_marker_id==7)
{
for(int p=1; p<2; p++)
{
pub_twist = node.advertise<geometry_msgs::Twist>("/cmd_vel", 1);
start_time =(double)ros::Time::now().toSec();

while ((double)ros::Time::now().toSec()< start_time+time)
{
pub_twist.publish(rotate_left);
ROS_INFO("rotating left");
}
}
```

```
}
if (current_marker_id==8)
{
for(int p=1; p<2; p++)
{
pub_twist = node.advertise<geometry_msgs::Twist>("/cmd_vel", 1);
start_time =(double)ros::Time::now().toSec();

while ((double)ros::Time::now().toSec()< start_time+time)
{
pub_twist.publish(rotate_right);
ROS_INFO("rotating right");
}
}
}
if (current_marker_id==9)
{
for(int p=1; p<2; p++)
{
pub_empty_land = node.advertise<std_msgs::Empty>("/ardrone/land", 1);
start_time =(double)ros::Time::now().toSec();

while ((double)ros::Time::now().toSec()< start_time+takeland_time)
{
pub_empty_land.publish(emp_msg);
ROS_INFO("landing");

}
}
```

}

-> Save the file and repeat " running the gazebo world with aruco markers " process to land and takeoff ardrone using markers.

6.10 Proposed Algorithm

6.10.1 Aruco Markers based localization and path planning

In this project Aruco readings provide unique Ids for the position and landmarks. Every Aruco Marker state, includes the full position of robot in world coordinates using angles to describe attitude. As the position estimated has a drift in horizontal, to have a prediction it is used increment of this position estimated. Hence input commands are:

$$Du(k) = p^{R(k)}_{R(k \quad 1)} = p^{WO}_{R}(k \quad 1)p^{WO(k \quad 1)}_{WO(k)} p^{WO}_{R}(k) \tag{6.1}$$

Assuming $p^{WO(k \quad 1)}_{WO(k)} = 0$ small drift between two time instants.

p^{WO}_{R} is position of odometry in world coordinates

$$Du(k) = p^{WO}_{R}(k \quad 1) \quad p^{WO}_{R}(k) \tag{6.2}$$

In terms of noise the input command is given by:

$$D'u(k) = Du(k) + n_u \tag{6.3}$$

The robot state is modified:

$$p^{W}_{R}(k) = p^{W}_{R}(k \quad 1) \quad p^{R(K)}_{R(k \quad 1)} \tag{6.4}$$

p^{W}_{R} is position of robot in world coordinates

The robot model is given by:

$$x_R(k) = x_R(k \quad 1) \quad D'u(k \quad 1) \tag{6.5}$$

x_R is robot state.

Landmark process model is given by:

$$p^W_{VMi}(k) = p^W_{VMi}(k \quad 1) \tag{6.6}$$

p^W_{VMi} is position of markers in world coordinates.

The model that allows to calculate the state of new aruco marker is given by:

$$p^W_{VMi} = p^W_R \quad p^R_C \quad p^R_{VMi} \tag{6.7}$$

p^R_C is camera calibration paramereter.

p^W_{VMi} is position of markers in world coordinates.

6.11 Practical Results

Below are some Practical results that we performed during the Aruco test. Fig6.9 and Fig6.10 shows that the ardrone turning left when Aruco Marker comes in front of drones camera.

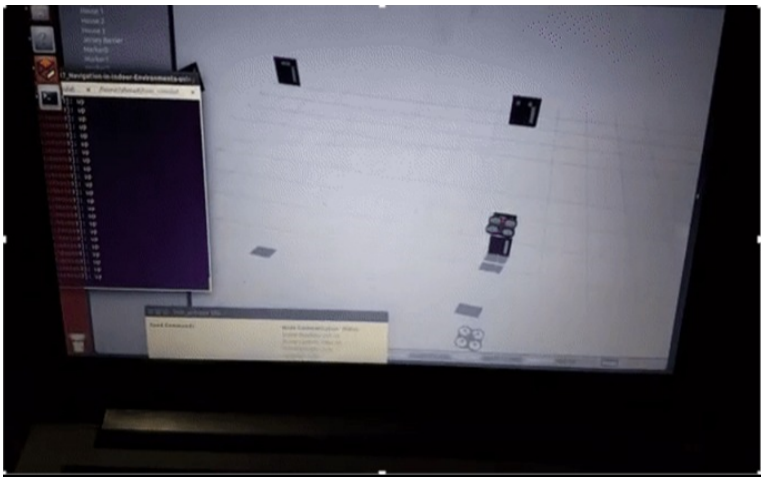

Figure 6.8: Software Simulation

Figure 6.9: Turning Left Using Aruco

Figure 6.10: Turning Left

Chapter 7

Aerostack

In this chapter we are going to discuss:-

- Introduction to Aerostack
- Installation and launch of Aerostack
- Graphical User Interface
- Creating Environment Map
- Behavior Tree
- Practical Results

7.1 Introduction to Aerostack

In FYP (II) our Jury raised the question that if we use Augmented Reality for hurdle avoidance and Path Planning in a warehouse setup, the whole environment of warehouse will be full of Aruco Markers and there will be a messy setup using these markers. Hence we need such a setup which minimizes the use of Aruco Markers. A setup which can avoid hurdles and locate the path autonomously without the use of Aruco Markers. Aerostack is the software which is built for developers to design the architecture of aerial robotics network. We integrated all our setup of Swarm, Open CV, Package localization, Augmented Reality in addition with Aerostack to create our required setup for warehouse environment. Using Aerostack [29] we can create a Map of our warehouse environment and the stationary hurdles which can come across the path of drones. We added the points where our quad copters are supposed to go, from the reference (takeoff) to the final destination. The quad copter will autonomously takeoff and go to the final location avoiding the stationary obstacles without the usage of Aruco Markers. Now Aruco Markers are only used for hurdle avoidance of moving objects and package localization. Hence we achieved our goal to minimize the usage of Aruco Markers.

7.2 Installation and launch of Aerostack

Aerostack [29] has being used and tested using the following setup:

- Ubuntu Linux 16.04 as an Operating System.
- Robot Operating System (ROS), Kinetic version.

The installation requires the following steps:

1. Open the teminal.
2. Download the file by just cloning it from git using the following command:

$ git clone https://github.com/Vision4UAV/Aerostack_installer ~/temp

3. Run it by the command:

$ ~/temp/installation_script.sh

4. Reopen the terminal and navigate to the aerostack folder by the command:

$ cd $AEROSTACK_STACK

5. Update sub modules by the following command:

$ git submodule update –init –recursive

6. Install dependencies or update the pre-installed dependencies by:

$ $AEROSTACK_STACK/installation/installation_dep_script.sh

Now to launch the first flight of aerostack first make sure of the following things.

- The drone is on and reset it if required.
- Connect the PC with drone's Wi-Fi.

1. Now open the terminal.
2. Turn on the master by the command:

$ roscore

3. Now navigate to the folder which have launcher files:

$ cd $AEROSTACK_STACK/launchers

4. Execute the following command to get the Graphical User Interface for all launchers:

$ ./ardrone_basic.sh

Now the installation and launch is complete and we can operate the drone by User Interface Controls Fig7.1.

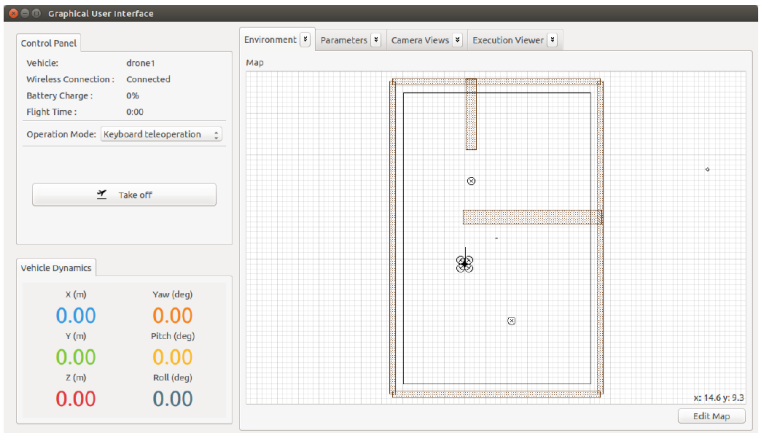

Figure 7.1: User Interface Controls

Now to terminate all the launchers we can use the following command in the terminal:

$ cd $AEROSTACK_STACK/launchers

$ ./stop.sh

To access this feature copy the following camera calibration files from config folder:

- ardrone_camera_bottom.yaml
- ardrone_camera_front.yaml

Now navigate to home

- Go to the folder named '.ros'.
- Create a folder named 'camera_info'.
- Press 'ctrl+v' to paste these calibration files.

7.3 Graphical User Interface

The Graphical User Intrerface [31] of aerostack [29] helps user to understand the both internal and external behavior of robot. The GUI offers both communications from user to robot as well as from robot to user. Both communications are important because:

- The communication from robot to the user is important because to monitor all the process and to look whether the given task to the robot is performing accordingly.
- The communication from user to the robot is important because if there is any mis-

conduct performed by the robots during the execution of mission, the GUI have some emergency options like land the robot or abort the mission and re-initiate the mission.

The Graphical user interface can perform the following tasks:

- Before starting the mission the user can specify what kind of behavior robot needs to follow (teleop operated, guided by behavior tree).
- To monitor the process during execution of mission. If there is any Mal function occurred during the process, it is possible to handle the emergency situation by stop the mission or repair the cause which affect the mission and re-initiate the mission.
- The user can also operate the basic commands manually simply by keyboard like (takeoff, turn left, turn right, rotate, land etc).
- The GUI can also collect the information from the front and bottom camera as well as form other hardware for further use.

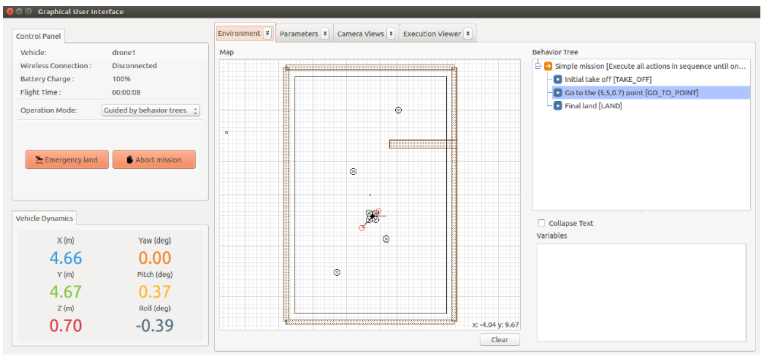

Figure 7.2: Graphical User Interface

7.4 Environment Map

This describes that how the user can create an environment map [31] of a mission to be executed using graphical tool. Using this tool we can create a map of our environment by setting the size and dimensions and by placing the static hurdles which can come across the path of robot. This also allows user to load or save the current map to the config files and to import or export the map.

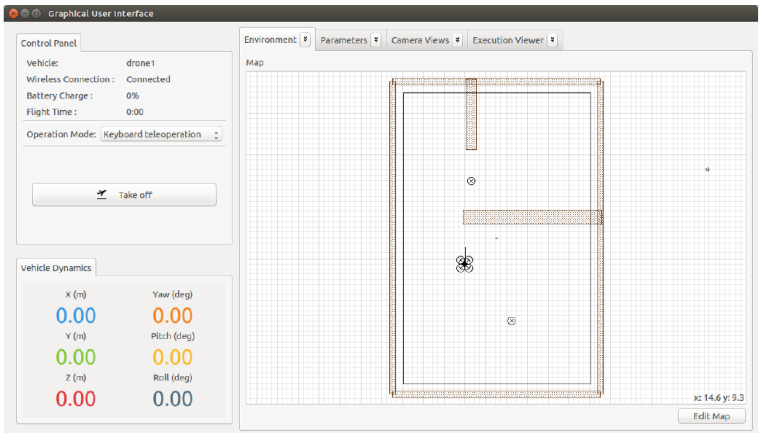

Figure 7.3: Environment Map

In order to edit the map click the ”Edit Map” icon in the lower right corner in environment tab Fig 7.3.

7.4.1 Creating a New Map

To create a new map the following steps needs to be followed:

- In environment viewer press “New” tab.
- A window will appear. Make the size of the map and the according to the need and the reference (takeoff) points.
- Press “Accept” and a new map will be created Fig7.4.

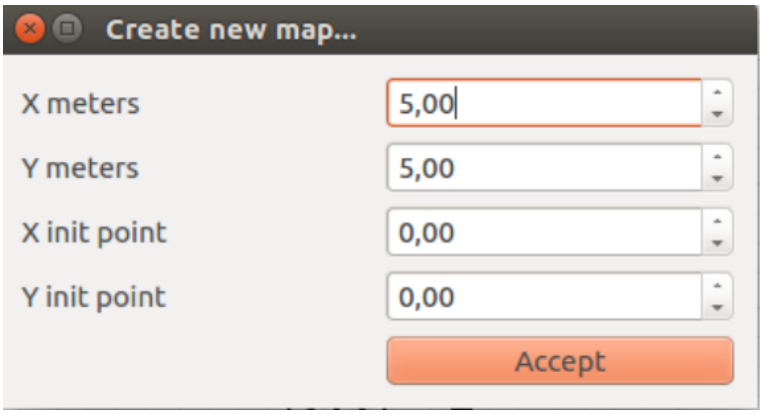

Figure 7.4

7.4.2 Creating Obstacles in the Map

A tool's set is created to add new obstacles in the map. To create new obstacles, select the tool with the required object and place it in the desired position.

Now to edit the existing obstacles, go through the following steps

- Click "Selection Tools" and click the desired obstacle in the map.
- This displays a dialog box in the right of the window in order to modify the selected obstacle.
- If we want to delete the obstacle, click on the obstacle and press "supr" key.

In in the following screenshots Fig7.5 Fig7.6 Fig7.7, a pole and a wall are created as obstacle. Keeping in mind that obstacles Id number must be different. If they are not the user will be asked to check them and setup properly.

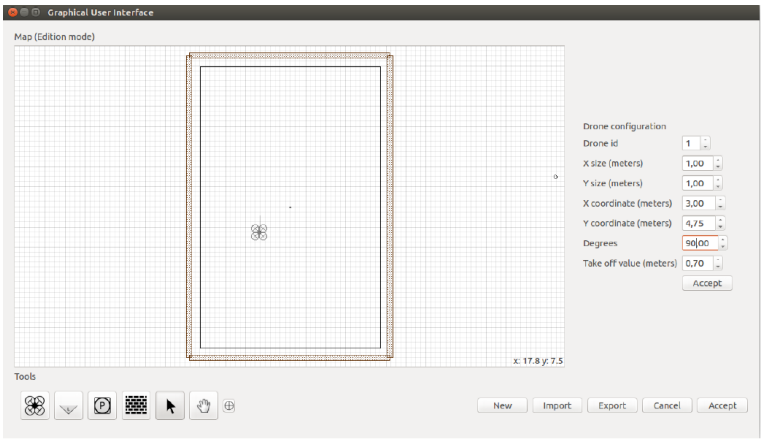

Figure 7.5

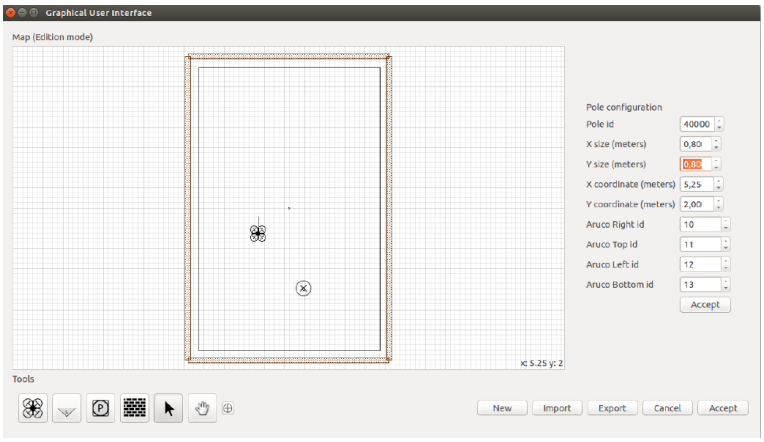

Figure 7.6

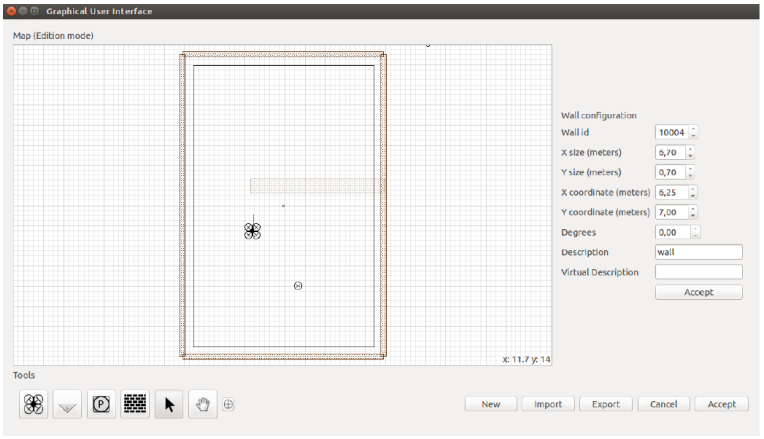

Figure 7.7

When we are finished with editing the environment, click on accept. This displays a window which tells us that a new environment cant be available until the Aerostack [29] is restarts as shown in the Fig7.8.

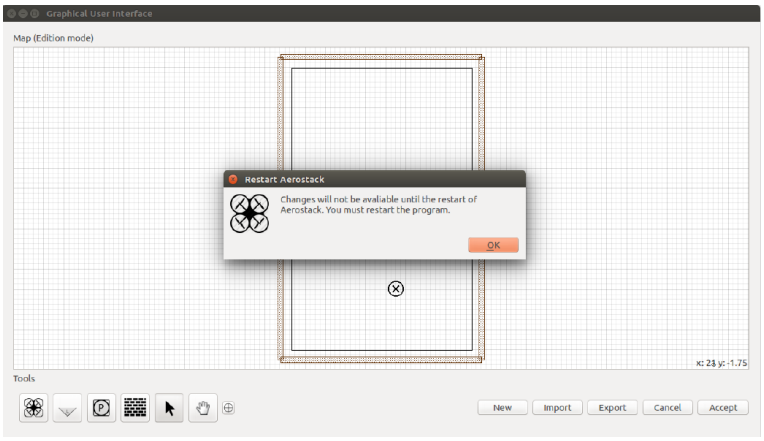

Figure 7.8

When we choose Ok, the latest map added to the Aerostack will be displayed as in the Fig7.9 we choose the map containing a pole and a wall.

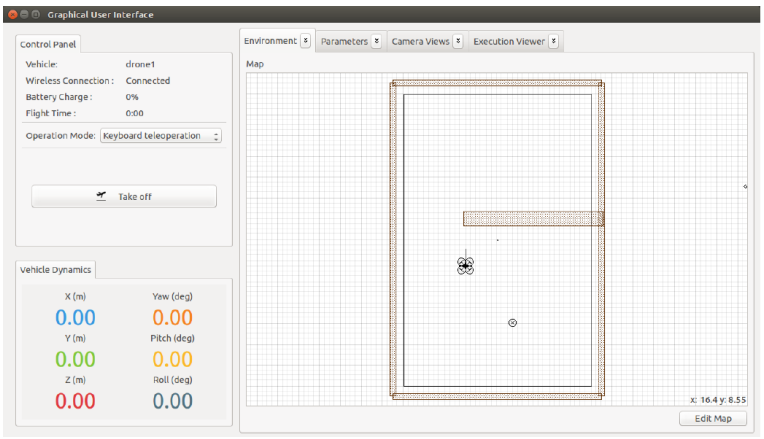

Figure 7.9

7.5 Behavior Tree

A behavior tree is a visual modeling that define graphical notation to show system's behavior. The behavior tree is represented by nodes hierarchy. Each node can successfully run or can fail. When the user creates the behavior tree, he/she gives a particular name to each node with a particular command.

7.5.1 Creating a Behavior Tree

The user can plan the mission by using the behavior tree of GUI of Aerostack. Following are the steps to create a behavior tree for the mission:

- Change operation mode from “keyboard teleoperation to Guided by Behavior Tree” from the control panel.
- Click on “Edit Mission” tap which is at bottom right corner to edit the current behavior or to create a new one.
- By clicking “New” we can create a new Behavior tree replacing the previous one. A window will appear asking for the name for new mission.
- After this “right click” on the desired node and a menu will appear. The menu asks us to add the new child node, modify the node or to delete the node as shown in Fig7.10 Fig7.11.

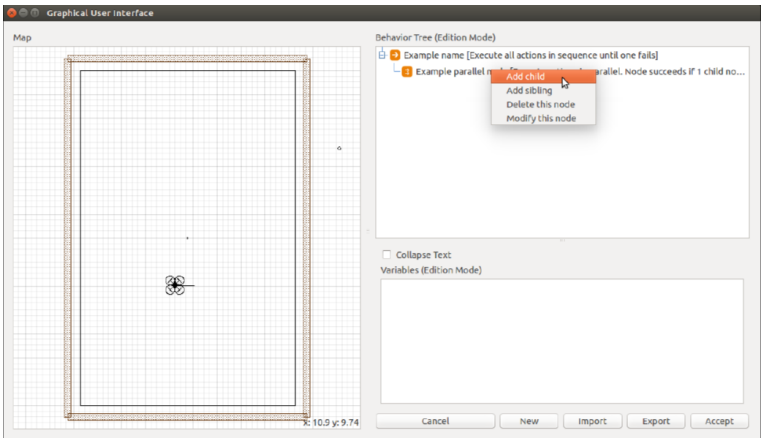

Figure 7.10

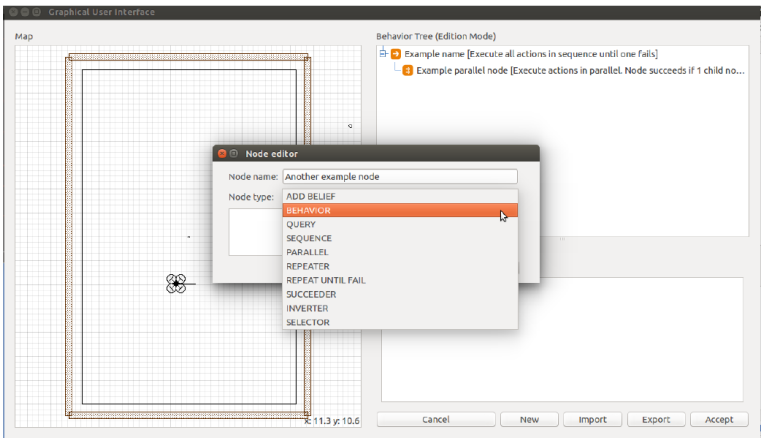

Figure 7.11

• The user can save the Behavior Tree in .yaml format by clicking “Export” and can simply access the Behavior Tree in the mission that is saved previously by clicking “Import”.

• Finally click Accept, and the Behavior Tree is ready to be executed as in Fig7.12.

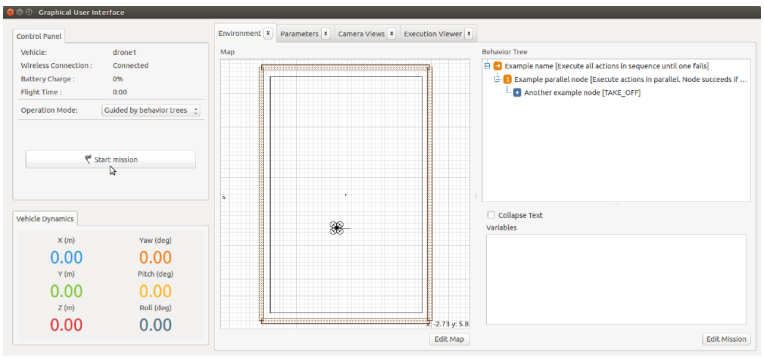

Figure 7.12: Behavior Tree With Single Child Node

Fig7.13 is the complete example of Behavior Tree with different nodes.

Figure 7.13: Behavior Tree With Multiple Child Nodes

7.6 Practical Results

The practical was performed with the stationary hurdle in front of ardrone. The hurdle was mapped in the environment of Aerostack [29]. When the mission was executed the quadcopter successfully avoids the hurdle and reaches its destination. The shots of which are shown in Fig 7.14 Fig7.15 Fig7.16 Fig7.17.

Figure 7.14

Figure 7.15

Figure 7.16

Figure 7.17

References

[1] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, Oxford,(1999).

[2] D. W. Corne, A. Reynolds, E. Bonabeau, Swarm Intelligence, in:Handbook of Natural Computing (Eds. G. Rozenberg, T. Bäck, J. N.Kok), Springer, pp. 1599-1622 (2012).

[3] L. Fisher, The Perfect Swarm: The Science of Complexity in Everday Life, Basic Books, (2009).

[4] P. Miller, Swarm theory, National Geographic, July 2007.

[5] M. W. Müller, S. Lupashin, and R. D'Andrea, "Quadrocopter ball juggling," in Proceedings of the 24th IEEE/RSJ International Conference on Intelligent Robot Systems, San Francisco, USA, Sept 2011, pp. 5113–5120.

[6] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," The International Journal of Robotics Research, vol. 31, no. 5, pp. 664–674, 2012.

[7] S. Shen, N. Michael, and V. Kumar, "Autonomous multi-floor indoor navigation with a computationally constrained mav," in Proceedings of the IEEE 2011 International Conference on Robotics and Automation - ICRA'11, Shanghai, China, May 2011, pp. 20–25

[8] S. Weiss, M. Achtelik, S. Lynen, M. Achtelik, L. Kneip, M. Chli, and R. Siegwart, "Monocular vision for long-term micro aerial vehicle state estimation: A compendium," Journal of Field Robotics, vol. 30, no. 5, pp. 803–831, 2013.

[9] F. Kendoul, Z. Yu, and K. Nonami, "Guidance and nonlinear control system for autonomous flight of minirotorcraft unmanned aerial vehicles," Journal of Field Robotics, vol. 27, no. 3, pp. 311–334, 2010.

[10] T. Krajnik, V. Vonasek, D. Fiser, and J. Faigl, “Ar-drone as a platform for robotic research and education,” in Research and Education in Robotics - EUROBOT 2011, ser. Communications in Computer and Information Science, D. Obdrzalek and A. Gottscheber, Eds. Springer, 2011, vol. 161

[11] J. J. Lugo and A. Zell, “Framework for autonomous on-board navigation with the ar.drone,” Journal of Intelligent and Robotic Systems, vol. 73, no. 1-4, pp. 401–412, 2014.

[12] A. A. R. Group, “Ardrone autonomy repository,”https://github.com/Autonomy-Lab/ardrone autonomy. accessed on August 12th 2013.

[13] T. A. R. Group, “Tum ardrone repository,” https://github.com/tum vision/tum ardrone. accessed on August 12th 2013.

[14] Engel, Jakob et all. “Accurate Figure Flying with a Quadrocopter Using Onboard Visual and Inertial Sensing”. Department of Computer Science, Technical University of Munich, Germany.http://ros.org/wiki/tum ardrone/.

[15] S. Cousins, “Exponential Growth of ROS,” IEEE Robotics and Automa-tion Magazine, no. March, pp. 19–20, 2011.

[16] E. Budianto, M.S. Alvissalim, A. Hafidh, A. Wi-bowo, W. Jatmiko, B. Hardian, P. Mursanto, A. Muis,“Telecommunication Networks Coverage Area Ex-pansion in Disaster Area Using Autonomous MobileRobots:Hardware and Software Implementation”,IEEE International Conference on Advanced Com-puter Science and Information Systems, pp. 113–118,2011.

[17] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” in Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA), 2011.

[18] J. Engel, J. Sturm, and D. Cremers. Camera-based navigation of a low-cost quadrocopter. In Proc. of the International Conference on Intelligent Robot Systems (IROS), Oct. 2012.

[19] M. M¨uller, S. Lupashin, and R. D'Andrea, "Quadrocopter ball juggling," in Proc. IEEE Intl. Conf. on Intelligent Robots and Systems (IROS), 2011.

[20] Q. Lindsey, D. Mellinger, and V. Kumar, "Construction of cubic structures with quadrotor teams," in Proceedings of Robotics: Science and Systems (RSS), Los Angeles, CA, USA, 2011.

[21] Aruco: a minimal library for augmented reality applications based on opencv. http://www.uco.es/ investiga/grupos/ava/node/26.

[22] "Parrot AR.Drone," 2012. [Online]: http://ardrone.parrot.com/.

[23] T. Krajn´ık, V. Von´asek, D. Fi˘ser, and J. Faigl, "AR-drone as a platform for robotic research and education," in Proc. Communications in Computer and Information Science (CCIS), 2011.

[24] Jesu´s Pestana. On-board control algorithms for Quadrotors and indoors navigation. Master's thesis, Universidad Polit´ecnica de Madrid, Spain, 2012

[25] J. Engel, "Autonomous Camera-Based Navigation of a Quadrocopter," Master's thesis, Technical University Munich, 2011.

[26] P. Rudol. Increasing autonomy of unmanned aircraft systems through the use of imaging sensors. Master's thesis, Linkoping Institute of Technology, 2011.

[27] J. De Schutter, J. De Geeter, T. Lefebvre, and H. Bruyninckx. Kalman Filters: A Tutorial, 1999.

[28] https://appreal-vr.com/blog/augmented-reality-and-drones/

[29] http://vision4uav.com/

[30] https://github.com/tentone/aruco

[31] https://github.com/Vision4UAV/Aerostack

[32] http://gazeboism.org

[33] https://www.ros.org