South Dakota State University
Open PRAIRIE: Open Public Research Access Institutional Repository and Information Exchange

Electronic Theses and Dissertations

2018

Development of an Indoor Multirotor Testbed for Experimentation on Autonomous Guidance Strategies

Kidus Guye
South Dakota State University

Follow this and additional works at: https://openprairie.sdstate.edu/etd

Part of the Aerospace Engineering Commons, and the Energy Systems Commons

Recommended Citation
Guye, Kidus, "Development of an Indoor Multirotor Testbed for Experimentation on Autonomous Guidance Strategies" (2018). *Electronic Theses and Dissertations*. 2972.
https://openprairie.sdstate.edu/etd/2972

This Thesis - Open Access is brought to you for free and open access by Open PRAIRIE: Open Public Research Access Institutional Repository and Information Exchange. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Open PRAIRIE: Open Public Research Access Institutional Repository and Information Exchange. For more information, please contact michael.biondo@sdstate.edu.

DEVELOPMENT OF AN INDOOR MULTIROTOR TESTBED FOR EXPERIMENTATION ON AUTONOMOUS GUIDANCE STRATEGIES

BY

KIDUS GUYE

A thesis submitted in partial fulfilment of the requirements for the

Master of Science

Major in Mechanical Engineering

South Dakota State University

2018

DEVELOPMENT OF AN INDOOR MULTIROTOR TESTBED FOR EXPERIMENTATION ON AUTONOMOUS GUIDANCE STRATEGIES

KIDUS GUYE

This thesis is approved as a creditable and independent investigation by a candidate for the Master of Science degree and is acceptable for meeting the thesis requirements for this degree. Acceptance of this thesis doesn't imply that the conclusion reached by the candidate are necessarily the conclusions of the major department.

Marco Ciarcià, PhD
Thesis/ Major Advisor

Date

Kurt Bassett, PhD
Head, Mechanical Engineering

Date

Dean, Graduate School

Date

ACKNOWELDGEMENTS

I would first like to thank my thesis advisor Dr. Ciarcià. He has been available to provide guidance and support whenever I had questions about my research approach or writing. Dr. Ciarcià consistently allowed this paper to be my own work but steered me in the right direction whenever he thought I needed it.

A special thanks to Dr. Venanzio Cichella for his generous technical support on the implementation of Bezier trajectories. His contribution was instrumental to develop the real time optimization software.

I would also like to thank members of ARTLAB who were involved in reviewing my paper and I am gratefully indebted to them for their very valuable comments on this thesis.

Finally, I must express my very profound gratitude to my mom, EMEBET MULATU and my whole family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

CONTENTS

LIST OF FIGURES vi

LIST OF TABLES viii

ABSTRACT ix

CHAPTER ONE 1

INTRODUCTION 1

1.1. Thesis Goals and Outline 4

CHAPTER TWO 6

QUADROTOR DYNAMIC MODELLING 6

2.1. Kinematics of a Quadrotor 8

CHAPTER THREE 11

MULTIROTOR TESTBED LAB 11

3.1. Streaming data from Motive to Simulink 14

3.2. Controlling AR Drone from MATLAB-Simulink 20

3.2.1. OptiTrack Model 21

3.2.2. Controller Model 23

CHAPTER FOUR 30

ACCURATE LANDING AND FORMATION FLIGHT 30

4.1. Accurate Landing 31

4.1.1. Phase 1: Take-off 32

4.1.2. Phase 2: Closing 32

4.1.3. Phase 3: Docking/Landing 35

4.2. Formation Flight 40

CHAPTER FIVE 45

COMPARISON OF NLP SOLVERS 45

5.1. Problem Statement 46
5.2. Nonlinear Programming Solver 46
5.3. Example Problems and Result 48
CHAPTER SIX 62
TRAJECTORY OPTIMIZATION 62
6.1. Previous Works 63
6.2. Bezier Curve 64
6.2.1. Determinations of Optimal Trajectory 65
6.3. Numerical Evaluation and Results 67
6.3.1. Pre-flight Simulation Results 68
6.3.2. Experimental Result 71
CHAPTER SEVEN 78
CONCLUSION 78
7.1. Summary 78
7.2. Recommendation 78
REFERENCES 80
APPENDIX I 84
APPENDIX II 89
APPENDIX III 90

LIST OF FIGURES

Figure 1. Fixed wing vehicles 1
Figure 2. Multirotor vehicles 2
Figure 3. AR. Drone 2.0 4
Figure 4. Body and inertial frame of reference and attitude angles for the quadrotor ... 6
Figure 5: Rotor actuation to execute: (a) hovering and vertical motion, (b) yaw angle variation, (c) longitudinal motion, (d) lateral motion 7
Figure 6. Multirotor testbed structure 12
Figure 7. Aerospace Robotics Testbed Laboratory (ARTLAB) 14
Figure 8. Camera Calibration Pane 16
Figure 9. (a) OptiTrack Calibration Wand (b) OptiTrack Calibration Square 17
Figure 10. Ground plane pane 18
Figure 11. MCS cameras and Rigid body representation in Motive's graphical interface 19
Figure 12. Data Streaming Engine Pane 20
Figure 13. OptiTrack Model 21
Figure 14. UDP Packet Output 23
Figure 15. Controller model 24
Figure 16. AR Drone Wi-Fi block diagram 25
Figure 17. Schematic diagram of a feedback control system 26
Figure 18. Overall Simulink diagram structure with MCS 29
Figure 19. Three phases of accurate landing 32
Figure 20. Rectangular landing stage 36
Figure 21. Demonstration of why extrapolation was used 37
Figure 22. Position plot on the x, y and z and yaw angle plot 38
Figure 23. 3D plot of the accurate landing path 39
Figure 24. The control outputs plot versus their feedback values 40
Figure 25. Hovering of two drones in a coordination flight 41
Figure 26. Illustration of coordination equation used 42
Figure 27. Formation flight plot 43
Figure 28. Drone 1 plot in the x, y and z direction 44

Figure 29. Drone 2 plot in the x, y and z direction 44
Figure 30. Quadratic Bezier curve 64
Figure 31. Optimized trajectory calculated offline 71
Figure 32. Simulink diagram of the OptiTrack model 72
Figure 33. Initial guess block 73
Figure 34. The duration of time changes as the drone flies 75
Figure 35. Optimized trajectory avoiding collision 76
Figure 36. The X, Y, Z and yaw angle plot 77
Figure 37. x and y position capture at $t = t_s$ 90
Figure 38. The second initial point extrapolation block 90
Figure 39. Initial guess starter block 91
Figure 40. For loop to calculate the initial guess 91
Figure 41. SGRA block content 92
Figure 42. Bezier Curve block 92
Figure 43. Bezier curve points selector 93
Figure 44. Extrapolation for the third and afterwards initial points 93

LIST OF TABLES

Table 1 Expression for the coefficients of x- axis..... 34
Table 2 Expression for the coefficients of y- axis..... 34
Table 3 Expression for the coefficients of z- axis..... 35
Table 4 Accurate landing values..... 38
Table 5 Formation flight results..... 43
Table 6 Comparison of solvers using problem 1 49
Table 7 Comparison of solvers using problem 2 50
Table 8 Comparison of solvers using problem 3 51
Table 9 Comparison of solvers using problem 4 52
Table 10 Comparison of solvers using problem 5 53
Table 11 Comparison of solvers using problem 6 54
Table 12 Comparison of solvers using problem 7 55
Table 13 Comparison of solvers using problem 8 56
Table 14 Comparison of solvers using problem 9 57
Table 15 Comparison of solvers using problem 10 58
Table 16 Comparison of solvers using problem 11 59
Table 17 Comparison of solvers using problem 12 60
Table 18 Comparison of solvers using problem 13 61

ABSTRACT

DEVELOPMENT OF AN INDOOR MULTIROTOR TESTBED FOR EXPERIMENTATION ON AUTONOMOUS GUIDANCE STRATEGIES

KIDUS GUYE

2018

Despite the vast popularity of rotary wing unmanned aerial vehicles and research centres that develop their guidance software, there are only a limited number of references that provide an exhaustive description of a step-by-step procedure to build-up a multirotor testbed. In response to such need, the first part of this thesis aims to describe, in detail, the complete procedure to establish and operate an autonomous multirotor unmanned aerial vehicle indoor experimental platform to test and validate guidance, navigation and control strategies. Both hardware and software aspects of the testbed are described to offer a complete understanding of the different aspects.

The second part of this thesis focuses on two benchmarks multirotor guidance, navigation and control problems. Initially, the guidance law for an accurate landing manoeuvre is studied. Multirotor usually have a flight time limited to a few minutes. Autonomous landing and docking to a charging station could extend the mission duration of these vehicles. Subsequently, the guidance strategy for the formation flight between two multirotors is considered. In this case, the fundamental goal is an accurate autonomous alignment between two vehicles, each of them behaving as a target and chaser simultaneously.

In the last part of this thesis, the problem of minimum energy manoeuvres is tackled. Again, in this case, the motive is to address the limitation in multirotor flight duration. The fundamental objective of this guidance, navigation and control strategy is to determine and implement, in real-time, the minimum energy control histories that transfer the multirotor from its initial point to a given final point. As opposed to conventional guidance strategies,

mostly based on proportional-integral-derivative laws, a minimum energy controller allows the vehicle to execute the manoeuvre with a minimum electrical power expenditure.

CHAPTER ONE

INTRODUCTION

The last two decades have witnessed a growing interest toward unmanned aerial vehicles (UAVs). Some of the most relevant applications relate to contribution to rescue missions, aerial inspection of structures, precision agriculture, aerial imaging/sensing, package delivery, etc. As a result, there is an increasing need of guidance, navigation, and control strategies for this category of aerial vehicles. UAVs fall into two main categories: fixed wing vehicles and rotary wing vehicles. The latter category has, in general, between one to eight rotors depending on design criteria [1].

Figure 1. Fixed wing vehicles

A flying vehicle which uses four rapidly spinning rotors to generate lift and thrust force in order to keep it in flight is usually called quadrotor or quadcopter. This allows the four-rotor UAVs to take off and land vertically and fly frontward, backward and sideways.

Figure 2. Multirotor vehicles

Unlike conventional helicopters, quadrotors are mechanically simpler and less expensive. Moreover, their smaller blade size mitigates the risk of damage to persons and nearby objects. All these aspects make them a popular choice over other UAVs categories.

Nowadays, quadrotors are often used as a standard platform for robotics research projects due mainly to their safety, smaller size/weight, lower cost, and higher manoeuvrability over other aerial vehicles [2]. For example, the AR. Drone 2.0 (Figure 3), built by the French company Parrot, is one of the most popular models of quadrotors that entered the drone market in the last decades. AR. Drone can be either controlled from a phone or tablet with their user-friendly app or can be programmed for autonomous manoeuvre execution.

As stated previously, there is an increasing need of guidance, navigation and control (GNC) strategies for UAVs. In these days, many research groups are addressing these research problems by carrying out experimental work on indoor testbeds that are usually composed by one or more UAV, a personal computer (PC) workstation, and a motion

capture system (MCS). Notably, the latter component allows the retrieval of information on UAV's position and attitude in real time.

As the interest in multirotor vehicles increased, the need to use UAVs for longer duration missions has also increased. Many companies that use multi-rotor aerial vehicles for commercial purposes today require their drones to carry out a longer mission. However, since these rotor crafts use energy from a battery source, it is highly unlikely that these types of missions will be completed with a single fly. Yet, by equipping the drones with the ability to recharge themselves autonomously, long-term missions can be carried out. To carry out an autonomous recharge, aerial vehicles should conduct an autonomous flight to a charging station and make an accurate landing at a docking station.

Together with other advantageous aspects, multirotors are characterized by a few limitations. For example, multiple smaller size blades, as opposed to a fix wing or a single rotary wing, induce a much less efficient flight. A work by Theys et.al. [3], comparing 2-blade and 3-blade propellers, showed that propellers with higher blade numbers are less efficient than those with a small number of blades. Multirotors consume a large amount of energy to generate the required lift and hovering force. These aerial robots have a very limited flight endurance of between 15 and 30 minutes [1].

To address such problem various research groups have invested a significant amount of effort toward two strategies. The first was the design of a quadrotor body structure using a lighter material to reduce the overall weight of a quadrotor and the second was the use of high energy density battery package to power a quadrotor. Strategies to distinguish and work on those regimes that are power starving have succeeded in reducing the operation on those regimes; however, no state-of-the-art technological advances are expected in this direction soon [1]. Finally, the most effective strategy for extending the flight duration of quadrotors is to develop a guidance strategy for calculating and carrying out minimum energy trajectory. This master thesis will focus mainly on this last aspect of reducing flight energy consumption.

Figure 3. AR. Drone 2.0

1.1. Thesis Goals and Outline

The goal of this thesis is to discuss development of an indoor multirotor testbed for experimentation on autonomous guidance strategies. The first part of this thesis presents the dynamics equation for quadrotors with X- configuration like that of AR Drone.

Chapter 3 contains the necessary steps needed to set up a testbed lab for experimentation on multi-rotor vehicles using an AR Drone 2.0 quadrotor and OptiTrack motion capture system. This section of the thesis details how to connect a ground station, a quadrotor and OptiTrack cameras. The last part of the section showed the steps to be taken to carry out an autonomous flight using a Simulink model as a controller.

In Chapter 4, accurate landing and coordinated drone flight are studied. A simple polynomial equation was used to calculate the trajectory for a quadrotor to fly to a mock-up charging station autonomously and make a safe landing. In the second part of Chapter 4, two AR drones conduct a formation flight in order to achieve the ultimate objective of both hovering at a fixed point.

Chapter 5 includes the comparison of different nonlinear programming optimization tools. We have chosen and compared six nonlinear programming solvers by calculating the CPU time each took to solve 13 different nonlinear problems.

In Chapter 6, I discuss a real time trajectory optimization technique for the minimization of energy for multirotor. With the motivation to solve the problem of limited endurance of multirotors, trajectory optimization was carried out to minimize the energy consumption.

In the last chapter, a summary of the work done in this thesis paper and a recommendation on future works are summarized.

CHAPTER TWO

QUADROTOR DYNAMIC MODELLING

In this chapter, a quadrotor's dynamical model was mainly derived according to [5] and [6] and briefly reported here for the sake of completeness. The body frame and the inertial frame of references are shown in Figure 4. A quadrotor movement is controlled by balancing the thrust force and the drag torque on each rotor [4]. As shown in Figure 5, the four rotors of a quadrotor generate the lifting force needed to create a motion by varying their speed. To perform hovering, each rotor rotates at the same angular rates, creating equal contributions to the total thrust, as schematized in Figure 5(a). When vertical motion is required, the quadrotor can move vertically by increasing or decreasing the speed of the propellers, thus creating higher or lower thrust values, with respect to the equilibrium, while maintaining the rotational balance of the rotorcraft.

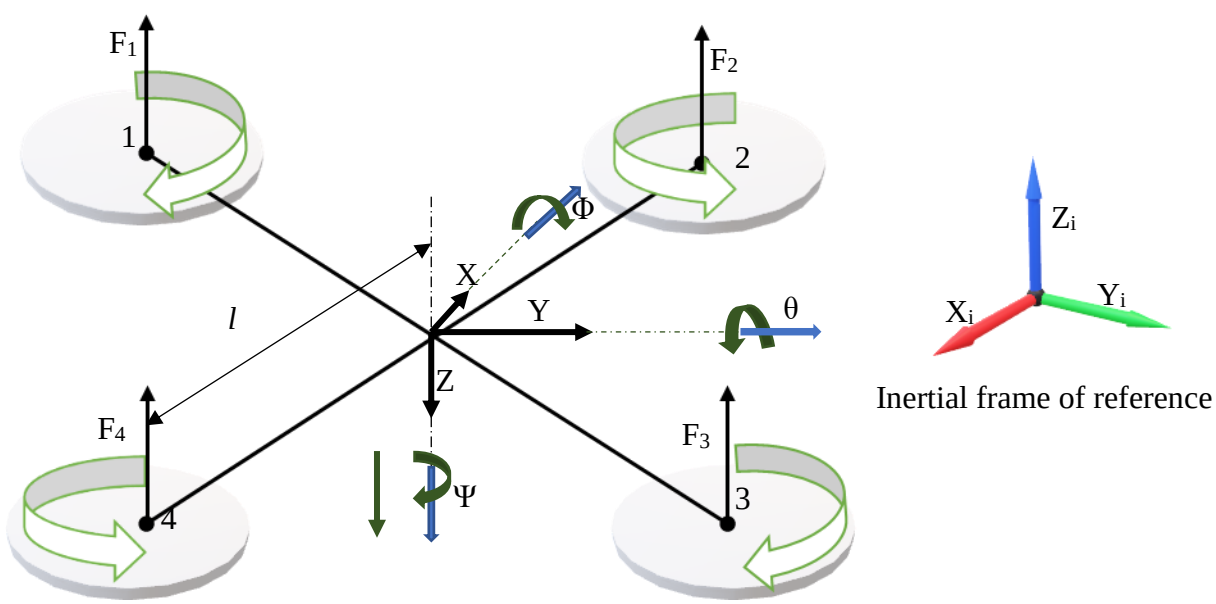

Figure 4. Body and inertial frame of reference and attitude angles for the quadrotor

The three Euler angles can be changed by varying the angular rates of the four rotors. For example, if a positive yaw angle is commanded, the speed of rotor 1 and 3 is diminished

and the speed of rotor 2 and 4 is equally increased with the final result of creating a positive reaction torque, on the multirotor body, and maintaining the same total thrust for vertical equilibrium. Correspondingly, if a negative yaw angle is commanded, the speed of rotor 1 and 3 will be increased while decreasing the rotor speed of 2 and 4 with an equal amount. Figure 5: shows schematic illustration of the above stated fact.

To execute a forward motion of the quadrotor, the pitch angle must be varied. To do this, the rotor speed of 1 and 2 increased with the same amount as the reduced rotor speed of 3 and 4, consequently making a negative pitch angle and moving the quadrotor forward as shown in Figure 5(c). On the other hand, to move the quadrotor backward or make a positive pitch angle the rotors speed of 3-4 and 1-2 increase and decrease with a similar sum, respectively. Similarly, increasing and decreasing the speed of the rotor pairs 1-4 and 2-4, drives the quadrotor in the lateral direction and changes the roll angle. The above particular stated fact is shown in Figure 5(d).

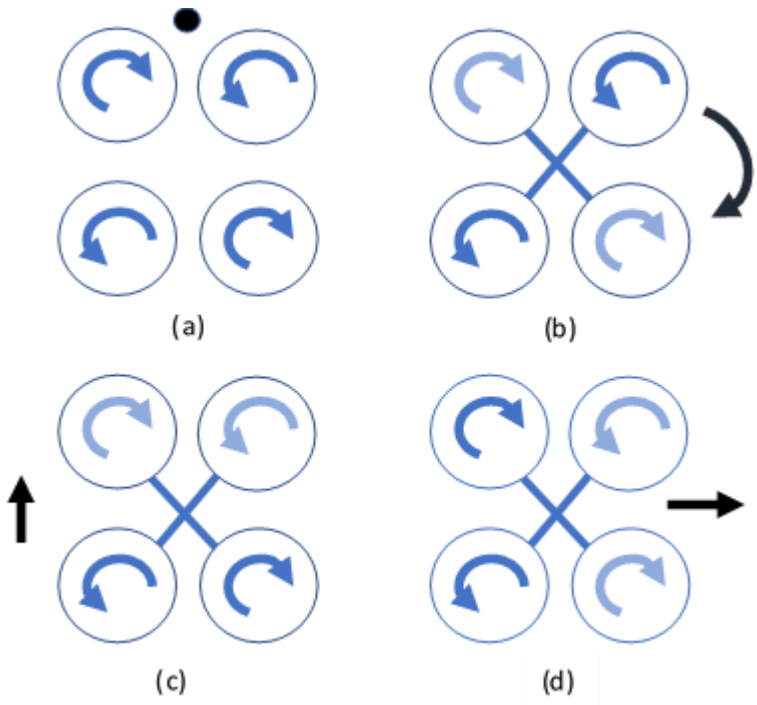

Figure 5: Rotor actuation to execute: (a) hovering and vertical motion, (b) yaw angle variation, (c) longitudinal motion, (d) lateral motion

2.1. Kinematics of a Quadrotor

The following assumptions have been taken into account during the dynamic model derivation of the quadrotor:

i. The structure of rotorcraft is rigid
ii. The propellers are rigid
iii. There is no blade flipping occurring
iv. The aircraft has a symmetric structure
v. The body frame origin and center of gravity of the quadrotor assumed to coincide

The propeller thrust force can be described in terms of the rotating speed considering a thrust factor b as follows [4]

$$T = b\Omega^2 \tag{1}$$

where, T is the thrust force, b is a thrust factor, and Ω is a rotor speed

The following set of four control variables are introduced as functions of four thrusts components and some geometric parameters.

- The total thrust u_z is the sum of thrust generated by each rotor

$$u_z = T_1 + T_2 + T_3 + T_4 \tag{2}$$

- The torque required to create a roll moment is given by

$$\tau_\emptyset = l(T_1 - T_2 - T_3 + T_4) \tag{3}$$

  where, l is the distance of the propeller axis from the center of gravity

- The torque required to create a pitch moment is produced by proportionally varying the front and back speed of the rotors

$$\tau_\theta = l(T_1 + T_2 - T_3 - T_4) \tag{4}$$

- The torque along the yaw angle is calculated by adding each thrust force on the rotors and multiplying it with a proportional constant

$$\tau_{\psi} = d(T_1 + T_2 + T_3 + T_4) \tag{5}$$

where, d is a proportional constant, namely, the ratio between the total thrust and the angular moment

There are two types of movement on quadrotors, translational and rotational. The general equation for the translation motion is given by

$$m\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = -mg\boldsymbol{Z_i} + \boldsymbol{R}F \tag{6}$$

where, $\boldsymbol{Z_i}$ is the vertical axis in the inertial frame, $\boldsymbol{R}$ denotes the rotation matrix used to project the vector from body frame into the inertial frame, m, g and F refers to the total mass, gravitational acceleration and total force applied on the vehicle respectively.

The rotation matrix $\boldsymbol{R}$ to transform from body-frame axes to inertial frame is calculated as

$$\boldsymbol{R} = \begin{bmatrix}1 & 0 & 0\\ 0 & c\phi & s\phi\\ 0 & -s\phi & c\phi\end{bmatrix}\begin{bmatrix}c\theta & 0 & -s\theta\\ 0 & 1 & 0\\ s\theta & 0 & c\theta\end{bmatrix}\begin{bmatrix}c\psi & s\psi & 0\\ -s\psi & c\psi & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}c\psi c\theta & c\psi s\phi s\theta - c\psi s\phi s\theta & s\psi s\phi + s\theta c\psi c\phi\\ c\theta s\psi & c\phi c\psi + s\psi s\theta s\phi & s\psi s\theta c\phi - c\psi s\phi\\ -s\theta & c\theta s\phi & c\theta c\phi\end{bmatrix} \tag{7}$$

with ϕ, θ, and ψ Eluer's angles, and c, s representing cosine and sine operators, respectively.

Similarly, the rotational motion equation can be expressed as

$$\boldsymbol{I\Omega} = -\boldsymbol{\Omega} \text{ x } \boldsymbol{I\Omega} + \boldsymbol{\tau} \tag{8}$$

with $\boldsymbol{I}$ inertia matrix of the multirotor as shown in equation (9), $\boldsymbol{\Omega}$ is the angular velocity of the airframe expressed in the body-fixed frame and total torque $\boldsymbol{\tau} = [\tau_{\psi}, \tau_{\theta}, \tau_{\emptyset}\,]^T$.

$$\boldsymbol{I} = \begin{bmatrix}I_{xx} & 0 & 0\\ 0 & I_{yy} & 0\\ 0 & 0 & I_{zz}\end{bmatrix} \tag{9}$$

where the I_{xx}, I_{yy}, I_{zz} are principal moment of inertia along x, y and z axes respectively.

Substituting the expression of the rotational matrix $\boldsymbol{R}$ in equation (6) and rearranging the derivative of the body frame velocities are given by

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \frac{u_z}{m} \begin{bmatrix} c\psi s\phi - s\theta c\phi s\psi \\ -c\psi s\theta c\phi - s\phi s\psi \\ c\theta c\phi \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{10}$$

The derivation of attitude in terms of the angular rate can then be formulated as

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & sc\theta s\phi & sc\theta c\phi \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{11}$$

where $p, q,$ and r are the derivates of angular rates in body frame reference and t, sc represent tangent and secant operators, respectively.

Correspondingly, the derivatives of angular rates can be expressed in terms of the inertial motion as [5]

$$\dot{p} = \frac{I_{yy} - I_{zz}}{I_{xx}} qr + \frac{I_r}{I_{xx}} q\eta + \frac{\tau_\phi}{I_{xx}} \tag{12}$$

$$\dot{q} = \frac{I_{zz} - I_{xx}}{I_{yy}} pr - \frac{I_r}{I_{yy}} p\eta + \frac{\tau_\theta}{I_{yy}} \tag{13}$$

$$\dot{r} = \frac{I_{xx} - I_{yy}}{I_{zz}} qp + \frac{\tau_\psi}{I_{zz}} \tag{14}$$

where, $\eta = \Omega_4 + \Omega_2 - \Omega_3 - \Omega_1$ is the counter clockwise residual rotor speed, I_{xx}, I_{yy}, I_{zz} principal moment of inertia, and I_r is moment of inertia along the radial axis.

CHAPTER THREE

MULTIROTOR TESTBED LAB

As outlined in the previous sections there is an increasing interest in experimental research to test and validate GNC strategies of multi-rotor vehicles. Nevertheless, there is a limited and fragmented amount of documentation available that shows a thorough approach to the development of an indoor multirotor testbed. In particular, there is a gap of information regarding the procedure to connect and control one or more multirotor using Simulink model with MCS which feedbacks the drone position and attitude data. In [7] the author provides a MATLAB toolbox that would enable to control AR Drone 2.0 using MATLAB 2015a and a Vicon MCS. A step-by-step description of how to communicate a Vicon MCS and Simulink model and deploy control points data from a proportional-integral-derivative (PID) controller to a drone was discussed. The Vicon system has a simple approach to directly send actual variable data to Simulink via user data protocol (UDP). However, unlike Vicon, other popular MCSs, like the Motive OptiTrack, have a different approach to communicate with Simulink. Motive OptiTrack provides a NatNet software development kit (SDK) with a MATLAB function file for streaming data from the MCS to MATLAB. This MATLAB file cannot, however, be used directly on the Simulink platform. Correspondingly, a level-2 s-function was created in [8] based on the NatNet SDK MATLAB function to solve the communication problem between Simulink and OptiTrack MCS.

In this part of the thesis, we will provide a detailed description of the procedures used to develop and operate an indoor multirotor testbed. In particular, we will present the steps taken to set up a testbed based on the AR Drone and an OptiTrack MCS in the Aerospace Robotics Testbed Laboratory (ARTLAB). ARTLAB is an experimental facility in the department of Mechanical Engineering at South Dakota State University. The research activities carried out in ARTLAB mainly focus on robotics, mechatronics, small satellites, nonlinear control and optimal control.

In ARTLAB, the multirotor testbed has eight “Prime 13” OptiTrack cameras. These cameras allow the tracking of real-time position and attitude of a rigid body using a set of retro reflective passive markers, as shown in Figure 3. The data from the cameras are streamed in real-time using Motive optical motion capture software which is a proprietary software platform from OptiTrack. The markers are shown on the Motive software as green dots. A MATLAB function from [8], which will be identified as NatNetsFunction (see Appendix I) throughout this paper, used to retrieve the position and attitude data of a rigid body in real-time from Motive optical motion capture software. A rigid body is created connecting at least three (or more) of those markers. NatNetsFunction is a level-2 MATLAB s-function code used to gather a captured data from Motive OptiTrack and stream it to Simulink at a specified sample time.

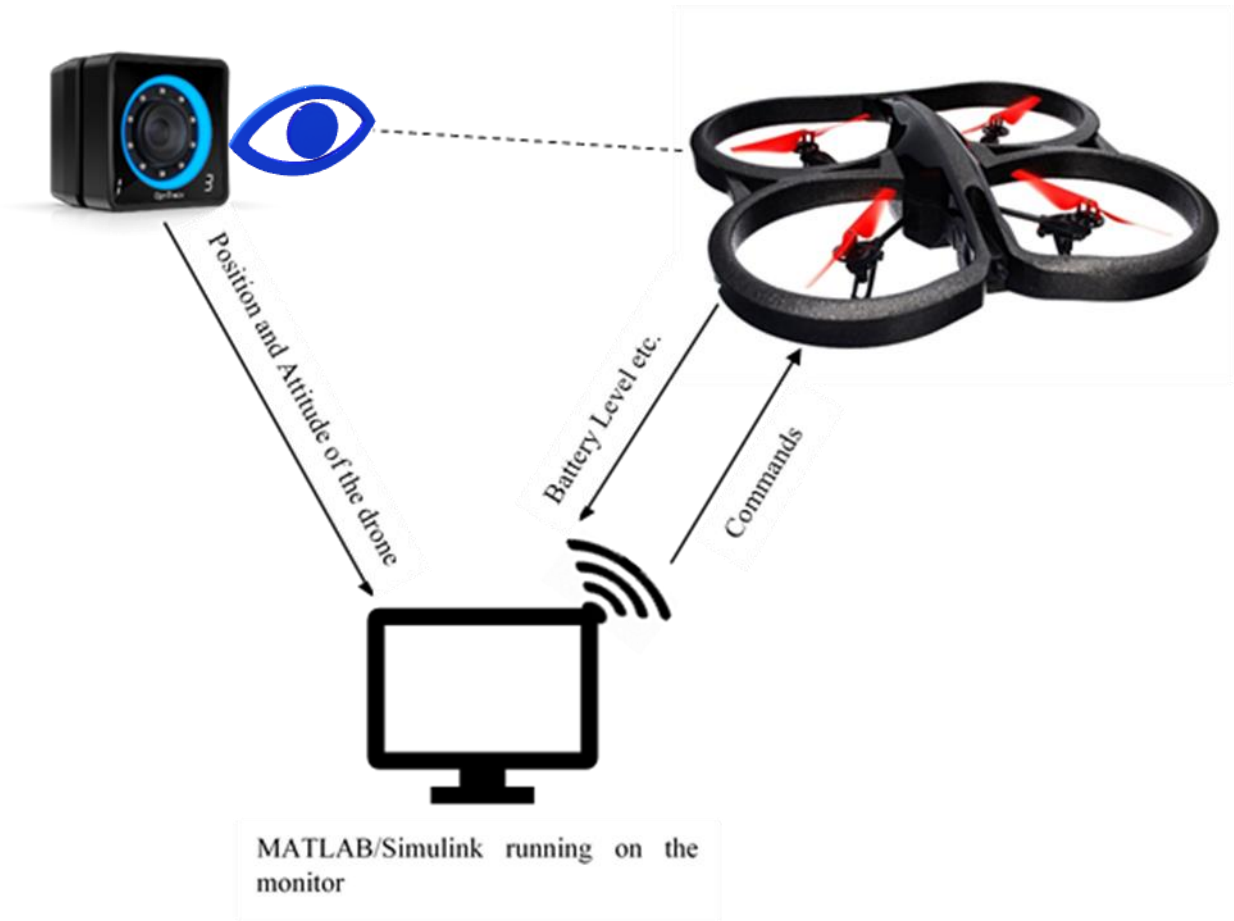

Figure 6. Multirotor testbed structure

The presented testbed is composed by a PC workstation, a set of AR Drone 2.0 Parrot Elite Editions, an OptiTrack MCS based on eight Prime 13 cameras with the following specifications:

- a resolution of 1.3 mega pixel
- frame rate of 240 frame per second and
- field of view of 42°,56°

Sanbria et.al. [9] provided a Simulink model based on the AR Drone 2.0 SDK to establish communication between the Simulink platform and AR Drone 2.0. In a similar context, a MATLAB project from [10] provides a different approach to establishing communication between Simulink and AR Drone 2.0 using an embedded coder. However, it is technically laborious to send, simultaneously, the captured data from the OptiTrack MCS to this Simulink project model and the control variable data from the Simulink model to the drone. Therefore, the former method was implemented on this thesis. To control the parrot, a PID controller from [11] has been improved and redesigned to use it to control the commands.

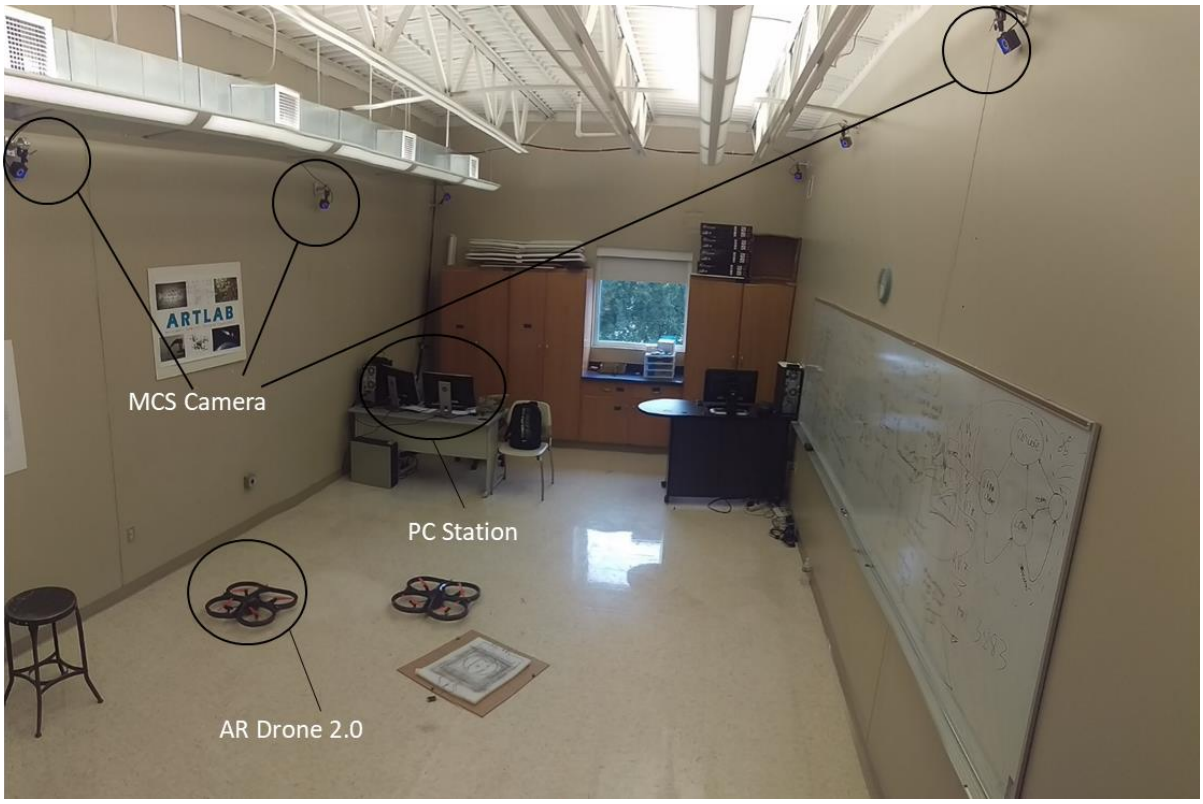

Figure 7. Aerospace Robotics Testbed Laboratory (ARTLAB)

3.1. Streaming data from Motive to Simulink

Motive offers multiple options to stream real-time tracking data onto external applications. There are streaming plugins for Visual3D, Autodesk Motion Builder, Unreal Engine 4, VRPN and NatNet SDK etc. NatNet SDK enables users to build custom clients to receive captured data.

An embedded level 2 s-function, written on the basis of the Motive NatNet SDK MATLAB code, is used to stream captured data from Motive OptiTrack to Simulink as described in the section above. The NatNetsFunction streams the captured actual position

and attitude data, shown in equation (15) and (16) respectively, from Motive to Simulink in real-time with step time of 1msec[1].

$$\boldsymbol{X_s} = [\mathrm{x}_s\ \mathrm{y}_s\ \mathrm{z}_s]^T \tag{15}$$

$$\boldsymbol{\theta_s} = [\theta_s\ \phi_s\ \psi_s]^T \tag{16}$$

This NatNetsFunction requires the same subordinate files that are used to receive data from Motive OptiTrack in the NatNet SDK MATLAB function file [12]. It is highly recommended to put all these files on the same folder path.

Follow these steps to stream data to Simulink:

1. Open NatNetsFunction MATLAB file
2. Define the path for the 'dll' file. To edit the path to the file location, make a change on the MATLAB code line that starts with 'dllPath = fullfile ()'.
3. If streaming of more than one rigid body data is required, make the following changes on NatNetsFunction MATLAB code
    - Set the number of output and input ports to be identical to the number of rigid bodies.
    - Specify the dimension of the input and output ports for each rigid body. The dimension for the input port is 1. The output ports dimension can extend from one to six depending on which state variable required to be streamed. But for the purpose of this project the output ports dimension was defined as six since all the six states variables ($\boldsymbol{X_s}$ and $\boldsymbol{\theta_s}$) were desired.
    - State the data streamed at the output ports for each rigid body (as shown in Appendix I, line 142, for two rigid bodies)
    - Describe frame of data for each rigid body (as shown in Appendix I, line 174)

[1] One millisecond is used for this project, but it can be changed if a different value is required. To make a change on the step time edit the 'block.SampleTimes' on the NatNetsFunction file (as shown in Appendix I, line 30).

- Define position and attitude data to be streamed for each rigid body (Appendix I, line 178 to 185)
- Define quaternion for each rigid body (Appendix I, line 187)

4. Open Motive OptiTrack
5. Follow the following steps to calibrate OptiTrack cameras and set an origin [13]

To calibrate:

- Click on the layout tab found on the top left side of Motive window. Select calibrate from the list shown under layout.
- Click on the camera calibration pane and select a calibration type from the list as shown below

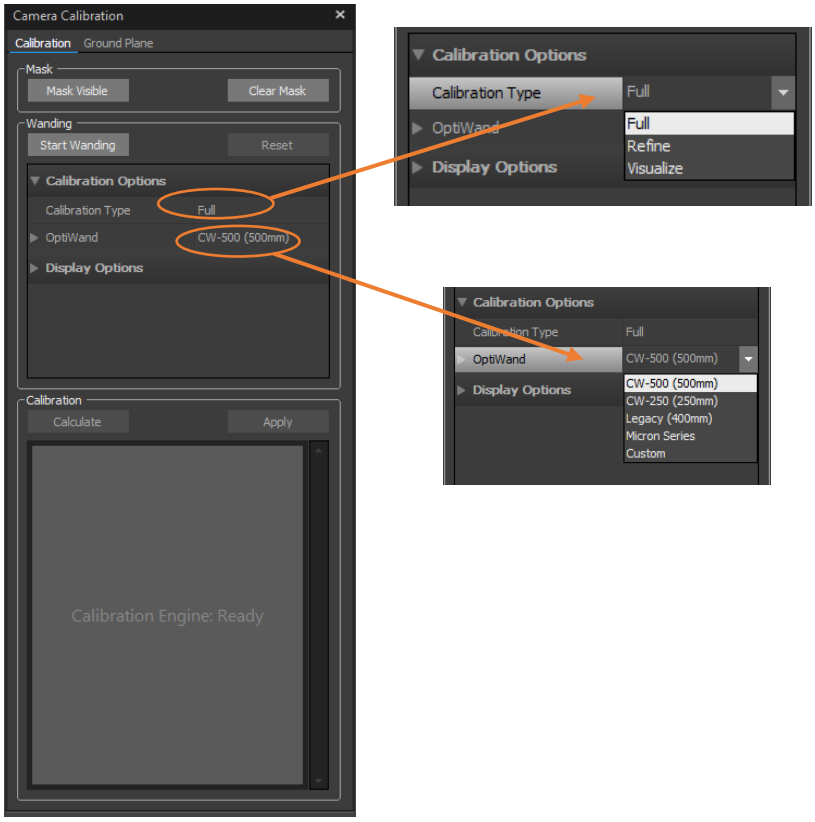

Figure 8. Camera Calibration Pane

- From the OptiWand section list, choose the correct calibration wand name
- Click on start wanding
- Start waving the Calibration wand, shown in Figure 9(a), across the entire capture volume. Each of the cameras light turns to green when enough samples are collected.
- After enough samples are collected, all the cameras lights will turn to green, click on the calculate button
- Save the calibration file

a b

Figure 9. (a) OptiTrack Calibration Wand (b) OptiTrack Calibration Square

To set the origin:

- First place the calibration square, provided from OptiTrack, in the capture area at a specific location, as shown in Figure 9(b).
- Align the calibration square in a desired axis orientation
- The long leg of the calibration square indicates the positive z-axis by default, while the shorter leg indicates the x-axis. The positive y-axis is directed upwards.
- Next adjust the level indicator on the calibration square to balance the calibration square
- Open ground plane on Motive

- Set the vertical offset. The vertical offset is used to compensate the difference between the actual ground plane and the center of markers on calibration square.
- Click on the set ground plane tab
- To further improve the leveling of the coordinate plane, place several markers with a pre-known radius on the ground. Next, adjust the vertical offset for the ground plane refinement with the marker's radius. This function refines the leveling using the marker's position.

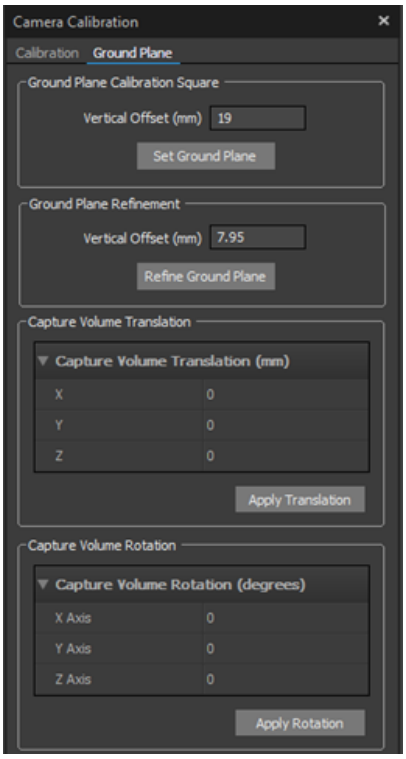

Figure 10. Ground plane pane

6. Create a rigid body from markers as shown in Figure 11. In order to create a rigid body, it is mandatory to link at least three markers together. Motive will automatically assign the geometrical center for each rigid body created.

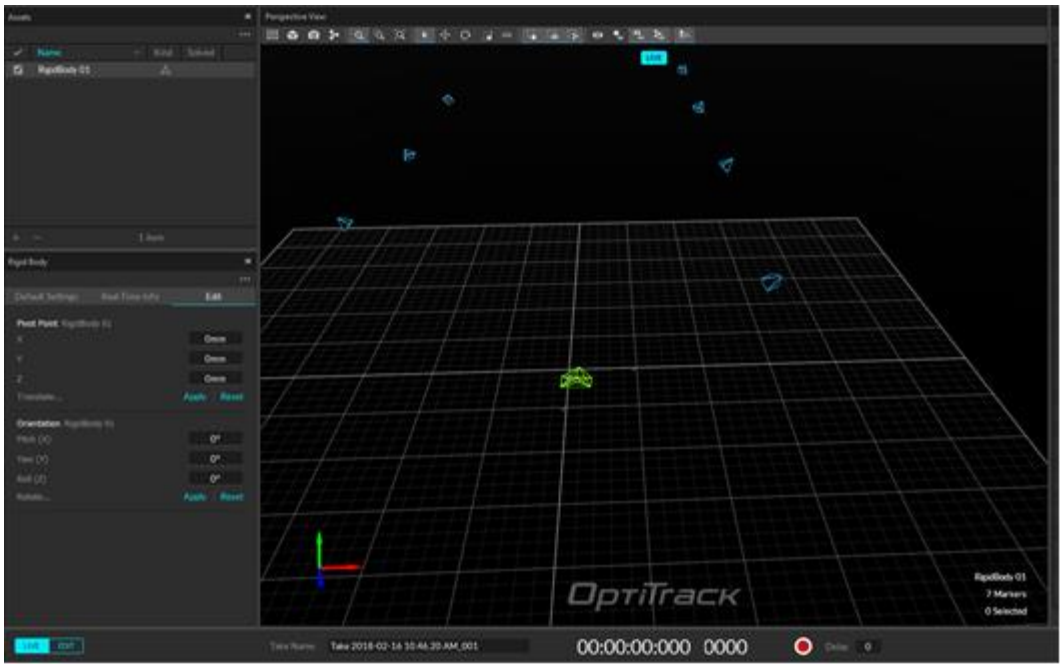

Figure 11. MCS cameras and Rigid body representation in Motive's graphical interface

7. Go to the view tab on the left side of Motive window and select data streaming pane from the list to open the OptiTrack streaming engine.
8. Enable the broadcast frame data on OptiTrack streaming engine as shown in Figure 12 below.
9. Set the local interface to loopback if the streaming occurs in the same computer. Otherwise, select the IP address of the computer in which the streaming should occur.

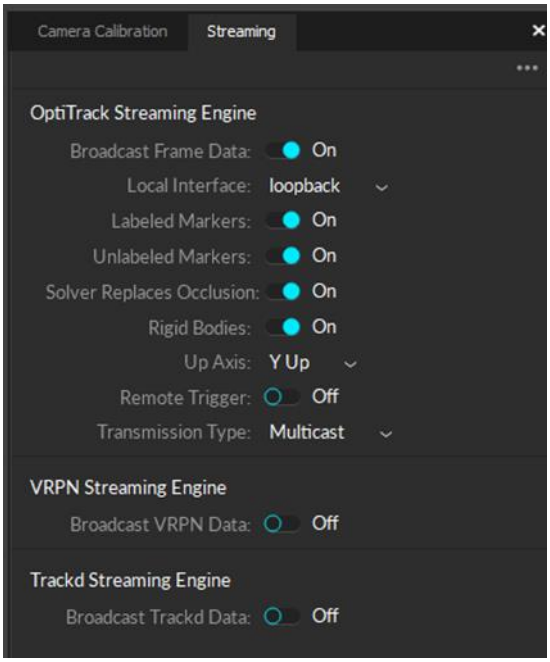

Figure 12. Data Streaming Engine Pane

10. Insert the IP address of the receiving computer on NatNetsFunction or set it to 127.0.0.1 if the same device is used (as shown in Appendix I, line 113).
11. Run the Simulink model corresponding with the NatNetsFunction to acquire the streamed data on Simulink platform.

3.2. Controlling AR Drone from MATLAB-Simulink

The focus of this section of the thesis is to explain in detail the step required to autonomously fly AR Drone using a Simulink-modelled control system. As described in the previous section, NatNetsFunction was created with a level-2 s-function. A model containing level-2 s-function requires a corresponding Target Language Compiler 'TLC' file to build it in Simulink and run it on a target hardware [14].

Two different models were created to address this shortcoming. In particular, these models are

- OptiTrack model
- Controller model

The OptiTrack model ran the NatNetsFunction embedded model in 'normal mode' without the need to build it. The Controller model, on the other hand, can be built without a separate TLC file because the model included in it is not created using a function requiring the TLC file. It is therefore possible to build and run the controller model on the target hardware. The two models will then share data in real time via Simulink Desktop Real Time's (SDRT) UDP communication.

3.2.1. OptiTrack Model

The OptiTrack model consists of the 'From Motion Capture System', 'Trajectory Generation', 'Controller Switch' and 'UDP Sender'. Each model is discussed below in detail.

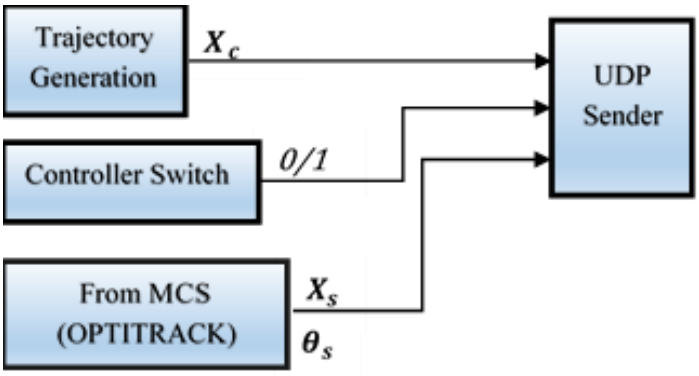

Figure 13. OptiTrack Model

a) From MCS:

This Simulink block is the embedded model of NatNetsFunction. As described in the previous section, this block helps to extract the actual state of the drone from Motive OptiTrack. In order to link the MATLAB s-function file to the Simulink model, the following steps must be followed

- Open s-function block from Simulink library browser.
- Click on the function block
- Write the NatNetsFunction name on the s-function name space.

b) Trajectory Generation:

The trajectory generation block is a reference path that is to be followed by the drone. The output signal from the block is a vector $\boldsymbol{X_c}$ which comprises of the commanded position along the three coordinates axis and a yaw angle.

$$\boldsymbol{X_c} = [\mathrm{x}_c\ \mathrm{y}_c\ \mathrm{z}_c\ \psi_c]^T \tag{17}$$

c) Controller Switch

The control switch is the manual switch which allows us to control the time of transmission of the control data to the drone. By switching it on, the data from the controller block can be transmitted to the drone while switching it off stops the process of sending the data. This manual switch can always be switched on if you need to start sending control data simultaneously with the start of the OptiTrack model.

d) UDP sender

UDP sender conveys all OptiTrack model's output signal data to the controller Simulink model in real time at a specified sample time. To set up the UDP sender port, the following steps must be followed:

- Select UDP input ports from Simulink library browser under Simulink Desktop Real-Time section.
- Double click on the UDP packet output.
- The block parameter will appear as shown in Figure 14.
- Select a board already installed if the list contains one or install a new one.
- Next click on the board setup.
- Set the local UDP port, remote address and remote UDP port to be 127.0.0.1, 36880 and 36884 respectively
- Set the sample time, maximum missed tricks, output packet size and output packet field data types. The output packet size is equal to the number of signals

transferred multiply by eight. The maximum missed trick for the model is the maximum number of missed data that is tolerated.

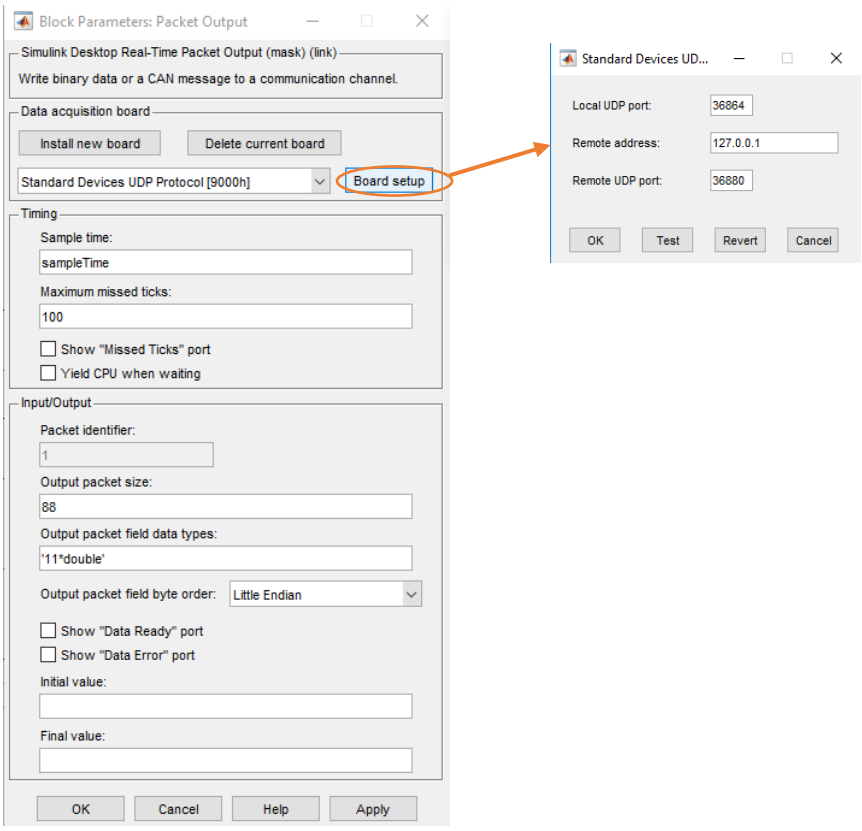

Figure 14. UDP Packet Output

3.2.2. Controller Model

The controller model includes the AR Drone Wi-Fi block (kit model), state observer, controller, UDP receiver and take off/land manual switch. This model runs in ‘external mode’, allowing Simulink to communicate with the model deployed on the drone board during runtime.

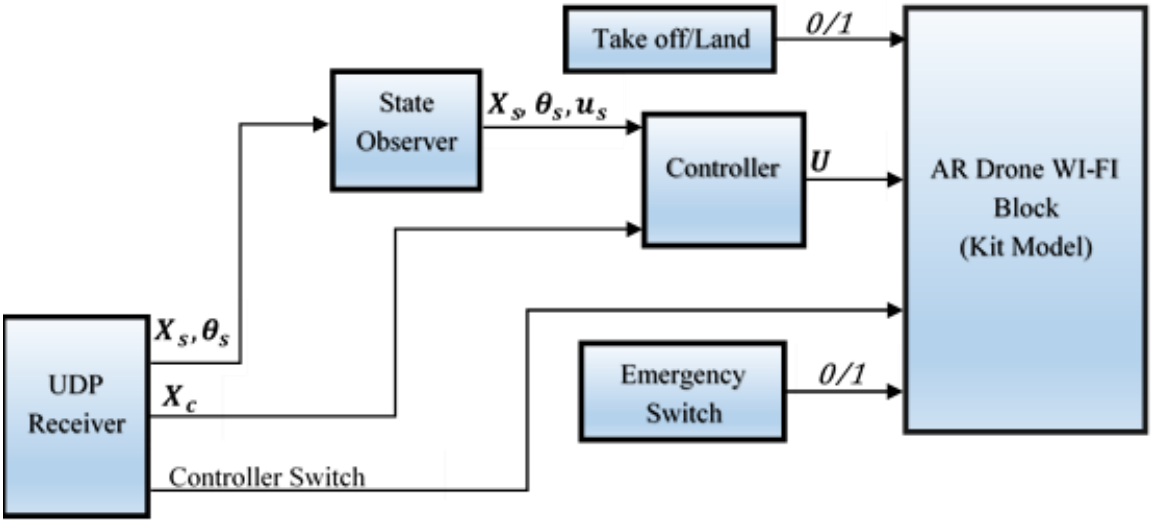

Figure 15. Controller model

a) UDP receiver

UDP receiver receives data from OptiTrack model in real time using the UDP communication technology. The same steps for the UDP sender described above were used to set up the UDP receiver. The IP address used for the UDP receiver (UDP packet input) is similar with UDP sender. However, the local UDP port and the remote UDP port have been swapped for the UDP receiver, respectively 36880 and 36864.

b) AR Drone Wi-Fi Block (Kit Model)

This block was taken from [9]. The kit model decodes and transmits the signal data of the controller in real time from the controller block to the AR Drone. The Wi-Fi block also includes a decoder for data streaming from the drone, such as battery level, etc.

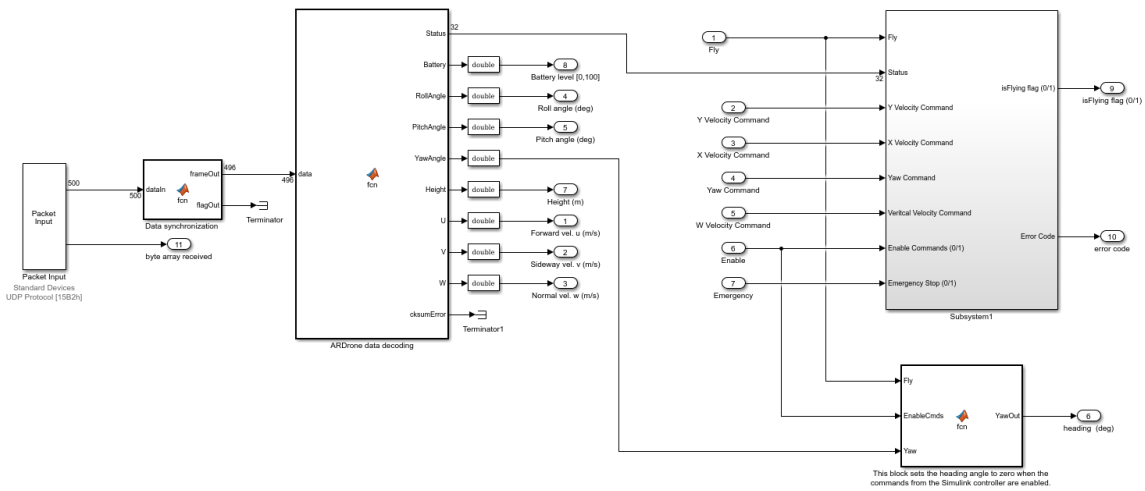

Figure 16. AR Drone Wi-Fi block diagram

c) State Observer

State observer derives the body frame velocity $\boldsymbol{u}_s$ from the components of the time derivative of the position vector, $\dot{\boldsymbol{X}}_s$ and the Euler angles, $\boldsymbol{\theta}_s$ [4].

$$\boldsymbol{u}_s = [u\ v\ w]^T \tag{18}$$

$$\dot{\boldsymbol{X}}_s = [\dot{x}\ \dot{y}\ \dot{z}]^T \tag{19}$$

$$\boldsymbol{\theta}_s = [\theta\ \phi\ \psi]^T \tag{20}$$

This block has three sets of output vectors: the current position $\boldsymbol{X}_s$, the current attitude angles $\boldsymbol{\theta}_s$ and the body frame velocities $\boldsymbol{u}_s$ vector. The body frame velocities are calculated using the rotation matrix from equation (7) as follows.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} cos\theta cos\psi & sin\phi sin\theta cos\psi - cos\phi sin\psi & cos\phi sin\theta cos\psi + sin\phi sin\psi \\ cos\theta sin\psi & cos\phi cos\psi + sin\phi sin\theta sin\psi & cos\phi sin\theta sin\psi - sin\phi cos\psi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

(21)

d) Controller

A controller applies a responsive correction in order to provide an output that exhibit the desired behavior using the feedback signals/state variables. A simple schematic diagram below describes the functionality of a controller.

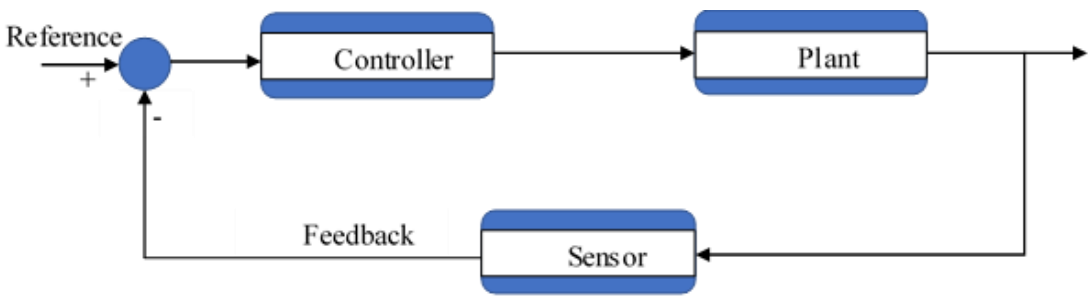

Figure 17. Schematic diagram of a feedback control system

As shown in the diagram above, the controller receives a difference between the feedback signal and the reference value and sends a command signal to the plant. A feedback data is then provided by a sensor that will be subtracted from the reference value and sent to the controller to close the control system loop.

The kit model, which is used to transmit generated controller data to the target drone, requires $U_{\dot{z}}$, $U_{\dot{\psi}}$, U_{φ} and U_{θ} as input variables. Therefore, a PID controller is used to generate these four control variables. The $\boldsymbol{U}$ control output vector is

$$\boldsymbol{U} = \begin{bmatrix} U_{\varphi} \\ U_{\theta} \\ U_{\dot{\psi}} \\ U_{\dot{z}} \end{bmatrix} \tag{22}$$

As described in CHAPTER TWO, the forward movement, along the x-axis, and lateral movement, along the y-axis, are controlled by generating pitch and roll angles, respectively. As a result, the four control components are formulated, in PID fashion, as follows [11]

$$U_{\dot{\psi}} = K_{p,\psi}(\psi_c - \psi_s) + (\psi_c - \psi_s)\frac{K_{i,\psi}}{s} \tag{23}$$

$$U_{zd} = -K_{p,z}(Z_c - Z_s) - (Z_c - Z_s)\frac{K_{i,z}}{s} \quad (24)$$

$$U_{\theta} = -K_{p,t}x_e - K_{d,t}V_{xe} - K_{anglet}\theta_s - \frac{K_{i,t}}{s}x_e \quad (25)$$

$$U_{\phi} = K_{p,f}y_e + K_{d,f}V_{ye} + K_{anglef}\phi_s + \frac{K_{i,f}}{s}y_e \quad (26)$$

where,

$K_{p,\psi}$ and $K_{i,\psi}$ are the proportional and integral gains for the yaw angle respectively.

ψ_c and ψ_s are the commanded and actual values of yaw angle respectively.

$K_{p,z}$ and $K_{i,z}$ are the proportional and integral gains for the altitude respectively.

Z_c and Z_s are the commanded and actual values of the altitude respectively.

$K_{p,f}$, $K_{d,f}$ and $K_{i,f}$ are the proportional, derivation and integral gains for the roll angle respectively.

K_{anglet} is a constant gain applied to minimize the controller error by subtracting a portion of the feedback pitch angle.

K_{anglef} is a constant gain applied to minimize the controller error by subtracting a portion of the feedback roll angle.

$y_e$and V_{ye} are the position and velocity error in the y – axis.

ϕ_s is actual value of the roll angle.

e) Take off/Land

This is a manual switch controlling the take-off and land command for a drone as the name suggests.

The connection between AR. Drone 2.0 and ground station is established via Wi-Fi. A modem/router or a USB Wi-Fi adapter can be used to create a connection with the AR Drone network. For this project a USB Wi-Fi adapter was used.

To fly an AR Drone using the coordination of the two Simulink models, i.e. the Controller model and OptiTrack model, the following steps must be followed:

1. First plug the USB adapter
2. Follow the necessary steps to set up the USB adapter
3. Connect your device (PC) with AR Drone network
4. Open the OptiTrack model and controller model
5. Make sure the NatNetsFunction is on the same path/folder as the rest of the files
6. Run ‘initial variables’ to reset all the variables such as the control gains (shown in Appendix II)
7. Build the Controller model using ctrl + B or clicking the run button on Simulink
8. After the Controller model started running, run the OptiTrack model
9. Switch on the take-off/land to take off drone
10. Switch on the Controller switch (enable reference switch) to start sending the controller data. Switching off the enable reference will stop the data sending process
11. Switch off the take-off/land switch to land the Parrot

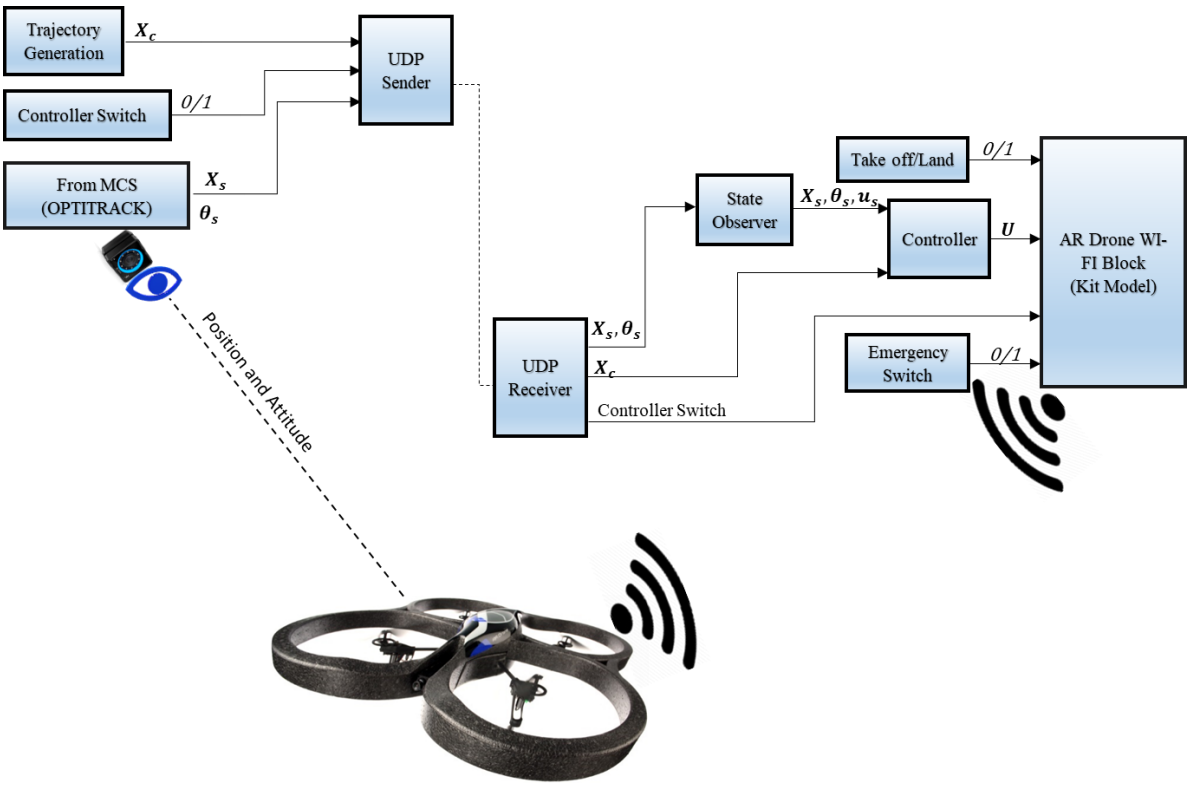

Figure 18. Overall Simulink diagram structure with MCS

CHAPTER FOUR

ACCURATE LANDING AND FORMATION FLIGHT

As mentioned earlier, this chapter briefly discusses the two-benchmark multirotor GNC problems. The first study tackled accurate landing guidance strategies. An emulated autonomous battery recharging experiment was conducted using a mock-up battery charging platform as a proof of concept for autonomous battery charging capability. Next, the formation flight between two multirotors is studied. The main goal behind performing the formation flight is to assess the accuracy of the controller to track, accurately, a desired moving target position.

The idea and interest on autonomous charging robotics vehicles started mid-20th century. As the applications of UAVs increased significantly, there is a rise in need of an extended flight duration to accomplish a mission that requires a longer flight duration. To address this problem, several research studies have now been carried out on an autonomous charging system. In [15] researchers at the Massachusetts Institute of Technology were able to perform autonomous landing and recharging of batteries of X-UFO and Draganflyer drones using a recharging station.

Some researchers have worked with the Wireless Power Transfer (WPT) technique to address this problem. WPT is a power transmission technology in which electrical energy can be transmitted without a wire connection. In [16] and [17] WPT technology is used to autonomously charge a multi-copter. The authors of [16] were able to wirelessly charge AR Drone with an average WTP efficiency of 75%. One advantage of WTP is that it does not require a very accurate approach to landing which makes it more valuable in this regard than a docking strategy. The docking method, however, requires having an accurate control system with a small error margin. Although the WTP reduces the need for a highly accurate control system for the landing operation in contrast to the docking technique, it still faces

a great deal of drawback on their efficiency. The author in [16] calculated the WTP efficiency as

$$E = \frac{V_0 I_0}{V_i I_i}\% \tag{27}$$

where, V_0 and V_i are the output and input voltage while I_0 and I_i are the output and input current.

In addition to the above research, authors of [18] used the cameras available on the drone at the front and at the bottom to navigate through and make an autonomous landing. Similarly, Carriera in [19] used a vision-based target localization to autonomously execute landing operation. These papers, however, were not concerned with actually charging a drone.

In the following sections, I will be describing the accurate landing strategies implemented in ARTLAB.

4.1. Accurate Landing

A strategy is applied in this work to achieve an accurate drone landing. The first step in the work of autonomously charging drones is to navigate where the charging platform is located. Then, the drone tracks the position of a designated platform and lands on the docking station safely and accurately.

The autonomous accurate landing of a drone is divided into the following three phases, as shown in Figure 19.

Phase 1. Take-off
Phase 2. Closing
Phase 3. Landing/Docking

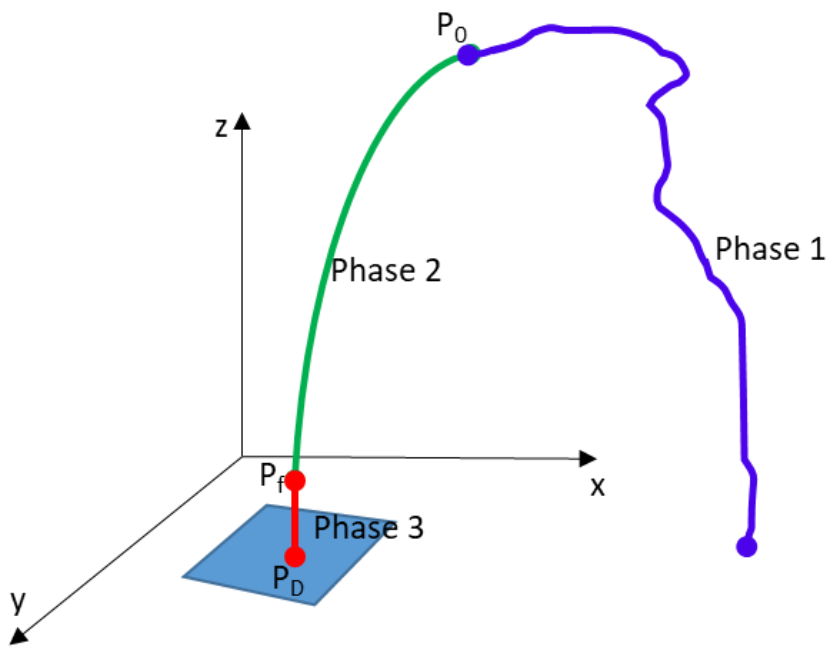

Figure 19. Three phases of accurate landing

4.1.1. Phase 1: Take-off

In this stage of the flight, the drone takes-off and hovers for a few seconds until the controller command is initiated. AR Drones are equipped with autopilot system which allows the aerial vehicles to have an autonomous take-off and hover in the air. For the purpose of this paper, the duration time set for the drone to hover before the second phase, i.e. closing phase, was 7 secs. Based on multiple experimental tests conducted, such duration was found sufficient for the drone to take-off and had stabilized hovering.

4.1.2. Phase 2: Closing

In phase 2, the drone closes to the docking configuration following a predefined trajectory. Such time trajectory was generated to control the speed at which the drone approaches the last point. In particular, it is assumed that the trajectory components can be described as fourth degree polynomials. The polynomial coefficients are calculated by imposing the assigned initial and final conditions on position, and velocity and final acceleration.

The general equation of the trajectory is given by,

$$\boldsymbol{X}(t) = \boldsymbol{A_0} + \boldsymbol{A_1}t + \boldsymbol{A_2}t^2 + \boldsymbol{A_3}t^3 + \boldsymbol{A_4}t^4 \tag{28}$$

where, $\boldsymbol{X} = [x\ y\ z]^T$ and $\boldsymbol{A_i} = [a_i\ b_i\ c_i]^T$ with $i = 0, \dots, 4$

As mentioned before, the coefficients $\boldsymbol{A_i}$ are determined by imposing different boundary conditions at initial[2] time and final time. The exact position and speed of the drone at the beginning of the closing phase or at the end of the take-off phase, which is taken from the MCS, was therefore taken as the initial position and speed of the vehicle.

The final positions on the X and Y axes were set to the X and Y coordinates of the landing platform respectively. The final altitude of the drone in the z coordinate axis was set to be the height of the docking stage from the ground plus 0.15 meters. The reason for the gap distance of 0.15 meters between the flying drone and the landing platform was to allow the drone to hover and stabilize safely before landing. The disturbing ground effects on the drone flight would increase dramatically at heights less than 0.15 meters. At the end of the closing stage, the acceleration of the drone shall be equal to zero.

$$\boldsymbol{X}(t_0) = \boldsymbol{X_0} \tag{29}$$

$$\boldsymbol{X}(t_f) = \boldsymbol{X_f} \tag{30}$$

$$\dot{\boldsymbol{X}}(t_f) = 0 \tag{31}$$

$$\dot{\boldsymbol{X}}(t_0) = \dot{\boldsymbol{X}}_{\boldsymbol{0}} \tag{32}$$

$$\ddot{\boldsymbol{X}}(t_f) = \boldsymbol{0} \tag{33}$$

In turn, by setting the above boundary conditions and solving for the coefficients, these unknown variables can be represented in terms of known parameters as follows.

[2] Note that the time considered as initial here is the seventh second which is the end of the first phase and the start of the second phase. Hence t means t_c-t_s, if t_c is the current time and t_s = 7sec.

For the x- axis coefficients:

Coefficients	Expressions
a_0	x_0
a_1	$\dot{x}_0$
a_2	$\frac{-3(2x_0 - 2x_f + t_f\dot{x}_0)}{t^2{}_f}$
a_3	$\frac{8(x_0 - x_f) + 3t_f\dot{x}_0}{t^3{}_f}$
a_4	$\frac{-(3x_0 - 3x_f + t_f\dot{x}_0)}{t^4{}_f}$

Table 1 Expression for the coefficients of x- axis

For the y- axis coefficients:

Coefficients	Expressions
b_0	y_0
b_1	y_{d0}
b_2	$\frac{-3(2y_0 - 2y_f + t_f\dot{y}_0)}{t^2{}_f}$
b_3	$\frac{8(y_0 - y_f) + 3t_f\dot{y}_0}{t^3{}_f}$
b_4	$\frac{-(3y_0 - 3y_f + t_f\dot{y}_0)}{t^4{}_f}$

Table 2 Expression for the coefficients of y- axis

For the z axis coefficients:

Coefficients	Expressions
c_0	z_0
c_1	z_{d0}
c_2	$\frac{-3(2z_0 - 2z_f + t_f \dot{z}_0)}{t^2{}_f}$
c_3	$\frac{8(z_0 - z_f) + 3t_f \dot{z}_0}{t^3{}_f}$
c_4	$\frac{-(3z_0 - 3z_f + t_f \dot{z}_0)}{t^4{}_f}$

Table 3 Expression for the coefficients of z- axis

The total time duration required for the drone to complete the maneuver was calculated based on the empirical observation that $t = 12.5$ seconds is a reasonable amount of time for the drone to cover 1 meter in translation. Based on the same assumption an average velocity was calculated as

$$V_{avg} = \frac{\Delta s}{\Delta t} = 0.08\ m/s \tag{34}$$

Then, the time required for any specific flight could be retrieved by calculating the distance from the initial to the final position and dividing it by the average velocity.

$$\Delta t = \frac{\|\boldsymbol{X_f} - \boldsymbol{X_0}\|}{V_{avg}} \tag{35}$$

4.1.3. Phase 3: Docking/Landing

Phase 3 is the last phase of the accurate landing. When the drone closes to the final landing area, it must hover above the landing area until a docking configuration is achieved. As described in the above section, the main goal of the accurate landing experimental

campaign is to dock on a mock-up landing stage. As shown in Figure 20, a set of wires passes through the rectangular plate which connects to a LED. As the drone lands on the platform, a cable placed underneath will close the circuit and the LED will light up to demonstrate that it landed with the required accuracy.

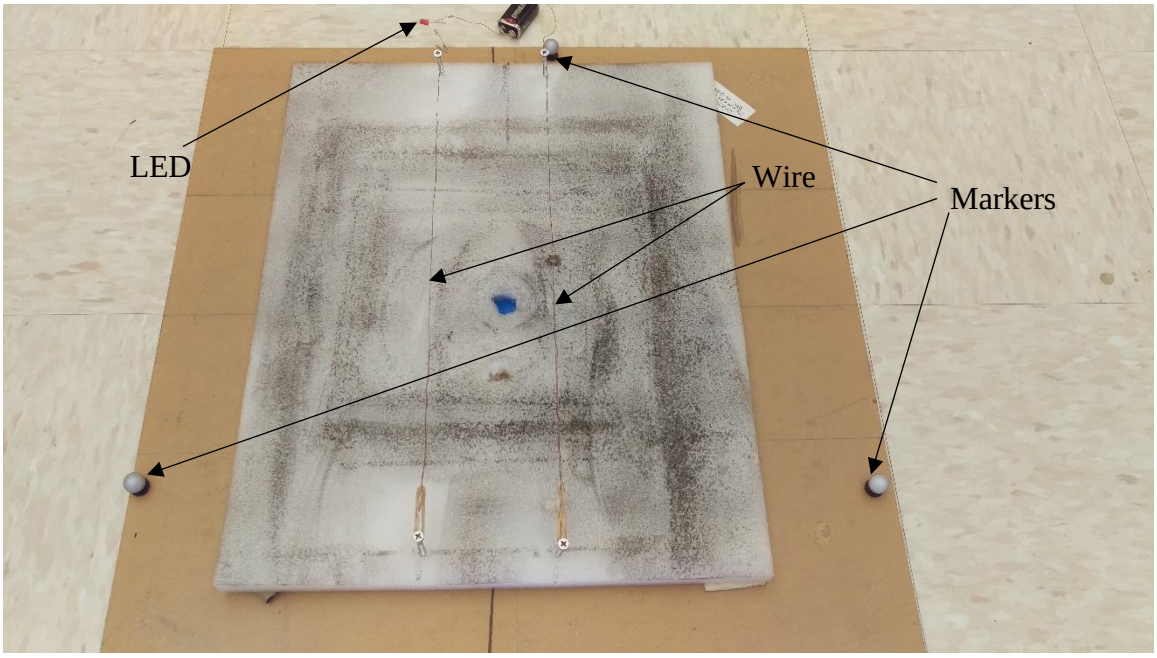

Figure 20. Rectangular landing stage

In order to increase the landing accuracy and to boost the success rate, the drone's landing logic is established by extrapolating the current critical state variables for a correct landing, namely distance to target and near zero speed. This approach allows to slightly anticipate favorable landing condition instead of delaying the rotor shut-off to land off of the docking stage. This theory is described in a Figure 21 below. Let assume the drone is flying towards the green dot and presently it is at the blue dot position. If the drone position is correctly extrapolated and predicted when the drone was at the blue point, the time it took the drone to travel from the blue to the green point would make up for the time it took the drone to meet the landing criteria and land safely. However, if the drone was made to land when it arrived at the green point, the drone could be moved to the red dot point during the time gap between meeting the logic and landing.

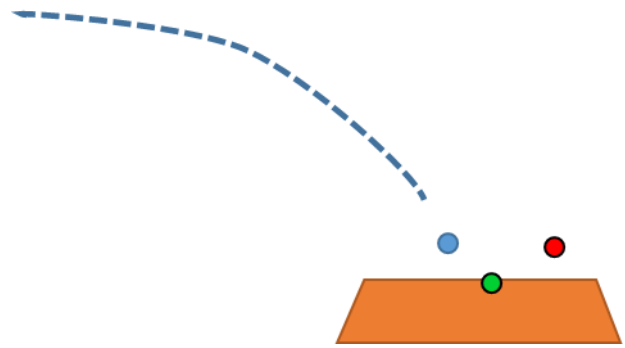

Figure 21. Demonstration of why extrapolation was used

The extrapolation equation employed for the position vector $\boldsymbol{X}$ and velocity vector $\dot{\boldsymbol{X}}$ to calculate the values at time t_{i+1} is given by

$$\boldsymbol{X}(t_{i+1}) = \boldsymbol{X}(t_i) + \frac{(\boldsymbol{X}(t_i) - \boldsymbol{X}(t_{i-1}))}{t_i - t_{i-1}} * K \tag{36}$$

$$\dot{\boldsymbol{X}}(t_{i+1}) = \dot{\boldsymbol{X}}(t_i) + \frac{(\dot{\boldsymbol{X}}(t_i) - \dot{\boldsymbol{X}}(t_{i-1}))}{t_i - t_{i-1}} * K \tag{37}$$

where, t_{i-1} and t_i are the previous and present time instants respectively and K is a compensation constant to be tuned during the experimental campaign.

By changing the constant K, the extrapolation values which gave a satisfactory result are adjusted. A separate logic was applied independently for the planar and the vertical motion. Changing the variables, values that gave a decent result were determined.

$$(x(t_{i+1}) - x_c)^2 < {\varepsilon_x}^2 \tag{38}$$

$$(y(t_{i+1}) - y_c)^2 < {\varepsilon_y}^2 \tag{39}$$

$$z(t_{i+1}) < z_c + \varepsilon_z \tag{40}$$

$$\dot{x}(t_{i+1})^2 + \dot{y}(t_{i+1})^2 < {\varepsilon_{\dot{x}\dot{y}}}^2 \tag{41}$$

$$\dot{z}(t_{i+1}) < \varepsilon_{\dot{z}} \tag{42}$$

where,

- x_c, y_c and z_c are the commanded values

- x, y, z and $\dot{x}, \dot{y}, \dot{z}$ are the position and velocity at t_{i+1}on the x, y and z coordinates.
- $\varepsilon_x, \varepsilon_y$ and ε_z are the error tolerated for the position in each three coordinates.
- $\varepsilon_{\dot{x}\dot{y}}$ and $\varepsilon_{\dot{z}}$ are the error tolerated for the velocity.

Tests proved that satisfactory results are achieved with

K	ε_x	ε_y	ε_z	$\varepsilon_{\dot{x}\dot{y}}$	$\varepsilon_{\dot{z}}$
0.2	0.03	0.03	0.05	0.04	0.05

Table 4 Accurate landing values

Figure 22 below shows the commanded and actual trajectory for the three coordinates and the yaw angle. It is important to note that the initial seven seconds of the maneuver are actually used to perform the take-off. The actual commanded trajectory begins right after the seventh second. As described in the previous section, this is the duration of time set until the control data allowed to be streamed.

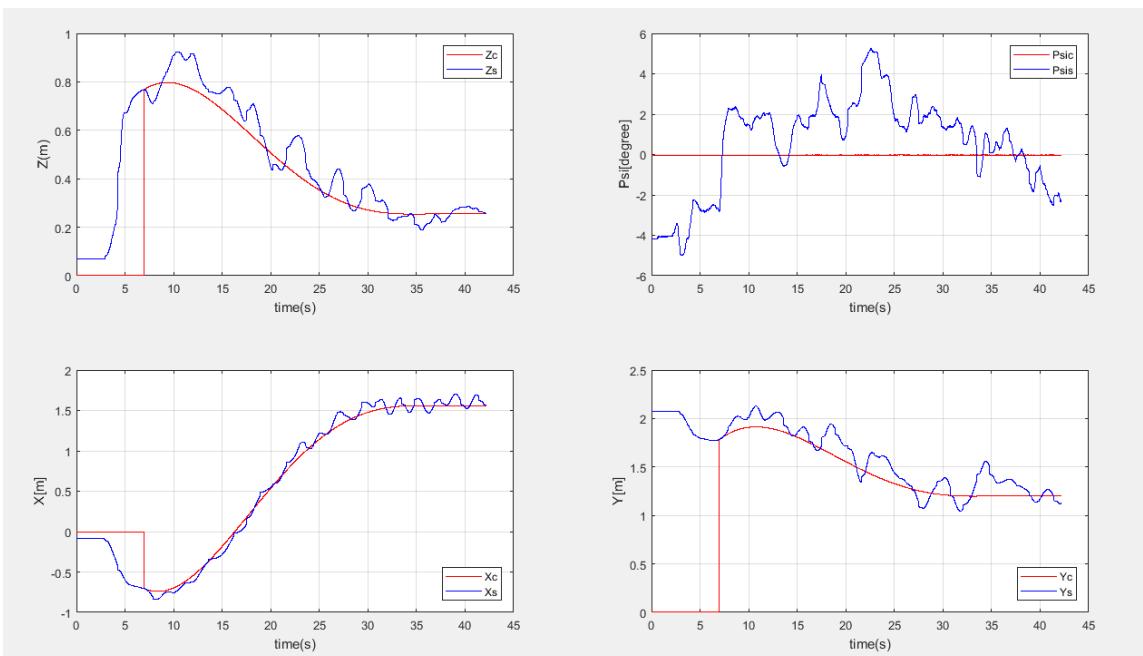

Figure 22. Position plot on the x, y and z and yaw angle plot

The 3D plot of the trajectory is shown in the following Figure 23.

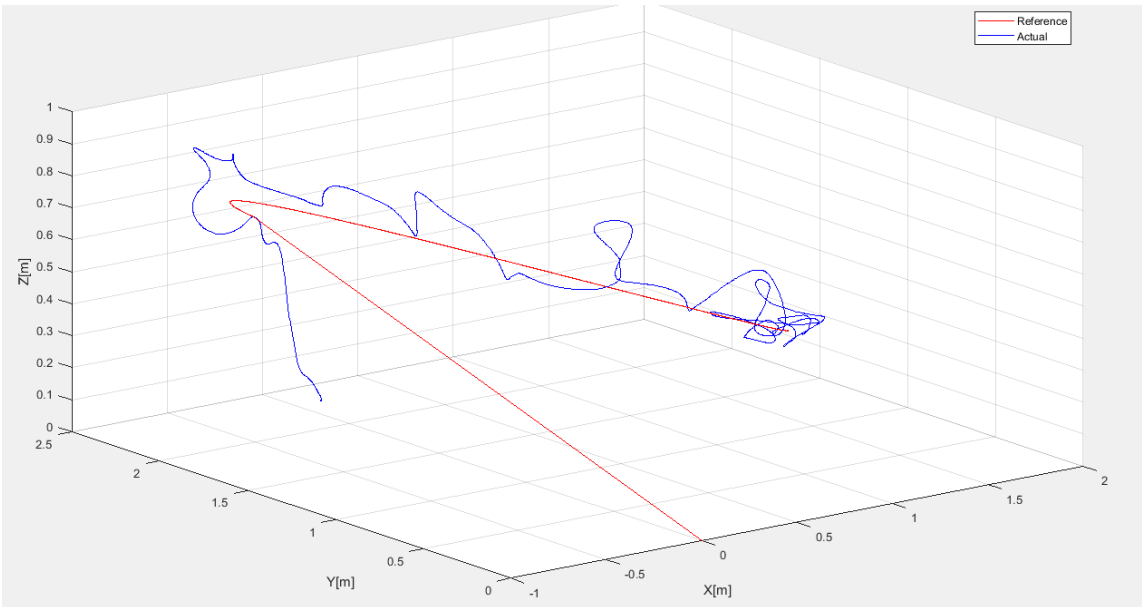

Figure 23. 3D plot of the accurate landing path

The histories of the control variables, $U_{\dot{z}}$, U_{θ} and U_{φ}, and the histories of actual vertical velocity, pitch and roll angles, are reported in Figure 24 below.

When I have introduced the OptiTrack and controller model in section 3.2.2, it was stated that the controller model should run first before the OptiTrack model. This procedure is reflected into a time gap shown in the Figure 24 below. In other words, it is the time lost before the OptiTrack model data begins to be sent to the controller model, which was caused by the time it took to run and compile the OptiTrack model[3] and the drones to start flying.

[3] OptiTrack model is run and compiled after the controller model was run and compiled

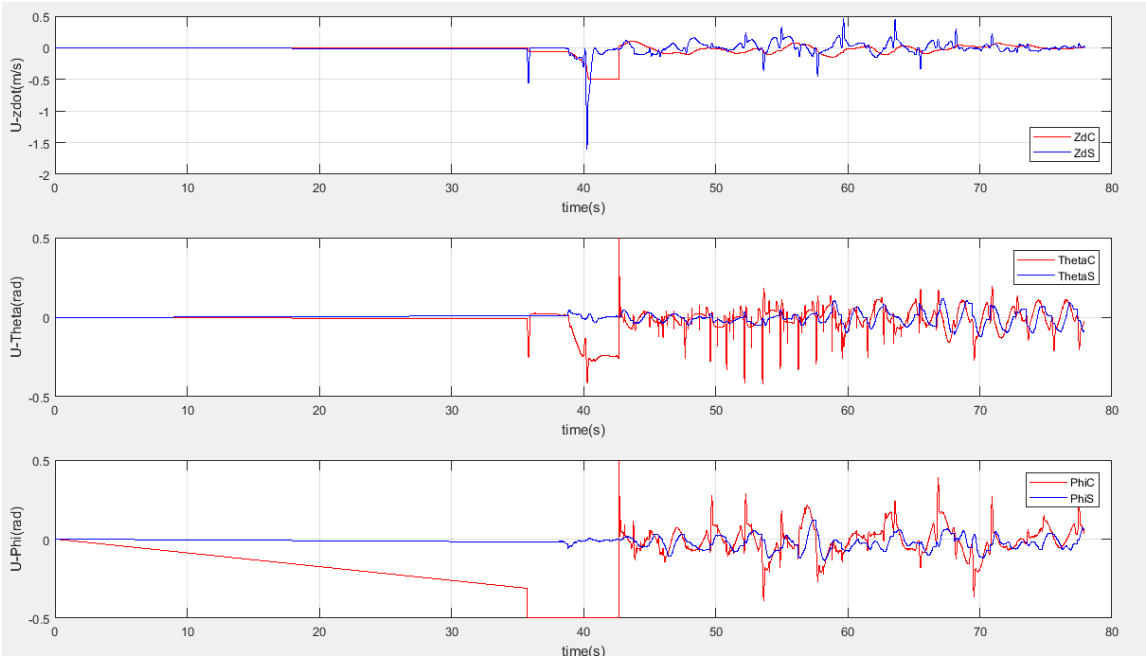

Figure 24. The control outputs plot versus their feedback values

4.2. Formation Flight

Another topic in this thesis paper is the autonomous formation flight of two mutirotors. Different research groups worked on drone coordination flight for multiple drones to accomplish different and articulated tasks, such as rescue operation, structures inspection, fire control missions, etc.

In this paper, with the main objective of testing the accuracy of the controller, two are requested to execute a formation flight to achieve a steady alignment. To perform such maneuver the two vehicles will behave simultaneously as target and chaser. As shown in the following Figure 25, the two drones start in two different positions, drone 1 from P_{1_o} and drone 2 from P_{2_o}. After takeoff each vehicle will start to chase the other vehicle and reach the final points of P_{1f} and P_{2f}. The two final points of the drones separated by a constant value of 2D in the X axis.

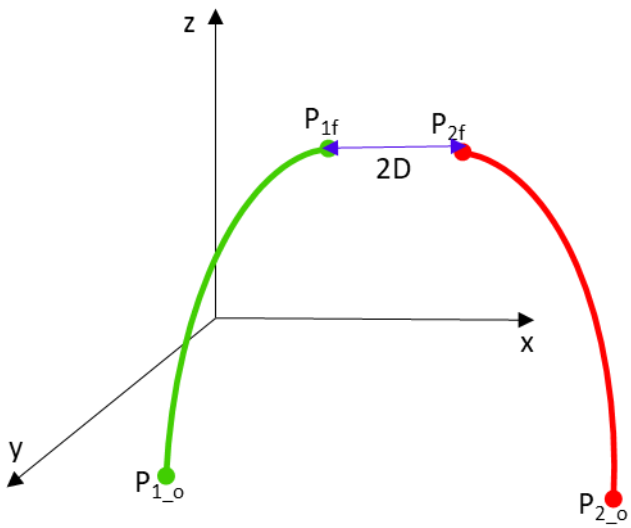

Figure 25. Hovering of two drones in a coordination flight

One of the challenges faced in the coordination of the two drones was that the drones continued to fly together instead of stabilizing and hovering together at a point fixed to the absolute frame of reference. As a result, the two vehicles will be prone to simultaneously drift toward some direction. Various techniques were used to tackle this problem. The one that delivered satisfactory results is discussed here.

For each coordinate, a simple equation was used in which each drone target position is the sum of its current position plus a portion of the relative position with respect to the other vehicle so that the gap between them continues to be minimized and converge toward the final assigned distance 2D. This can be further illustrated on the $x - y$ plane in the Figure 26 below. The first drone moves from the blue dot to the black dot position adding a percentage of the difference between the initial position of the first and second drone. At the same time, the second drone also adds a percentage of the distance between the first positions of the two drones to reach at the black dot position. This process continues until both drones reach at the same point.

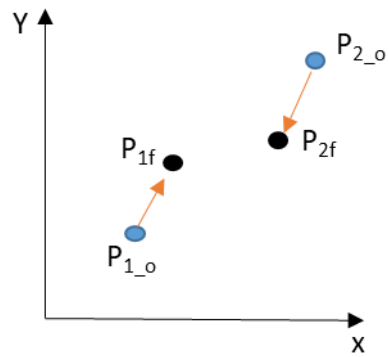

Figure 26. Illustration of coordination equation used

The illustration above can be mathematically described as follow. Let dt be the maneuver time interval, then the target location of drone 1 is

$$x_1(t+dt) = (x_2(t) - x_1(t))k_x + x_1(t) - D \tag{43}$$

$$y_1(t+dt) = (y_2(t) - y_1(t))k_y + y_1(t) \tag{44}$$

$$z_1(t+dt) = (z_2(t) - z_1(t))k_z + z_1(t) \tag{45}$$

Similarly, for the second drone

$$x_2(t+dt) = (x_1(t) - x_2(t))k_x + x_2(t) + D \tag{46}$$

$$y_2(t+dt) = (y_1(t) - y_2(t))k_y + y_2(t) \tag{47}$$

$$z_2(t+dt) = (z_1(t) - z_2(t))k_z + z_2(t) \tag{48}$$

As mentioned above, the value of x, y and z values at $(t+dt)$ is the current position of either drone plus a constant (k_x or k_y or k_z) multiplying the difference between the two drones. The constants k_x, k_y and k_z are compensation constants for the components x, y and z, respectively, which can be tuned experimentally to achieve satisfactory performances.

Based on experimental campaign results the following values have been selected

k_z	k_y	k_x	$D\ [m]$
0.7	0.4	0.4	0.6

Table 5 Formation flight results

In Figure 27 are shown the results for the formation flight in each axis. In particular, the three figures report the commanded and actual position component histories for both drones. Separately, the time histories of the position components are represented in Figure 28. Similarly, the plots for the second drone are shown in Figure 29.

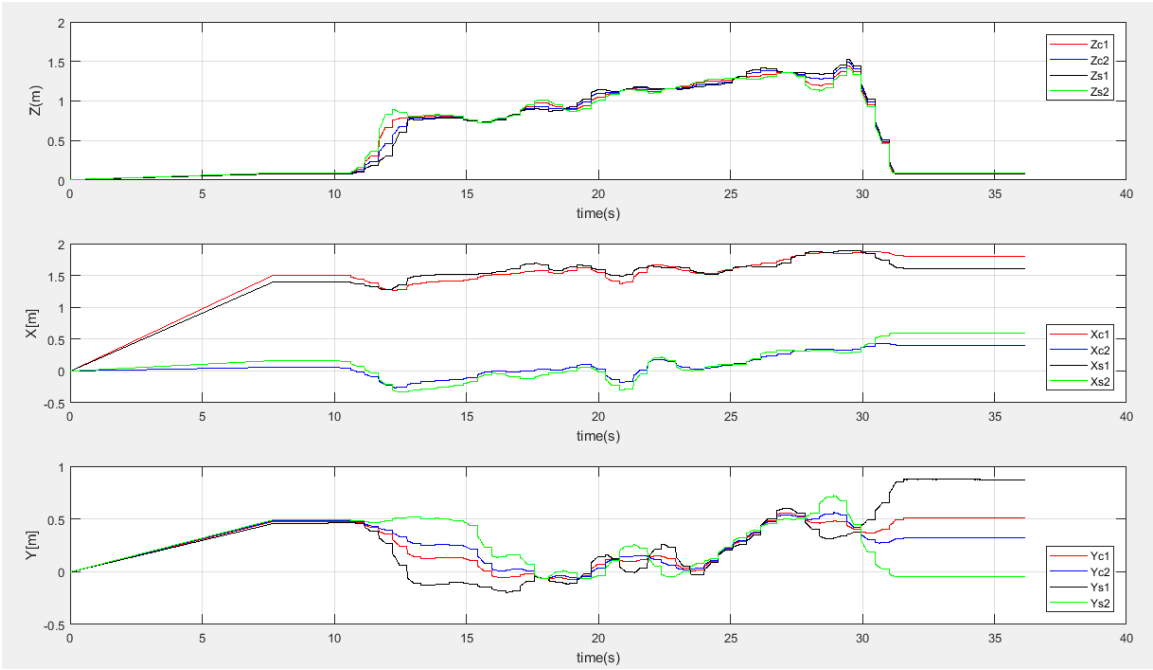

Figure 27. Formation flight plot

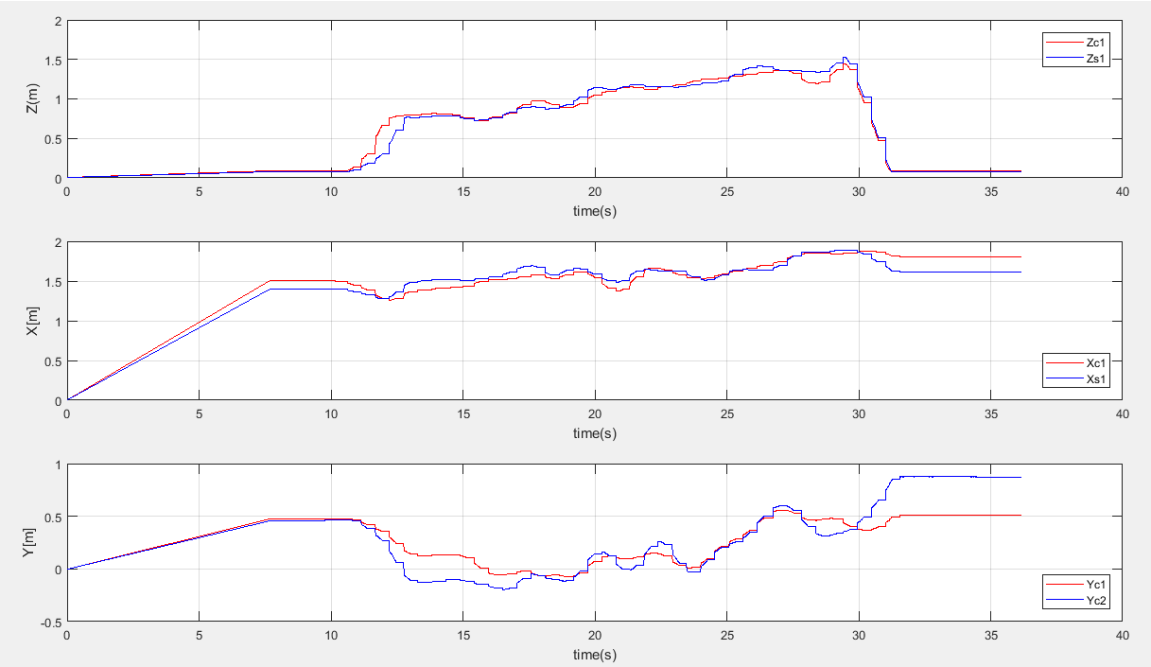

Figure 28. Drone 1 plot in the x, y and z direction

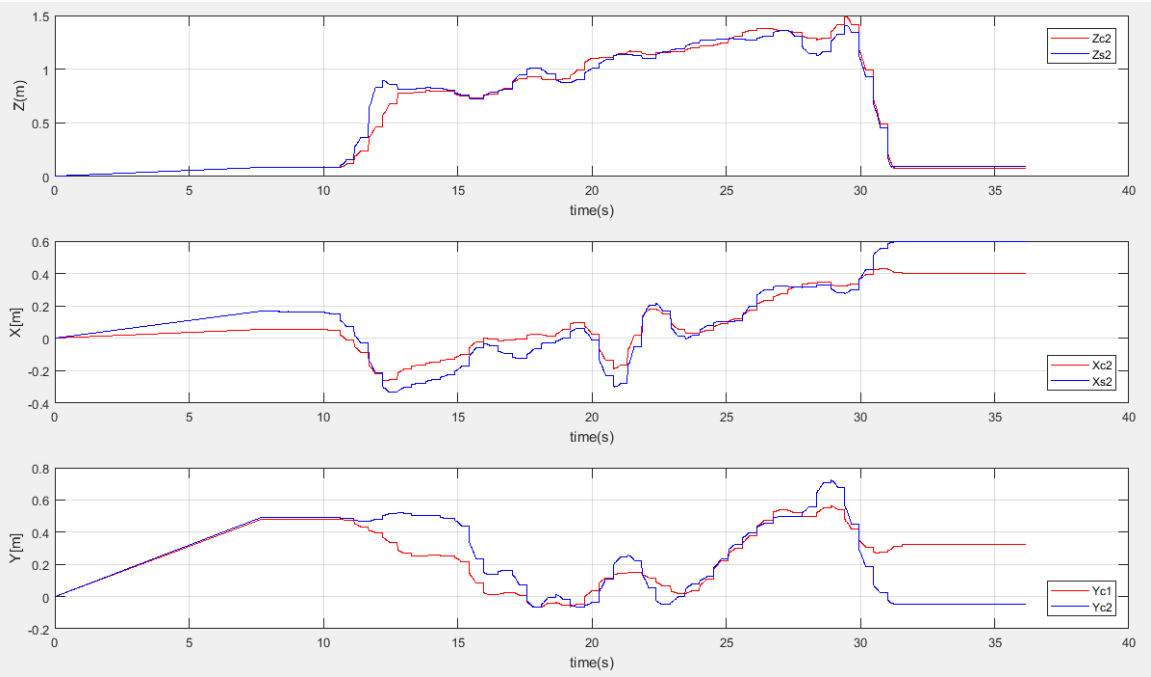

Figure 29. Drone 2 plot in the x, y and z direction

CHAPTER FIVE

COMPARISON OF NLP SOLVERS

In the last decades, there has been a growing interest towards fast numerical methods to solve nonlinear programming problems (NLP), namely, to find the minimum of a scalar function $f(x)$ subject to a constraint $\emptyset(x) = 0$ [20]. This particular interest reflects into a growing demand of advanced NLP solver methods capable of solving large-scale optimization problems. A variety of nonlinear programming solvers are currently available to address optimization problems, including ARTELYS KNITRO, IPOPT, MIDACO, Dlib, MATLAB-FMINCON, and SGRA.

Several of these NLP solvers are commercial products and are hardly available online as free source software. However, there are a few that can be easily downloaded. Some have a free trial version for a specific time allotment. The software for NLP solvers is often written in popular computer programming languages such as C or Python. A few of the software can be integrated and executed in MATLAB as well.

In this paper the performance of Sequential Gradient-Restoration Algorithm (SGRA) was compared with a variety of other NLP solvers that can be executed in MATLAB. The testing was based on performance comparison using various benchmark problems. There are very few research publications that worked on a similar topic. In [20] the author compares the properties of multiple algorithms based on their computational time and ability to converge to the solution using 16 numerical examples as a test problem. Similarly, using benchmark problems containing as many as twenty variables, the performance of eight different optimization methods were evaluated in [21]. However, in that particular paper the optimization method discussed are applied to unconstrained optimization problems.

Six different NLP solvers were considered in this thesis based on online availability and MATLAB compatibility. The comparison was done using a collection of 13 different benchmark test problems retrieved from [20] and [22]. The comparison was then based on

the reliability and efficiency of each solver, measured by the time the calculation took to reach a certain level of precision.

5.1. Problem Statement

Nonlinear programming is a particular branch of optimal control theory and it is based on the minimization of a scalar objective function subjected to a set of inequality constraints. The general form of nonlinear programming problem can be stated as:

$$\text{Min } f = f(\boldsymbol{x}) \tag{49}$$

subject to:

$$\emptyset(\boldsymbol{x}) \geq 0$$

with $f: R^n \to R$ and $\emptyset: R^n \to R^q$ nonlinear functions and $n > q$, for a bonafide optimization problem. The vector $\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T$ contains all the optimization variables.

5.2. Nonlinear Programming Solver

As mentioned in the first part of this chapter, there are many commercially available NLP solvers. Nevertheless, only a few can be freely downloaded and used. In addition, not all of them supports MATLAB which is a software platform used in this paper to run the NLP optimization approaches.

The NLP solvers discussed in this paper are listed below.

I. SGRA (Sequential Gradient-Restoration Algorithm) is a nonlinear programming solver algorithm composed of a sequence of gradient phases and restoration phases. I coded SGRA using MATLAB software.

II. ARTELYS KNITRO[4] is a commercially available software package for solving nonlinear optimization problems that has developed since 2001 by Zienna

[4] Since the trial version of this software was used in this paper the performance might be different than the full version

Optimization. The solver name KNITRO is a short form for 'nonlinear interior point trust region optimization'. KNITRO has presented an interface to use the software in MATLAB. KNITRO can be used using two different algorithms.

a. Interior-point (IP)
b. Active-set (AS)

The trial version of the software is used in this paper.

III. MMA (Method of Moving Asymptotes) is a freely available sequential convex approximations which was coded on MATLAB. This algorithm was taken from [23] and takes into account a problem of form optimization:

$$\text{Minimize } f_0(\boldsymbol{x}) + k_0 z + \sum_{j=1}^{n}(a_j y_j + \tfrac{1}{2} b_j {y_j}^2) \tag{50}$$
$$\text{subject to } f_j(\boldsymbol{x}) - k_j z - y_j \leq 0, \text{ for each } \; j = 1, \dots, n$$
$$\boldsymbol{x} = (x_1, x_2, \dots, x_m)^T \in \{R^m \;\&\; {x_j}^{min} \leq x_j \leq {x_j}^{max}\}, j = 1, \dots, m\,,$$
$$\boldsymbol{y} = (y_1, y_2, \dots, y_n)^T \geq 0, z \geq 0$$

k_0, a_j, b_j and $k_j$are real numbers given that satisfy $k_0 > 0, a_j \geq 0, b_j \geq 0, k_j \geq 0$ and $a_j + b_j > 0$ for all j. Also, $k_j a_j > k_0$ for all j with $k_j > 0$.

The following adjustment was made to make this problem equivalent to the form shown in equation (49) and to use this algorithm to solve it.

- Let $k_0 = 1$ and $k_j = 0$ for all $j > 0$
- $z = 0$ in any optimal solution
- Let $b_j = 1$ and a_j equals a large number, so that the value of $y_j$becomes insignificant for each j
- Then $\boldsymbol{y} = 0$ in any optimal solution of equation (50) and the respective $\boldsymbol{x}$ is an optimal solution for the problem of the form shown in equation (49).

IV. GCMMA is the global convergent version of MMA

V. MATLAB-FMINCON is a MATLAB optimization toolbox used to solve nonlinear programming problems. FMINCON uses five different algorithms from which four of them will be used for this thesis. The fifth algorithm, the trust-region-reflective

algorithm, requires bound or linear equality constraints, and since the example problems in this paper contain non-linear constraints, the trust-region-reflective algorithm has been excluded from this research. The four algorithms used are listed below

a. Interior-point (IP)
b. Sequential quadratic programming (SQP)
c. Sequential quadratic programming -legacy (SQPL)
d. Active-set (AS)

VI. MIDACO[5] (Mixed Integer Distributed Ant Colony Optimization) is an optimization solver which is based on the ant colony optimization algorithm. The trial version of MIDACO that only works with a maximum of four variables problem was used in this paper.

Of these NLP solvers I, III, IV and VI requires the gradient of the objective function and the constraints. No gradient function is required for the rest. To solve an optimization problem, MMA and GCMMA always need a lower and upper bound of the variable.

5.3. Example Problems and Result

Thirteen different problems are studied to compare the solver performance. The variables number n ranges from two to twelve. Similarly, the number of constraints q varies from one to four. Information about each solver performance is tabulated following the expression for each problem. The CPU times shown in the tables are the average value of three consecutive simulation run.

[5] Since the trial version of this software was used in this paper the performance might be different than the full version

Problem 1:

$$\text{Min } f(x) = {x_1}^2 + {x_2}^2 + {x_3}^2 + {x_4}^2 + {x_5}^2 + {x_6}^2 + {x_7}^2 + {x_8}^2 + {x_9}^2 + {x_{10}}^2 + {x_{11}}^2 + {x_{12}}^2 \quad (51)$$

subject to the constraints:

$$\emptyset_1(x) = x_1 + x_3 x_2 - 1$$
$$\emptyset_2(x) = x_2 + x_5 - 2x_7$$
$$\emptyset_3(x) = x_8 + x_4 - x_9 + 3$$
$$\emptyset_4(x) = x_{10} + x_{11} + x_{12} - 1$$

With initial guess of $x = ones(12,1)$, the calculation time and final and the initial function value for each of the solvers is tabulated below.

Initial guess $x = ones(12,1)$				
Name		Initial function value	Final function value	CPU time
FMINCON	SQP	12	4.3333	0.2520
	SQP-legacy			0.276
	Active-set			0.2796
	Interior-point			0.4267
SGRA			4.3335	0.0099
KNITRO	Interior-point		4.3333	0.0512
	Active-set			0.1489
MMA			4.383	49.3021
GCMMA			4.3819	29.023223

Table 6 Comparison of solvers using problem 1

From the table above SGRA converges, with a calculation time of 0.0099 sec, faster than the rest of the optimization approaches. MMA and GCMMA didn't converge to the optimal solution and as number of iterations for increased the solution even diverges further from the optimal solution.

Problem 2:

$$\text{Min } f(x) = -x_1 x_2 x_3 x_4 \tag{52}$$

subject to the constraints:

$$\emptyset_1(x) = {x_1}^3 + {x_2}^2 - 1$$
$$\emptyset_2(x) = {x_1}^2 + x_4 - x_3$$
$$\emptyset_3(x) = {x_4}^2 - x_2$$

Initial guess $x = ones(4,1)$				
Name		Initial function value	Final function value	CPU time
FMINCON	SQP	-1	-0.25	0.137303
	SQP-legacy			0.2456
	Active-set			0.275
	Interior-point			0.3654
SGRA				0.0017
KNITRO	Interior-point			0.0375
	Active-set			0.0358
MMA				1.4054
GCMMA				3.9768
MIDACO			-0.249	9.7018

Table 7 Comparison of solvers using problem 2

From the table above for the benchmark problem 2, the SGRA still converges faster than the rest of the solvers with a CPU time of 0.0017 sec. For KNITRO, however, the AS algorithm has a smaller benefit in CPU time than the internal point algorithm in this unlike problem 1.

Problem 3:

$$\text{Min } f(x) = (x_1 - 1)^2 + (x_2 - 2)^2 + (x_3 - 3)^2 + (x_4 - 4)^2 \tag{53}$$

subject to the constraints:

$$\emptyset_1(x) = x_1 - 2$$

$$\emptyset_2(x) = {x_3}^2 + {x_4}^2 - 2$$

Initial guess $x = ones(4,1)$				
Name		Initial function value	Final function value	CPU time
FMINCON	SQP	14	13.8579	0.134257
	SQP-legacy			0.2559
	Active-set			0.2761
	Interior-point			0.3733
SGRA			13.8579	0.0074
KNITRO	Interior-point		13.8579	0.0342
	Active-set			0.0369
MMA			13.8588	0.3509
GCMMA			13.8579	9.4765
MIDACO			13.8534	8.2057

Table 8 Comparison of solvers using problem 3

MMA solves the problem faster than the FMINCON-IP algorithm. FMINCON, however, solved the problem without applying the upper and lower limits, while MMA needs the upper and lower limits to solve the problem.

Problem 4:

$$\text{Min } f(x) = (x_1 - x_2)^2 + (x_2 + x_3 - 2)^2 + (x_4 - 1)^2 + (x_5 - 1)^2 \tag{54}$$

subject to the constraints:

$$\emptyset_1(x) = x_1 + 3x_2$$
$$\emptyset_2(x) = x_3 + x_4 - 2x_5$$
$$\emptyset_3(x) = x_2 - x_5$$

Initial guess $x = ones(5,1)$				
Name		Initial function value	Final function value	CPU time
FMINCON	SQP	0	4.093	0.14149
	SQP-legacy			0.253
	Active-set			0.272
	Interior-point			0.3788
SGRA			4.093	0.0057
KNITRO	Interior-point		4.093	0.0349
	Active-set			0.0357
MMA			4.0934	1.5638
GCMMA			4.0931	12.0248

Table 9 Comparison of solvers using problem 4

Problem 5:

$$\text{Min } f(x) = (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^4 \tag{55}$$

subject to the constraints:

$$\emptyset_1(x) = x_1({x_2}^2 + 1) + {x_3}^4 - 4 - 3\sqrt{2}$$

Initial guess $x = ones(3,1)$				
Name		Initial function value	Final function value	CPU time
FMINCON	SQP	0	0.0326	0.14154
	SQP-legacy			0.2504
	Active-set			0.2686
	Interior-point			0.3734
SGRA			0.0326	0.007
KNITRO	Interior-point		0.0326	0.0404
	Active-set			0.0366
MMA			0.0334	8.9878
GCMMA			0.0344	818.5
MIDACO			0.0326	2.1517

Table 10 Comparison of solvers using problem 5

If either the lower or the upper bound were not applied, MIDACO would not converge to the optimum solution for the above problem. In the table above, a lower limit of zero is used for solving the problem by MIDACO which converged more quickly than MMA.

Problem 6:

$$\text{Min } f(x) = (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6 \qquad (56)$$

subject to the constraints:

$$\emptyset_1(x) = {x_1}^2 x_4 + \sin(x_4 - x_5) - 2\sqrt{2}$$

$$\emptyset_2(x) = x_2 + {x_3}^4 {x_4}^2 - 8 - \sqrt{2}$$

Initial guess $x = ones(5,1)$				
Name		Initial function value	Final function value	CPU time
FMINCON	SQP	0	0.2415	0.154386
	SQP-legacy			0.2627
	Active-set			0.2821
	Interior-point			0.3844
SGRA			0.2415	0.0031
KNITRO	Interior-point		0.2415	0.0418
	Active-set			0.0366
MMA			0.3974	76.7057
GCMMA			0.4337	238.1397

Table 11 Comparison of solvers using problem 6

MMA and GCMMA displayed a disagreement from the optimal solution while the other solvers calculated the solution with the same trend observed for the other problems.

Problem 7:

Min $f(x) = (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2 + (x_3 - x_4)^4 + (x_4 - x_5)^4$ (57)

subject to the constraints:

$$\emptyset_1(x) = x_1 + {x_2}^2 + {x_3}^2 - 2 - 3\sqrt{2}$$

$$\emptyset_2(x) = x_2 - {x_3}^2 + x_4 + 2 - 2\sqrt{2}$$

$$\emptyset_3(x) = x_1 + x_5 - 2$$

Initial guess $x = ones(5,1)$				
Name		Initial function value	Final function value	CPU time
FMINCON	SQP	0	0.1623	0.142924
	SQP-legacy			0.2543
	Active-set			0.284
	Interior-point			0.3789
SGRA			0.1623	0.0087
KNITRO	Interior-point		0.1623	0.0366
	Active-set			0.0398
MMA			0.1623	12.26
GCMMA			0.1623	202.165

Table 12 Comparison of solvers using problem 7

Problem 8:

$$\text{Min } f(x) = 0.01(x_1 - 1)^2 + (x_2 - {x_1}^2)^2 \tag{58}$$

subject to the constraints:

$$\emptyset_1(x) = x_1 + {x_3}^2 + 1$$

Initial guess $x = ones(3,1)$				
Name		Initial function value	Final function value	CPU time
FMINCON	SQP	0	0.0404	0.140382
	SQP-legacy			0.2707
	Active-set			0.2897
	Interior-point			0.3934
SGRA			0.0404	0.0156
KNITRO	Interior-point		0.04	0.04
	Active-set			0.0408
MMA			0.0425	0.2453
GCMMA			0.0401	2.7936
MIDACO			0.041411	1.2939

Table 13 Comparison of solvers using problem 8

Problem 9:

$$\text{Min } f(x) = -x_1 \tag{59}$$

subject to the constraints:

$$\emptyset_1(x) = x_2 - {x_1}^3 - {x_3}^2$$
$$\emptyset_2(x) = {x_1}^2 - x_2 - {x_4}^2$$

Initial guess $x = ones(4,1)$				
Name		Initial function value	Final function value	CPU time
FMINCON	SQP	-1	-1	0.141759
	SQP-legacy			0.2562
	Active-set		-1.0622	0.3241
	Interior-point		-1	0.3838
SGRA			-1	0.0105
KNITRO	Interior-point		-1	0.0355
	Active-set			0.0378
MMA			-1	1.3648
GCMMA			-1	238.3986
MIDACO			-0.97989	4.9125

Table 14 Comparison of solvers using problem 9

Problem 10:

$$\text{Min } f(x) = \log(1 + {x_1}^2) - x_2 \tag{60}$$

subject to the constraints:

$$\emptyset_1(x) = (1 + {x_1}^2)^2 + {x_2}^2 - 4$$

Initial guess $x = zeros(2,1) + 2$				
Name		Initial function value	Final function value	CPU time
FMINCON	SQP	-0.3906	-1.7321	0.141832
	SQP-legacy			0.2482
	Active-set			0.2665
	Interior-point			0.43
SGRA			-1.7321	0.012776
KNITRO	Interior-point		-1.7321	0.0406
	Active-set			0.0345
MMA			-1.7321	1.0169
GCMMA			-1.7325	790.7121
MIDACO			-1.7322	0.1757

Table 15 Comparison of solvers using problem 10

Problem 11:

$$\text{Min } f(x) = (x_1 - 20)^2 + (x_2 + 20)^2 \tag{61}$$

subject to the constraints:

$$\emptyset_1(x) = (\frac{x_1}{a})^2 + (\frac{x_2}{b})^2 - 1$$

$$a = 10; b = 10$$

Initial guess $x = [4, -5]$				
Name		Initial function value	Final function value	CPU time
FMINCON	SQP	481	334.3146	0.140921
	SQP-legacy			0.2389
	Active-set			0.2552
	Interior-point			0.358
SGRA			334.3146	0.007402
KNITRO	Interior-point		334.3146	0.0323
	Active-set			0.037
MMA			334.3146	0.0802
GCMMA			334.3146	0.3817
MIDACO			334.3146	0.7344

Table 16 Comparison of solvers using problem 11

Problem 12:

The objection function is the same as problem 11. For the constraint function the constants values in the constraint's equation was changed using following values.

$$a = 10; b = 4$$

Name		Initial function value	Final function value	CPU time
FMINCON	SQP	481	452.4044	0.136228
	SQP-legacy			0.2436
	Active-set			0.2607
	Interior-point			0.3581
SGRA			452.4044	0.011706
KNITRO	Interior-point		452.4044	0.031
	Active-set			0.0322
MMA			452.4044	0.0605
GCMMA			452.4044	0.6896
MIDACO			452.2719	0.6946

Table 17 Comparison of solvers using problem 12

Problem 13:

Similarly, for problem 13, the same objection and constraint function as problem 11 was used with constraint equation constants of,

$$a = 10; b = 1$$

Name		Initial function value	Final function value	CPU time
FMINCON	SQP	617	496.1121	0.142002
	SQP-legacy			0.2471
	Active-set			0.2616
	Interior-point			0.3762
SGRA			496.1121	0.012376
KNITRO	Interior-point		496.1124	0.049
	Active-set			0.0356
MMA			496.1124	0.1584
GCMMA			496.1124	1.7066
MIDACO			496.0119	0.6306

Table 18 Comparison of solvers using problem 13

Of the methods implemented to solve NLP problems, SGRA was the most consistent and efficient from the tabulated results of 13 different example problems. This algorithm has proven to be a very powerful solver by completing all the problems in a much smaller calculation time. The ARTELY KNITRO method had a higher calculation time than SGRA. However, despite that, it performed better than the rest of the methods, for both the interior-point and active set algorithm. In some examples MMA performed better than the algorithms used by FMINCON but considering the consistency and the ability to use FMINCON without the need for upper and lower bound as well as a gradient function, it can be concluded that FMINCON is more efficient and reliable than MMA, GCMMA and MIDACO.

CHAPTER SIX

TRAJECTORY OPTIMIZATION

Trajectory generation for multirotors is based on the construction of a set of space points that describe the transfer path of the vehicle from an initial position to a final position. In general, a particular criterion must be selected to generate a desired trajectory. For example, in section 4.1, I have used polynomial expressions to describe the trajectory components that transfer the vehicle from its initial position to a desired final position. The coefficients of these polynomials were chosen so to satisfy the boundary conditions on position, velocity, and acceleration. In this section, I will be describing a trajectory generation strategy that aims to address the limited flight duration of a multirotor. In particular, I will investigate optimal trajectories which minimize the energy required by a multirotor to transfer from its current position to a final position within a given time.

With this goal in mind, the maneuver energy optimization is achieved through the minimization of jerk *J*, which is the rate of change of acceleration.

$$J = \frac{da}{dt} = (\frac{dF}{dt})/m \tag{62}$$

providing that the body mass m is constant.

A work from Mueller et.al. [24], proved that minimizing the control effort on a quadrotor is equivalent to minimizing the manoeuvre jerk. In particular, the authors proved that minimizing the jerk is equivalent to minimizing the upper bound of the product of the control inputs (total thrust and body angular rates).

$$\int_0^T (f(t)^2 \|\omega(t)\|^2) dt \leq \sum_{i=1}^{3} J_i \tag{63}$$

where J is the jerk, f is the total thrust, i is the axis subscript and ω is the body angular rate

Hence minimizing the jerk would minimize the product of the control inputs. The control inputs, however, are the major force that consume the power energy of the drone.

Consequently, minimizing these control inputs will also reduce the power consumption. As a result, the jerk minimization reflects into minimizing the power consumption.

6.1. Previous Works

A number of solutions to enhance endurance have recently been proposed in the literature. Most of these projects focused on improving the mechanical design and power systems, whereas other research teams developed a minimum energy path guidance strategy. In [25], MIT researchers proposed a hardware platform for a battery swamp strategy that enables UAVs to autonomously swamp batteries and perform a long–duration task. In a similar context, Ure et.al. developed an automatic system in [26] that can switch drained batteries with a charged battery while recharging several other batteries simultaneously. Another group of researchers in [27] attempted to address the endurance performance of electrically powered UAVs by dumping batteries that were depleted. This can, however, raise concerns about environmental pollution and safety issues.

In addition to those studies that attempted to solve the endurance limitations of UAVs by improving the power system, several other studies have also been carried out to improve the mechanical design of UAVs. One example of these projects is reported in [28], where Gurdan et.al. created a low-weight flying robot that is energy efficient. Furthermore, the authors in [29] described a new energy efficient rotor configuration for a quadrotor.

The generation of an energy-optimal trajectory to deal with the problem of energy consumption received some attention in the rotorcraft literature only in recent years. Yacef et. al. in [30] proposed an optimal control problem using an energetic model to optimize the UAV's trajectory which minimizes the consumption of energy. In that work, a NLP solver software called GPOPS-II was used to solve the proposed optimal control offline. Similarly, in [1] the author, by leveraging electrical model of a brushless DC motor, determined minimum energy trajectory using ACADO Toolkit to solve optimal control problem for DJI Phantom 2. The solver took more than 65 sec in average to calculate the optimal solution as stated in the same paper, which hinders the ability to apply it in real-time. This forced the author to solve optimal problem offline and deploy the trajectory. There have been also researchers, such as in [31], who introduced a software system by the

name of Green Flight that can acquire battery data during a mission and make suggestions on how to optimize the consumption. Although this software system approach is quite different than generating trajectory that would minimize energy, it would give the user the ability to decide what to do based on the battery energy analysis delivered by the system.

The guidance strategy addressed by multiple researchers on this topic showed that they executed the optimization calculation offline and deploy the trajectory afterwards. However, in real environmental situations, it would be difficult to achieve optimal value as the environment highly affect the trajectory. Hence, working towards real-time optimization is the preferable approach. In this thesis real time guidance strategy for minimum energy is introduced.

6.2. Bezier Curve

Bezier curve is a particular parametric curve between two end points. This curve had its initial applications in computer graphics and related fields. It is also practical in mechanical and electrical engineering applications. An example of a quadratic Bezier curve is shown in Figure 30 below. P_0 and P_2 are the end points of the Bezier curve while P_1 is the control point.

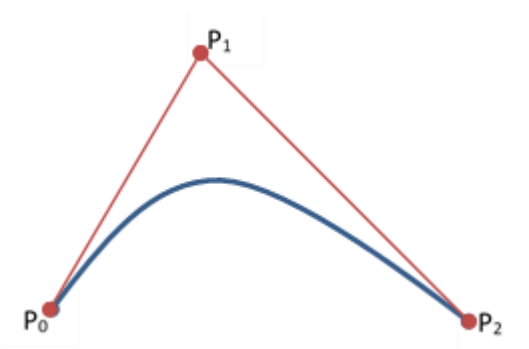

Figure 30. Quadratic Bezier curve

Lately, there have been a growing interest in the use of Bernstein polynomials for aerial vehicle trajectory generation. In the work of Cichella et.al. [32], a Bernstein approximation was used to propose a minimum jerk optimal trajectory for a differentially flat aerial

vehicle. That paper presents a collision avoidance trajectory generation system using MATLAB FMINCON as NLP solver. Notably, the NLP problem was derived from the original optimal control problem formulated to achieve the collision avoidance mission while minimizing the jerk. The fundamental equations discussed on that paper are reported in the following section. For further detailed mathematical derivation it is recommended to refer to [32].

A Bezier curve function can be defined as the summation of a control points multiplied by the Bernstein polynomial.

$$B_n(t) = \sum_i^n c_i p_{i,n(t)} \tag{64}$$

where n is the Bernstein polynomial degree, c_i are control points ($i = 0, 1, \dots, n$), and the polynomial

$$p_{i,n(t)} = \frac{\binom{n}{i}(t_f - t)^{n-i} t^i}{{t_i}^n} \tag{65}$$

is known as the Bernstein polynomial of degree n.

The set of all i combination from n elements $\binom{n}{i}$ can be written using factorial as

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \tag{66}$$

6.2.1. Determinations of Optimal Trajectory

In this section the optimal control problem for differential system, as stated in [32], is introduced and its transcription into a nonlinear programming problem is also provided. In this scenario the SGRA, introduced in section 5.2, can be used as solver to generate the optimal trajectories in real time.

A general optimal control problem is expressed as

Minimize $$\tilde{I}\big(\boldsymbol{x}(t),\boldsymbol{u}(t)\big) = \tilde{E}\left(\boldsymbol{x}(0),\boldsymbol{x}\big(t_f\big)\right) + \int_0^{t_f} \tilde{F}(\boldsymbol{x}(t),\boldsymbol{u}(t))dt \tag{67}$$

subject to,

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t),\boldsymbol{u}(t)\big), \tag{68}$$

$$\tilde{\boldsymbol{e}}\left(\boldsymbol{x}(0),\boldsymbol{x}\big(t_f\big)\right) = \boldsymbol{0}, \tag{69}$$

$$\tilde{\boldsymbol{h}}\big(\boldsymbol{x}(t),\boldsymbol{u}(t)\big) \leq \boldsymbol{0} \tag{70}$$

The differential constraints in equation (68), represents the dynamics of the system, the equality constraint in equation (69) represents boundary conditions on the state, and the inequality constraint in equation (70) represents bounds on control variables and, if presents, path constraints.

The system expressed in equation (68) is assumed to be differentially flat system. In particular, the function on equation (68) is flat if there exists an output $\boldsymbol{y}$ expressed as

$$\boldsymbol{y}(t) = \boldsymbol{\beta}\left(\boldsymbol{x}(t),\boldsymbol{u}(t),\dot{\boldsymbol{u}}(t),\dots,\boldsymbol{u}^{\boldsymbol{d}}(t)\right) \tag{71}$$

such that

$$\boldsymbol{x}(t) = \boldsymbol{\beta_1}\left(\boldsymbol{y}(t),\dot{\boldsymbol{y}}(t),\dots,\boldsymbol{y}^{(s-1)}(t)\right)$$

$$\boldsymbol{u}(t) = \boldsymbol{\beta_2}\left(\boldsymbol{y}(t),\dot{\boldsymbol{y}}(t),\dots,\boldsymbol{y}^{(s)}(t)\right)$$

The above optimal control problem can then be formulated as a calculus variations equation with

$$z(t) = [\big(\boldsymbol{y}(t)^{\boldsymbol{T}},\dot{\boldsymbol{y}}(t)^T,\dots,\boldsymbol{y}^{(s)}(t)^T\big)]^T \tag{72}$$

Substituting equation (71) and (72) in to the optimal control problem the calculus variations problem can be transcribed for all $t \in [0, t_f]$ as

$$I\big(\boldsymbol{y}(t)\big) = E\left(z(0),z\big(t_f\big)\right) + \int_0^{t_f} F(\boldsymbol{z}(t))dt \tag{73}$$

subjected to:

$$\boldsymbol{e}\left(z(0), z\left(t_f\right)\right) = \mathbf{0},$$
$$\boldsymbol{h}\left(z(t)\right) \leq \mathbf{0}$$

The above calculus variation problem then can be approximated using Bezier curves and Bernstein polynomial basis as follows.

For $\delta p \in (0,1)$**,** a Bezier curve degree of n and control points of $c = [c_0, \dots, c_n]$, determine c that minimizes

$$I(\boldsymbol{c}) = E\left(z_n(0), z_n(t_n)\right) + w \sum_{i=0}^{n} F(z_n(t_i)) \tag{74}$$

subject to

$$\left\|\boldsymbol{e}\left(z_n(0), z_n(t_n)\right)\right\| \leq n^{-\delta p}$$

$$\boldsymbol{h}\left(z_n(t_i)\right) \leq n^{-\delta p}\mathbf{1}$$

where $w = \frac{t_f}{n+1}$ and $i = 0, \dots, n$

These equations identify a nonlinear programming problem which can be solved, after some minor modification, by the SGRA to calculate the control points c. The final trajectory can be subsequently generated by substituting these control points in the Bezier curve formula, equation (64).

6.3. Numerical Evaluation and Results

In the literature review, it was stated that previous researches performed trajectory optimization for the minimization of maneuver energy, but the solution of these problems were calculated offline. This was due to the difficulty in the optimization solvers to achieve a faster calculation of the optimal solution. In CHAPTER FIVE, several NLP solvers were compared based on reliability and speed of evaluation. SGRA took the least amount of time to solve the optimization problems compared to the other methods. For these reasons, SGRA will be applied to address this problem.

In order to apply the optimization problem in SGRA the problem must be expressed in the form of NLP. In the last section, it was shown how to change the control problem equation for differential flat system to the NLP form, that can be used by SGRA directly.

The final nonlinear problem used by SGRA is

Determine $\boldsymbol{x}(t)$ that minimizes

$$f = f(x_1, x_2, \dots x_n) \tag{75}$$

subject to

$$\emptyset_{\boldsymbol{i}}(x_1, x_2, \dots x_n) + {s_i}^2 = 0$$

where s_i is a slack variable added to convert the inequality constraints to equality constraints. $i = 1,2, \dots$ q, where q is number of scalar constraints.

The constraints also include the condition to avoid collision throughout the flight.

6.3.1. Pre-flight Simulation Results

In this part, an offline simulation is performed for a selected initial and final position before the experimental flight algorithm is executed.

For the pre-flight offline example, the following conditions were taken,

- The drone starts from initial position of $[x_0, y_0] = [2.5,2]$.
- The final position of the drone would be $[x_f, y_f] = [0,0]$.
- A static virtual circular obstacle with a radius of 1 unit having the center at $[x_c, y_c] = [1,1]$ was considered.

For this problem, a Bernstein polynomial of degree 6 is used, as this value is the minimum degree that can give a sounding result. Therefore, the control points to be optimized are 5, which makes the total variables 10, as there are 5 points for the x coordinate and 5 points for the y coordinate. For the constraints, 13 constraints were applied, which also makes the slack variables 13. The total variables to be minimized are then become the 10 variables plus 13 slack variables which summed to be 23.

To calculate the initial conditions for the SGRA a straight line with the initial and final points as the extreme ends was created. The line then divided into ten points to be used as initial guess for the variables. The algorithm for the initial guess is shown below.

Algorithm 1: Initial guess prediction

Input: The initial and final position vector, $X_0 = [x_0, y_0]$, $X_f = [x_f, y_f]$, the final time, t_f, and the degree of Bernstein polynomial, N

Output: Initial guess vector

```
function x_0 = InitGuess (X_f, X_0, t_f, N)
time = 0:t_f/N: t_f;
    for t_in = 2:N
        t = time (t_in);

        %Straight line from initial to the final points
        x_gs(:, t_in − 1) = t * (X_f(:)−X_0(:))/t_f + X_0(:);
end
[m, n] = size(x_gs);
x_0 = reshape(x'_gs, [2 * n, 1]);
end
```

The SGRA solved the optimization equation with an average CPU time of 0.06 sec. As discussed in section 6.2.1, the solution from the sequential gradient algorithm is used in the Bezier curve for control points to create a trajectory. The algorithm to calculate the Bezier curve is described below.

Algorithm 2: Bezier Curve

Input: The vector of the control points (the optimization solutions) $\boldsymbol{x_0}$, the final time t_f and the Bernstein polynomial degree N

Output: Bezier curve points

```
function B_z = Bezier (x_0, t_f, N)
    st = 0.01; %Sample Time
    t_0 = 0; %Initial Time
time = t_0: 0.01: t_f;
    k = length (time);
B_z = [ ];
    for m = 1: k
        B_zr = 0;
        for i = 0: N
        comb = factorial(N) / (factorial(i)*facorial(N-i));
        B_zr = (comb * t_f - time(m))^(N-i) * ((time(m)-t_0)^i) / (t_f)^N * x_0(:, i + 1) + Bzr;
        end
        B_z = [B_zr]; %Bezier Curve
end
```

The X Y plot of the results with the control points will then be,

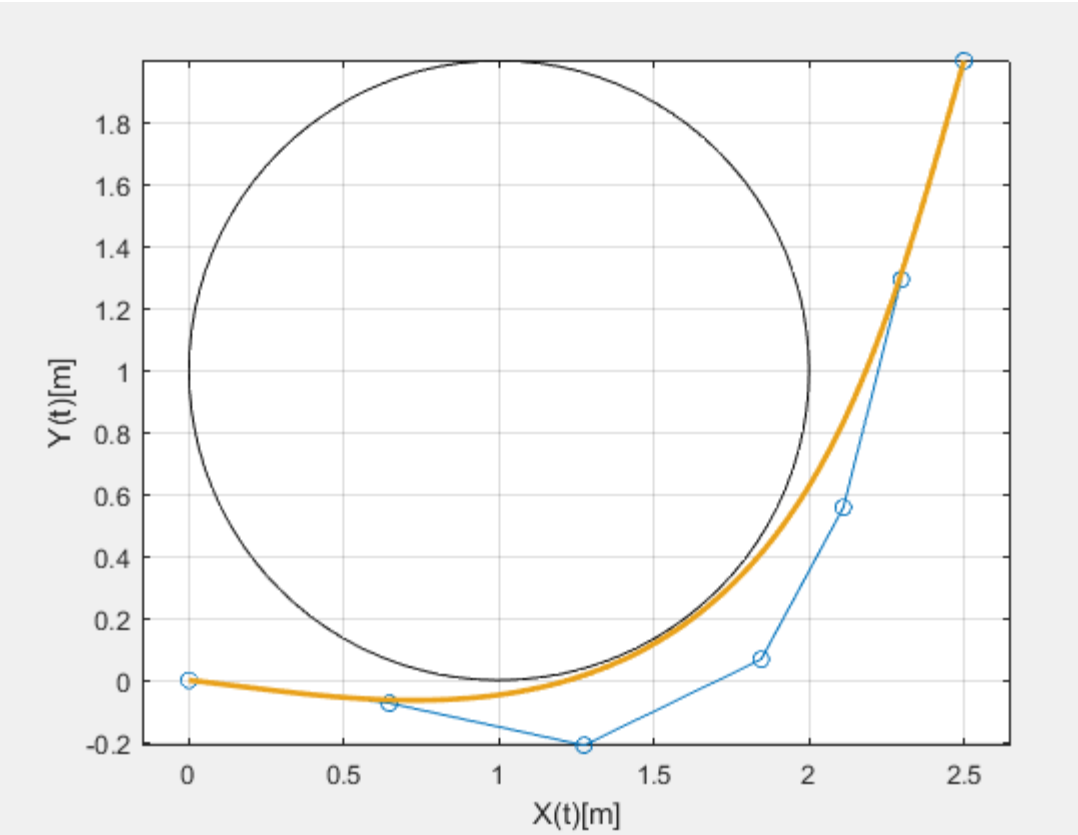

Figure 31. Optimized trajectory calculated offline

On the plot the points connected by a line are the control points. The circle, as described in the start of this section, is the virtual obstacle. The idea behind the Bezier curve above is that the smooth curve shown in orange above would have been achieved if an infinite number of control points were used.

6.3.2. Experimental Result

We have created a function block in Simulink to use the SGRA algorithm to solve the control points and stream data to the drone in real time. Since the calculation time could be different for different scenarios, a sample time of 0.3 sec was used to run the SGRA Simulink block in order to give SGRA enough time to solve the problem in each calculation. However, the sample time running the OptiTrack model and the Controller model is 1millisecond. As a consequence, some changes were made to the Bezier curve generation system model to make it compatible with other models, which will be discussed

in detail later. The OptiTrack model Simulink diagram is shown below for this experimental flight.

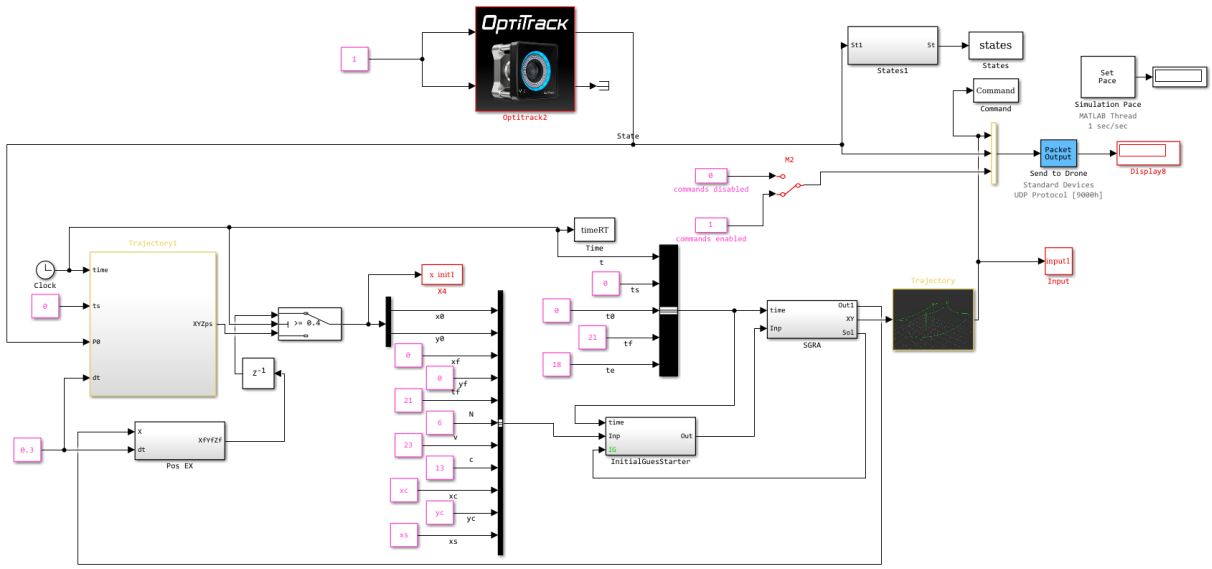

Figure 32. Simulink diagram of the OptiTrack model

To run the Simulink model, the drone's initial position must be known. The x and y position of the drone was extracted from the OptiTrack data at the initial time of the trajectory, which is $t = t_0$. These points have been used as the starting point of a planned trajectory. The initial position of the drones must be sent to SGRA every 0.3 sec, the sample time at which the SGRA block is running. The second position data meaning the initial position data at $t = 0.3$ sec is calculated by extrapolating the data from the previous position. One reason why extrapolation is preferred to capture data from the OptiTrack MCS is due to the complexity of creating an algorithm to capture data at that exact time. The other is that even if the data could be captured at that time, a time delay occurs when it is sent to the next block that causes the data gap and creates an error in the entire simulation. The second initial location data was therefore extrapolated from the first. The third and subsequent initial positions were calculated by extrapolating the Bezier curve's final points. The Bezier were calculated to return a point vector for 0.3 seconds. Therefore, it is possible to calculate the initial position at $t = 0.6$ sec using the last two elements of

this vector, which are the $t = 0.29$ sec and $t = 0.3$ sec points. The same technique was used to calculate the other values of the initial position.

The algorithm used to extract the first initial position points from the MCS data points is displayed as follows

---

Algorithm 3: Capturing data at $t = t_s$ sec

Input: time t, the specific time the data required at t_s, a flag F_{in}, the position vector P_0 and P_{inm}

Output: Flag out F_{out} and the captured data vector P_{outm}

$function\ [P_{outm}, F_{out}] = fcn(P_0, P_{inm}, F_{in}, t, t_s)$
$F_{out} = F_{in};$
$P_{outm} = P_{inm};$
$if\ t \geq t_s\ \&\ F_{in} = 0$
    $P_{outm} = P_0;$
    $F_{out} = 1;$
end

The initial guess calculation method discussed in the pre-flight simulation was also used in the experimental test to calculate the initial guess. The block calculating the initial estimation points can be seen in the Figure 33 below.

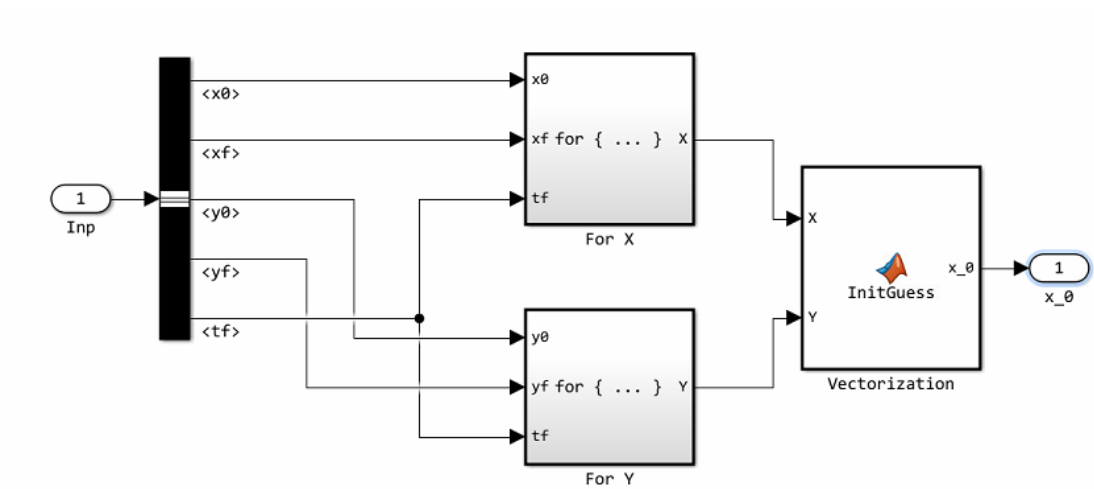

Figure 33. Initial guess block

For the first five points, x-coordinate points, the for-loop block is calculated in the 'For X ' block, while the last five are calculated in the ' For Y ' block. The internal structure for 'For X' is shown in Figure 40 of Appendix III. The vectorization function block creates a 10 x 1 vector from the solution of each block. These points were used as an initial guess for the 10 variables. For the remaining variables, which are the 13 slack variables, the same vector is used for the first 10 variables and 0.4 was used as an initial guess for the last three variables.

As previously described, the SGRA block and the Bezier curve are calculated at 0.3 sec. The Bezier curve points calculated each time for a duration of 0.3 sec. The data package calculated for the 0.3 sec is 30 points because each point was calculated as if the sample time was 0.01 sec. The vector elements therefore correspond to the time with an increase of 0.01. The vector is a matrix of 2 x 30. The vector elements are streamed to the controller model every 0.01 sec using a column selector block via UDP sender. A rate transition is used to exchange data between blocks using a 0.3 sec sample time and 0.01 sec sample time.

Algorithm 4: Bezier Curve calculation for x axis

Input: The vector of SGRA solution xyf, the time t, the final time of the trajectory t_f, initial time t_0 and the degree of Bernstein polynomial N

Output: The Bezier curve points Bzr

$function$ Bzr $=$ bezier(xyf, t, t_0, t_f, N)

Bzr $= 0$;

if $t_f - t \geq 0.1$

for $i = 0{:}N$

$$\text{comb} = \frac{\text{factorial(N)}}{\text{factorial(i)} * \text{factorial(N} - \text{i)}};$$

$$\text{Bzr} = \frac{\text{comb} * (t_f - t)^{N-i} (t - t_0)^i}{(t_f - t_0)^N} \text{xyf}(1, i + 1) + \text{Bzr};$$

end

$else$

Bzr $= 0$;

end

When the Bezier curve vector is calculated every 0.3 seconds, the initial time changes each time. To describe it in detail, see Figure 34 below. At first the drone was at point 0. During the first flight the drone moves from point 0 to 1 in 0.3 sec. This means that the flight duration of 0.3 sec was already completed, therefore the initial time for the next flight is the time at point 1. When the drone moves from point 1 to point 2, the same time, 0.3 sec, is used to cover the track. Then the next initial time becomes 0.6 sec or the time at point 2. It will follow the same trends until the initial time is equal to the final time which is the time the drones reached the destination.

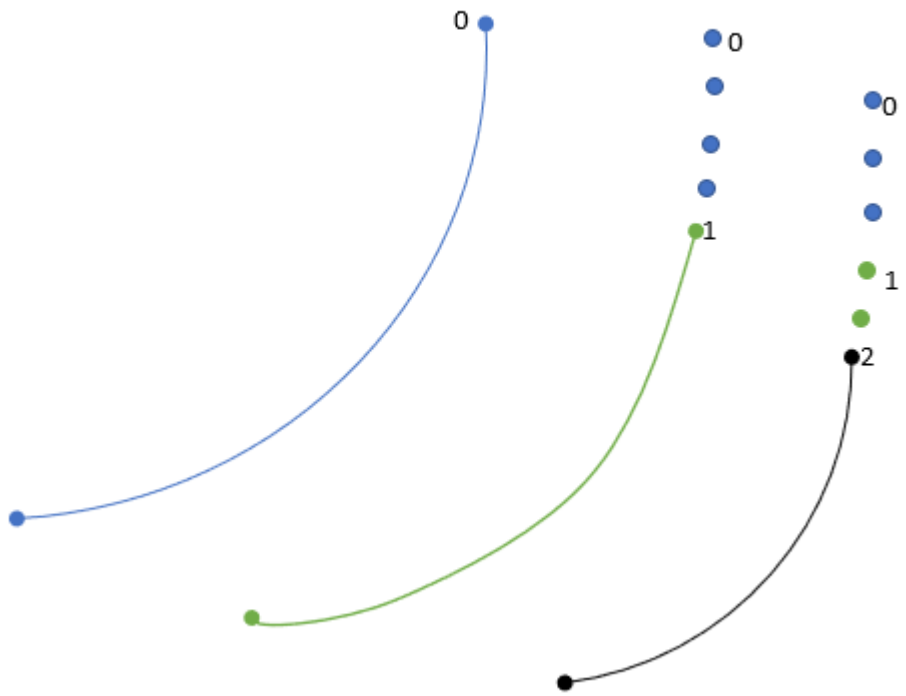

Figure 34. The duration of time changes as the drone flies

As discussed earlier, the last two elements of the Bezier vector, 29th and 30th, are sent to the extrapolation block to calculate the future points that will be used as the initial position for the next calculation of the Bezier curve.

It became difficult for SGRA to solve a problem on time as the drones reach the destination points. To deal with this problem, we have created another block that calculates the curves of Bezier for the last 3 seconds. Therefore, the drone will use this trajectory for the last three minutes instead of the one calculated every 0.3 sec. An automatic switch will be used to switch between these two blocks.

The following commands were executed which the drone follows. The inputs are shown here.

- The final destination points are $[x_f, y_f] = [0,0]$
- The altitude was set to 0.6 m
- The yaw angle was set to zero
- A virtual obstacle placed at $[x_c, y_c] = [1,0.5]$.

The x-y plane plot is then shown below.

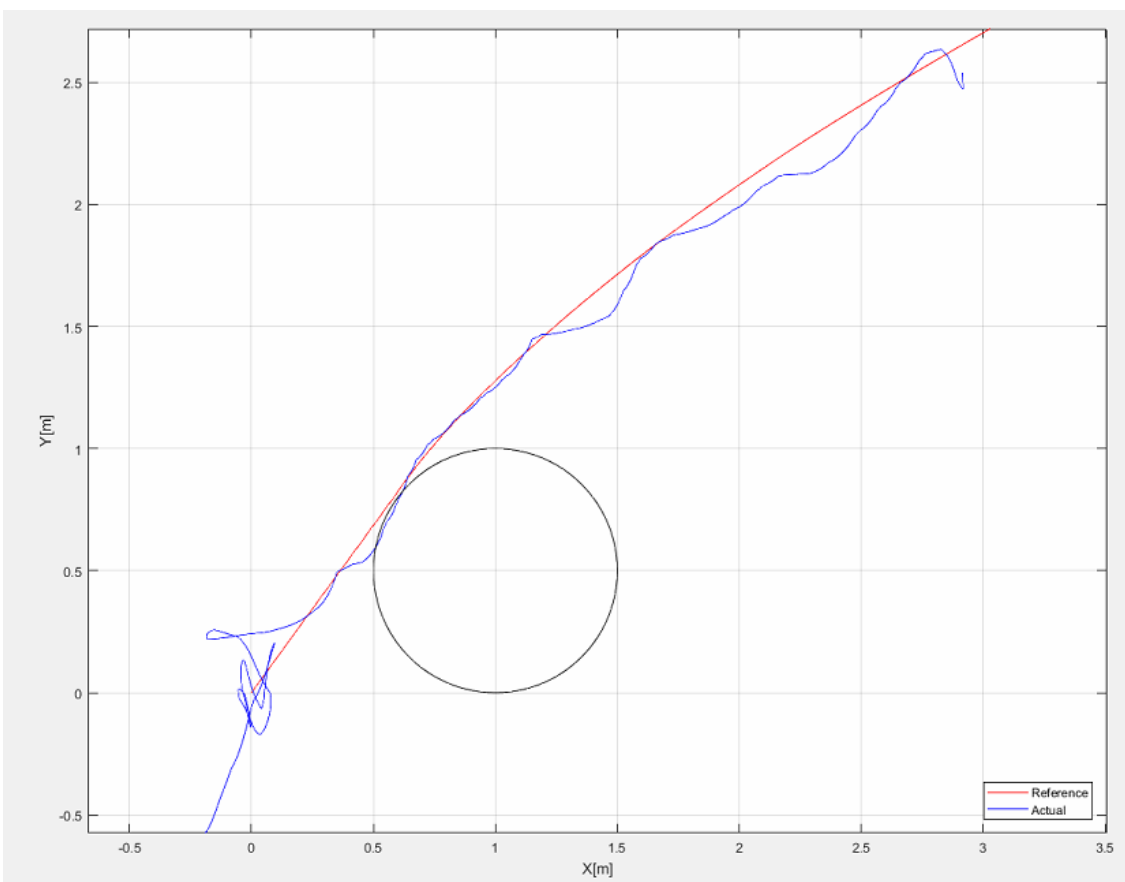

Figure 35. Optimized trajectory avoiding collision

The x, y, and z plot separately is then shown in the following diagram.

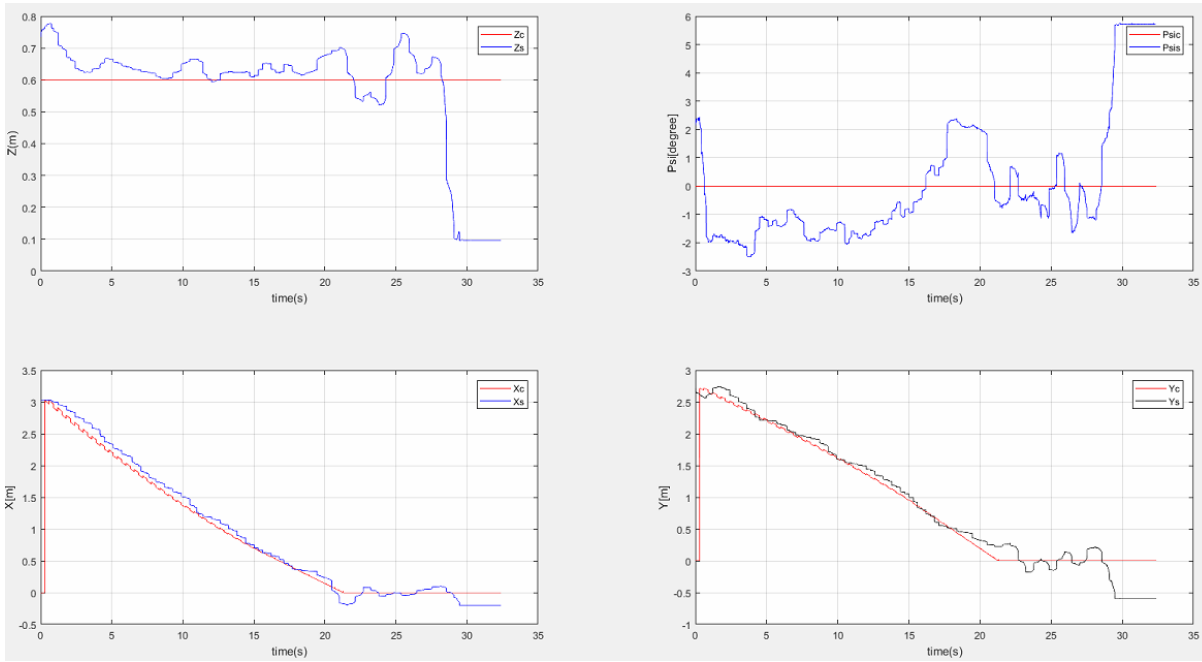

Figure 36. The X, Y, Z and yaw angle plot

CHAPTER SEVEN

CONCLUSION

7.1. Summary

This thesis has presented the development of an indoor multirotor testbed lab and guidance, navigation and control benchmark problems on multirotor. In chapter 3, a detailed description of how to develop an experimental testbed laboratory for multirotor was presented. Furthermore, I described how to control and fly an AR Drone using Simulink as a ground controller and an OptiTrack MCS as system to retrieve position and attitude information. In chapter 4, a strategy to execute an autonomous and accurate landing was discussed. The obtained results show that this approach could be used for autonomous battery charging purpose. In the same section I have also reported a methodology for formation flight execution between two drones to reach an accurate steady alignment. In chapter 5 a comparison between different nonlinear programming methods presented. The performance of each solvers was compared based on the reliability and computational.

In chapter 6 one of the major sets back of battery powered drones was tackled through the application of trajectory optimization. A trajectory was generated using a Bezier curve by minimizing a jerk function. Minimization of jerk would have an indirect effect on the power consumption.

7.2. Recommendation

The project included in this thesis paper, including accurate landing and optimization of trajectories, is concerned with solving the energy consumption problem. In the precise landing study, it was demonstrated that a drone can execute a precise landing on a mock-up charging station. For this particular work, however, an actual battery charge was not performed. With the landing accuracy achieved in this study, an actual autonomous battery charging can be performed by building a charging station.

Another work included in this thesis paper was the use of trajectory optimization to minimize energy consumption to solve the power problem with multirotor. In this area, we have tried to reduce energy consumption by optimizing the trajectory minimizing the jerk. How much this jerk reduction in trajectory generation would reduce energy consumption should be studied for future work.

In addition, more work on optimizing SGRA and using it for real-time trajectory optimization would solve the problem of finding an optimization solver that is fast enough to solve an NLP so that it can be applied in real time.

REFERENCES

[1] Morbidi, F., Cano, R., and Lara, D., "Minimum-Energy Path Generation for a Quadrotor UAV", *IEE International Conference on Robotics and Automation*, Stockholm: 2016.

[2] Mahony, R., Kumar, V., and Corke, P., "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor", *IEEE Robotics & Automation Magazine*, Vol. 19, No. 3, 2012, pp. 20-32.

[3] Theys, B., Dimitriadis, G., Hendrick, P., and Schutter, J. D., "Influence of propeller configuration on propulsion system efficiency of multi-rotor Unmanned Aerial Vehicles," *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2016.

[4] Tsay, T., "Guidance and Control Laws for Quadrotor UAV", *WSEAS TRANSACTIONS on SYSTEMS and CONTROL*, Vol. 9, 2014, pp. 606-613.

[5] Li, Q., "Grey-Box System Identification of a Quadrotor Unmanned Aerial Vehicle," thesis, 2014.

[6] YAOU, Z. H. A. N. G., WANSHENG, Z. H. A. O., TIANSHENG2 , L. U., and LI, "The attitude control of the four-rotor unmanned helicopter based on feedback linearization control," *WSEAS TRANSACTIONS on SYSTEMS*, vol. 12, Apr. 2013.

[7] L'Afflitto, A., and Anderson, R., "Control AR Drone Parrot 2.0 with Matlab 2015a and Vicon - File Exchange - MATLAB Central", *Mathworks.com* Available: https://www.mathworks.com/matlabcentral/fileexchange/59177-control-ar-drone-parrot-2-0-with-matlab-2015a-and-vicon.

[8] Cherfan, R., "MotiveExample - File Exchange - MATLAB Central", *Mathworks.com*Available: https://www.mathworks.com/matlabcentral/fileexchange/49085-motiveexample.

[9] Sanabria, D., "AR Drone Simulink Development-Kit V1.1 - File Exchange – MATLAB Central", *Mathworks.com* Available: https://www.mathworks.com/matlabcentral/fileexchange/43719-ar-drone-simulink-development-kit-v1-1.

[10] Lee, D., "AR.Drone 2.0 Support from Embedded Coder - File Exchange – MATLAB Central", *Mathworks.com* Available: https://www.mathworks.com/matlabcentral/fileexchange/48558-ar-drone-2-0-support-from-embedded-coder.

[11] Cichella, V., Kaminer, I., Xargay, E., Dobrokhodov, V., Hovakimyan, N., Aguiar, A., and Pascoal, A., "A Lyapunov-based approach for Time-Coordinated 3D Path-Following of Multiple Quadrotors", *51st IEEE Conference on Decision and Control*, Maui: 2012, pp. 1776-1781.

[12] "Developer Tools Downloads," *OptiTrack*Available: https://optitrack.com/downloads/developer-tools.html.

[13] "Calibration - NaturalPoint Product Documentation Ver 2.0", *V20.wiki.optitrack.com* Available: https://v20.wiki.optitrack.com/index.php?title=Calibration.

[14] "Write Level-2 MATLAB S-Functions- MATLAB & Simulink", *Mathworks.com*Available: https://www.mathworks.com/help/simulink/sfg/writing-level-2-matlab-s-functions.html#brgscav-1.

[15] Valenti, M., Dale, D., How, J., Farias, D. P. D., and Vian, J., "Mission Health Management for 24/7 Persistent Surveillance Operations," *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2007.

[16] Junaid, A. B., Lee, Y., and Kim, Y., "Design and implementation of autonomous wireless charging station for rotary-wing UAVs," *Aerospace Science and Technology*, vol. 54, 2016, pp. 253–266.

[17] Junaid, A., Konoiko, A., Zweiri, Y., Sahinkaya, M., and Seneviratne, L., "Autonomous Wireless Self-Charging for Multi-Rotor Unmanned Aerial Vehicles," *Energies*, vol. 10, 2017, p. 803.

[18] Sani, M. F., and Karimian, G., "Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors," *2017 International Conference on Computer and Drone Applications (IConDA)*, 2017.

[19] Carreira, T. G., "Quadcopter Automatic Landing on a Docking Station," thesis, 2013.

[20] Levy, A. V., and Guerra, V., *On the optimization of constrained functions: comparison of sequential gradient-restoration algorithm and gradient projection algorithm*, Houston: Rice University, 1975.

[21] Box, M. J., "A Comparison of Several Current Optimization Methods, and the use of Transformations in Constrained Problems," *The Computer Journal*, vol. 9, Jan. 1966, pp. 67–77.

[22] Hock, W., and Schittkowski, K., "The Test Problems," *Lecture Notes in Economics and Mathematical Systems Test Examples for Nonlinear Programming Codes*, 1981, pp. 23–127.

[23] Svanberg, K., 2007.

[24] Mueller, M. W., Hehn, M., and Dandrea, R., "A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[25] Michini, B., Toksoz, T., Redding, J., Michini, M., How, J., Vavrina, M., and Vian, J., "Automated Battery Swap and Recharge to Enable Persistent UAV Missions," *Infotech@Aerospace 2011*, 2011.

[26] Ure, N. K., Chowdhary, G., Toksoz, T., How, J. P., Vavrina, M. A., and Vian, J., "An Automated Battery Management System to Enable Persistent Missions With Multiple Aerial Vehicles," *IEEE/ASME Transactions on Mechatronics*, vol. 20, 2015, pp. 275–286.

[27] Chang, T., and Yu, H., "Improving Electric Powered UAVs' Endurance by Incorporating Battery Dumping Concept," *Procedia Engineering*, vol. 99, 2015, pp. 168–179.

[28] Gurdan, D., Stumpf, J., Achtelik, M., Doth, K.-M., Hirzinger, G., and Rus, D., "Energy-efficient Autonomous Four-rotor Flying Robot Controlled at 1 kHz," *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007.

[29] Driessens, S., and Pounds, P., "The Triangular Quadrotor: A More Efficient Quadrotor Configuration," *IEEE Transactions on Robotics*, vol. 31, 2015, pp. 1517–1526.

[30] Yacef, F., Bouhali, O., and Hamerlain, M., "Optimization of Energy Consumption for Quadrotor UAV," *International Micro Air Vehicle Conference and Flight Competition*, 2017, pp. 215–222.

[31] Fronza, I., Corral, L., Ioini, N. E., and Ibershimi, A., "Towards Optimization of Energy Consumption of Drones with Software-Based Flight Analysis," *Proceedings of the 28th International Conference on Software Engineering and Knowledge Engineering*, Jan. 2016.

[32] Cichella, V., Kaminer, I., Walton, C., and Hovakimyan, N., “Optimal Motion Planning for Differentially Flat Systems Using Bernstein Approximation,” *IEEE Control Systems Letters*, vol. 2, 2018, pp. 181–186.

[33] Eager, D., Pendrill, A.-M., and Reistad, N., “Beyond velocity and acceleration: jerk, snap and higher derivatives,” *European Journal of Physics*, vol. 37, 2016, p. 065008.

[34] Smith, G., “Newton's Philosophiae Naturalis Principia Mathematica,” *Stanford Encyclopedia of Philosophy*Available: https://plato.stanford.edu/entries/newton-principia/index.html#NewLawMot.

[35] Jeurgens, N., "Identification and Control Implementation of an AR.Drone 2.0", Masters, Eindhoven University of Technology, 2017.

[36] Bangura, M., and Mahony, R., "Real-time Model Predictive Control for Quadrotors", *The International Federation of Automatic Control*, Cape Town: 2014, pp. 11773-11780.

[37] Mahony, R., Kumar, V., and Corke, P., "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor", *IEEE Robotics & Automation Magazine*, Vol. 19, No. 3, 2012, pp. 20-32.

[38] Mellinger, D., "Trajectory Generation and Control for Quadrotors", PhD, University of Pennsylvania, 2012.

[39] "UAV Applications /// Drones for Inspection, Surveying & more", Ascending Technologies Available: http://www.asctec.de/en/uav-uas-drone-applications/.

[40] Veeraklaew, T., Piromsopa, P., Chirungsarpsook, K., and Pattaravarangkur, C., “A Study on the Comparison between Minimum Jerk and Minimum Energy of Dynamic Systems,” *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC06).*

APPENDIX I

NATNETSFUNCTION

```
function NatNetsFunction1(block)
% Level-2 MATLAB file S-Function for unit delay demo.
%   Copyright 1990-2009 The MathWorks, Inc.
%   $Revision: 1.1.6.3 $

setup(block);

function setup(block)

block.NumDialogPrms  = 1;

%% Register number of input and output ports
block.NumInputPorts  = 2;
block.NumOutputPorts = 2;
%edited by Kidus for 2 outports
%% Setup functional port properties to dynamically
%% inherited.
%block.SetPreCompInpPortInfoToInherited;
%block.SetPreCompOutPortInfoToInherited;

block.InputPort(1).Dimensions        = 1;
block.InputPort(2).Dimensions        = 1;
%added by Kidus for two ports
block.InputPort(1).DirectFeedthrough = false;

block.OutputPort(1).Dimensions       = 6;
block.OutputPort(2).Dimensions       = 6;

%% Set block sample time to [0.01 0]
block.SampleTimes = [0.01 0];
%%%%%%%%%%%%%%%%%%%%%%%%%% Changed from 0.01 to 0.02 %%%%%%%%%%%%%%%
%%Set the block simStateCompliance to default(i.e.,same as a built-in block)
block.SimStateCompliance = 'DefaultSimState';

%% Register methods
block.RegBlockMethod('PostPropagationSetup', @DoPostPropSetup);
block.RegBlockMethod('InitializeConditions', @InitConditions);
block.RegBlockMethod('Outputs',                 @Output);
block.RegBlockMethod('Update',                  @Update);
block.RegBlockMethod('Terminate',               @Terminate);

%added by Kidus
block.RegBlockMethod('SetInputPortSamplingMode',@SetInputPortSamplingMode);
%      block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
%      block.RegBlockMethod('Outputs', @Output);
```

```
47
48
49    function SetInputPortSamplingMode(block, idx, fd)
50    block.InputPort(idx).SamplingMode = fd;
51    block.InputPort(idx).SamplingMode = fd;
52
53    block.OutputPort(1).SamplingMode = fd;
54    block.OutputPort(2).SamplingMode = fd;
55    %endfunction added by Kidus
56
57    function DoPostPropSetup(block)
58
59    %% Setup Dwork
60    block.NumDworks = 1;
61    block.Dwork(1).Name = 'x0';
62    block.Dwork(1).Dimensions      = 1;
63    block.Dwork(1).DatatypeID      = 0;
64    block.Dwork(1).Complexity      = 'Real';
65    block.Dwork(1).UsedAsDiscState = true;
66
67    %endfunction
68
69    function InitConditions(block)
70
71    %% Initialize Dwork
72    block.Dwork(1).Data = block.DialogPrm(1).Data;
73
74    display('NatNet Sample Begin')
75
76    global theClient;
77    global frameRate;
78    %     global TimerData;
79    lastFrameTime = -1.0;
80    lastFrameID = -1.0;
81    % approach 1 : poll for mocap data in a tight loop using
GetLastFrameOfData
82    usePollingLoop = false;
83    % approach 2 : poll using a Matlab timer callback ( better for
UI based apps )
84    usePollingTimer = false;
85    % approach 3 : use event callback from NatNet (no polling)
86    useFrameReadyEvent = true;
87    %  useUI = true;
88
89    % Add NatNet .NET assembly so that Matlab can access its
methods, delegates, etc.
90    % Note : The NatNetML.DLL assembly depends on NatNet.dll, so
make sure they
91    % are both in the same folder and/or path if you move them.
92    display('[NatNet] Creating Client.')
93    % TODO : update the path to your NatNetML.DLL file here :
94    %%%%%%%%%%%%%%%%% CHANGED HERE %%%%%%%%%%%%%%%%%%
95    %dllPath =
fullfile('c:','NatNetSDK2.5','Samples','bin','NatNetML.dll');
96    dllPath = fullfile('c:','Users','kidus.guye','Desktop','Thesis
Softwares',...
97        'NatNetSDK','lib','x64','NatNetML.dll');
```

```
assemblyInfo = NET.addAssembly(dllPath);

% Create an instance of a NatNet client
theClient = NatNetML.NatNetClientML(0); % Input = iConnectionType: 0 = Multicast, 1 = Unicast
version = theClient.NatNetVersion();
fprintf( '[NatNet] Client Version : %d.%d.%d.%d\n', version(1),...
    version(2), version(3), version(4) );

% Connect to an OptiTrack server (e.g. Motive)
display('[NatNet] Connecting to OptiTrack Server.')
hst = java.net.InetAddress.getLocalHost;


%%%%%%%%%%%%%%%%% CHANGED HERE %%%%%%%%%%%%%%%%%%
%HostIP = char(hst.getHostAddress);
HostIP = char('127.0.0.1');%Get IP
flg = theClient.Initialize(HostIP, HostIP); % Flg = returnCode: 0 = Success
if (flg == 0)
display('[NatNet] Initialization Succeeded')
else
display('[NatNet] Initialization Failed')
end

% print out a list of the active tracking Models in Motive
GetDataDescriptions(theClient)

% Test - send command/request to Motive
[byteArray, retCode] = theClient.SendMessageAndWait('FrameRate');
if(retCode ==0)
byteArray = uint8(byteArray);
frameRate = typecast(byteArray,'single');
end
global theClient;
global data;
data = theClient.GetLastFrameOfData();

function Output(block)

%block.OutputPort(1).Data = block.Dwork(1).Data;
%         global theClient;
global data;
%         java.lang.Thread.sleep(0.1);
%         data = theClient.GetLastFrameOfData();
D=ProcessFrame(data);

block.OutputPort(1).Data=double([D.x;D.y;D.z;D.angleX;D.angleY;D.angleZ]);
block.OutputPort(2).Data = double([D.x1;D.y1;D.z1;D.angleX1;D.angleY1;...
    D.angleZ1]);
%edited by Kidus Guye to check for two rigid
%         bodies at the same time

```

```
148
149
150   function Update(block)
151   global theClient;
152   global data;
153   block.Dwork(1).Data = block.InputPort(1).Data;
154   data = theClient.GetLastFrameOfData();
155
156   %endfunction
157
158   function Terminate(block)
159   global theClient;
160   %       global TimerData;
161   theClient.Uninitialize();
162   %      % Sheng's test on polling data CLEAN UP!!
163   %      stop(TimerData);
164   %      delete(TimerData);
165   %      % eng Sheng's clean up
166
167   disp('NatNet Sample End')
168
169
170   %endfunction
171
172   function [D] = ProcessFrame( frameOfData )
173
174   rigidBodyData = frameOfData.RigidBodies(1);
175   rigidBodyData1 = frameOfData.RigidBodies(2);
176   % Position
177
178   D.x = rigidBodyData.z;
179   D.y = -rigidBodyData.x;
180   D.z = -(rigidBodyData.y);
181   %second rigid body
182   D.x1 = rigidBodyData1.z;
183   D.y1 = -rigidBodyData1.x;
184   D.z1= -(rigidBodyData1.y);
185   %edited Kidus for two rigid
186
187   q = quaternion( rigidBodyData.qx, rigidBodyData.qy,
rigidBodyData.qz,...
188       rigidBodyData.qw );
189
190   qRot = quaternion( 0, 0, 0, 1);% rotate pitch 180 to avoid
180/-180 flip for nicer graphing
191   q = mtimes(q, qRot);
192   %
193   %%edited by Kidus to get the yaw, pitch and roll angles
194   angles = EulerAngles(q,'zyx');
195   D.angleX = -angles(1)* 180.0 / pi;   % must invert due to 180
flip above
196   D.angleY = angles(2)*180.0 / pi;
197   D.angleZ = -angles(3)*180.0 / pi;   % must invert due to 180
flip above
198   %added by Kidus
199   q1 =
quaternion(rigidBodyData1.qx,rigidBodyData1.qy,rigidBodyData1.qz,...
```

```
200         rigidBodyData1.qw );
201
202     qRot1 = quaternion( 0, 0, 0, 1); % rotate pitch 180 to avoid
180/-180 flip for nicer graphing
203     q1 = mtimes(q1, qRot1);
204     %
205      %edited by Kidus to get the yaw, pitch and roll angles
206     angles1 = EulerAngles(q1,'zyx');
207     D.angleX1 = -angles1(1)* 180.0 / pi;   % must invert due to 180
flip above
208     D.angleY1 = angles1(2)*180.0 / pi;
209     D.angleZ1 = -angles1(3)*180.0 / pi;
210     %Edited by Kidus for the second RGB
211
212     function GetDataDescriptions( theClient )
213
214     dataDescriptions = theClient.GetDataDescriptions();
215
216     % print out
217     fprintf('[NatNet] Tracking Models : %d\n\n',
dataDescriptions.Count);
218     for idx = 1 : dataDescriptions.Count
219     descriptor = dataDescriptions.Item(idx-1);
220     if(descriptor.type == 0)
221     fprintf('\tMarkerSet \t: ');
222     elseif(descriptor.type == 1)
223     fprintf('\tRigid Body \t: ');
224     elseif(descriptor.type == 2)
225     fprintf('\tSkeleton \t: ');
226     else
227     fprintf('\tUnknown data type : ');
228     end
229     fprintf('%s\n', char(descriptor.Name));
230     end
231
232     for idx = 1 : dataDescriptions.Count
233     descriptor = dataDescriptions.Item(idx-1);
234     if(descriptor.type == 0)
235     fprintf('\n\tMarkerset : %s\t(%d markers)\n',
char(descriptor.Name),...
236         descriptor.nMarkers);
237     markerNames = descriptor.MarkerNames;
238     for markerIndex = 1 : descriptor.nMarkers
239     name = markerNames(markerIndex);
240     fprintf('\t\tMarker : %-20s\t(ID=%d)\n',
char(name),markerIndex);
241     end
242     elseif(descriptor.type == 1)
243     fprintf('\n\tRigid Body : %s\t\t(ID=%d, ParentID=%d)\n',...
244         char(descriptor.Name),descriptor.ID,descriptor.parentID);
245     elseif(descriptor.type == 2)
246     fprintf('\n\tSkeleton : %s\t(%d bones)\n',
char(descriptor.Name),...
247         descriptor.nRigidBodies);
248     %fprintf('\t\tID : %d\n', descriptor.ID);
249     rigidBodies = descriptor.RigidBodies;
250     for boneIndex = 1 : descriptor.nRigidBodies
```

```
251   rigidBody = rigidBodies(boneIndex);
252   fprintf('\t\tBone : %-20s\t(ID=%d, ParentID=%d)\n',
char(rigidBody.Name),...
253       rigidBody.ID, rigidBody.parentID);
254   end
255   end
256   end
257   %endfunction
```

APPENDIX II

INITIAL VARIABLES

```
1     sampleTime = 0.01;
2
3     kz = 1;
4     kzi = 0;
5     %pitch/forward velocity controller gains
6     Kpt = 0.4;%pid proportional gain
7     Kit = 0;%pid integral gain
8     Kdt =0.45;%pid derivative gain
9     Kanglet = 0.3;
10    %roll/lateral velocity controller gains
11    Kpf = 0.45;%pid proportional gain
12    Kif = 0;%pid integral gain
13    Kdf =0.45;%pid derivative gain
14    Kanglef = -0.2;
15
16    %yaw rate controller gains
17    kpsi = 1.9;
18    kpsil =0.075;
```

APPENDIX III

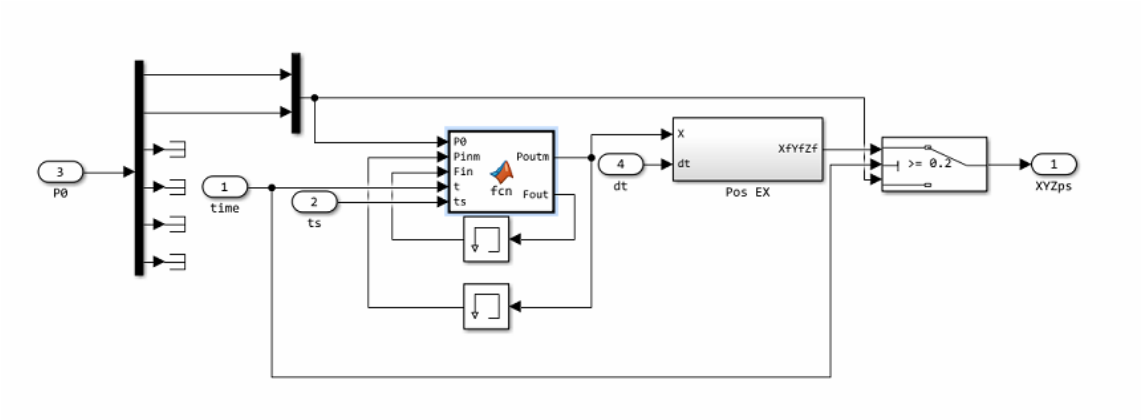

Figure 37. x and y position capture at t = t_s

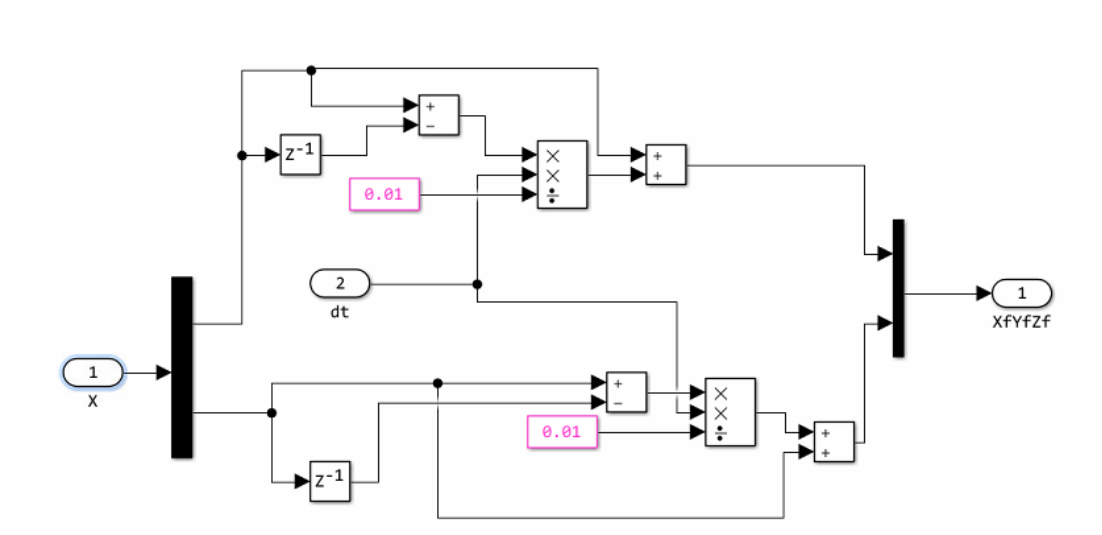

Figure 38. The second initial point extrapolation block

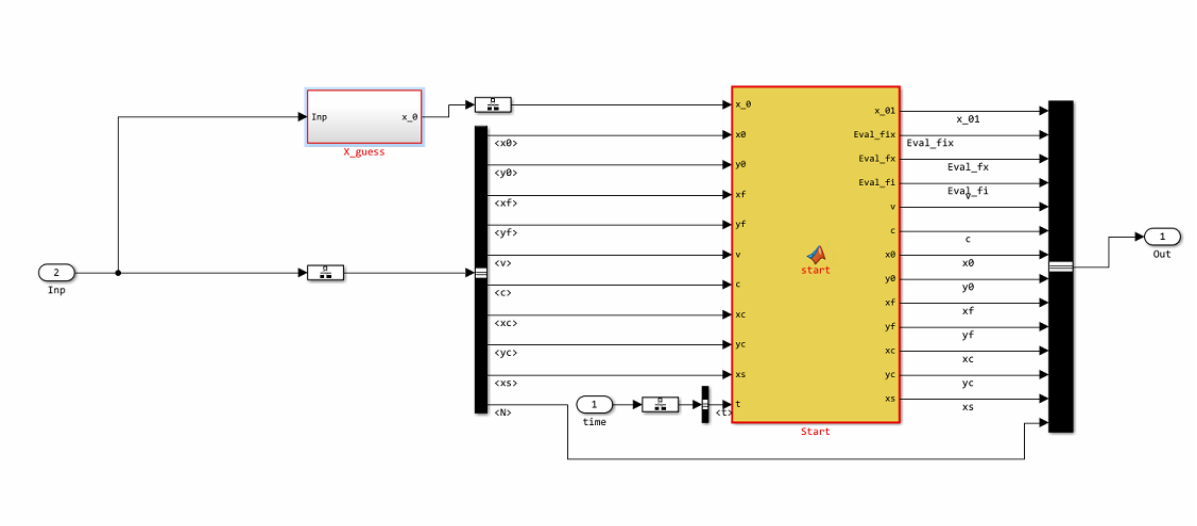

Figure 39. Initial guess starter block

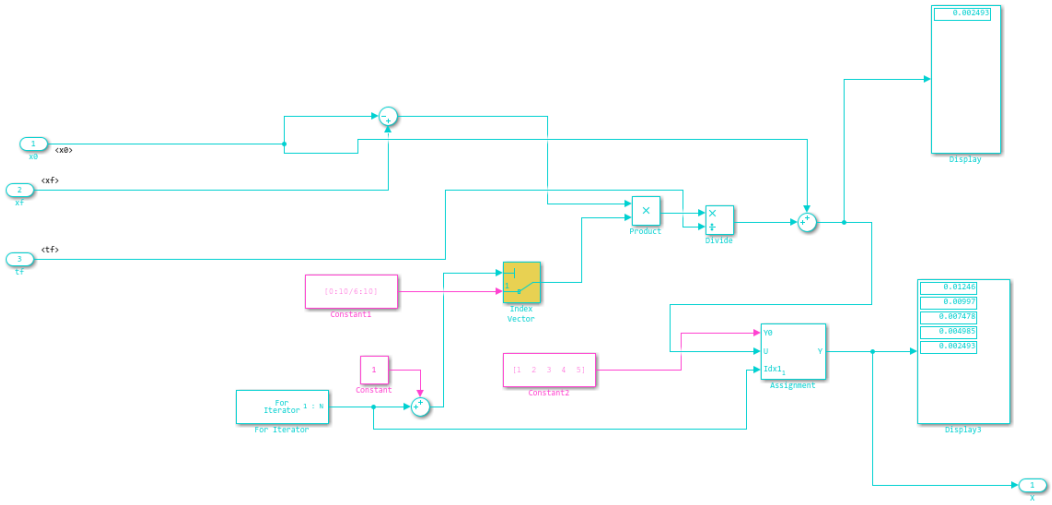

Figure 40. For loop to calculate the initial guess

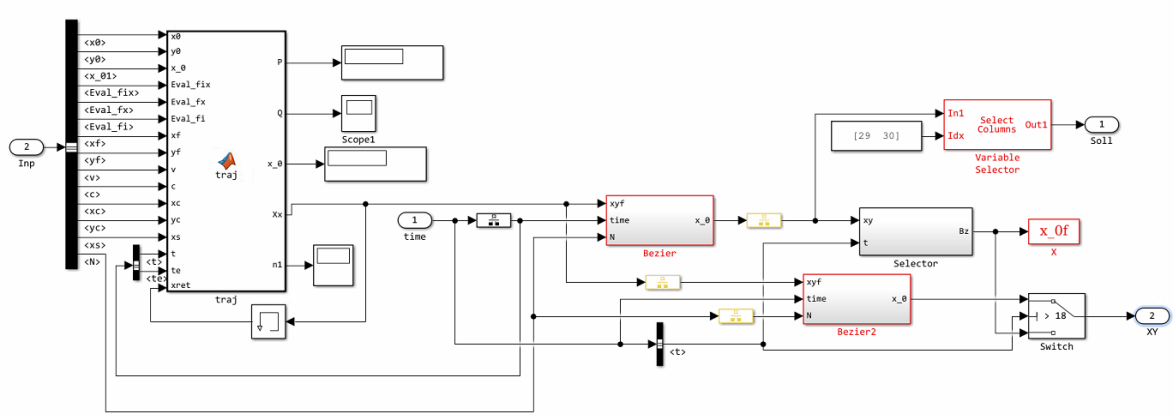

Figure 41. SGRA block content

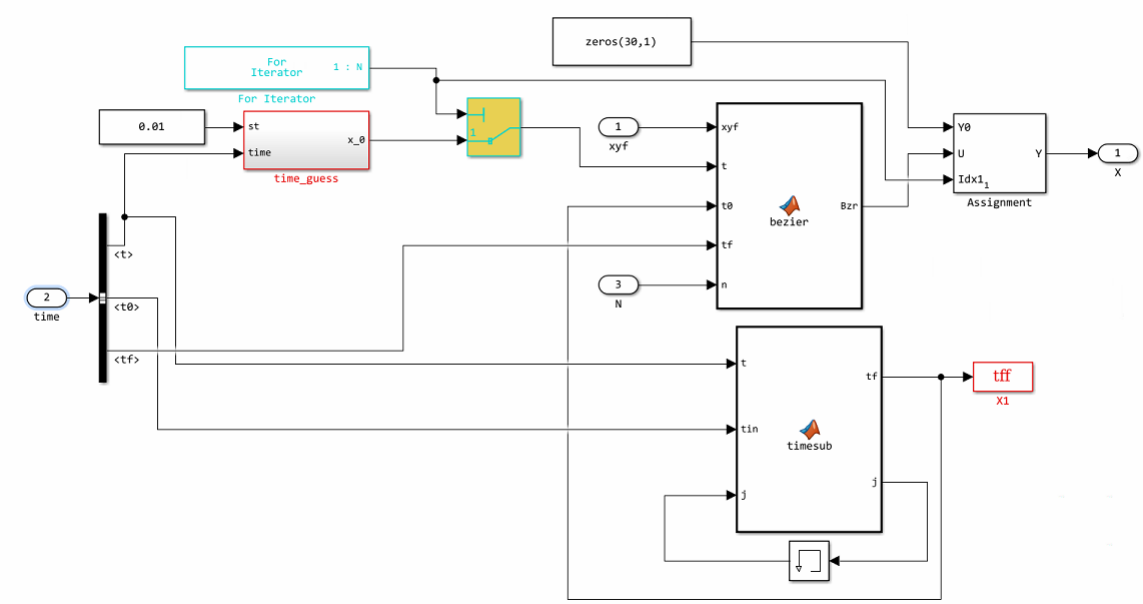

Figure 42. Bezier Curve block

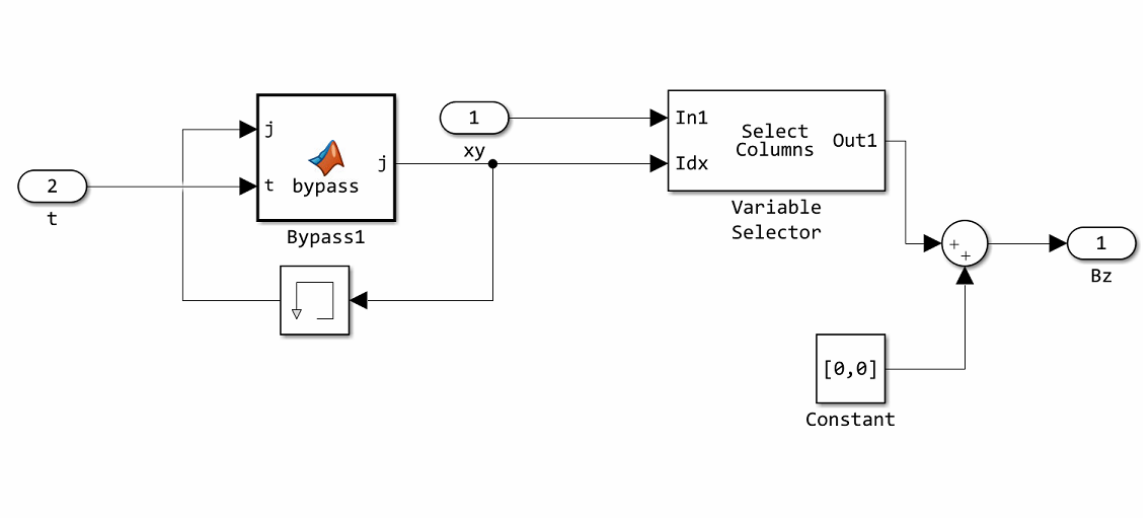

Figure 43. Bezier curve points selector

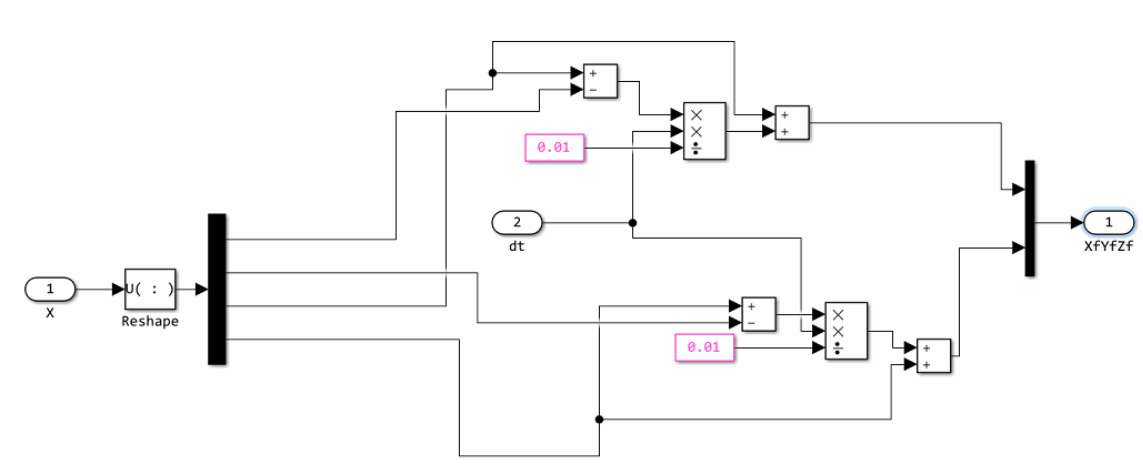

Figure 44. Extrapolation for the third and afterwards initial points