5-2013

MARIT : the design, implementation and trajectory generation with NTG for small UAVs.

Yinan Cui
University of Louisville

MARIT: THE DESIGN, IMPLEMENTATION AND TRAJECTORY GENERATION
WITH NTG FOR SMALL UAVS

By

Yinan Cui
B.S., Zhejiang University, Hangzhou, China, 2005, EE
M.S., Royal Institute of Technology (KTH), Stockholm, Sweden, 2007, EMIS

A Dissertation
Submitted to the Faculty of the
J.B. Speed School of Engineering of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering

May 2013

MARIT: THE DESIGN, IMPLEMENTATION AND TRAJECTORY GENERATION
WITH NTG FOR SMALL UAVS

Submitted by

Yinan Cui

A Dissertation Approved on

18 April 2013

by the Following Reading and Examination Committee:

_____

Tamer Inanc, Ph.D., Thesis Director

_____

Jacek Zurada, Ph.D.

_____

Cindy Harnett, Ph.D.

_____

Ibrahim Imam, Ph.D.

_____

Robert Powers, Ph.D.

ABSTRACT

MARIT: THE DESIGN, IMPLEMENTATION AND TRAJECTORY GENERATION
WITH NTG FOR SMALL UAVS

Yinan Cui

2013-04-18

This dissertation is about building a Multiple Air Robotics Indoor Testbed (MARIT) for the purpose of developing and validating new methodologies for collaboration and cooperation between heterogeneous Unmanned Air Vehicles (UAVs) as well as expandable to air-and-ground vehicle teams. It introduces a mathematical model for simulation and control of quadrotor Small UAVs (SUAVs). The model is subsequently applied to design an autonomous quadrotor control and tracking system.

The dynamics model of quadrotor SUAV is used in several control designs. Each control design is simulated and compared. Based on the comparison, the superior control design is use for experimental flights. Two methods are used to evaluate the control and collect real-time data.

The Nonlinear Trajectory Generation (NTG) software package is used to provide optimal trajectories for the SUAVs in MARIT. The dynamics model of the quadrotor is programmed in NTG and various obstacle avoidance scenarios are modeled to establish a platform for optimal trajectory generation for SUAVs. To challenge the capability of NTG for real-time trajectory generation, random obstacles and disturbances are simulated. Various flight simulations validate this trajectory tracking approach.

Key words: UAV, Testbed, Quadrotor, Dynamic Modeling, Optimal Control,

Trajectory Generation, Trajectory Tracking.

ACKNOWLEDGMENTS

TABLE OF CONTENTS

LIST OF FIGURES

xi

CHAPTER 1
INTRODUCTION

Unmanned aerial vehicles (UAVs), also known as drones, or remotely piloted vehicles (RPVs) have become one of the fastest growing sectors in aero-space industry. A UAV is capable of completing controlled, sustained level flight, while reducing the risk of human life and lowering operational costs. According to the estimation of a recent report [7], the UAVs market will double in the next 10 years from current worldwide UAV R&D and procurement expenditures of $5.9 billion to $15.1 billion. The dynamic growth results from the great potential of UAVs in vast areas. In this section, a general background of UAVs, including a brief history of UAVs and some of their successful applications in various industries will be presented. A Micro UAV (MAV) is a special type of UAV with small size and light weight. The research in MAVs makes another growing field in the UAV industry. The MAVs can be carried by humans or any vehicle and operate in various situations. Figure 1.1 shows several MAV products.

1

FIGURE 1.1: Examples of Micro UAVs (MAVs).

Controlling Micro-unmanned Air Vehicles (MAVs) in a confined environment provides a convenient platform for the study of developing and validating new technologies for collaboration and cooperation between heterogeneous Unmanned Air Vehicles (UAVs). Because of its low cost, simple components and indoor capability, the MAVs have become the research subjects of more and more institutions. While MAVs usually come with simpler sub-systems and components, controlling one is not a trivial task. Even flying a hobby RC plane requires training and practice. It is desired to have a platform in which multiple MAVs can be controlled easily with minimum risk of damage.

This research is devoted to constructing a Multiple Air Robotics Indoor Testbed (MARIT) and the development of control algorithms and trajectory generation for MAVs. An indoor testbed was chosen as opposed to an outdoor one because the latter is subject to change due to weather, temperature and many other factors. On the other hand, an indoor testbed, once established, is always available to use. Moreover, flying vehicles outdoors brings dangers to both human operators and the vehicles themselves; an indoor

testbed makes it easier to build protection mechanisms so damage could be reduced.

With the testbed constructed, multiple MAVs will be controlled automatically to fly within an enclosed area. The control signals are sent from RC controllers that are connected to PCs where the control commands are generated. The testbed is equipped with overhead high speed Cameras so that the fast movements of the MAVs can be captured. Captured data is sent to the controller units to form a closed-loop system, so the MAVs can be controlled automatically. Users of the testbed can develop their own controller programs and evaluate the performance. The six degrees of freedom (DOF) of a MAV provides the experiments with various features including sharp turns, high speed obstacle avoidance and optimal trajectory tracking. Figure 1.2 illustrates the feedback mechanism of the testbed.
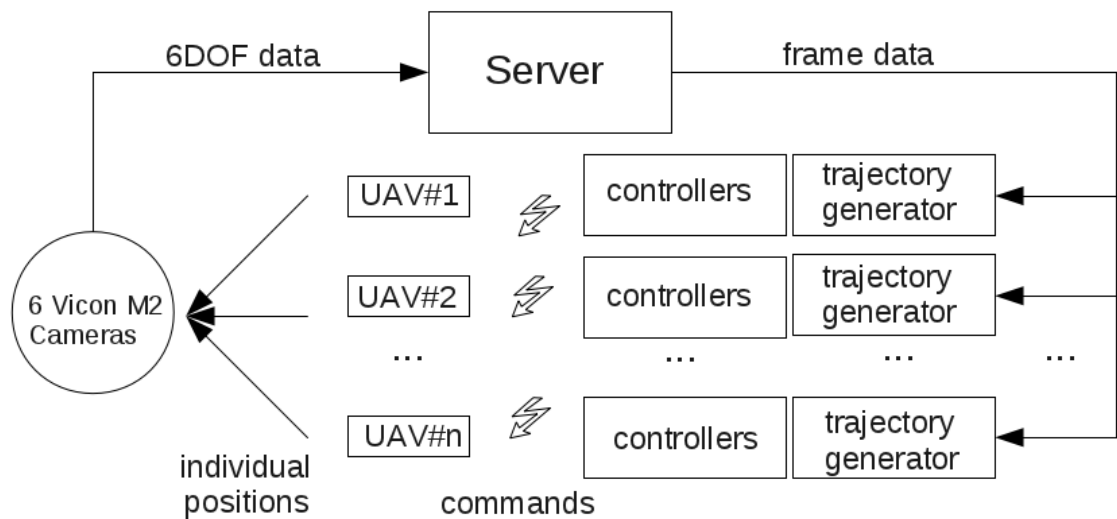
FIGURE 1.2: An overview of the structure of MARIT. A closed-loop UAV control system is build with Vicon Motion Capture systems [18], Draganflyer quadrotors [20], controller and trajectory generating units, and the data hosting server.

1.1 UAV: A Brief History

MAVs originate from the UAV family. The research and development of UAVs

3

can find its way back to as early as 1848. The Austrians used unmanned battle balloons to attack the Italian city of Venice [1]. A large group of pilot-less Austrian balloons loaded with explosives were launched towards Venice, see Figure 1.3.

ЧЕРТЕЖИ КЪ СТ. ВОЗДУХОПЛАВАНIЕ.
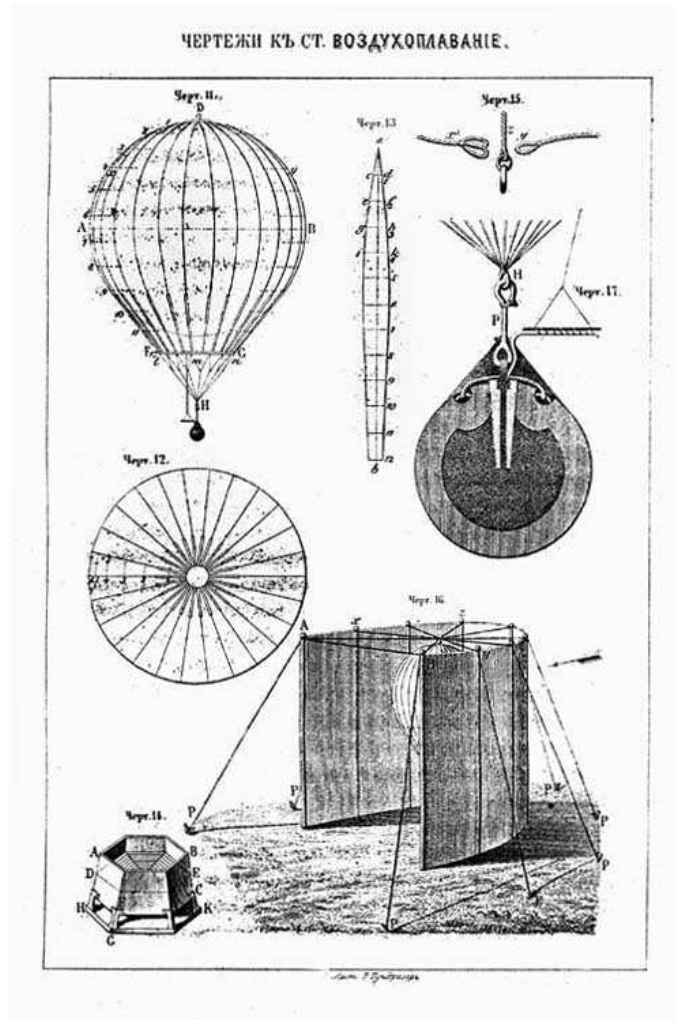
FIGURE 1.3: Bombing by Balloon, 1848. An creative weapon the Austrians used against the Italian city of Venice. Unmanned balloons carrying explosives caused back-fires due to undesirable winds.

With the help of favorable winds and timed explosive fuses, some of the bombs exploded as planned. This action, though these balloons do not qualify as modern UAVs, might be the first instance of its type. Similar ideas were tested in many other occasions.

For example in the American Civil War, the North and South launched balloons carrying explosives that would drop into the other side's ammunition depot and make damage. These early examples took the advantage of favorable high altitude winds, which is a highly uncertain factor. While these ideas proved to be not effective and causing backfires, the exploration in the UAV area never stopped.

The development in radio control (RC) in the early 20th century made remote control of a UAV possible. Several traditional aircrafts were modified and converted into RC UAVs. Examples are Standard E-1 drone, a modification to an early American Army fighter aircraft [8], the Larynx (Figure 1.4), a British unmanned aircraft used as a guided anti-ship weapon, and the Fairey Queen RC target aircraft, a model based on the British reconnaissance biplane model Fairey III. These early experiments generated promising results, which brought the development of UAVs into a new era.
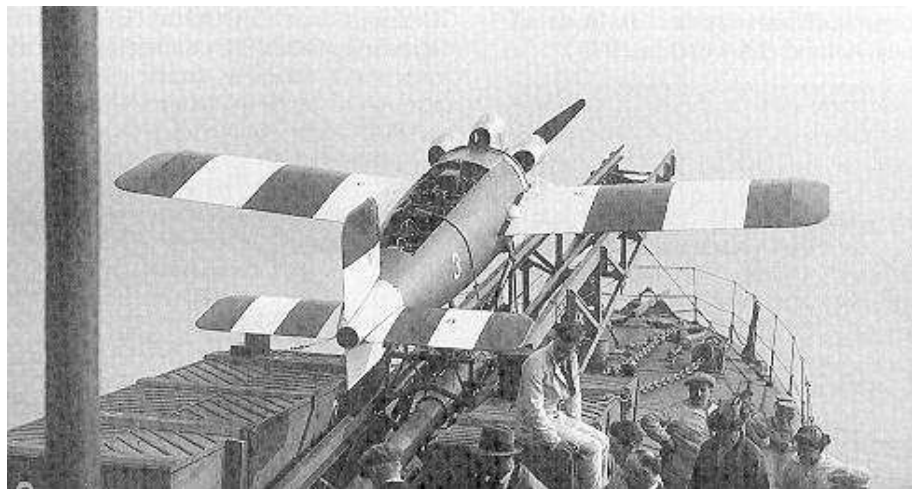
FIGURE 1.4: Larynx unmanned aircraft, the prototype of the Standard E-1 drone, was used as a guided anti-ship weapon.

In the 1930s, the US Navy developed an RC aircraft that could be remotely controlled from another aircraft, and named it "N2C-2" (Figure 1.5). Soon after the invention of N2C-2, the U.S. Army Air Force (USAAF) adopted the concept and performed many successful experiments, including delivering torpedo attack remotely, and crash-

ing into moving objects. In World War II the US utilized RC aircrafts in the combat against Japan in the Russel Islands and Solomon Islands to attack Japanese merchant ships, and most were effective [9].

FIGURE 1.5: N2C-2, first US remotely piloted aircraft (1936), Delmer Fahrney Collection

The research in UAVs continued after World War II. More manufacturers produced UAVs in their own areas. The "OQ-2" UAV drone by Radioplane were modified and used as basic training target. The "ADM-20 Quail" by McDonnell Douglas, shown in Figure 1.6, could be carried by Boeing B-52 Stratofortress bombers and used as decoys. Several UAVs even contributed to nuclear tests. The B-17 Flying Fortress by Boeing and Grumman F6F Hellcat were both sent to fly over nuclear clouds directly above the explosion to collect data. Reconnaissance is also an area that UAVs are widely used, examples are the Boeing "Compass Copes" and the Lockheed "D-21". The US used thousands of Ryan reconnaissance drones in the Vietnam War [10].

FIGURE 1.6: McDonnell ADM-20C-40-MC "Quail" could be carried by Boeing B-52 bombers and used as decoys.

In the modern era, more and more UAVs equipped with advanced technology are being applied in vast fields. As a modern battlefield UAV, the "Predator", shown in Figure 1.7, by General Atomics, which is able to stay in the air for 40 hours, was the first deployed UAV to the Balkans in 1995, and to Iraq in 1996. Modern UAVs are not limited to military uses. Many UAVs have already been successfully applied in agricultural industry, weather research, mineral exploration, cost watch, as well as robotics exploration of remote planets and moons by next generation air robotics (aerobots) rovers [3].

FIGURE 1.7: The Predator RQ-1L UAV (General Atomics) is capable of staying for as much as 40 hours in the air.

The advanced technologies nowadays makes the production of small sized UAVs (SUAVs) possible. MAVs are those SUAVs that can be carried by humans because of their small size and light weight. In 1997, the Defense Advanced Research Projects Agency (DARPA) began a multi-year development program to develop MAVs. The goal was to develop very small UAVs that would perform tasks such as carrying night imager with high endurance. Figure 1.8 displays several modern MAVs being produced today. Most of them have vertical take-off and landing (VTOL) design. The MAV being controlled in this research is a similar model called Draganflyer quadrotor [20], which will be described in Chapter 2.

	Name	Size	Weight	Manufacturer
	DelFly Micro	10 cm	3.07 g	MavLab
	Skypbotix CoaX	34cm	280 g	Skybotix
	Aeryon Scout	80cm	1400 kg	Aeryon Labs
	SQ-4 RECON	24 cm	198 g	BCB International
	AIRROBOT	100 cm	900 g	Westminster International
	ZALA 421-21	22 cm	1500 g	Zala Aero

FIGURE 1.8: Several modern MAVs.

1.2 Previous and Parallel Work

In recent years, a lot of work has been done in MAV research. The General Robotics, Automation, Sensing and Perception (GRASP) Laboratory at University of Pennsylvania has been doing research in quadrotor control for more than ten years

[4, 5]. In 2002, E. Altug et al. from GRASP lab used a ground camera to estimate the 6 DOF pose (position and orientation) of a quadrotor, and tested controllers such as backstepping-like control law with simulations. The same group later improved the pose estimation by using dual camera visual feedback, in which an onboard camera was added to the quadrotor. With this improvement, the group were able to apply their proposed pose estimation algorithm and nonlinear control techniques to a tethered quadrotor. During recent years, the GRASP lab upgraded their visual feedback sensors and developed high accuracy control algorithms. Vicon Motion Capture system with high frame rate cameras were introduced to capture the 6 DOF data of Hummingbird quadrotors [11]. The new quadrotors are equipped with onboard micro chip which can implement fast looping attitude control. With these improvement, the GRASP lab is able to display complicated maneuvers of autonomous flights with multiple quadrotors. The Hummingbird quadrotors were able to carry out tasks such as building simple structures with light weight blocks, doing 720 degree somersault while avoiding crashing.

The RAVEN (Real-time indoor Autonomous Vehicle test ENvironment) project in Aerospace Controls Laboratory at Massachusetts Institute of Technology uses a multi-vehicle platform to provide a facility for testing low-level control algorithms [12, 13]. Raven also used Vicon system to capture movements with high frame rate, but introduced a different type of quadrotor called "Draganflyer" [20]. Since January 2006, more than 2500 vehicle experiments have been performed in RAVEN, including approximately 60 flight demonstrations during a 16-h period at the Boeing Technology Exposition at Hanscom Air Force Base near Lexington, Massachusetts [13]. The RAVEN team also solved the battery duration problem of the quadrotor by including charging stations to the arena. After each task was completed, the quadrotors would autonomously fly back and dock onto the charging stations.

The Flying Machine Arena at ETH Zurich features a large-size indoor workspace to enable impressive aerobatics research [14]. The research team performed some inter-

esting tasks with quadrotors such as playing piano with a stick, dancing to the rhythms of music, and controlling the quadrotors with human hand gestures [15].

The Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control (STAR-MAC) project uses an outdoor platform to investigate multi-agent control of quadrotors in real-world scenarios [16]. The Vanderbilt Embedded Computing Platform for Autonomous Vehicle (VECPAV) project at Vanderbilt University built an autonomous intelligent control system that replaces human operators [17].

1.3   Overview and Statement of Contributions

A brief summary and thesis contributions by chapter:

- *Chapter 2*: The designing process is presented in this chapter. The mathematical model of the control subject is built, a simulator of the dynamics model is developed to evaluate the responses to inputs. The general design of controllers based on the model is described, and finally the components of the testbed are explained in details.

- *Chapter 3*: This chapter focuses on the development of controllers of the MARIT quadrotors. Several controllers are designed base on the mathematical model built in Chapter 2, and for each of the controllers, a simulator is developed to evaluate the controller's performance. Moreover, a comparison is performed to discuss the different controllers. Finally, based on the comparison and experiments performed on the actual testbed, the proposed controller is introduced.

- *Chapter 4*: In this chapter, the Nonlinear Trajectory Generation software package (NTG) by Milam et al. [34] is incorporated into MARIT for optimal trajectory generation. The mathematical background of NTG is presented, and MARIT is modeled and programmed in NTG. Various simulations are executed to evaluate

the co-operation of NTG and the system model. Finally, the controller developed in Chapter 3 are used for trajectory tracking.

CHAPTER 2
SYSTEM DESIGN

2.1 System Modeling

Controlling highly compact electronic devices has potential dangers. This is especially true when working with delicate devices like the Draganflyers. An important approach to lower the possibility of damage is acquiring an adequate dynamic model. This section describes the modeling of the dynamics of the quadrotor and the design of controllers and experiments.

2.1.1 Quadrotor Dynamics

A quadrotor has a square body frame and a rotor blade installed on each corner. The opposite pair of blades rotate in the same direction, while the neighboring pair rotate in opposite directions. This configuration removes the need of a tail rotor, which makes the quadrotor different from a typical helicopter. For most quadrotors, all blades are fixed-pitched and parallel, with their air-flow pointing down to get the lift force pointing up. The quadrotor has 6 DOF, namely (defined in earth frame) moving forward/backward, left/right, up/down, and (defined in body frame) rotating around three perpendicular axes (roll, pitch and yaw). As the coordinate systems shown in Figure 2.1, the earth frame is denoted by E and the body frame by B.

13
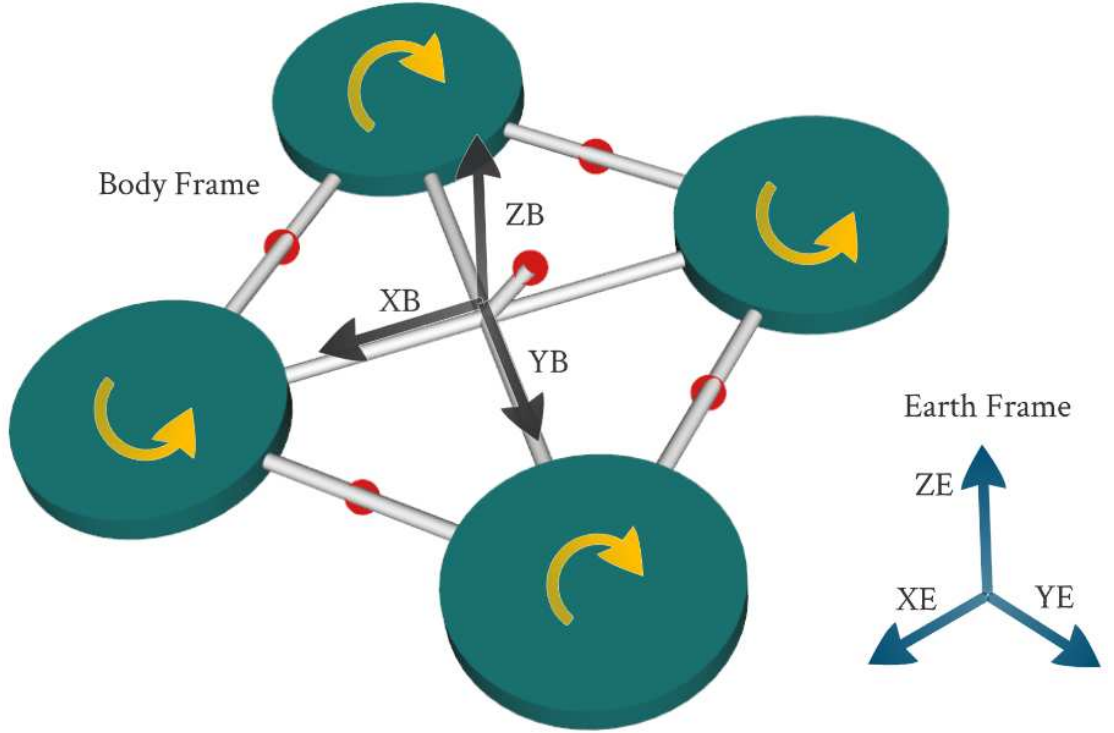
FIGURE 2.1: Body frame and earth frame of a quadrotor. The four plates installed on each corner are the rotors. The opposite pair of blades rotate in the same direction, and the neighboring pair rotate in opposite direction. The small dots on the body frame are markers installed for orientation.

Since the actuators (blade propellers) operate in the B-frame, it is necessary to establish a mapping from the B-frame to E-frame. Following the $Z-Y-X$ convention, to go from E to B, one needs to first rotate around Z_E by angle ψ (yaw), then rotate around Y_E by angle θ (pitch), and at last rotate around X_E by angle ϕ (roll). The $Z-Y-X$ rotation matrix that maps from B back to E is given by $R \in SO3$, as shown in equation (2.1), in which s denotes $sin()$ and c denotes $cos()$.

$$
R = \begin{bmatrix}
c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & s\psi s\phi + c\psi s\theta c\phi \\
s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\
-s\theta & c\theta s\phi & c\theta c\phi
\end{bmatrix}
\tag{2.1}
$$

TABLE 2.1

m	mass of quadrotor
F_i	force on blade propeller
Ω_i	rotor speed
b	thrust factor
d	drag factor
l	lever length
g	gravitational acceleration
τ	torque on frame body
ω	body angular speed
L	diagonal length of the quadrotor
I	body inertia
$I_{xx,yy,zz}$	I around x, y, and z axis
ϕ, θ, ψ	roll, pitch, yaw angles
p, q, r	angular velocity in B-frame

The symbols that will be used in the modeling are listed in Table 2.1. The Newton equations in the E-frame are obtained from the R matrix as:

$$
m \begin{bmatrix} \ddot{x}_E \\ \ddot{y}_E \\ \ddot{z}_E \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ \sum F_{iB} \end{bmatrix} \tag{2.2}
$$

where

$$
\sum F_{iB} = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2)
$$

Expanding (2.2) yields the motion equations in the E-frame as:

$$\ddot{x}_E = (s\psi s\phi + c\psi s\theta c\phi)\frac{(\Omega_1^2+\Omega_2^2+\Omega_3^2+\Omega_4^2)b}{m}$$

$$\ddot{y}_E = (s\psi s\theta c\phi - c\psi s\phi)\frac{(\Omega_1^2+\Omega_2^2+\Omega_3^2+\Omega_4^2)b}{m} \tag{2.3}$$

$$\ddot{z}_E = c\theta c\phi\frac{(\Omega_1^2+\Omega_2^2+\Omega_3^2+\Omega_4^2)b}{m} - g$$

Due to the symmetry of the quadrotor's frame body, the body inertia of the quadrotor could be expressed as:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{2.4}$$

The Euler equations in B-frame is given by

$$I\dot{\omega} + \omega \times (I\omega) = \tau$$

inserting B-frame angular variables, one obtains

$$I\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} bl(\Omega_4^2 - \Omega_2^2) \\ bl(\Omega_3^2 - \Omega_1^2) \\ M \end{bmatrix} \tag{2.5}$$

where $M = d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2)$. From equation (2.4) and (2.5), The quadrotor equations of motion in B-frame is given by :

$$\dot{p} = \frac{I_{yy}-I_{zz}}{I_{xx}}qr + \frac{bl(\Omega_4^2-\Omega_2^2)}{I_{xx}}$$

$$\dot{q} = \frac{I_{zz}-I_{xx}}{I_{yy}}pr + \frac{bl(\Omega_3^2-\Omega_1^2)}{I_{yy}} \tag{2.6}$$

$$\dot{r} = \frac{I_{xx}-I_{yy}}{I_{zz}}pq + \frac{d(-\Omega_1^2+\Omega_2^2-\Omega_3^2+\Omega_4^2)}{I_{zz}}$$

2.2 Controller Design

To construct a closed-loop system, MARIT uses Vicon Motion Capture System [18] with 6 high frequency overhead cameras to capture the 6 DOF data of the quadrotors. A server hosts the frames of all the vehicles and distributes the data to each quadrotor's controller PC through a local network. The controller commands are generated on the controller PCs in real-time and sent to the Optic 6 transmitters by Hitec RC transmitter [21]. The RC transmitter operates on 72.79 Hz. Controller commands are sent through the transmitter's four channels to control the quadrotors. Figure 2.2 illustrates the structure of MARIT.
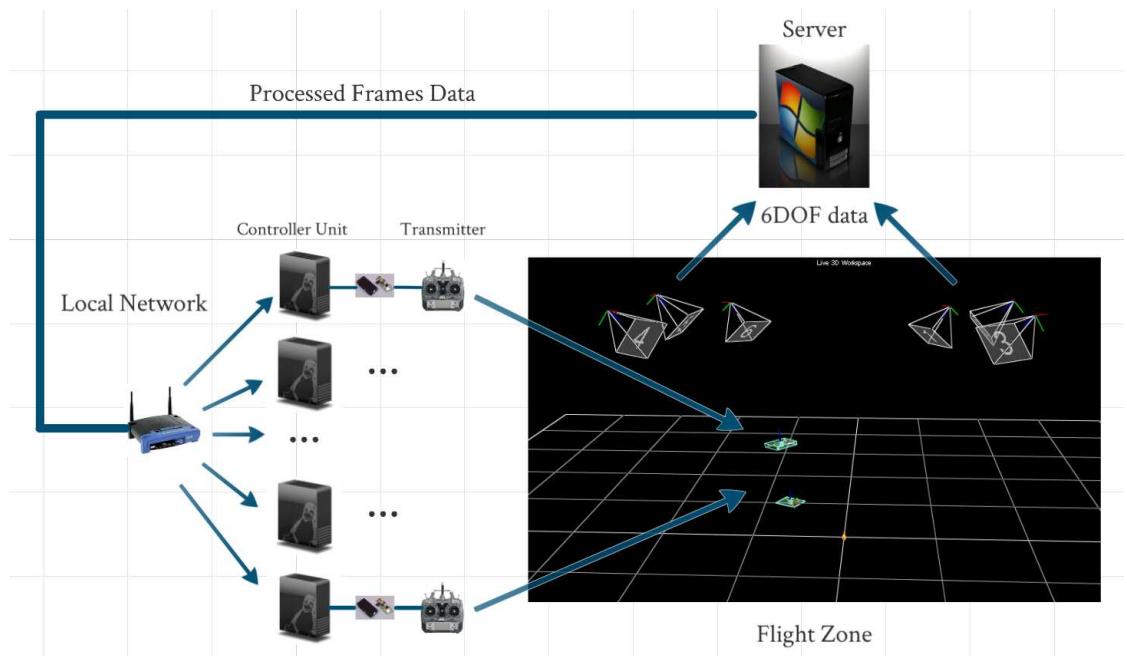
FIGURE 2.2: MARIT control loop. This closed-loop control system consists of high speed cameras, a hosting server, a local network, control units and the quadrotors.

The inputs to the system are the 4 RC channels on each transmitter. Figure 2.3 illustrates the layout of these channels. The four inputs are denoted as u_1, u_2, u_3, and u_4 respectively as in (2.7).

$$u_1 = b(-\Omega_2^2 + \Omega_4^2)$$
$$u_2 = b(\Omega_1^2 - \Omega_3^2)$$
$$u_3 = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \tag{2.7}$$
$$u_4 = d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2)$$

FIGURE 2.3: The RC transmitter channels. Channels 1 to 4 are roll, pitch, height, and yaw controls respectively. The fact that there are only four inputs to control a 6DOF vehicle makes the system under-actuated. Fortunately due to the symmetry of the quadrotor, the controls could be decoupled.

Following the Z-Y-X convention, the body angular speed p, q and r are related to the derivatives of the angles of rotation, i.e. roll (ϕ), pitch (θ) and yaw (ψ) as follows:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -s\theta \\ 0 & c\phi & s\phi c\theta \\ 0 & -s\phi & c\phi c\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.8}$$

When the quadrotor is in a nominal hovering state, ϕ and θ are small, the follow-

ing can be assumed:

$$\phi \approx \theta \approx 0, \psi \approx \psi_0 \tag{2.9}$$

Then the transition matrix in (2.8) approximates a unity matrix. The system is modeled under this circumstance and the controllers are designed with the following assumption:

$$\dot{\phi} \approx p$$
$$\dot{\theta} \approx q \tag{2.10}$$
$$\dot{\psi} \approx r$$

The objective of the control is to maintain and track the 6 DOF values of each quadrotor as desired. With the system inputs u_1 to u_4, one can build the state-space model of the system by choosing:

$$X = [\dot{x}\ \dot{y}\ \dot{z}\ x\ y\ z\ \dot{\phi}\ \dot{\theta}\ \dot{\psi}\ \phi\ \theta\ \psi]^\top \tag{2.11}$$

$$U = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^\top \tag{2.12}$$

$$Y = [x\ \ y\ \ z\ \ \phi\ \ \theta\ \ \psi]^\top \tag{2.13}$$

X is the states vector, U is the input vector and Y is the output vector. When yaw (ψ) is controlled to maintain a reference $\psi_0 \approx 0$, with the assumption of (2.10), from (2.3),

(2.6) and (2.7) one can linearize the model to the form:

$$\dot{x}_1 = \ddot{x} = g\theta = gx_{11}$$

$$\dot{x}_2 = \ddot{y} = -g\phi = -gx_{10}$$

$$\dot{x}_3 = \ddot{z} = \sum F/m - g = u_3/m - g$$

$$\dot{x}_4 = \dot{x} = x_1$$

$$\dot{x}_5 = \dot{y} = x_2$$

$$\dot{x}_6 = \dot{z} = x_3$$

$$\dot{x}_7 = \ddot{\phi} \approx \dot{p} \approx u_1 L/I_{xx}$$

$$\dot{x}_8 = \ddot{\theta} \approx \dot{q} \approx u_2 L/I_{yy}$$

$$\dot{x}_9 = \ddot{\psi} \approx \dot{r} \approx u_4/I_{zz}$$

$$\dot{x}_{10} = \dot{\phi} = x_7$$

$$\dot{x}_{11} = \dot{\theta} = x_8$$

$$\dot{x}_{12} = \dot{\psi} = x_9$$

$$(2.14)$$

Following the above linear model, the system can be designed to be controlled by a nested controller. The general form of the control diagram is shown in Figure 2.4.
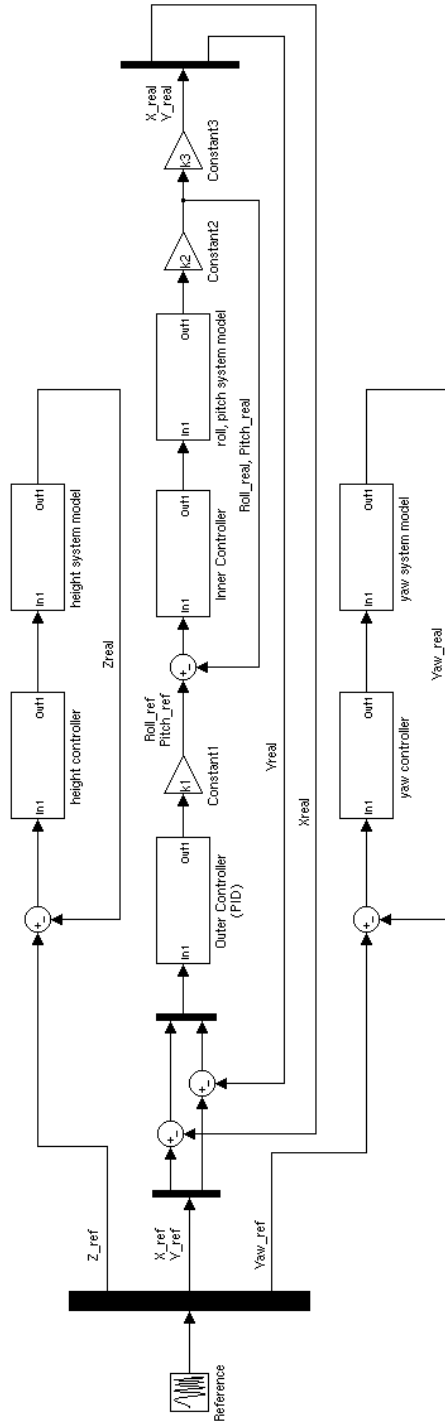
FIGURE 2.4: General system controller diagram. Altitude/Height and yaw control are decoupled from the rest. Longitude and latitude (XY) are controlled in a nested structure.

Reference values are provided to the controller to generate the control commands. The reference consists of three components, the height reference (Z), the yaw reference (ψ), and the XY position reference. The height and yaw controls use similar structure and are separated from the others. The XY position control is implemented in a nested structure. From the linear model 2.14 one can achieve the control of X with the control of pitch (θ), and control Y with the control of roll (ϕ). The XY position errors are calculated in real-time and go through an outer controller to generate the desired pitch/roll angle values. The desired angles are then further used as the reference to the inner (attitude) controller. The outer controller in the XY control structure is PID all through this research, but different controllers will be designed for the inner controller. When the desired angles are reached, according to the system dynamics from (2.6), the trajectory is tracked. In the following chapter different close-loop inner controllers will be designed and simulated.

2.3   Testbed Structure

To implement the above design, the hardware components of MARIT are constructed with a series of electronic devices to establish the structure illustrated in Figure 2.2. The software system of MARIT includes the driver program for each hardware, the control algorithms, and the data transmission. This section is devoted to describing the hardware and software system of MARIT.

2.3.1   Hardware Components

The general mechanism of this setup shows that while the MAVs are flying in the testbed area, their movements are captured and the frame data are sent to a workstation/server PC where the individual data of each MAV is directed to its own controller PC. This data feedback pipeline closes the control loop thus the MAVs are effectively

controlled to follow a desired trajectory. The details of each component is covered as follows:

The Vicon Motion Capture System [18] provides accurate motion capturing and was successfully applied in several testbeds [11, 13, 14]. MARIT is equipped with 6 Vicon M2 high speed cameras with a superior frame rate of up to 120 fps as shown in Figure 2.5. The cameras are aligned such that any object that is attached with reflective markers can be captured and tracked in real-time. Within the enclosed area, the Vicon cameras offer robust motion capture.

FIGURE 2.5: A Vicon M2 Camera with a maximum frame rate of 120 fps. The camera emits infrared to the field and receives reflections to locate and capture the objects.

Once the frame data is captured, it is sent to Vicon V8 data station for further processing and transferring. The Vicon V8 data station, shown in Figure 2.6, is capable of collecting data from 24 Vicon cameras simultaneously.

FIGURE 2.6: The Vicon V8 Datastation is the Frame Data Collection Unit

The frame data is sent to the workstation PC (Figure 2.7) through an ether-net connection from the Vicon V8. To retrieve the frame data of any objects in the field, the users could write their own programs on the server or on the controller PCs using Vicon RealTime Software Development Kit (SDK). Figure 2.8 shows the motion capture window of Vicon iQ running on workstation PC (server).

FIGURE 2.7: The Workstation Server PC (with yellow tag) and two controller PCs
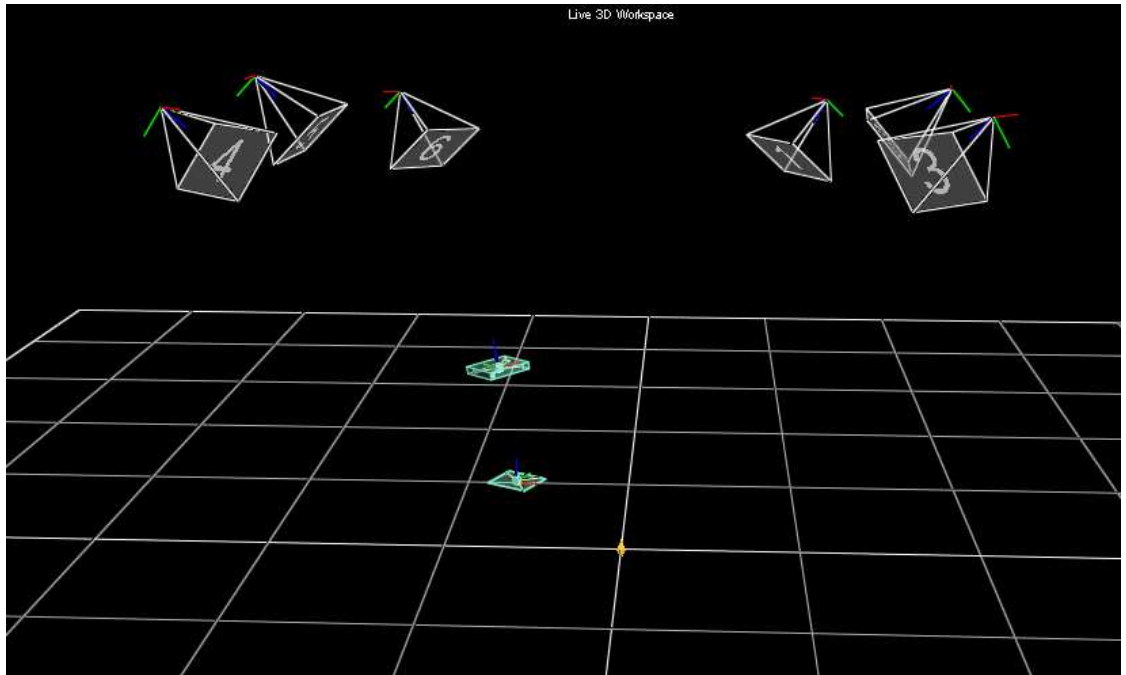
FIGURE 2.8: Live View Window of Vicon iQ. 6 cameras are used to detect the motions of quadrotors. The two square boxes are the quadrotor models built in Vicon iQ. The live view is reconstructed by Vicon iQ automatically.

A Server-Client Local Network is established to transfer the frame data from the workstation server to the controller clients. On the hardware level, all the PCs are connected to a wired network router. On the software level, Vicon RealTime Engine (RTE) runs on the server and open socket service to the local network. With the help of the client sample codes Vicon RTE supplies and knowledge of socket programming, an experienced C/C++ programmer should be able to develop programs that share data within this framework.

The Controller Client PCs run the controller programs and are connected to the RC transmitters. Frame data is transfered to the clients to be used in the control algorithms. The controller clients generate control commands from the algorithms and send them to the RC transmitters. All controller clients run on Linux-kerneled operating systems (currently Fedora 16).

The control commands generated by the controller clients are sent to the RC transmitters through **a PC to RC unit**. As shown in Figure 2.9, the SC-8000 connector [19] is used to connect the RC transmitter to the client PC. Once connected, the connector appears as a serial device in the PC's hardware list. In order to send desired commands to the connector (and further to the RC transmitter), the user needs to initialize the device to proper settings and normalize the values to be sent.[1]

FIGURE 2.9: SC-8000SP (front) Connects RC Transmitter (back) to Controller Client PC

For an indoor testbed with constrained space like MARIT ($3.6 \times 3.0 \times 2.7$ m^3), quadrotors offer advantages over fixed-wing UAVs in that the quadrotors are able to hover and maintain their positions. While building a quadrotor from scratch costs time and is out of the scope of this research, we chose to use the Draganflyer V Ti PRO

---

[1]With the help from the ACL lab at MIT, the SC-8000 driver programs on Linux was developed.

RC Gyro Stabilized Electric Helicopter from Draganfly Innovations Inc., as shown in Figure 2.10. This quadrotor is 76cm in diameter, weighs 525g, and offers flight time of 12 - 15 minutes [20]. Each Draganflyer comes with an RC transmitter. However, the original transmitter is not compatible with the SC-8000 connector well, so the Optic 6 transmitters by Hitec [21], seen in Figure 2.9, were purchased and used instead. The Optic 6 transmitter provides 6 FM radio control channels, but only 4 of them are used in the research, namely roll, pitch, yaw and throttle control. The transmitters are connected to the controller PC through SC-8000 connector interface to receive and send commands automatically.

FIGURE 2.10: Draganflyer V Ti PRO Quadrotors

2.3.2 Software System Structure

The testbed consists of several hardware units, with each unit running its own software programs simultaneously. This section is devoted to describing the software systems that keep the data flow. The data transfer diagram of the system is given by

Figure 2.12.

The data flow is first captured by the Vicon Motion Capture System and then transferred to the data station where it is relayed on to the workstation/server. This process is realized by running programs developed with the Vicon RealTime SDK on the server. At early stage, a socket server program is developed independently to achieve better flexibility (for better data manipulation). However, later testings proved that this program brought in large time delay. As a result, the original Vicon RealTime Engine (RTE) is used instead, since the RTE comes with a socket server thread to do the same thing.

Server: The software programs running on the server include Vicon iQ, the Vicon RTE and control clients (in the initial work). The Vicon iQ is provided by Vicon Inc. and must be run in order to initialize the camera system. It has a graphical interface in which the operator could manage camera settings, e.g. updating frequency and threshold values, set up data station connection (IP address etc.) and many other powerful features. Since most of the features that Vicon iQ provides do not apply to this research, no more details will be discussed here. Figure 2.11 shows Vicon iQ and RTE running on the workstation/server. The Vicon RTE is also provided by Vicon Inc. and is triggered automatically once the Vicon iQ main program is started. The RTE runs on background and starts a socket server to share the live frame data. This socket server is the crucial component that makes server/client communication happen. Vicon RTE listens to the port 800 and sends frame data to corresponding clients over network once a request is submitted.
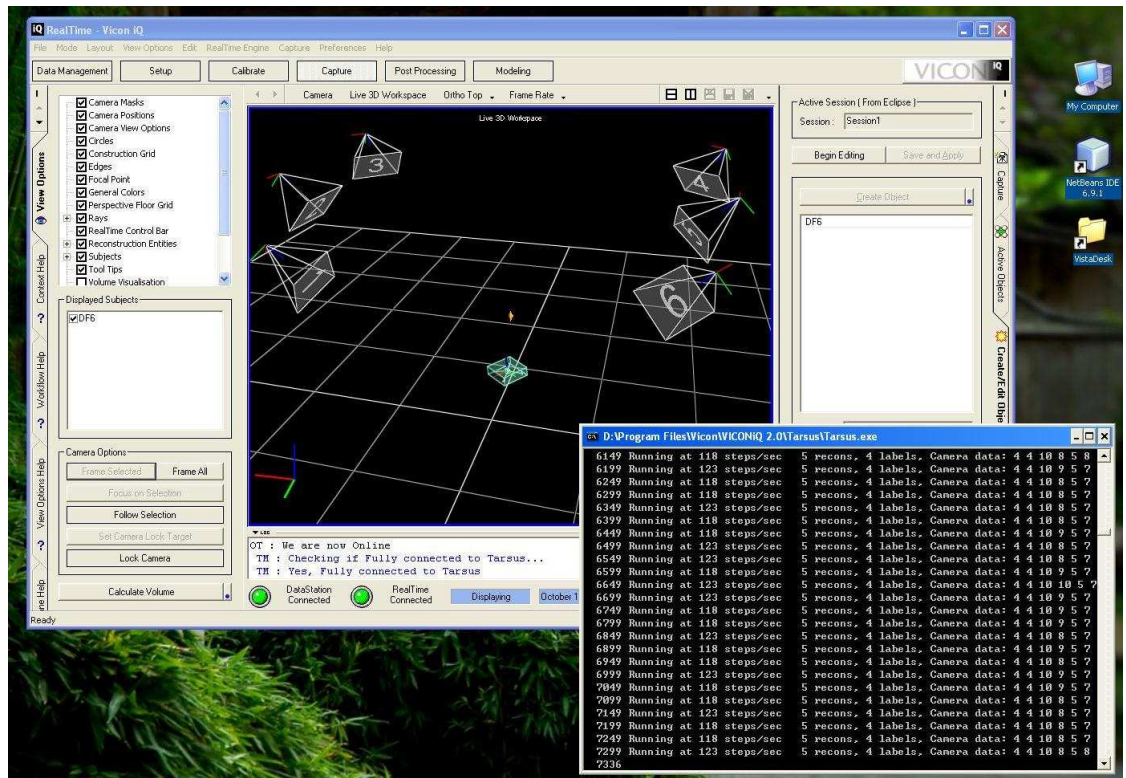
FIGURE 2.11: Vicon iQ and RTE Running on Workstation/server

Clients: For each control clients in the testbed, the corresponding software programs contain a socket client, the control algorithms and a graphical user interface. The socket client is developed independently using Vicon RealTime SDK and its function is to get live data frame by frame in real time. In order to realize real time data transfer, both the socket client (on the control client units) and the socket server (on the workstation/server) must be run simultaneously as shown in Figure 2.12. Testings proved the socket client program is able to update data with adequate frequency (50Hz) and brings negligible time delay.

A graphical user interface is designed and developed for the convenience of the operator. Operating the testbed on the client side requires some routines such as initializing PC-RC connectors and setting up IP address. Wrapping the trivial tasks and display the real time information in a graphical interface improves the operating effi-

ciency. Figure 2.13 shows the initial design of the GUI. The operator is able to modify the server's IP and socket port number on the GUI and click the "connect" button to start a new connection. This would be more complex to do with a command line console. Furthermore, one can achieve basic hovering of an individual MAV by simply clicking the "TestHover" button. The reserved display area could be used to feedback the frame data, or as a live video displayer if the clients are far away from the test ground. Reconstructing the MAVs from the live frame data enables the client operator to view the flight in real time. The client GUI is developed on Fedora Linux mainly using Qt framework and written in C++. More functionalities and optimizations will be made for better operating experience.
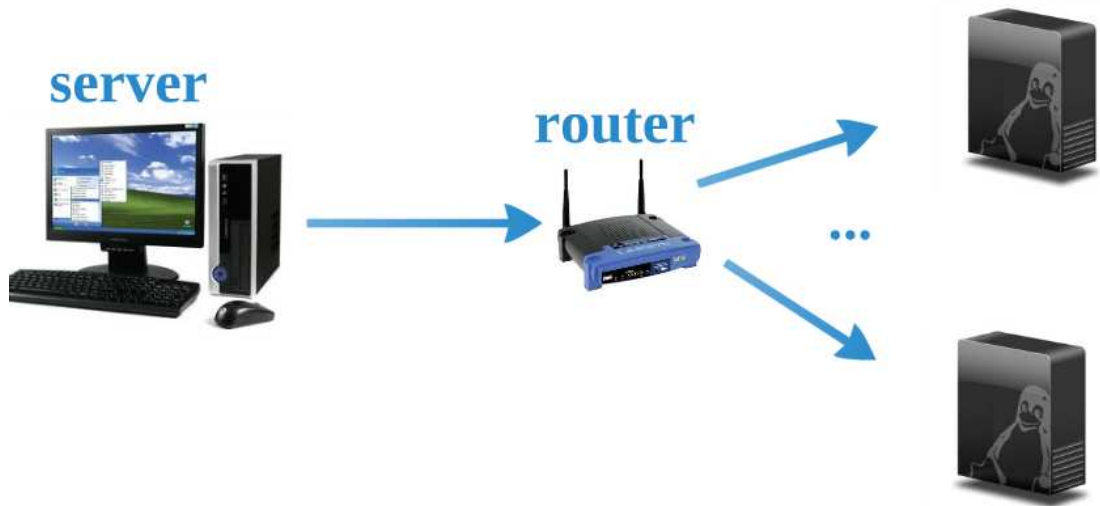
FIGURE 2.12: A Local Network is Established to Connect Server to Clients

reserved for display

connect to server

server IP

server port

hover command
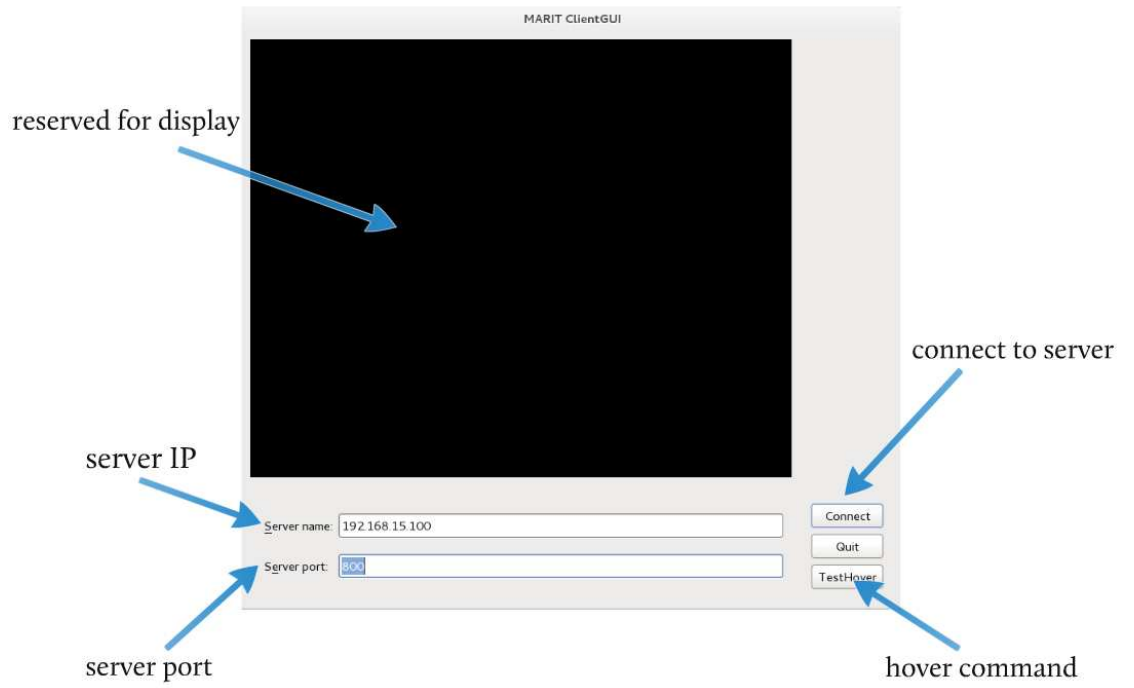
FIGURE 2.13: The Prototype Design of Control Client GUI
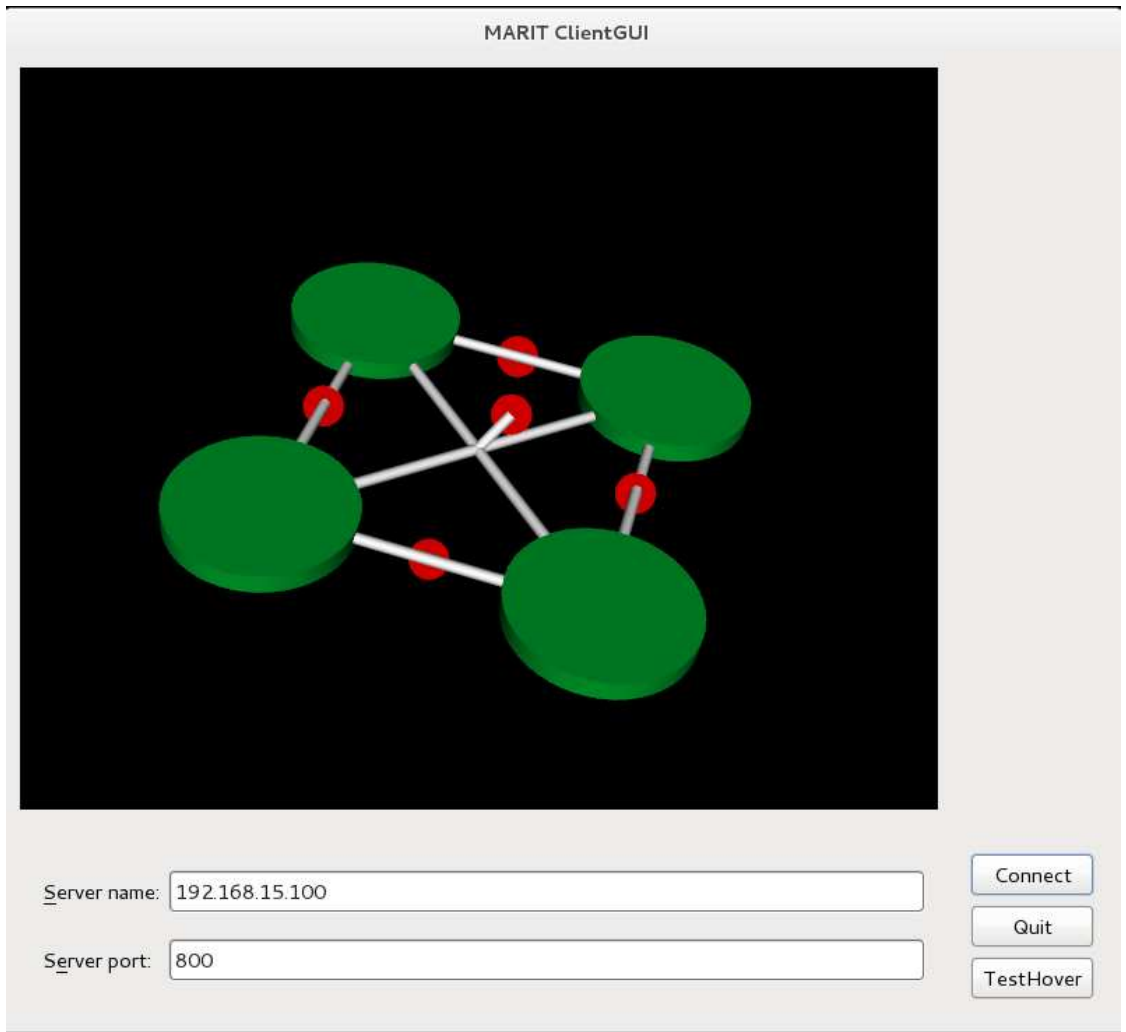
FIGURE 2.14: A 3D reconstruction of the Draganflyer in the client GUI

CHAPTER 3
SYSTEM CONTROL

To control a dynamics system, the first step is to build the mathematical model. In the previous chapter, the linearized dynamics model of the quadrotor in MARIT is established as (2.14). From the states variables given in (2.11) and by substituting u_3 with $\hat{u}_3 = u_3 + mg$ in the input vector given by (2.12), the linearized model can be expressed in state space as:

$$\dot{X} = AX + B\hat{U}$$
$$Y = CX + D\hat{U}$$

(3.1)

The matrix A is 12×12 as below:

$$
A = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\tag{3.2}
$$

The matrix B is 12×4 :

$$
B = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{m} & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\frac{L}{I_{xx}} & 0 & 0 & 0 \\
0 & \frac{L}{I_{yy}} & 0 & 0 \\
0 & 0 & 0 & \frac{L}{I_{zz}} \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\tag{3.3}
$$

The C matrix is 6×12 :

$$C = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

The Matrix D is 6×4 and is all zeros:

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.5}$$

To check controllability of the system, the controllability matrix has the form:

$$M_{ctrl} = \begin{bmatrix} B & AB & A^2B & \dots & A^{11}B \end{bmatrix} \tag{3.6}$$

M_{ctrl} has the rank of 12, thus the system with the model of (3.1) is controllable.

To check observability of the system, the observability matrix has the form:

$$M_{obsv} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \dots \\ CA^{11} \end{bmatrix} \tag{3.7}$$

M_{obsv} has the rank of 12, thus the system with the model of (3.1) is observable.

As mentioned in Chapter 2, the general controller design use feedback controllers to maintain and track altitude/height (z) and yaw (ψ) angle. In the following

sections, these two variables are controlled by two separate PID feedback controllers. The x and y are controlled by a nested structured controller as shown in Figure 2.4. The nested controller consists of an inner loop to maintain rotation/attitude angles (ϕ and θ), and an outer loop to control the positions. This design will realize position control when the system is close to the state of linearity as described in (2.14). In the following controller designs, PID controllers will be used for outer loop control in all designs. The inner loop that controls the rotation/attitude angles will differ from each other.

3.1 Control using Lyapunov Theory

In this section the Lyapunov controller is used to maintain the desired attitude angles due to its performance observed in [28]. Other researchers applied backstepping technique based on Lyapunov stability theory to design their quadrotor control systems [29], [30].

3.1.1 Design

The positive-definite Lyapunov function (3.8) is determined by combining the states from (2.11).

$$V_x = \frac{1}{2}[(x_{10} - \dot{x}_{10})^2 + x_7^2 + (x_{11} - \dot{x}_{11})^2 + x_8^2 + (x_{12} - \dot{x}_{12})^2 + x_9^2] \quad (3.8)$$

It's derivative can be expressed as

$$\dot{V_x} = (x_{10} - \dot{x}_{10})x_7 + x_7\frac{L}{I_{xx}}u_1 +$$
$$(x_{11} - \dot{x}_{11})x_8 + x_8\frac{L}{I_{yy}}u_2 +$$
$$(x_{12} - \dot{x}_{12})x_9 + x_9\frac{L}{I_{zz}}u_4 \quad (3.9)$$

The control law is determined by choosing

$$u_1 = -\frac{I_{yy}}{L}(x_{10} - \dot{x}_{10}) - k_1 x_7$$

$$u_2 = -\frac{I_{xx}}{L}(x_{11} - \dot{x}_{11}) - k_2 x_8 \qquad (3.10)$$

$$u_4 = -\frac{I_{zz}}{1}(x_{12} - \dot{x}_{12}) - k_4 x_9$$

When k_1, k_2, k_4 are positive, the following stands:

$$\dot{V}_x = -k_1 \frac{L}{I_{xx}} x_7^2 - k_2 \frac{L}{I_{yy}} x_8^2 - k_4 \frac{1}{I_{zz}} x_9^2 < 0 \qquad (3.11)$$

3.1.2   Simulation

By Lyapunov theorem, the simple stability for equilibrium is now ensured. The simulator is shown in Figure 3.1. Since only the inner loops of x and y adopt Lyapunov controllers, the height and yaw controller are identical to the one described in Figure 2.4 and are not shown here. The reference values containing the desired xy position are compared with the respective real values to calculate the errors. Then the errors pass through an outer controller (PID) to generated desired pitch and roll values. The inner controller uses Lyapunov theory to achieve the control of roll, pitch angles. The system dynamics model is defined according to (2.14). The trajectory tracking simulation is shown in Figure 3.2.

38
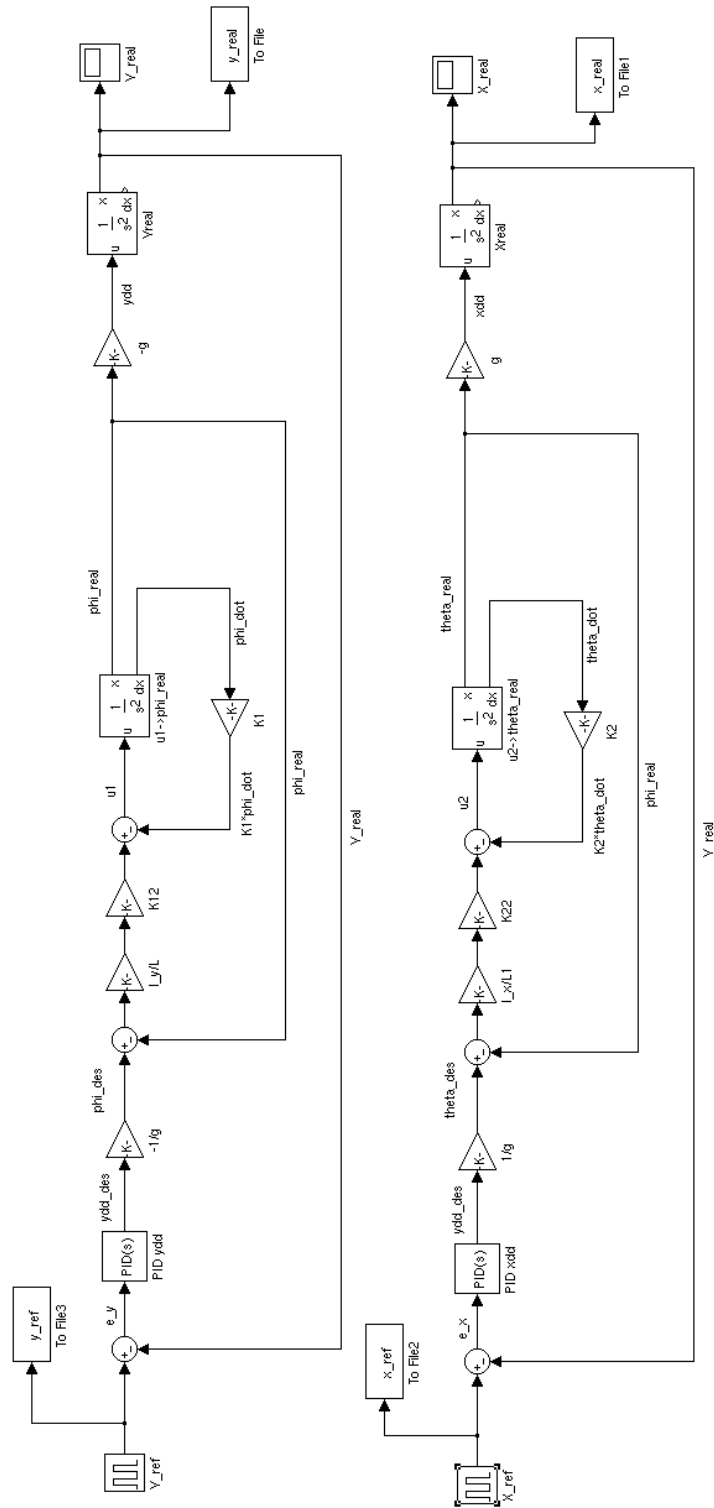
FIGURE 3.1: Lyapunov attitude controller simulator. The inner loop uses controllers based on Lyapunov stability theory to maintain attitude angles. The outer loop uses PID controllers.
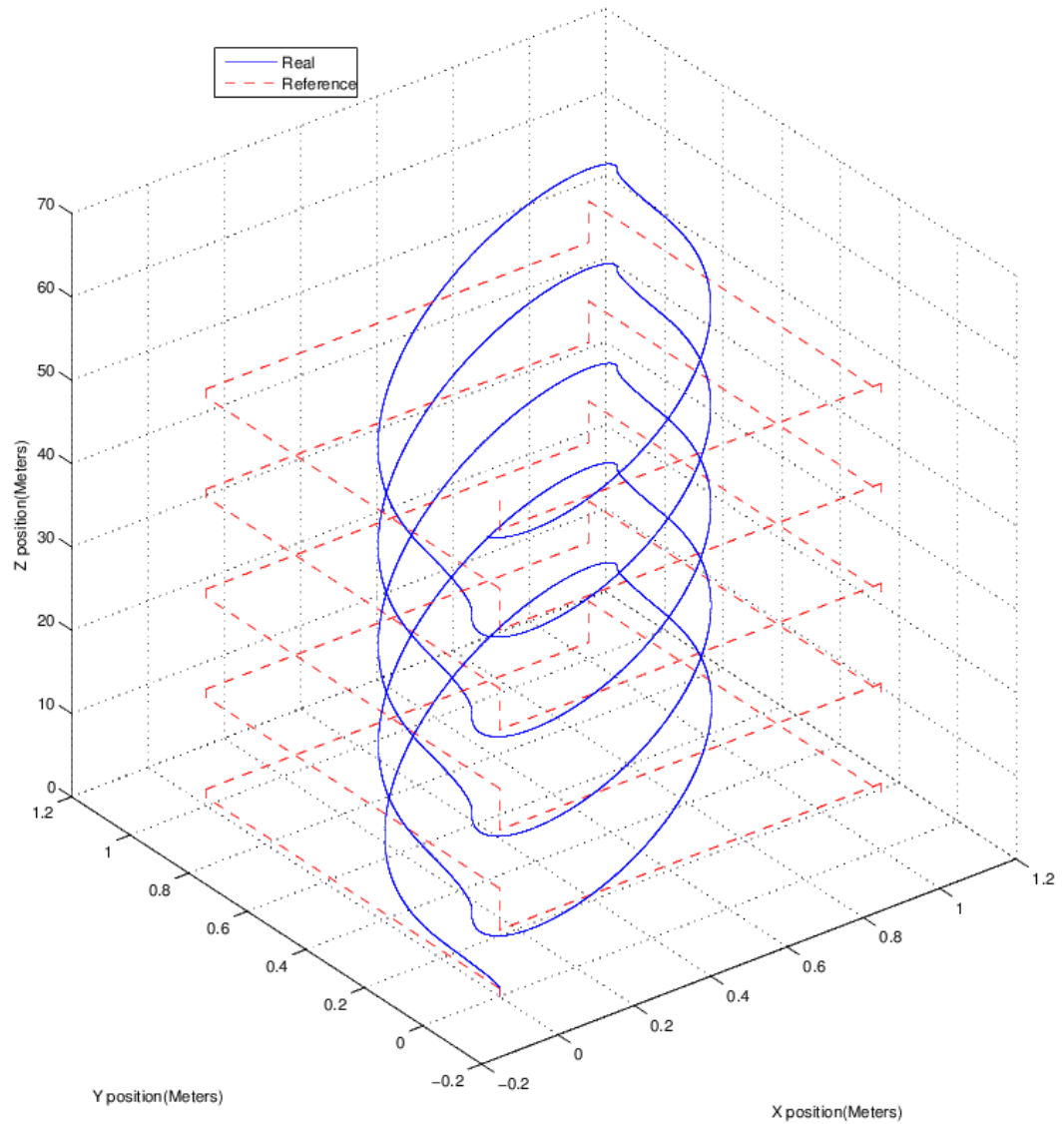
FIGURE 3.2: Quadrotor trajectory tracking using Lyapunov attitude controller

3.2   Control using LQR Controllers

The LQR is chosen because of its reliable performance observed in [13], in which the authors state the controllers optimizes the vehicle's capabilities in hover, while ensuring the vehicle can respond quickly to position errors.. In another research [27], by using LQR technique to minimize the running cost which is proportional to the velocity,

the authors are able to send the air vehicle down to a mineshaft.

3.2.1 Design

Since only the attitude controller (inner loop) is concerned, the states in (3.1) can be reselected as:

$$
\begin{aligned}
X_{att} &= [\dot{\phi}\ \dot{\theta}\ \dot{\psi}\ \phi\ \theta\ \psi]^T \\
U_{att} &= [u_1\ u_2\ u_4]^T \\
Y_{att} &= [\phi\ \theta\ \psi]^T
\end{aligned}
\tag{3.12}
$$

a linear model in state space is built:

$$
\begin{aligned}
\dot{X}_{att} &= A_{att}X_{att} + B_{att}\hat{U}_{att} \\
Y_{att} &= C_{att}X_{att} + D_{att}\hat{U}_{att}
\end{aligned}
\tag{3.13}
$$

The matrices $A_{att}, B_{att}, C_{att}$ and D_{att} are acquired from the linear model given in (2.14), and their values are shown in (3.14).

$$
A_{att} =
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\tag{3.14}
$$

$$
B_{att} =
\begin{bmatrix}
\frac{L}{I_{xx}} & 0 & 0 \\
0 & \frac{L}{I_{yy}} & 0 \\
0 & 0 & \frac{L}{I_{zz}} \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix}
\tag{3.15}
$$

$$C_{att} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

$$D_{att} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.17}$$

The structure of the LQR controller is shown in Figure 3.3. The controller is designed in Simulink and the matrix K is tuned as given in (3.18).
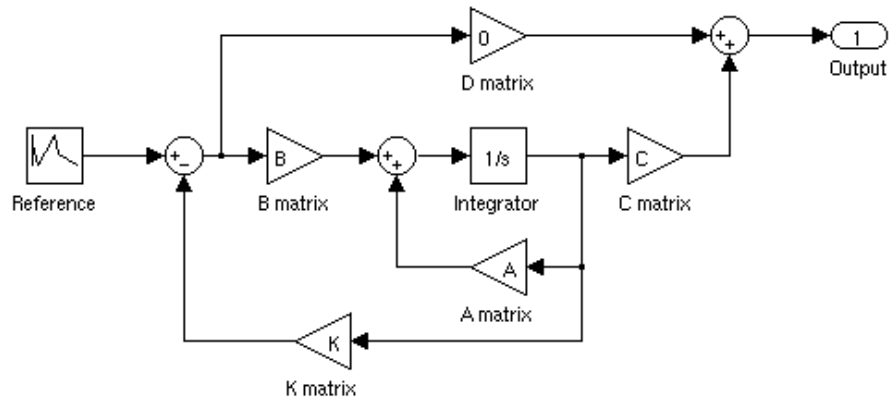
FIGURE 3.3: LQR controller block diagram. This general form follows the state space model given in (3.13). The LQR design feeds the states back and apply a parameter matrix K to generate the control law.

$$K = \begin{bmatrix} 0.2544 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0.2544 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.2544 & 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

3.2.2 Simulation

Similar to the case of Lyapunov controller, the LQR controller is simulated in Simulink, and the simulation model is shown in Figure 3.4. The height controller is identical to the one described in Figure 2.4, thus is omitted here. The reference values containing the desired XY position and yaw angle are compared with the respective real values to calculate the errors. Then the errors pass through an outer controller (PID) to generated desired angle values. The inner controller uses LQR to achieve the control of roll, pitch, and yaw angles. The system dynamics model is defined according to (2.14). The performance of tracking the same reference trajectory is shown in Figure 3.5.
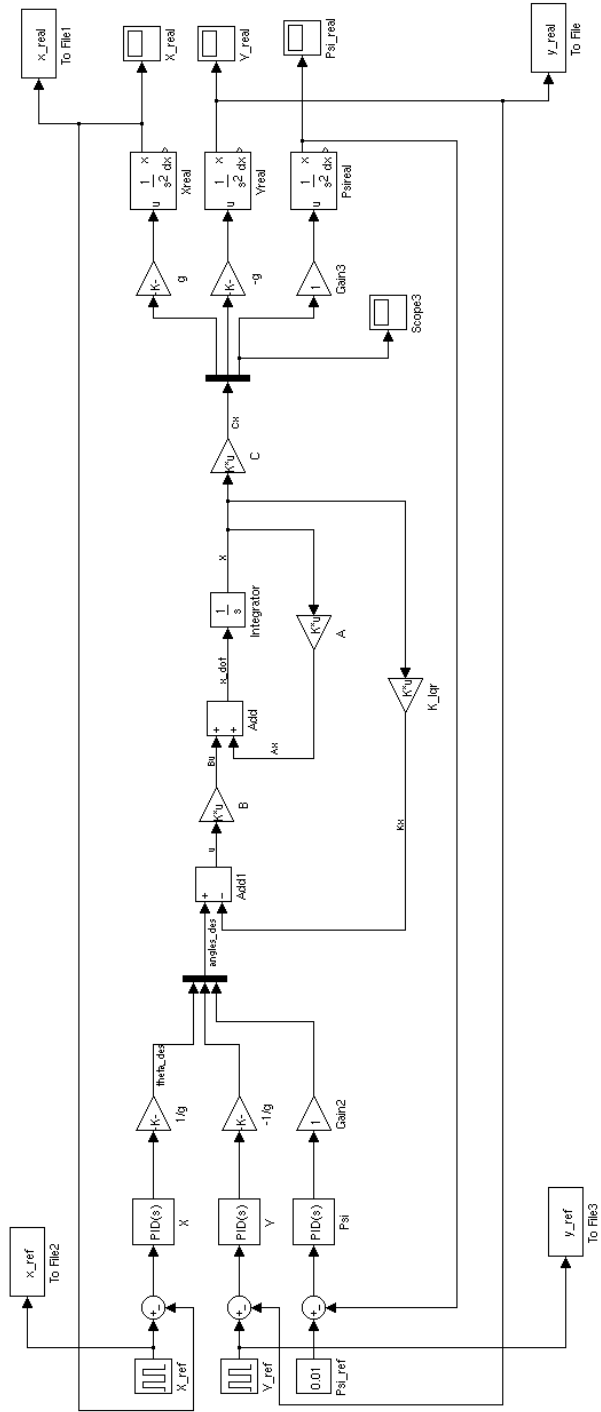
FIGURE 3.4: LQR attitude controller simulator. The inner loop that controls the attitude angles adopts LQR technique. The yaw angle is also included in the inner loop because of the convenience in design, although it is decoupled from the other variables.

44
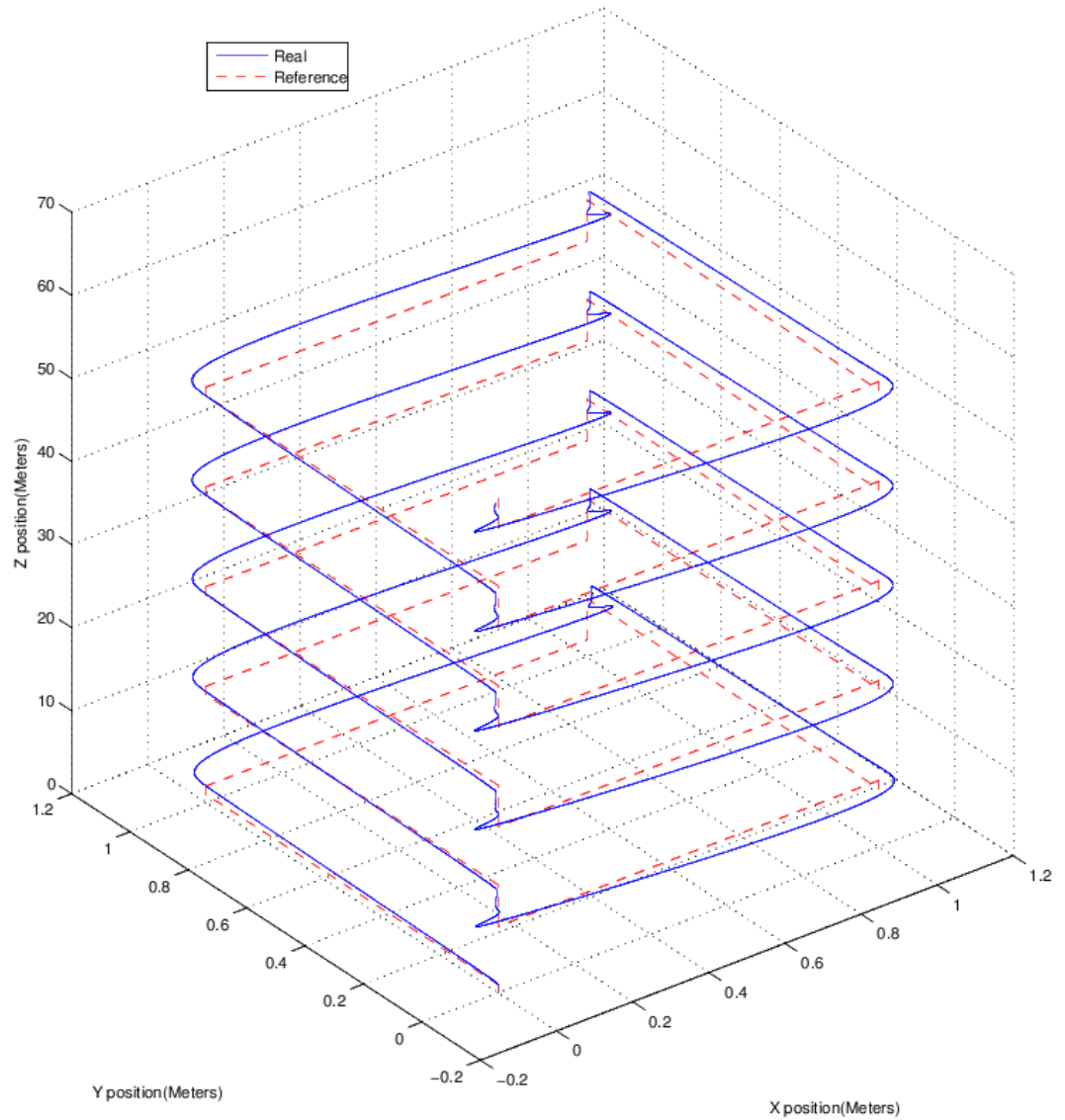
FIGURE 3.5: Quadrotor trajectory tracking using LQR attitude controller

3.3   Control using PID Technique

The PID controller is designed to maintain the desired attitude angles. It is cho-
sen because of its robustness as observed in [24] and [25]. According to [24], the au-
thors applied PD controllers to maintain positions, and added an accumulator for height
control due to battery power loss. Others combined backstepping technique with PID

controllers to achieve aggressive maneuvers [11]. Our previous work also shows the PID controller is able to maintain the angles and restrict the errors within 0.1rad [6]. PID controller takes the errors in real-time and multiply them with tuned parameters.

3.3.1 Design

PID controller uses feedback errors to generate control inputs. The error of each variable is processed and multiplied by the PID control parameters. The control law is given by equation (3.19), in which k_p, k_i, and k_d represent proportional, integral, and derivative parameters respectively.

$$
\begin{aligned}
u_{1att} &= k_{p\phi}e_\phi + k_{d\phi}e_{d\phi} + k_{i\phi}e_{i\phi} \\
u_{2att} &= k_{p\theta}e_\theta + k_{d\theta}e_{d\theta} + k_{i\theta}e_{i\theta} \\
u_{4att} &= k_{p\psi}e_\psi + k_{d\psi}e_{d\psi} + k_{i\psi}e_{i\psi}
\end{aligned}
\tag{3.19}
$$

e denotes the error, e_d is the derivative of error, and e_i is the integral of error. Each error is calculated as:

$$
\begin{aligned}
e_\phi &= \phi_{ref} - \phi_{real} \\
e_{d\phi} &= e_{\phi current} - e_{\phi previous} \\
e_{i\phi} &= \sum e_\phi.
\end{aligned}
\tag{3.20}
$$

3.3.2 Simulation

A simulator is created as in Figure 3.6 to simulate the trajectory tracking. Controller parameters are tuned with tools in Simulink. The height controller is omitted since it is identical to the one described in Figure 2.4. As is covered in the previous section, the xy position controller takes a nested form. The outer controller uses PID and a constant $(1/g)$ to obtain the desired roll and pitch angles. In this specific controller, the inner control also uses PID to maintain the angles as desired. The parameter values used in both inner controller and outer controller are listed below.

Parameter	Inner	Outer
k_p	5.11	6.67
k_i	2.45	0.46
k_d	122.16	9.07

The tuned system is able to track trajectories as shown in Figure 3.7, in which the dashed line being the reference trajectory, the solid line the simulated tracking trajectory. Detailed comparison of different controllers will be covered next.
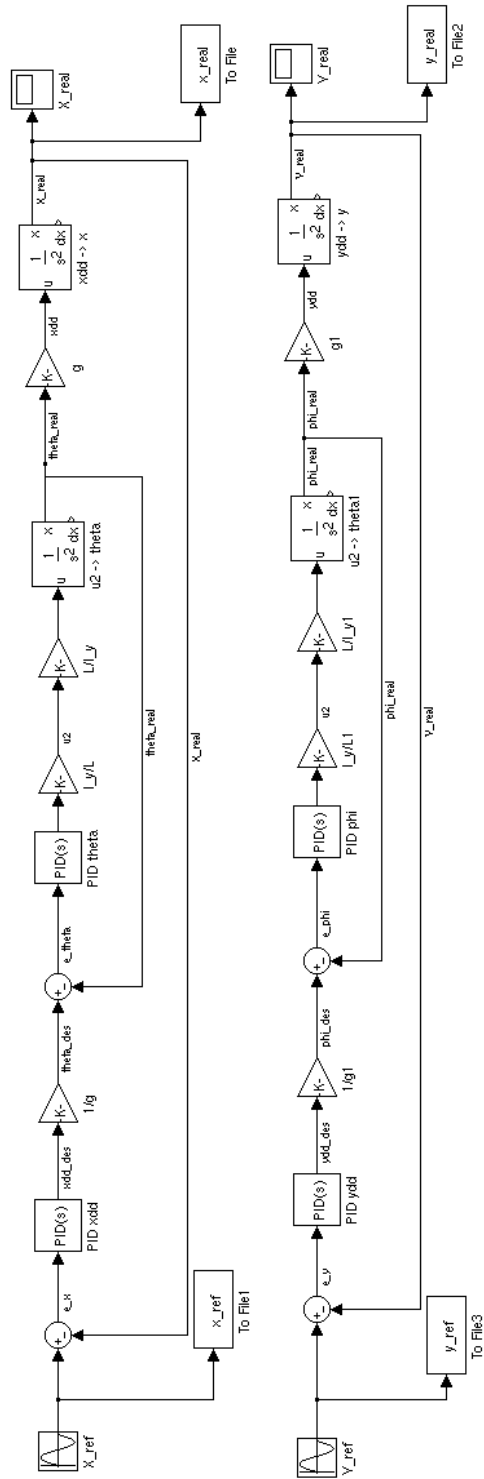
FIGURE 3.6: PID attitude controller simulator
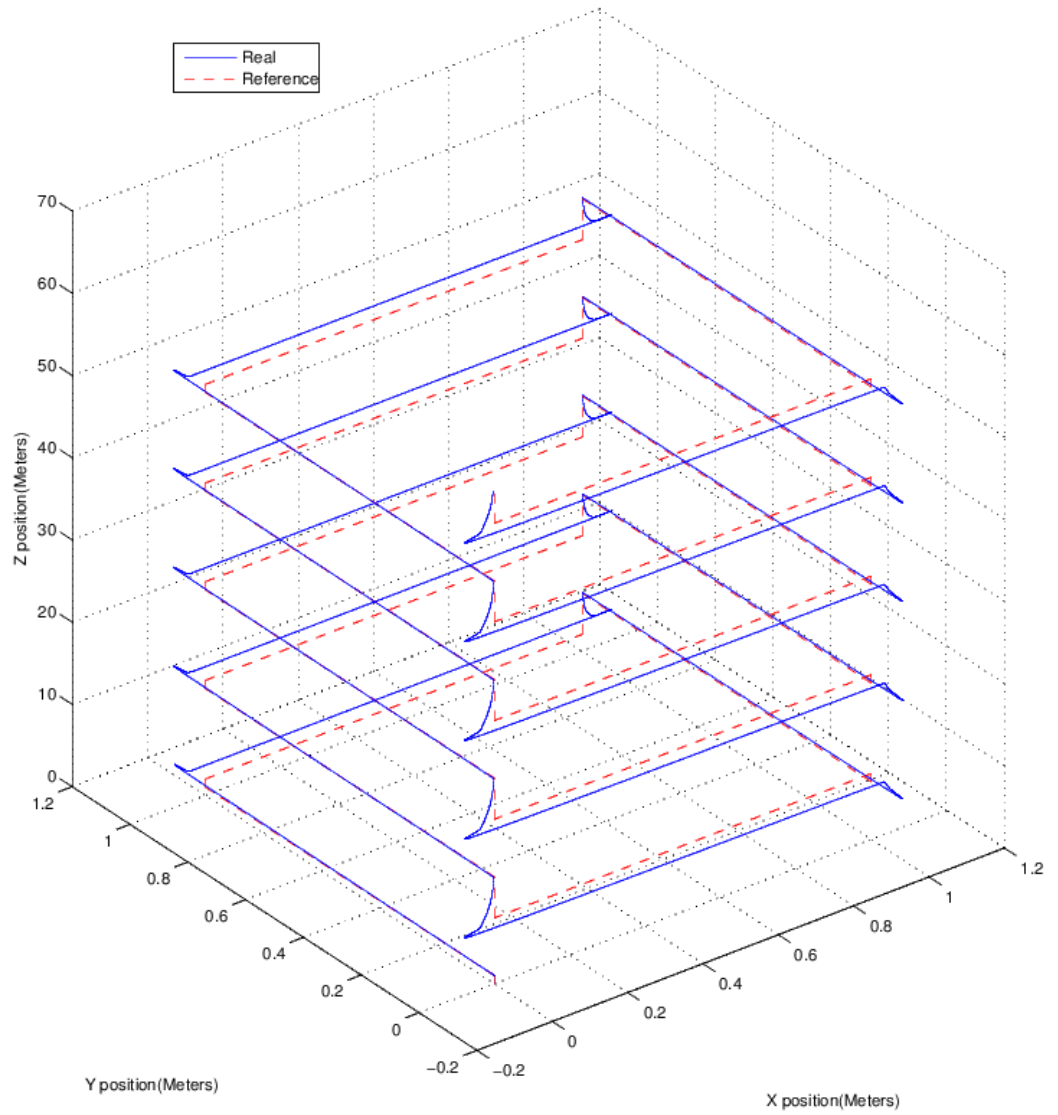
FIGURE 3.7: Quadrotor trajectory tracking using PID attitude controller

3.4 Comparison of Different Controllers

In the above sections three controller designs are developed. The following Figure 3.8 specifies the control technique used for each variable in the three designs.

Design	Outer Loop		Inner Loop		Altitude	Yaw
	x	y	phi	theta	z	psi
Lyapunov	PID		Lyapunov		PID	PID
LQR	PID		LQR		PID	LQR
PID	PID		PID		PID	PID

FIGURE 3.8: The controller used for each variable in different designs. The inner loop controllers differ from each design, most of the others adopt PID controller.

The performance of the above controllers are calculated and compared. By evaluating the step response of each controller, comparison is made as regard to factors such as rise time, settling time, percent overshoot and steady-state error (SSE).

To evaluate the quality of the trajectory tracking, a trajectory tracking error function is calculated for each controller. It sums up the square of the difference between the reference and real trajectory as follows:

$$E_{track} = \sum_{i=1}^{n} [(x_{ref_i} - x_{real_i})^2 + (y_{ref_i} - y_{real_i})^2 + (z_{ref_i} - z_{real_i})^2] \quad (3.21)$$

The step responses of the controllers are plotted in Figure 3.9. The readings are acquired from simulations and listed in Table 3.1. The PID controller excels in rise time, settling time and trajectory tracking error. The LQR controller has smallest percent overshoot, but gives longer settling time. The Lyapunov controller has the longest settling time (more than 15 seconds), similar "lagging" is also observed in [29]. However, the Lyapunov controller provides slightly better SSE than LQR.

TABLE 3.1

	unit	PID	LQR	Lyapunov
Rise Time	sec	0.135	1.759	1.985
Settling Time	sec	1.89	4.05	>15.00
Overshoot	percent	0.108	0.099	0.147
SSE	meter(m)	0.0004	0.0232	0.0200
E_{track}	m^2	853	4752	15070
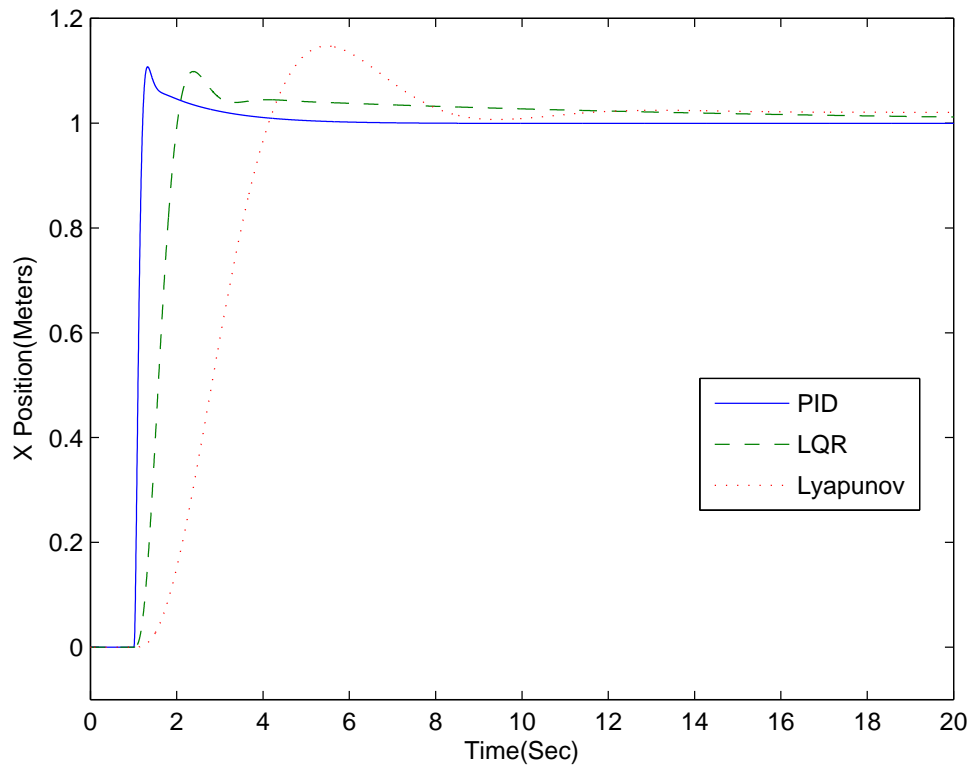
FIGURE 3.9: Step responses of the controllers.

3.5 Experiment with PID Design

From the comparison in the previous section, the PID design excels in rise time, settling time, and trajectory tracking error. Thus PID controllers are proposed to be applied in the experiment of hovering. Two different kinds of experiments are performed using different sets of hardwares. In the first method, hovering control is done by connecting the RC transmitters to the server. The server not only receive and host the frame data of the UAV, but also runs the control algorithms and deliver the control commands. This compact arrangement allows fast debugging for programs and equipment because fewer hardwares and communication pipes are involved. The structure is illustrated in Figure 3.10.
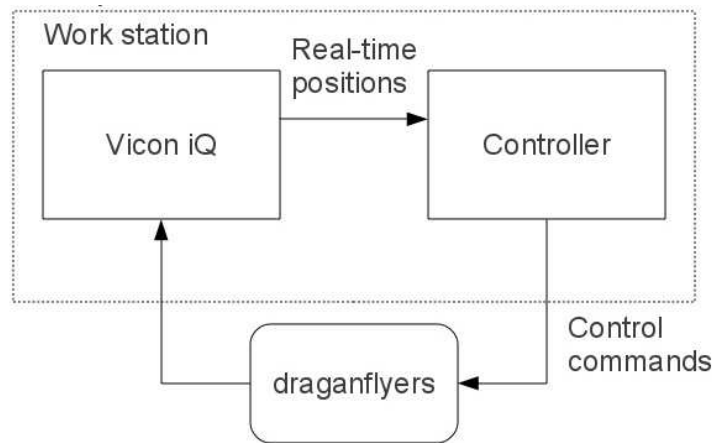
FIGURE 3.10: Early stage experiment diagram. The dotted frame represents the work station PC (server). In this method, both the server program (Vicon iQ) and the controllers are installed on the same machine to provided fast debugging and testing.

The second method used adopts the server-client structure. Multiple controller PCs are connected to the server via a local network. Data is shared among various units within the network. Each quadrotor is controlled by a single controller PC. The workstation (server) only runs the hosting program and distribute the frame data of each quadrotor to the local network. The transmission protocol is TCP/IP and POSIX socket interface is used for sending and receiving the data. The server program is provided

by Vicon software package and has multi-thread feature to enable the data of different quadrotors to be sent simultaneously. The "NTG" unit stands for Non-linear Trajectory Generation, which is a software package for optimal trajectory generation and will be covered in the next chapter. NTG generates trajectories for each quadrotor and sends reference data to the controller units for control commands calculation. Once calculated, the commands will be sent to the quadrotors using R/C transmitters, which are connected to the controller units. This method is illustrated in Figure 3.11.
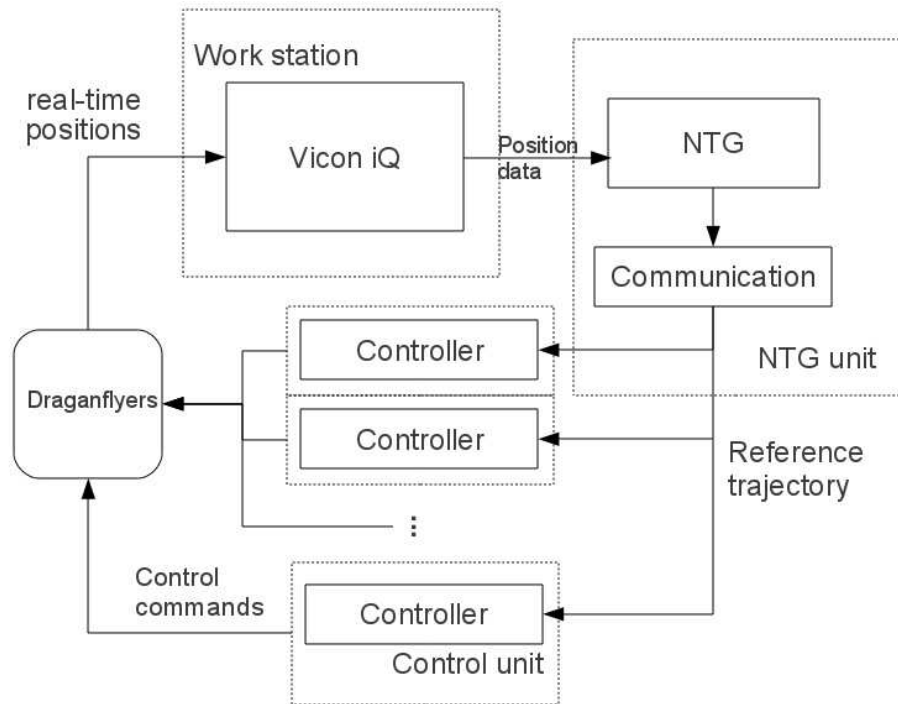
FIGURE 3.11: Server-client mode experiment diagram.

To analyze the results and evaluate the performance of the algorithms, data needs to be collected accurately. Data generated from the experiments were collected in real time.

Reference trajectories, namely $x_{ref}, y_{ref}, z_{ref}, \phi_{ref}, \theta_{ref}$ and ψ_{ref}, are determined arbitrarily (e.g. for hovering) or generated automatically (e.g. for trajectory tracking). For basic hovering, the reference trajectory is a group of constant values.

Thus collecting them is trivial.

Real time 6 DOF data sets, namely $x_{real}, y_{real}, z_{real}, \phi_{real}, \theta_{real}$ and ψ_{real} are measured using the Vicon M2 cameras in every cycle. Acquiring the data is done by programming with the Vicon RealTime Engine (RTE) Software Development Kit (SDK). The real time data is directed to a file on the local disk drive. This method produces a bottle neck in code execution time and slows down the algorithms when the looping frequency is high. However, it does not seem to affect the hovering when the program is looping at 20Hz.

Control commands are generated from the control algorithms and sent to the actuators during each loop. These values are updated at the same frequency as the real time data, and are collected in the same way.

The following graphs show the data collected from tethered experiments (string attached) on the Draganflyer quadrotor in MARIT. The tethered string prevents the quadrotor from flying away or drop to the ground unexpectedly.

Figure 3.12 shows the initial performance of the attitude controllers trying to maintain Euler angles at zero. In general the controllers are able to keep the outputs within the error range of ±0.1 rad. The red horizontal dotted lines in each plot represent the mean value of the outputs. The mean value of the roll and pitch angles shifted from zero by a small offset. This offset was caused by the unsymmetrical shape of the quadrotor (mainly because of the placement of the battery). The yaw angle has a large initial error and the responding time is slower than roll and pitch controllers.
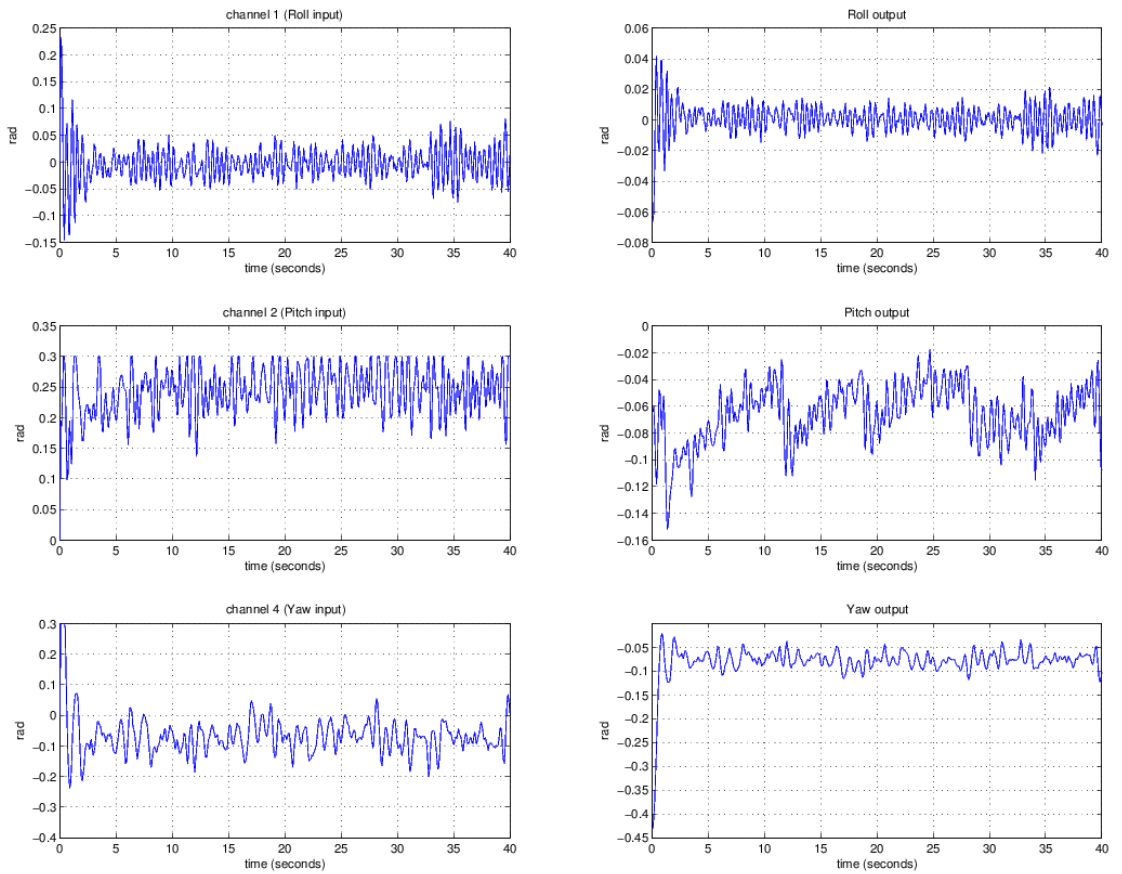
FIGURE 3.12: PID attitude controllers trying to maintain the rotation angles at 0, controlled from the server (method 1). Control loop runs at 12.8Hz.

Figure 3.13 shows the performance of the nested PID controllers maintaining X and Y positions in the earth frame at $(x = 0, y = 0)$.

55
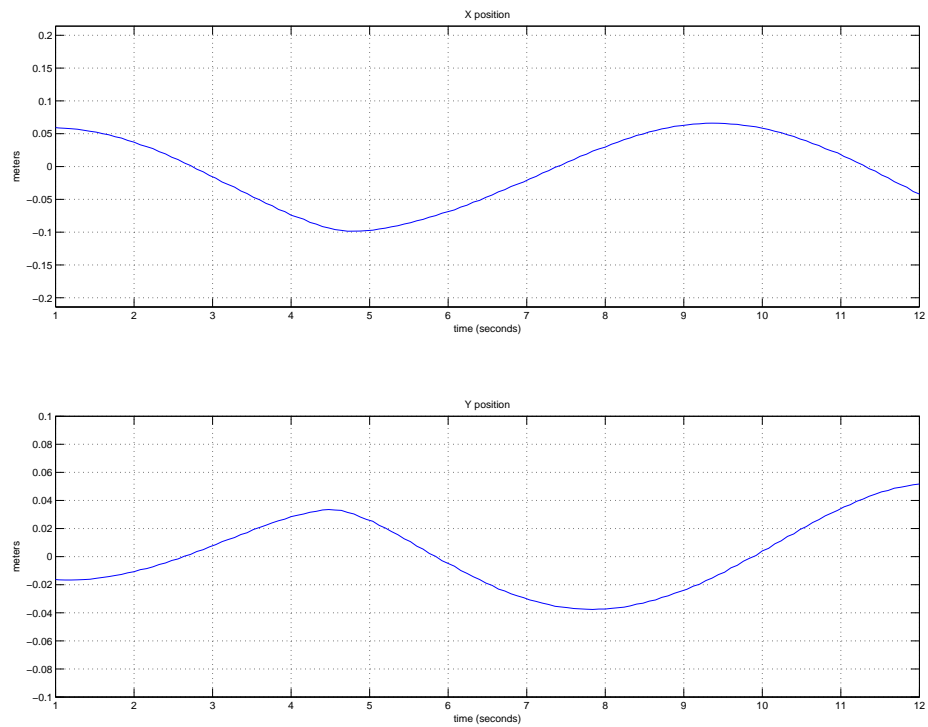
FIGURE 3.13: The X, Y positions maintained by nested PID controllers at $x = 0, y = 0$, controlled from the server (method 1). Control loop runs at 12.8Hz.

Although maintaining the position at given xy values does not control the rotation angles to be strictly zero, a hovering state should have small rotation angles. Figure 3.14 shows the values of rotation angles when the nested PID controller maintains the position at the above given point.
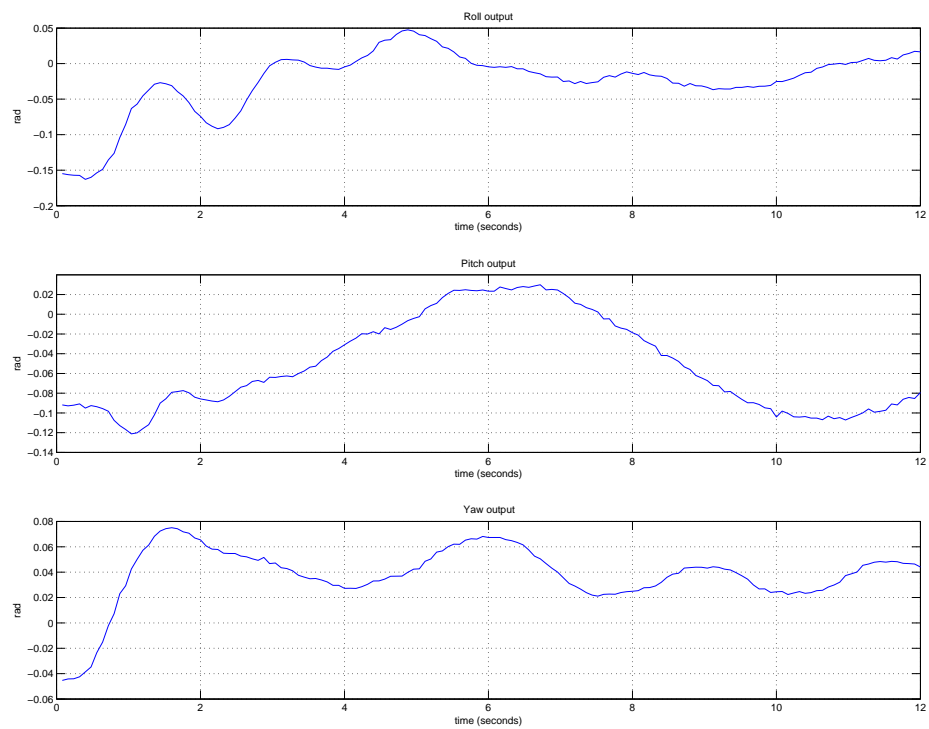
FIGURE 3.14: The rotation angles when nested PID controller is doing position control, controlled from the server(method 1). Control loop runs at 12.8Hz.

With the completion of the server/client framework, data is able to be shared and controllers are installed on the client computers. The server/client structure lessens the burdens on the server but inevitably brings larger time delay. Figure 3.15 shows the performance of the same attitude controllers running from the clients. The delay in the control of yaw caused the yaw data to be out of range within the first several seconds. The offsets of roll and pitch angles mentioned earlier were reduced by tuning the PID parameters.
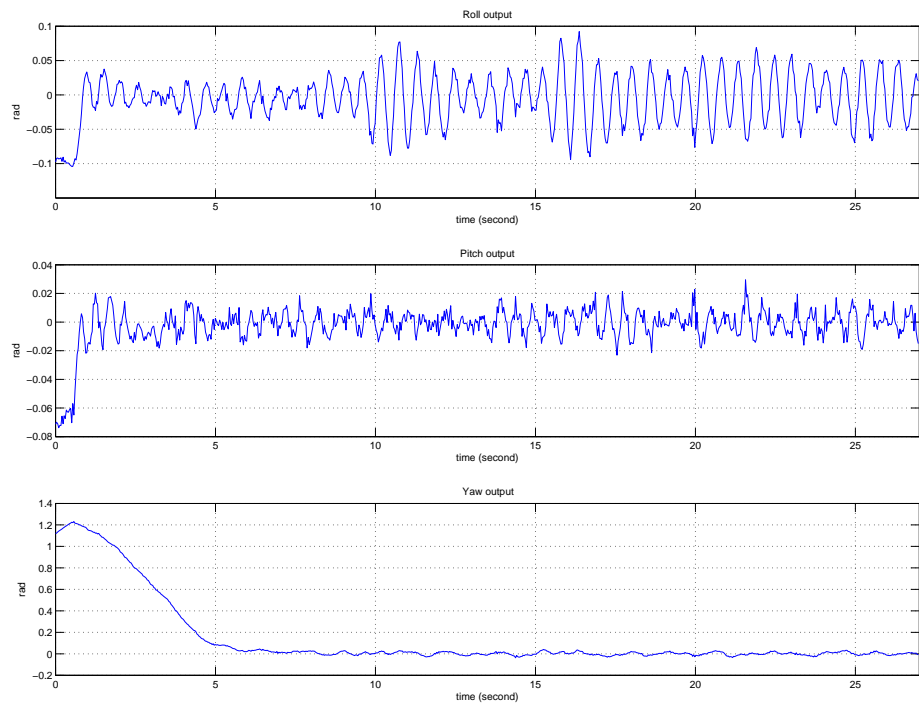
FIGURE 3.15: The rotation angles maintained by PID attitude controllers, controlled from the client through local network (method 2). Control loop runs at 30Hz.
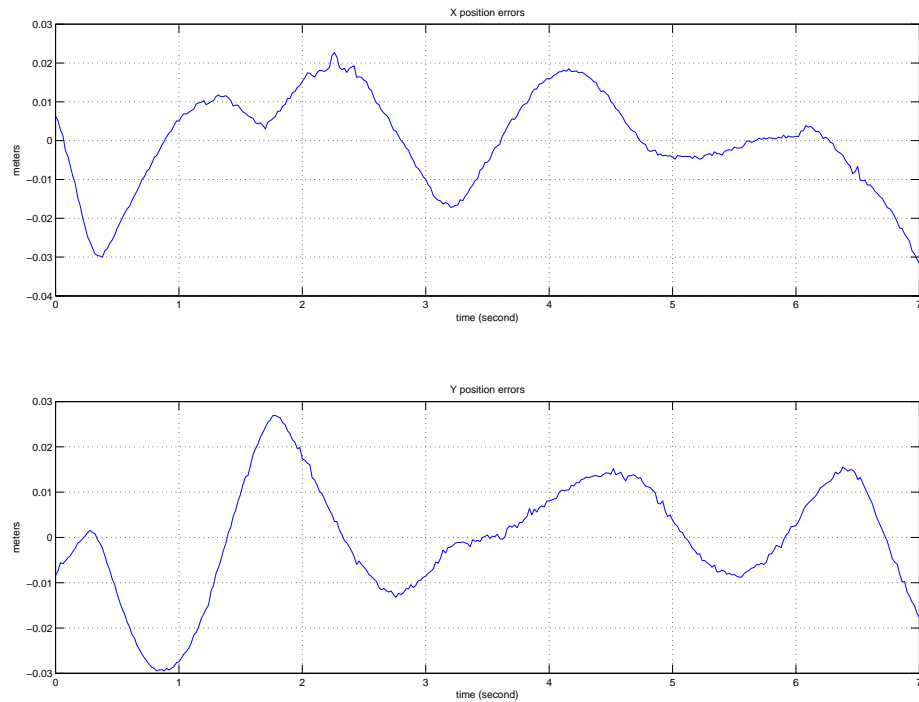
FIGURE 3.16: The X, Y positions maintained by nested PID controllers at $x = 0, y = 0$, controlled from the client through local network (method 2). Control loop runs at 30Hz..

3.6 Conclusion

In this chapter, three quadrotor controllers are designed for the Draganflyer quadrotor in MARIT. For each of the design, the controller is simulated in Simulink and a test is done to perform a simple trajectory tracking. Then the comparison is done to evaluate the controllers' performance according to several criteria. The PID design is chosen for the experiment due to its superior performance. Two methods are used in the experiments and data is collected from the tethered flights in real-time. Results are plotted to show the controller's performance.

CHAPTER 4
TRAJECTORY GENERATION

In the industrial world and military field, a large group of problems involve planning and following trajectories. Examples are usually systems with noise and uncertainty and requires accurate control, such as autonomous vehicles maneuvering in city streets, mobile robots performing tasks on factor oors (or other planets), manufacturing systems that regulate the ow of parts and materials through a plant or factory, and supply chain management systems that balance orders and inventories across an enterprise [33].

In linear systems, a standard technique is to separate the controller into a feedforward compensator and a feedback compensator. This structure, which provides nominal input used to track the reference trajectories with the feedforward compensator, and corrects errors between the reference and real trajectories with the feedback compensator, is referred to as *two degree of freedom* controller [33].

In nonlinear systems, the two degree of freedom design decouples the trajectory generation and asymptotic tracking problems. To illustrate this controller design, from (2.14) the following state space model is constructed:

$$\dot{x} = f(x, u), \qquad x \in \mathbb{R}^n, u \in \mathbb{R}^m$$
$$y = h(x, u), \qquad y \in \mathbb{R}^p$$

(4.1)

x, u and y are the system's states, input and output, respectively. To make the system track the reference trajectory given by x_d, a two degree of freedom controller shown by Figure 4.1 could be applied. This controller is configured with a trajectory

generator to generate both the reference trajectory x_d and the nominal input u_d.
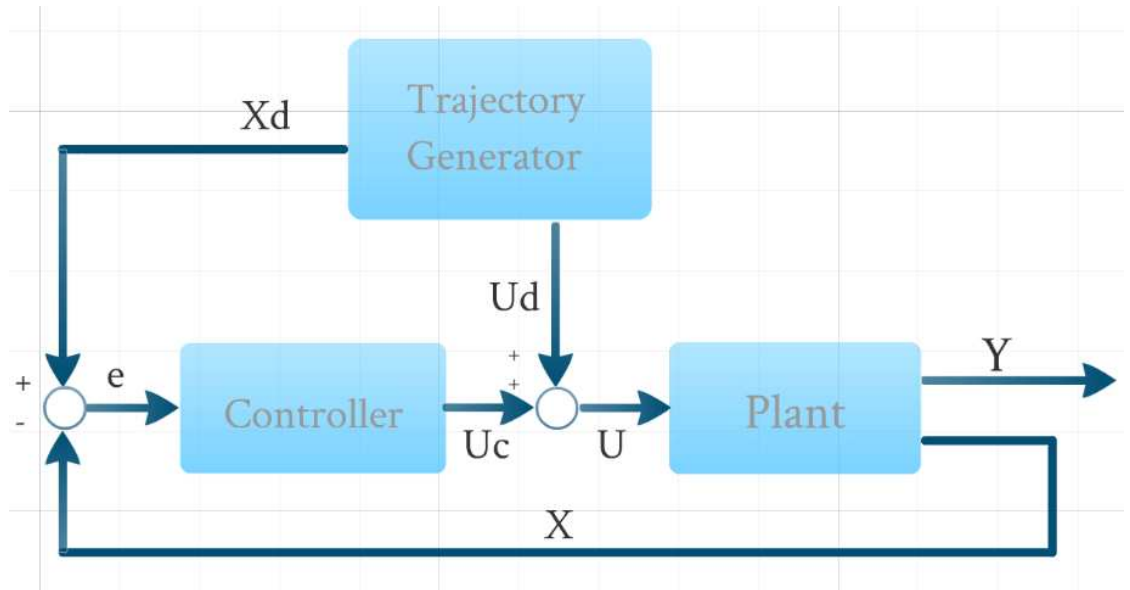
FIGURE 4.1: Two degree of freedom controller structure. The trajectory generator provides U_d, the feedforward nominal input for tracking, and X_d, the reference trajectory. The controller computes the error from X_d and the system feedback X and generates control command U_c. The actual input U is generated by combining U_d and U_c.

In the following sections, the above controller design is used for the purpose of linearized quadrotor model control. The controllers used are described in the previous chapter, and the trajectory generator is the *Nonlinear Trajectory Generation* software package, which will be covered next.

4.1   Nonlinear Trajectory Generation

Nonlinear Trajectory Generation (NTG), developed at Caltech by Mark Milam [34], is a tool to solve optimal control problems. NTG generates optimal trajectories in real-time for nonlinear systems. Successful applications of NTG include navigating under-water gliders guided by non-neglectable ocean currents [35], flight path optimization in the presence of wind or multiple radars [36].

Consider the system described by (4.1), $x(t)$ represents the state of the system, and $u(t)$ is the control input for $t \in [t_0, t_f]$. In the realm of real-analytic, it is desired to find a trajectory x_d to minimize the cost:

$$J = \phi_f(x(t_f), u(t_f)) + \phi_0(x(t_0), u(t_0)) + \int_{t_0}^{t_f} L(x(t), u(t))dt \qquad (4.2)$$

subject to

$$
\begin{aligned}
lb_0 &\leq \psi_0(x(t_0), u(t_0)) \leq ub_0, \\
lb_f &\leq \psi_f(x(t_f), u(t_f)) \leq ub_f, \\
lb_t &\leq S(x, u) \leq ub_t,
\end{aligned}
\qquad (4.3)
$$

respectively. The cost function J can be break down to three components: ϕ_f represents the final condition; ϕ_0 is the initial condition, and L is an integral cost over the trajectory. The above equations represent a standard optimal control problem. With complicated constraints and dynamics, this kind of problems become too difficult to be solved analytically. Fortunately, with the availability of large number solvers in nonlinear programming (NLP), the above problem could be solved by transforming them into NLP problems, which is the method that NTG employs.

The NTG approach consists of three main steps [34]. The first is mapping the system (4.1) to a lower dimensional output space. This means determining a set of output so the cost function (4.2) and constraints (4.3) can be mapped to a lower output space. The second step is to parameterize the outputs in terms of B-spline curves. Finally, nonlinear programming is used to solve the B-splines coefficients in output space, which will minimize the cost subject to the constraints.

Mapping outputs

The purpose of this step is to find an output z of system described by (4.1) in the form of:

$$z = \alpha(x, u, u^{(1)}, \ldots, u^{(r)}) \qquad (4.4)$$

where $u^{(i)}$ is the ith derivative of u with respect to time. The system needs to be *differentially flat* in order to be further investigated. A system of the form as in (4.1) is said to be differentially flat [33] if (x, u) can be completely recovered from:

$$
\begin{aligned}
x &= \beta(z, z^{(1)}, \dots, z^{(s)}) \\
u &= \gamma(z, z^{(1)}, \dots, z^{(s)})
\end{aligned}
\tag{4.5}
$$

where $z^{(i)}$ is the ith derivative of z with respect to time. A necessary condition for the existence of such an output can be found in [38], cases when a flat output cannot be determined are discussed in [39]. For a differentially flat system, all of the feasible trajectories for the system can be written as functions of a flat output z and its derivatives [33].

Parameterization with B-Splines

The second step of NTG is to represent the outputs in terms of B-Spline curves [39]. The reason for this step is that the system model (4.1) and the constraints (4.3) are usually so complicated that to minimize the cost function (4.2) becomes very difficult. One approach to solving optimal control problems, which NTG employs, is to transform them into NLP problems using B-spline functions, and solve the problem numerically.

In this step, the outputs found in the previous step are parameterized with a finite-dimensional approximation. B-splines are desirable as basis of functions to parameterize the outputs because of their compact (local) support, ease of enforcing continuity at breakpoints, and numerical stability. This process is performed as for each output z_i, with order k_i, continuity C^s or smoothness s_i, and knot breakpoints $\tau_i = t_0, \dots, t_{K_i}$ will be selected in consideration of the maximum derivative that occurs in the output and the number of desired decision variables [41]. The detailed parameterization is done as follows, and a sample spline trajectory is depicted in Figure 4.2.

$$z_1 \quad = \quad \sum_{i=1}^{q_1} B_{i,k_1}(t) C_i^1 \text{ for the knot breakpoint sequence } \tau_1$$

$$z_2 \quad = \quad \sum_{i=1}^{q_2} B_{i,k_2}(t) C_i^2 \text{ for the knot breakpoint sequence } \tau_2$$

$$\vdots$$

$$z_n \quad = \quad \sum_{i=1}^{q_n} B_{i,k_n}(t) C_i^n \text{ for the knot breakpoint sequence } \tau_n$$

where $q_i = l_i(k_i - s_i) + s_i$ and l_i is the number of knot intervals for the ith output. $B_{i,k_i}(t)$ is the B-spline basis function. With the outputs parameterized as above, the coefficients C will be found using nonlinear programming.
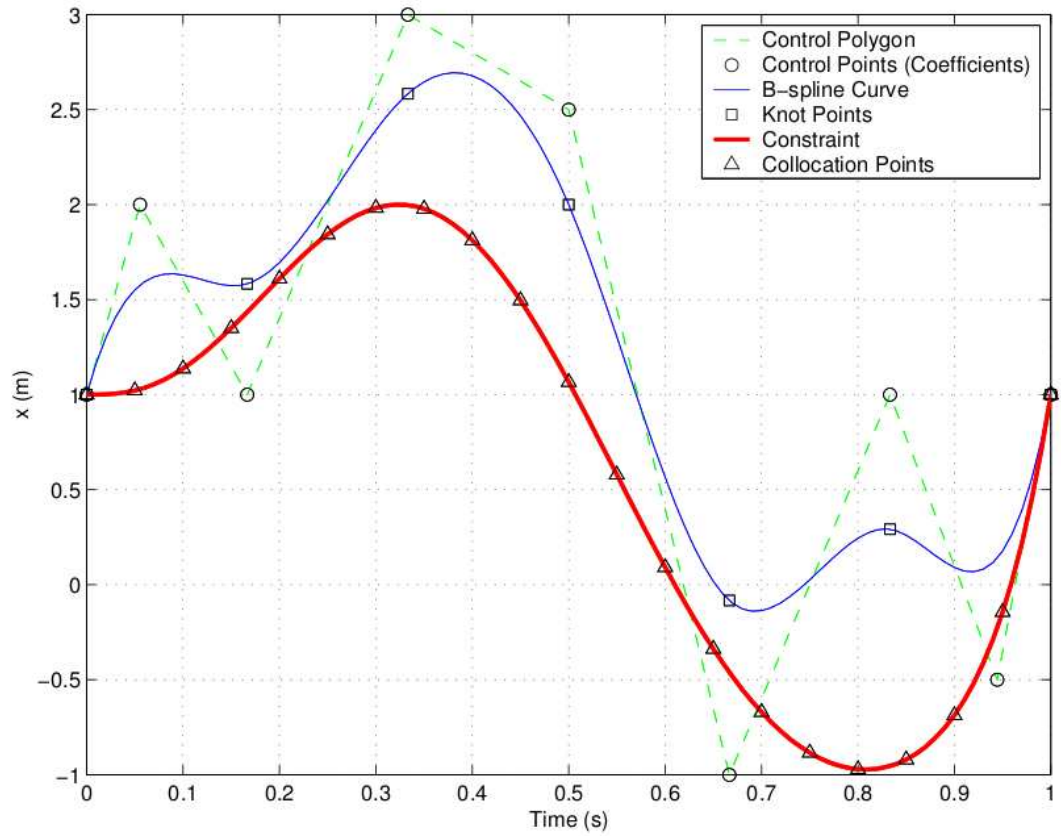
FIGURE 4.2: In this hypothetical problem, the B-spline curve has six intervals ($l = 6$), fourth order ($k = 4$), and is C^3 at the breakpoints (or smoothness $s = 3$). The constraint on the B-spline curve (to be larger than the constraint in this example) will be enforced at the 21 collocation points. The nine control points are the decision variables [41].

Transformation into Nonlinear Programming Problem

The final step in NTG is to solve the B-spline coefficient using sequential quadratic programming packages such as CFSQP and NPSOL [39] [40]. Suppose (4.2) and (4.3) are evaluated discretely between time interval $[t_0, t_n]$, they can be translated into the following NLP problem in C_j [40]:

$$\mathtt{min} F(\overrightarrow{C}), \qquad \overrightarrow{C} \in \mathbb{R}^p \tag{4.6}$$

subject to

$$L \leq G(\overrightarrow{C}) \leq U \tag{4.7}$$

where $F(\overrightarrow{C})$ is the transformed cost function (4.2), and $\overrightarrow{C} = [C_1 \ldots C_p]^T$. $G(\overrightarrow{C})$ is the transformed constraints (4.3). Now the only thing to do is solving the transformed problem with any nonlinear programming tool, for example, NPSOL.

In this section the trajectory generator that MARIT replies on, NTG, is introduced. In the next section, several trajectory tracking problems in MARIT will be modeled and programmed with NTG.

4.2 Trajectory Generation in MARIT

MARIT controls multiple UAVs in a closed-loop. According to the dynamics analysis done in previous chapters, the quadrotor control system is considered as linear when the angles of rotation are small, as is shown in (2.14). To maintain such a linear system at hovering, linear control techniques, such as LQR and simple feedback controllers are theoretically sufficient. This approach is frequently referred to as a *one degree of freedom* design. The one degree of freedom design for MARIT is shown in Figure 4.3, it is also the approach that the previous chapter followed to designing and

testing the controllers.
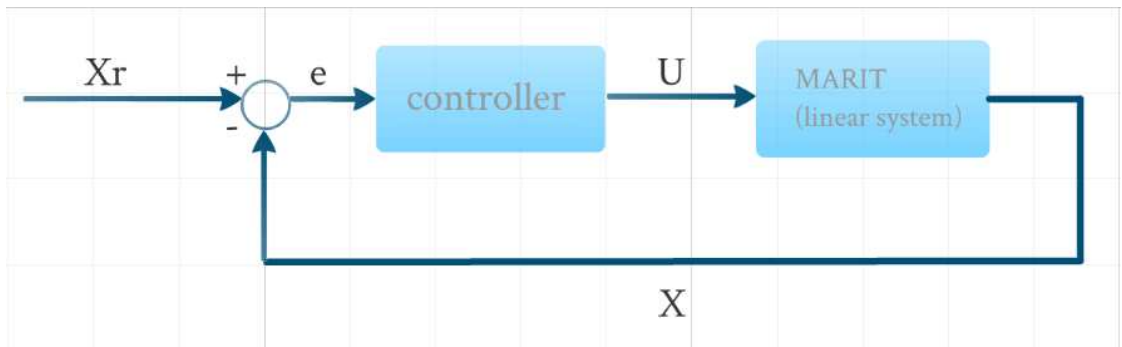
FIGURE 4.3: One degree of freedom design for MARIT. X_r is the provided reference, X is the real-time states feedback, e is the error, and U is the input generated from the controller. The system is linearized as is shown in (2.14).

However, when optimization is taken into consideration, especially optimizing with nonlinear constraints must be performed, the one degree of freedom design generally does not work well since the system is likely tracking drifting equilibrium configuration, which is an infeasible trajectory of the system [34]. The two degree of freedom design as is described in the previous section is a possible solution. Consisting of a trajectory generator and a feedback controller, the two degree of freedom design provides both feasible feedforward reference and feedback stabilization. This more complex structure enables the system to track feasible trajectory while maintaining stability. Figure 4.4 illustrates the two degree of freedom design for MARIT.
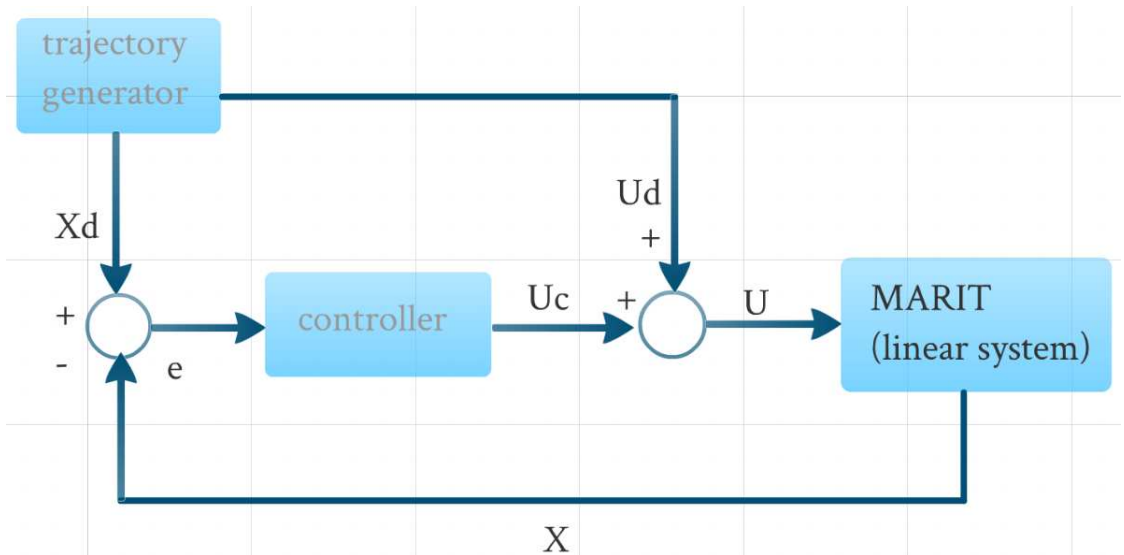
FIGURE 4.4: Two degree of freedom design for MARIT. The trajectory generator calculates and provides the feasible trajectory X_d, including the desired 3D trajectory and yaw angle, and the nominal input U_D, which is a feedforward term. The controller takes the error e calculated by subtracting real-time feedback X from X_d and generates input U_c. The final input to the system is generated by adding up U_d and U_c.

It is to be noted that, while the MARIT system itself is treated as linear, the trajectory generator might be nonlinear. This is why NTG is considered for generating feasible trajectories for the MARIT testbed.

4.2.1 Paring NTG with MARIT

Figure 4.5 illustrates how NTG functions in the MARIT environment. NTG takes in constraints from the operator and generates trajectory for MARIT. These constraints include the initial, trajectory and final constraints, and cost functions to minimize. Details will be covered in the next section.

Generally the controls of the quadrotors in MARIT are grouped into two sections because of their similarity in design. One is the latitude and longitude (x, y) control, the

other is the altitude (z) and yaw (ψ) control. The design of the controllers are described in the last chapter, and this section will focus on the trajectory generation.

The goal of control is to track trajectories given in 3D. Considering the dynamics of the quadrotor, it is impossible to control the positions x, y directly, so the nested controller structure was designed as described in the last chapter. This brings up an underlying constraint that the angles of rotation must be kept small during the whole trajectory tracking, because only when the angles are small, the system can be considered linear. These constraints are specified in the next section.

In the upper half of Figure 4.5, the NTG component provides feasible reference for altitude (z) and yaw angle (ψ), and also the nominal inputs for both. In the lower half NTG does the same thing to the x and y positions, but provides two sets of nominal inputs, U_{xd}, U_{yd} and U_{thetad}, U_{phid}. This is because the linear relationships between the positions x, y and angles θ, ϕ as described in (2.14) result in a nested controller structure.
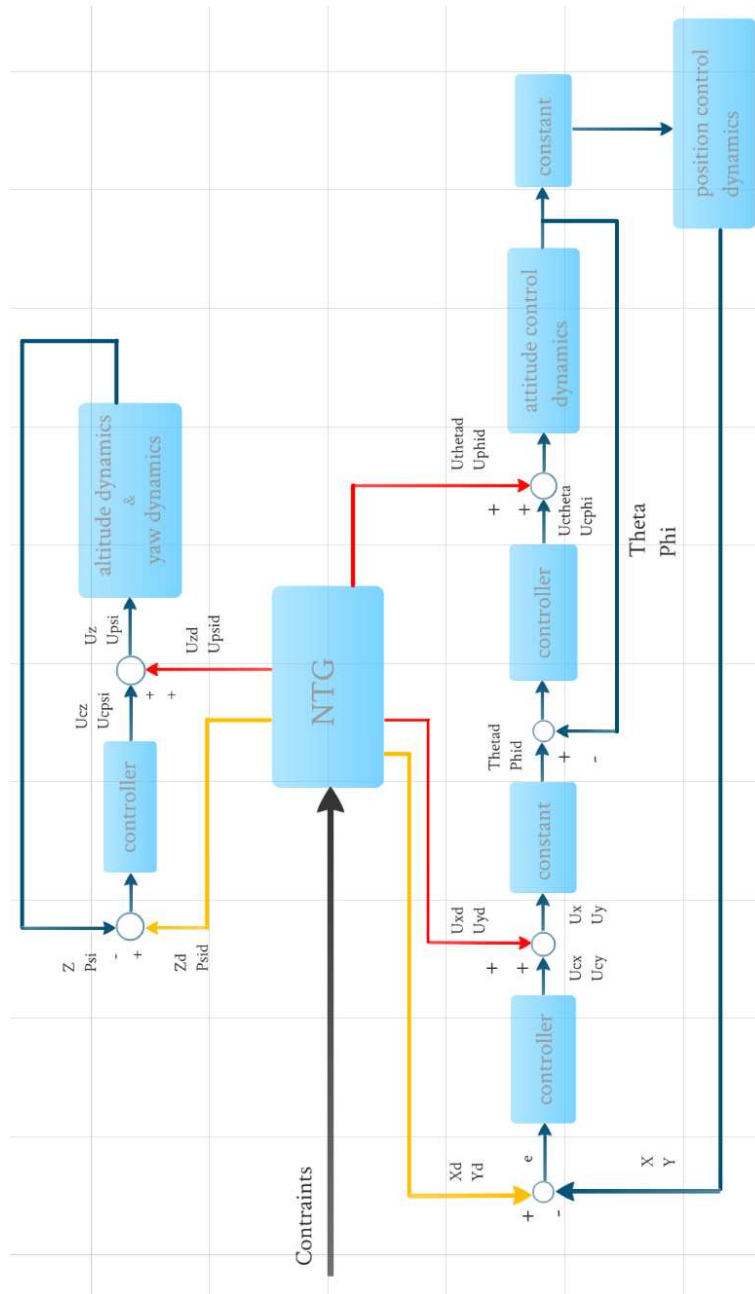
FIGURE 4.5: Incorporating NTG in MARIT control design.

4.2.2 Trajectory Generation

Typical applications of trajectory generation include obstacle avoidance by a robotic vehicle, minimum time missile interception of an agile target, formation flight

of micro satellites with coverage constraints, and a rapid change of attitude for an un-
manned flight vehicle to evade a dynamic threat [41]. In the MARIT testbed envi-
ronment, a series of simulations were designed and conducted by combining obstacle
avoidance with other constraints to evaluate the performance of NTG. The dynamics
system is modeled as shown by (2.14), and the feedback controllers are nested PID
control designed as described in the previous chapter.

The general goal of these simulations are stated below:

- Moving from the initial location to the destination in 3D space

- Avoiding multiple (randomly appearing) obstacles during the flight

- Visiting several fixed waypoints

- Minimize trajectory length and/or kinetic energy during the flight

- Guarantee precision of initial and final location

- Maintain the angles of rotation as needed to ensure the linear model is valid

- Limiting the speed and accelerations along x, y and z

- Limiting the speed and accelerations of angles of rotation ϕ, θ and ψ as needed

Figure 4.6 shows a possible scenario of obstacle avoidance in MARIT. The
quadrotor will start from the initial location (the green dot at bottom left) and fly to the
destination (the green dot at upper right). There are four obstacles (colored big spheres)
located in the way to prevent the quadrotor from flying directly to the destination. Keep-
ing a safe distance from the obstacles is required. Two waypoints (green dots) are to be
visited during the flight. To make the trajectory harder to generate and track, one of the
waypoints (waypoint#1) is located very close one of the obstacles (on the surface). The
above mentioned constraints are used to regulate the trajectory generation, and the flight
time is fixed between neighboring waypoints.
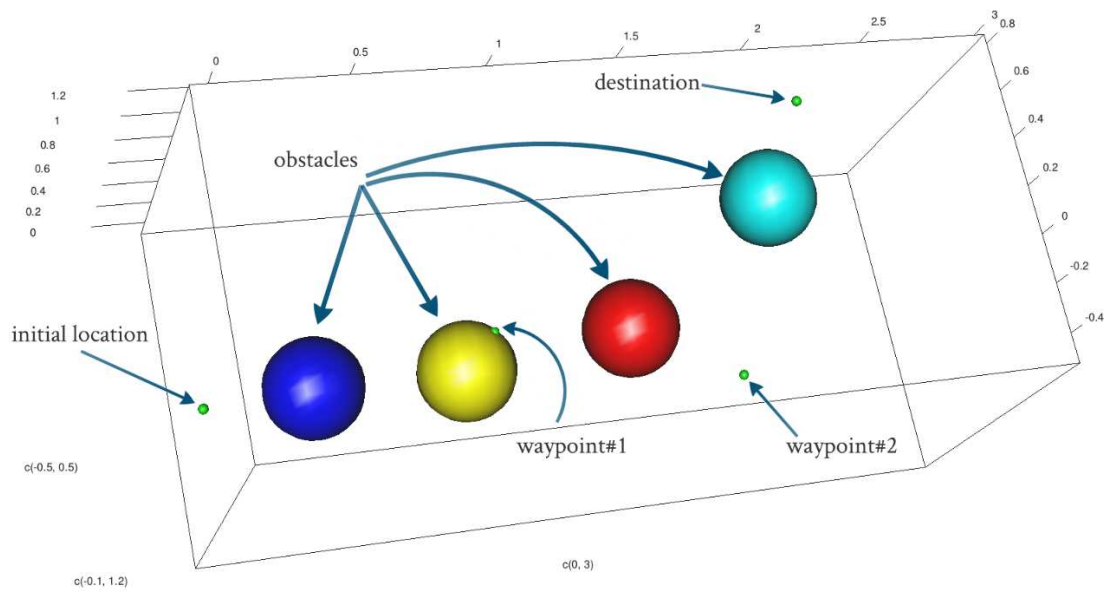
FIGURE 4.6: Objective of obstacle avoidance in MARIT. The UAV is supposed to fly from the initial location to the destination in a 3D space. During the flight, two waypoints are set to be visited. In this illustration, four obstacles depicted as spheres are to be avoided.

As described in the previous section, NTG follows three steps to generate the optimal trajectory. The first one is to map the outputs to a lower dimension. In order to do this, the system needs to be differentially flat. In order to prove the differential flatness of the MARIT quadrotor model, the states of the system are listed below, with r_x, r_y and r_z denoting the distance along x, y and z axis, and ϕ, θ and ψ denoting the angles of rotation:

$$
x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \\ x_{17} \\ x_{18} \end{bmatrix} = \begin{bmatrix} r_x \\ \dot{r}_x \\ \ddot{r}_x \\ r_y \\ \dot{r}_y \\ \ddot{r}_y \\ r_z \\ \dot{r}_z \\ \ddot{r}_z \\ \phi \\ \dot{\phi} \\ \ddot{\phi} \\ \theta \\ \dot{\theta} \\ \ddot{\theta} \\ \psi \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} \tag{4.8}
$$

and the inputs to the system is:

72

$$u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{r}_z \\ \ddot{\psi} \end{bmatrix} \tag{4.9}$$

By choosing the output z as:

$$z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \\ \psi \end{bmatrix} \tag{4.10}$$

According to the linearization of (2.14), $\ddot{r}_x$ is in linear relationship with θ, and $\ddot{r}_y$ is in linear relationship with ϕ, it is obvious by observation that x and u is able to be written as the functions of z and its finite number of derivatives. Thus the rule stated by (4.5) is satisfied. Now it is shown that the MARIT quadrotor dynamics model defined above by (4.8) and (4.9) with linearization (2.14) is differentially flat.

With the outputs mapped, the constraints are ready to be specified in terms of flat outputs. In the scenario illustrated by Figure 4.6, the constraints and cost function are defined as in Table 4.1. Each of the constraints and cost function is listed below, where G is the gravitation acceleration, determined to be 9.81:

TABLE 4.1

Item Name	Abbreviation	count
Linear Initial Constraints	LIC	16
Linear Trajectory Constraints	LTC	12
Linear Final Constraints	LFC	6
Nonlinear Initial Constraints	NLIC	0
Nonlinear Trajectory Constraints	NLTC	4
Nonlinear Final Constraints	NLFC	0
Cost Function	N/A	1 or 2

LIC

$$z_0(0) = r_x(0), z_1(0) = \dot{r}_x(0),$$

$$z_2(0) = \ddot{r}_x(0) = G\theta(0), z_3(0) = \dddot{r}_x(0) = G\dot{\theta}(0),$$

$$z_4(0) = \ddddot{r}_x(0) = G\ddot{\theta}(0), z_5(0) = r_y(0),$$

$$z_6(0) = \dot{r}_y(0),$$

$$z_7(0) = \ddot{r}_y(0) = -G\phi(0),$$

$$z_8(0) = \dddot{r}_y(0) = -G\dot{\phi}(0),$$

$$z_9(0) = \ddddot{r}_y(0) = -G\ddot{\phi}(0),$$

$$z_{10}(0) = r_z(0),$$

$$z_{11}(0) = \dot{r}_z(0),$$

$$z_{12}(0) = \ddot{r}_z(0),$$

$$z_{15}(0) = \psi(0),$$

$$z_{16}(0) = \dot{\psi}(0),$$

$$z_{17}(0) = \ddot{\psi}(0)$$

(4.11)

The linear trajectory constraints are shown in (4.12). These constraints are the

ones that the UAV needs to satisfy during the flight. Each output z falls into a close range between a minimum and a maximum value. The r_x, r_y and r_z are limited within an area to prevent the UAV from flying out of the MARIT indoor environment. The angles of rotation are restricted within a small range to ensure the model to be linear. The speed and accelerations are constrained to make simulation realizable.

$$
\begin{aligned}
&LTC \\
&z_0(t) = r_x(t) \in [r_{xmin}, r_{xmax}], \\
&z_5(t) = r_y(t) \in [r_{y_{min}}, r_{y_{max}}], \\
&z_{10}(t) = r_z(t) \in [r_{z_{min}}, r_{z_{max}}], \\
&z_1(t) = \dot{x}(t) \in [\dot{r}_{xmin}, \dot{r}_{xmax}], \\
&z_5(t) = \dot{y}(t) \in [\dot{r}_{ymin}, \dot{r}_{ymax}], \\
&z_{10}(t) = \dot{z}(t) \in [\dot{r}_{zmin}, \dot{r}_{zmax}], \\
&z_2(t) = G\theta(t) \in [G\theta_{min}, G\theta_{max}], \\
&z_7(t) = -G\phi(t) \in [-G\phi_{max}, -G\phi_{min}], \\
&z_{15}(t) = \psi(t) \in [\psi_{min}, \psi_{max}], \\
&z_9(t) = \ddot{\phi}(t) \in [\ddot{\phi}_{min}, \ddot{\phi}_{max}], \\
&z_4(t) = \ddot{\theta}(t) \in [\ddot{\theta}_{min}, \ddot{\theta}_{max}], \\
&z_{17}(t) = \ddot{\psi}(t) \in [\ddot{\psi}_{min}, \ddot{\psi}_{max}],
\end{aligned}
\tag{4.12}
$$

The linear final constraints are listed in (4.13). The final states of the 6DOF are provided by the operator. The values of these final states should be checked to insure the feasibility of the trajectory. For example, a location or angle too far or too large will

be infeasible to achieve with other constraints satisfied.

$$LFC$$
$$z_0(f) = r_x(f),$$
$$z_5(f) = r_y(f),$$
$$z_{10}(f) = r_z(f), \tag{4.13}$$
$$z_2(t) = G\theta(f),$$
$$z_7(t) = -G\phi(f),$$
$$z_{15}(t) = \psi(f),$$

In this scenario no nonlinear initial constraints and nonlinear final constraints are present. The nonlinear trajectory constraints are listed in (4.14). The x_{ob}, y_{ob} and z_{ob} are the location of the obstacles, d_{min} and d_{max} are the distance bounds to keep from the obstacles.

$$NLTC$$
$$\sqrt{(r_x - x_{ob1})^2 + (r_y - y_{ob1})^2 + (r_z - z_{ob1})^2)} \in [d_{min}, d_{max}],$$
$$\sqrt{(r_x - x_{ob2})^2 + (r_y - y_{ob2})^2 + (r_z - z_{ob2})^2)} \in [d_{min}, d_{max}], \tag{4.14}$$
$$\sqrt{(r_x - x_{ob3})^2 + (r_y - y_{ob3})^2 + (r_z - z_{ob3})^2)} \in [d_{min}, d_{max}],$$
$$\sqrt{(r_x - x_{ob4})^2 + (r_y - y_{ob4})^2 + (r_z - z_{ob4})^2)} \in [d_{min}, d_{max}],$$

According to the purpose of the operation, it is up to the operator to decide which cost function to use. Two different cost functions have been tested. One is to minimize the kinetic energy during the flight, the other is to minimize the total displacement during the flight. These are shown in (4.15) and (4.16) below, where m denotes the mass of the UAV, I_{xx}, I_{yy} and I_{zz} denote the moment of inertia around three body frame axes.

$$\frac{1}{2}m(\dot{r}_x)^2 + \frac{1}{2}m(\dot{r}_y)^2 + \frac{1}{2}m(\dot{r}_z)^2 + \frac{1}{2}I_{xx}(\dot{\phi})^2 + \frac{1}{2}I_{yy}(\dot{\theta})^2 + \frac{1}{2}I_{zz}(\dot{\psi})^2 \tag{4.15}$$

$$\int_0^T \left[\sqrt{(\dot{r}_x)^2 + (\dot{r}_y)^2 + (\dot{r}_z)^2} \right] \mathrm{d}t \tag{4.16}$$

TABLE 4.2

Parameter	Notation	Value
intervals	l	4
order	k	5
smoothness	s	3
number of coefficients	q	l(k - s) + s
breakpoints	nbps	45

With all the constraints and cost functions defined, the second step in NTG is to parameterize the outputs with B-splines as the form of $z_i(t) = \sum_{j=1}^{q_i} B_{j,r_i}(t)C_j^i$. To do this, the following parameters need to be determined by the operator, the values are chosen through trial and error to achieve good performance in the MARIT model.

After the outputs are parameterized, the problem is ready to be solved by NTG using nonlinear programming tool. NTG uses NPSOL [42] to solve the nonlinear programming problems that are converted from the original optimal control problems. NTG is programmed to solve the trajectory generation task for MARIT in a "discretized" method. The process flow is shown in Figure 4.7. NTG runs in a controlled looping to generate trajectories. Obstacles information will be updated in each loop, constraints are adjusted accordingly, and new trajectories are generated to fulfill the updated constraints.
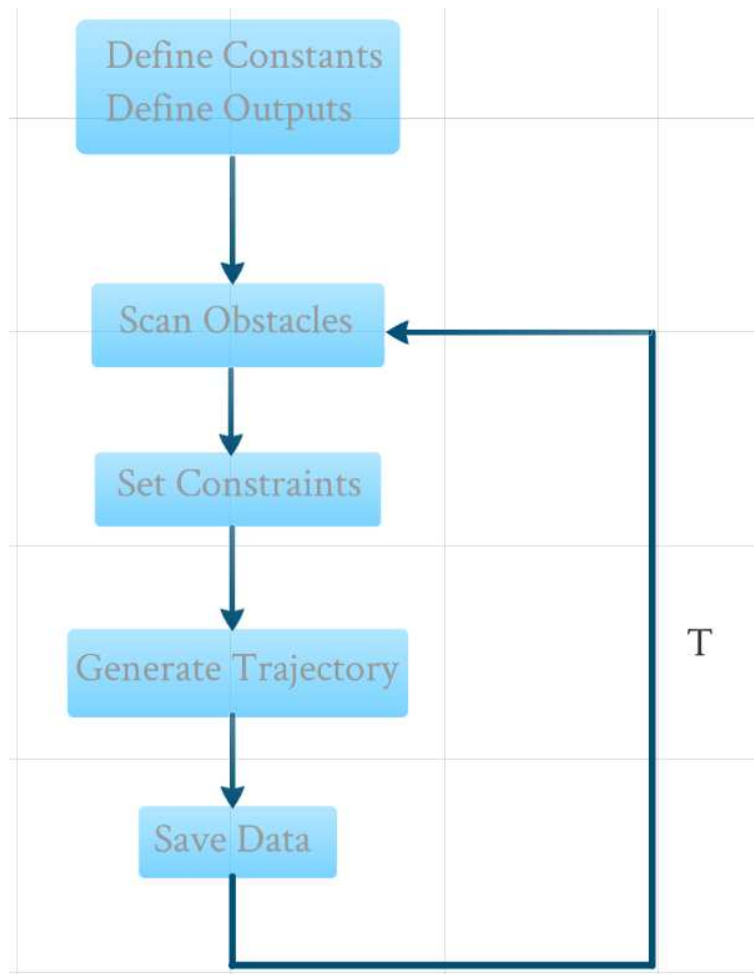
FIGURE 4.7: NTG starts by defining the constants and outputs needed by every loop. In entering the main loop, the obstacles locations and dimensions are updated, and corresponding constraints are set and updated. The trajectory is generated according to the constraints and data is saved for each loop before going into the next loop. Looping time T is fixed and can be set by the operator.

To evaluate NTG's ability of real-time trajectory generating in MARIT environment, two sets of designs are carried out in the simulation. Both are for obstacle avoidance. The first one scan for obstacle at preset waypoints and generates optimal reference trajectories. The second one scan for obstacles at a higher frequency and updates reference trajectories once new obstacle is detected.

"Mode 1" scenario has no obstacle blocking the path, NTG generates optimal trajectory without obstacle avoidance. The generated trajectory starts from the initial location and passes through two waypoints, and arrive at the destination. Figure 4.8 illustrates the trajectories generated for mode 1 from four different perspectives.
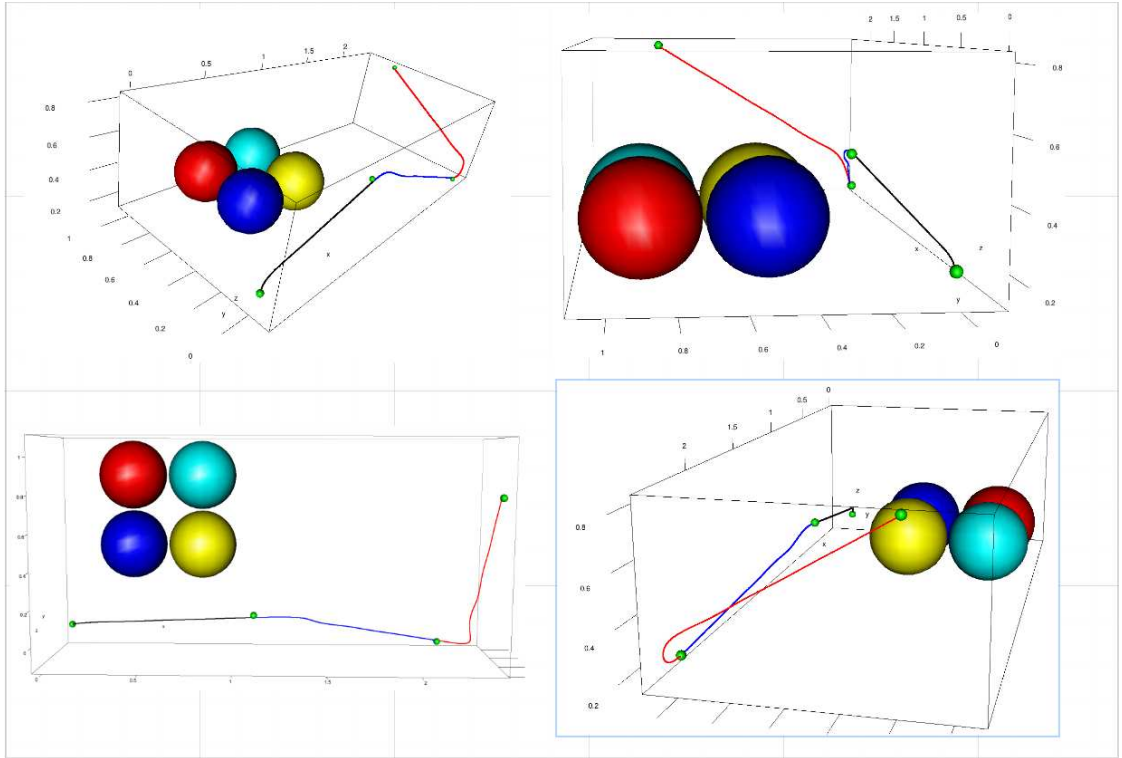
FIGURE 4.8: Mode 1 scenario, no obstacle lies on the path. The generated trajectory minimizes displacement during the flight. Since no obstacle avoidance is performed, the trajectory between each waypoints is close to a straight line.

In "Mode 2" there are 1 obstacle located on the path between the initial location and the first waypoint (the small green dot) near the initial location. NTG is able to recognize the obstacle and plan trajectory that avoids it and satisfying the other constraints. Trajectories are generated to minimize the displacement during the flight. Figure 4.9 shows the trajectory and obstacles in 4 different perspectives.
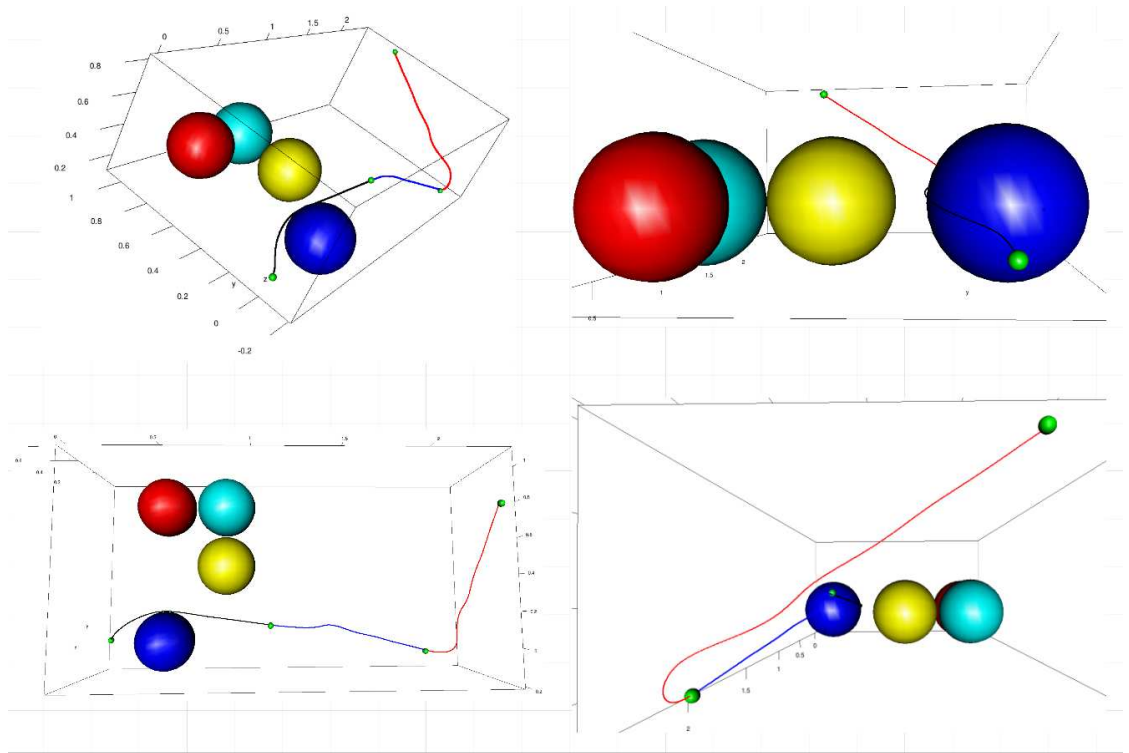
FIGURE 4.9: Mode 2 scenario, one obstacle (the blue sphere) has moved to block the path, locating close to the initial location. NTG detected the existence of the obstacle and generates trajectories that avoid the obstacle. Cost function is to minimize the displacement during the flight. After avoiding the blue sphere, the rest of the trajectories are close to straight lines.

"Mode 3" has two obstacles located on the path (the blue sphere and yellow sphere). One is in between the initial location and the first waypoint, the second is located between the first and the second waypoint. Figure 4.10 illustrates the trajectories generated for scenario 3. NTG reacts to the existence of the yellow sphere by flying the UAV to the left and passes the yellow sphere closely from the left side.

80
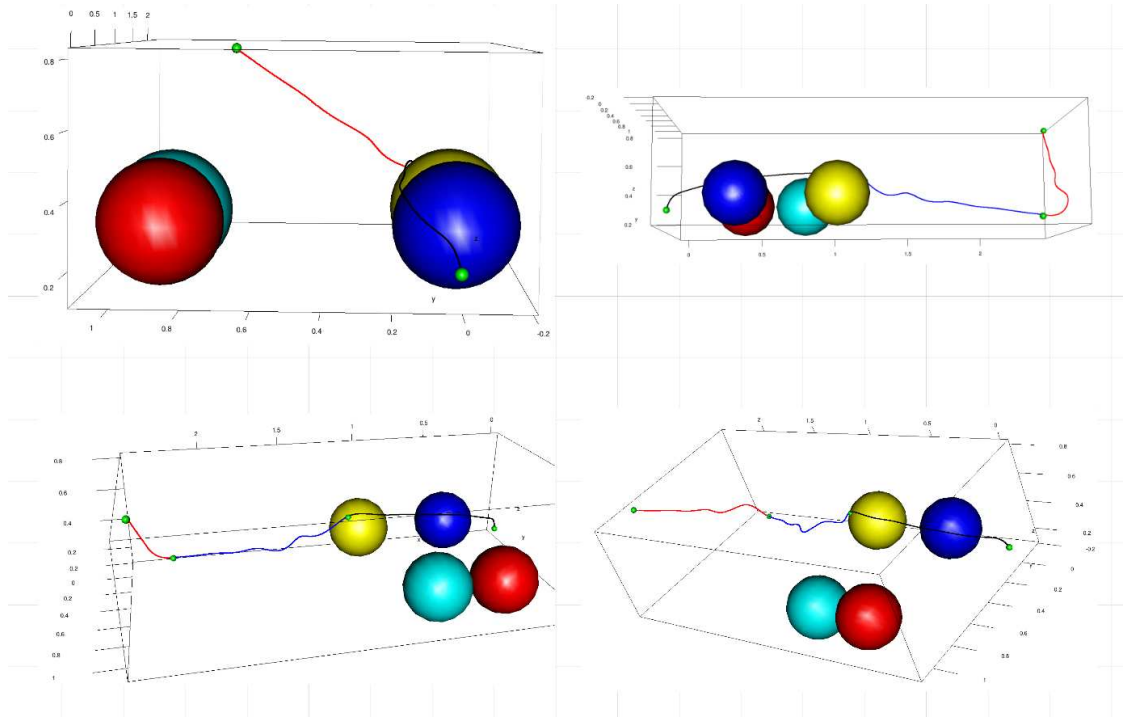
FIGURE 4.10: Mode 3 scenario has two obstacles blocking the path. NTG is able to avoid both obstacles and reach the first waypoint even though it is located on the surface of the second obstacle. After avoiding the yellow sphere, the trajectory continues and passes through the second waypoint and finally arrives at the destination.

Figure 4.10 shows mode 4 trajectory generation. Three obstacles appear on the path and NTG is able to avoid all of them and generate the optimal trajectory while minimizing the whole displacement during the flight.
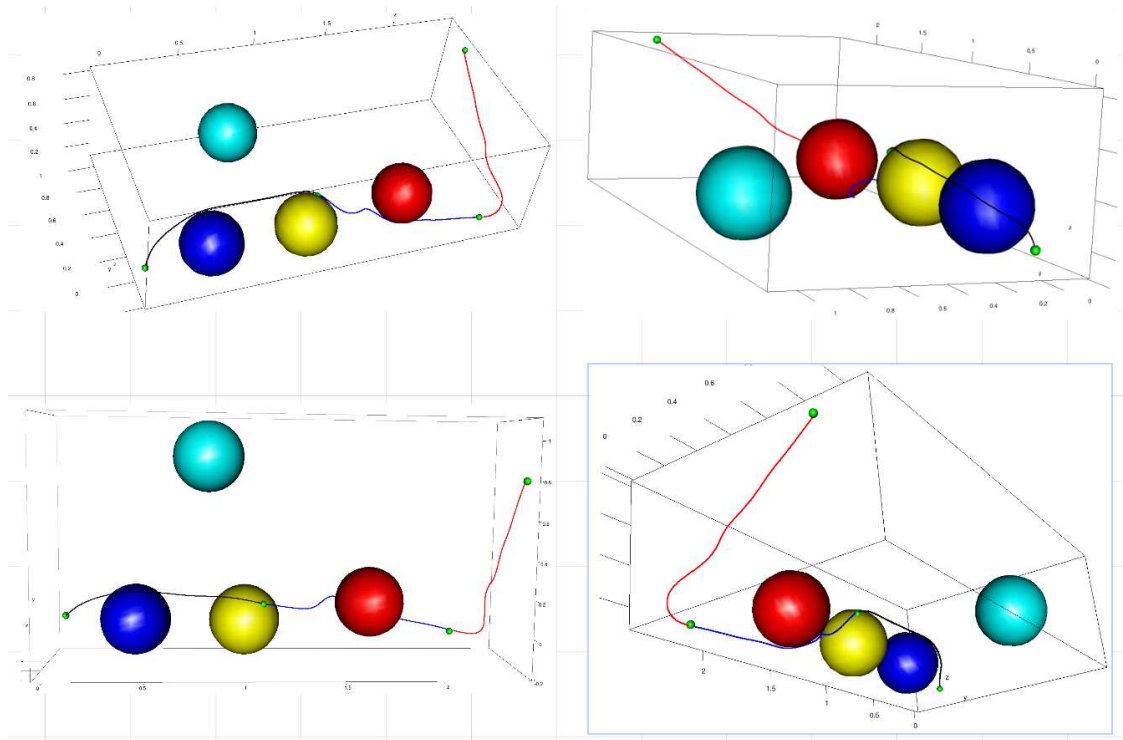
FIGURE 4.11: Mode 4 scenario features three obstacles, in addition to the two appeared in Mode 3, a third obstacle (the red sphere) is moved to between waypoint 1 and waypoint 2. NTG reacts to the existence of the red sphere and makes adjustments in planning the trajectory by flying the UAV under the red sphere.

"Mode 5" places four obstacles on the path, in addition to the 3 obstacles in Mode 4, a fourth obstacle (the cyan sphere) is placed between the second waypoint and the destination location. Figure 4.12 shows the different perspectives of the trajectory.
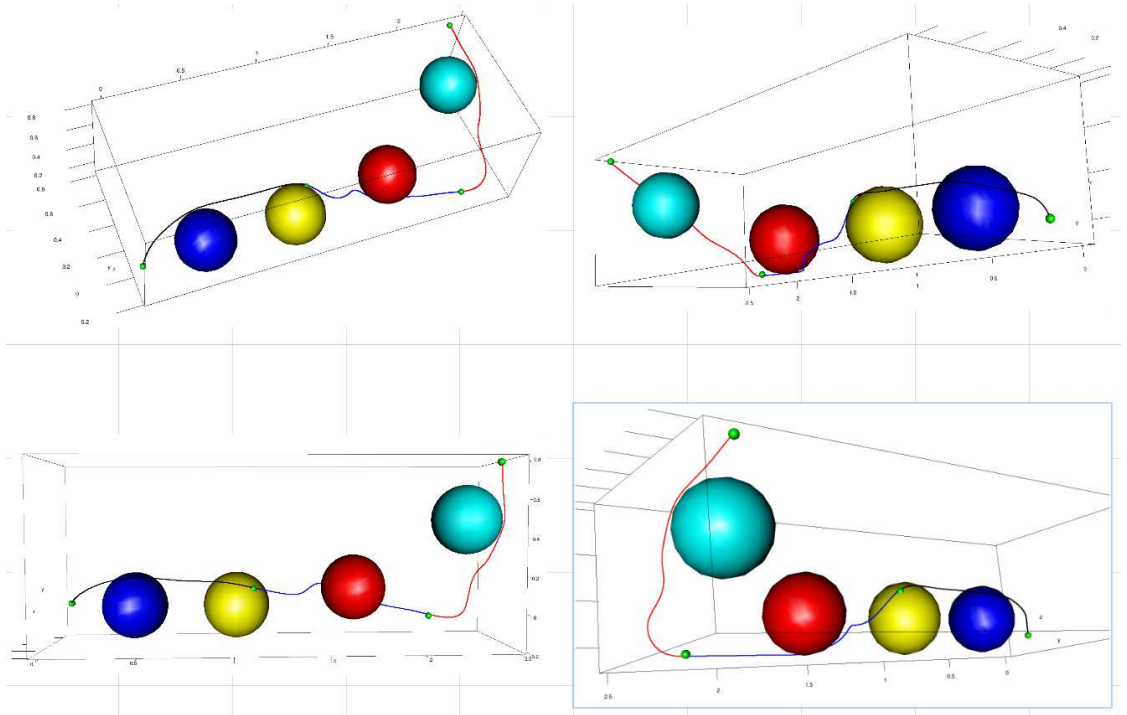
FIGURE 4.12: Mode 5 adds one more obstacle between the second waypoint and the final location. NTG reacts to the fourth obstacle by dodging to the right side of the obstacle. Cost function is still minimizing the displacement during the flight.

Up to this point, NTG is able to complete obstacle avoidance for all 5 obstacle placement scenarios. With the configuration determined in Table 4.2 and constraints provided, NTG proves to be competent for optimal trajectory generation for different obstacle situations in the MARIT simulation environment.

Figure 4.13 shows the several different trajectories NTG produces. The goal is identical to the previous settings, which is to go from the initial location (upper right green dot in this graph), pass through 2 waypoints, and arrive at the destination (lower left green dot). While minimizing the displacement during the flight, NTG ensures the trajectory satisfies the various linear and nonlinear constraints provided previously.
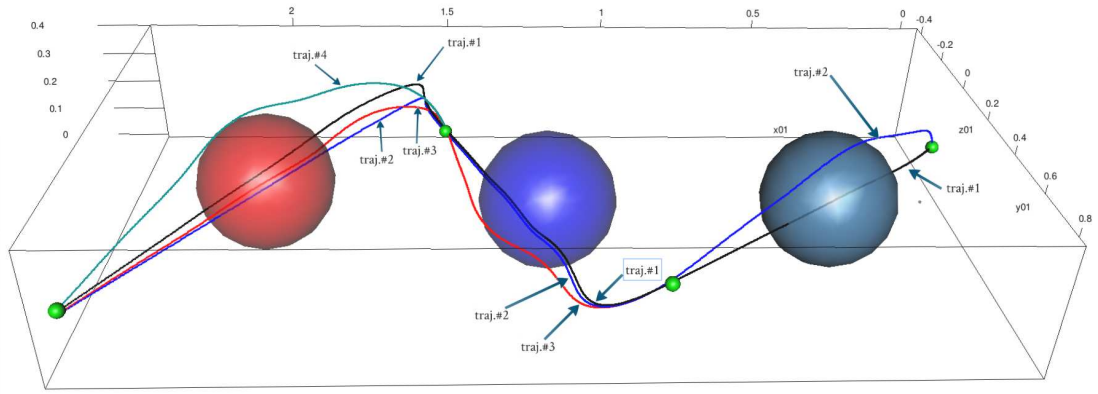
FIGURE 4.13: Real-time obstacle avoidance trajectories. Trajectory 1 does not avoid any obstacle, simply pass through the waypoints and arrives at the destination. Trajectory 2 avoids the first obstacle (the gray sphere) and ignores the other two obstacles. Trajectory 3 avoids both the first and second obstacle (gray and the blue sphere) but ignores the third obstacle. Trajectory 4 avoids all 3 obstacles including the red sphere.

To reflect the real-time calculation with less distractions, a simpler scenario is shown in Figure 4.14. Obstacles may appear between the waypoints, but only one obstacle will be present. When the UAV reaches the first waypoint (the green dot at bottom right), if the smaller obstacle is present, NTG generates the red (lower) trajectory for the UAV to avoid it. If the bigger one is present instead, NTG generates the black trajectory to go over the obstacle. These two different piece of trajectory merge from and into the same trajectories at the first and second waypoint.
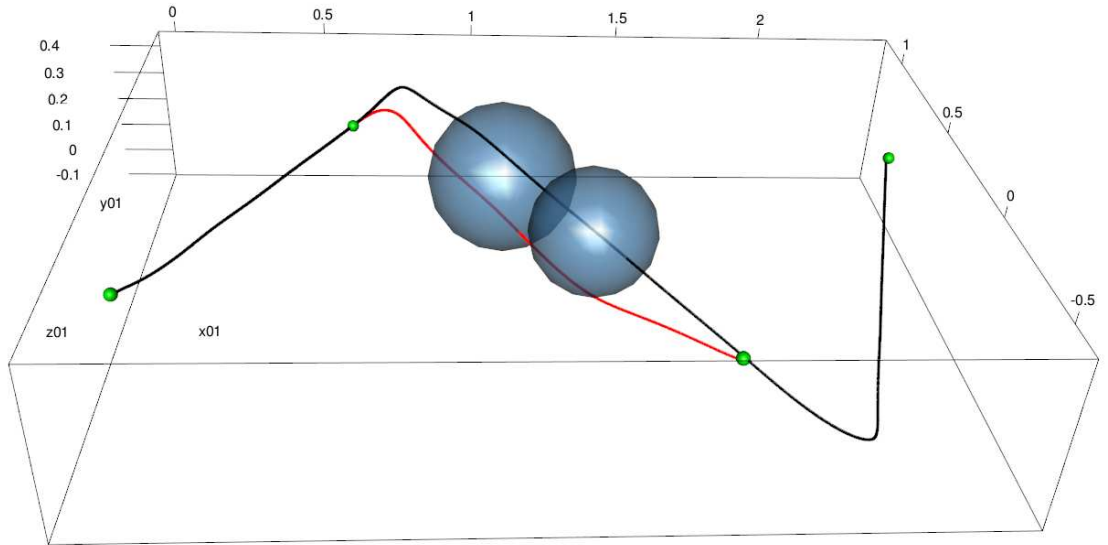
FIGURE 4.14: Another presentation of NTG generating real-time trajectories.

Figure 4.15 shows the view from nearby the second waypoint. The two different trajectories merge into the same trajectory when arriving at the second waypoint.
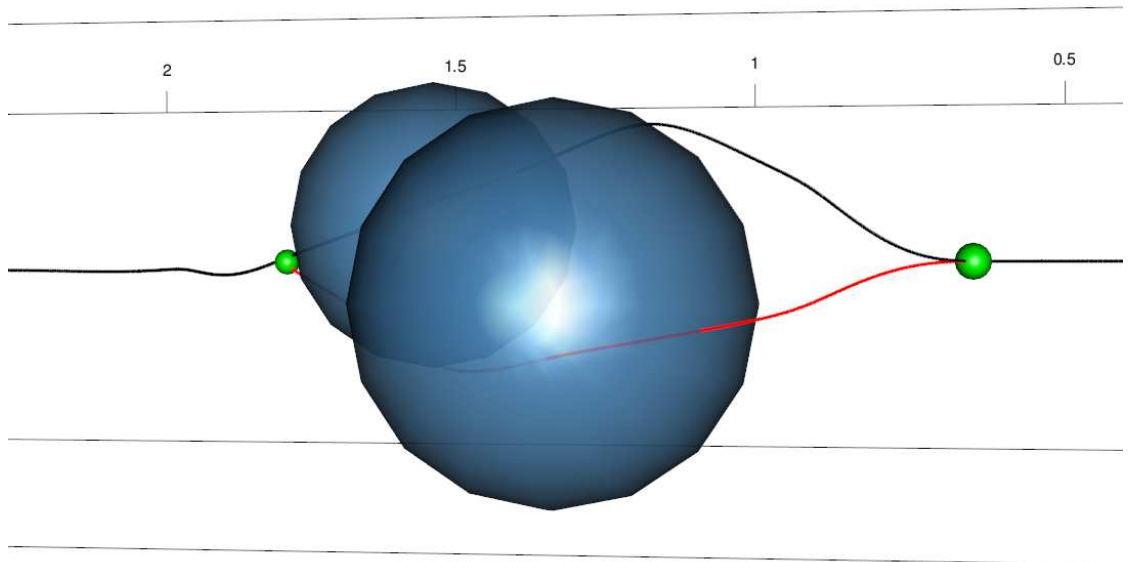
FIGURE 4.15: A different perspective of NTG generating real-time trajectories.

The above control work flow scans obstacles at each waypoint (including the initial point). However, the fact that the location of the obstacles are updated at these points has potential problems. First, if the UAV is on its way from one waypoint to another, it does not scan for new obstacles, which may lead to crushing if an obstacle appears during this period. Second, NTG may take some time for generating new trajectories at each waypoint, then the UAV has no reference until new trajectories are generated. To solve these potential problems, shown in Figure 4.16, a new work flow is designed.
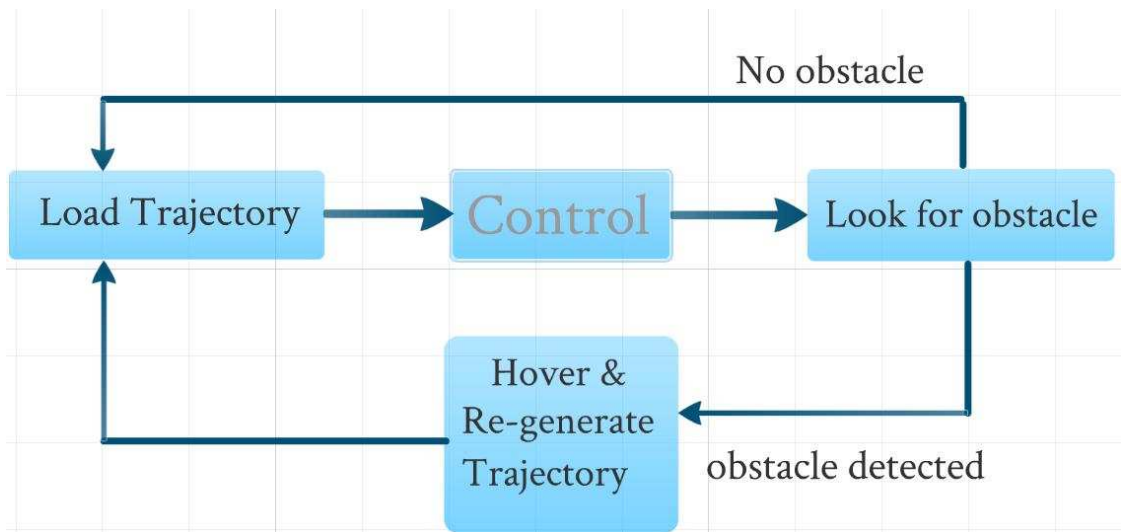
FIGURE 4.16: New work flow to ensure avoidance of obstacles in real-time.

This new work flow begins with loading the default trajectory and use it as the reference in its operations. It scans for obstacles at the end of each control loop, which usually runs at high frequency for air vehicles control. If no obstacle is found, the program goes back to loading reference trajectories and continue the loop; if obstacles are detected, the program sets the UAV to hovering, and re-generate new reference trajectories. Following this new design, the program is able to detect and avoid new obstacles whenever it appears, not just at the waypoints. A simulation of this design is illustrated in Figure 4.17.
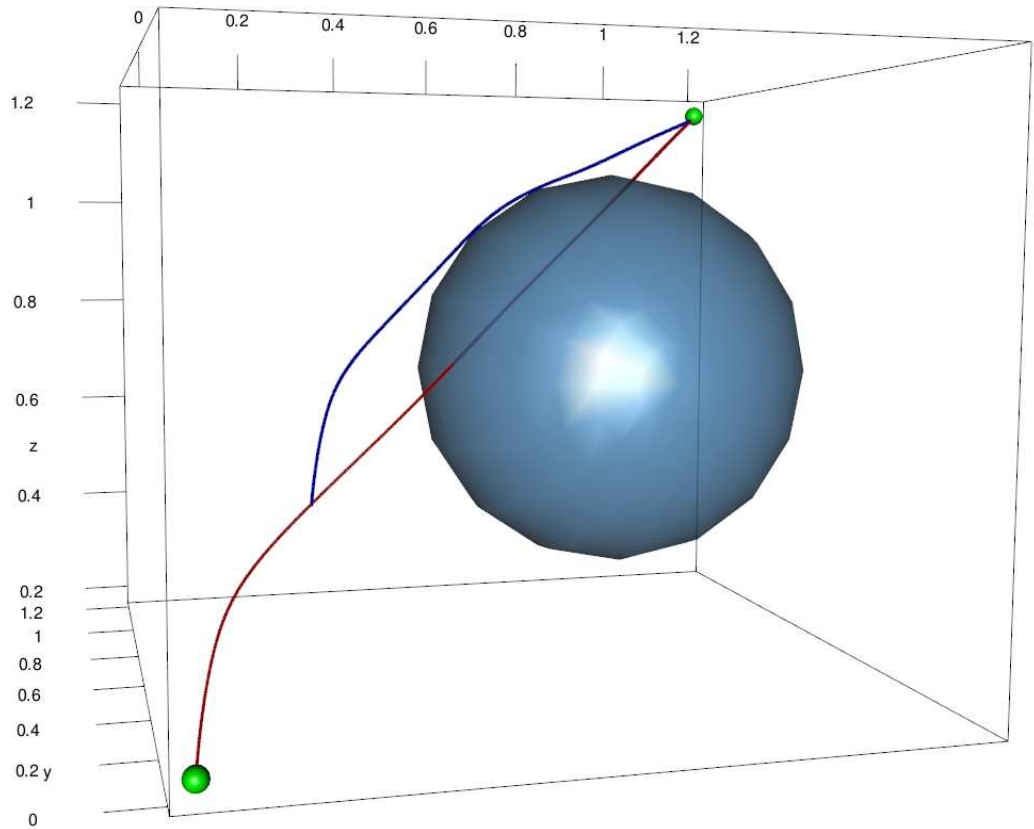
FIGURE 4.17: The program detects and avoid obstacles in real-time. Scanning for obstacles at the end of each control loop, the UAV is set to hovering if new obstacle is found and wait for NTG to re-generate new trajectories. The default reference is red and new trajectory is blue.

From the examples above, NTG proves to be able to generate real-time obstacle avoiding optimal trajectories for the MARIT quadrotor model. In the next section, controllers designed in the previous chapter will be applied to *track* the trajectories generated by NTG in this section.

4.2.3 Trajectory Tracking

The previous sections shows how NTG generates optimal trajectories for MARIT in various situations. The section will describe how the controllers designed in the pre-

vious chapters works with NTG to track the reference trajectories. Figure 4.5 displays the incorporated structure of nested controllers working with NTG to track trajectories. The tracking adopts the two degree of freedom design as shown in Figure 4.4. NTG provides the controllers with not only the reference trajectories, but also the nominal inputs to produce the reference trajectories. The reference trajectories are used to combine with the feedback to generate the controller input, and the nominal input is used as feedforward term to add to the controller input. The final input to the system is the sum of controller input and the feedforward nominal input. The altitude and yaw control are performed separately from the latitude and longitude control.
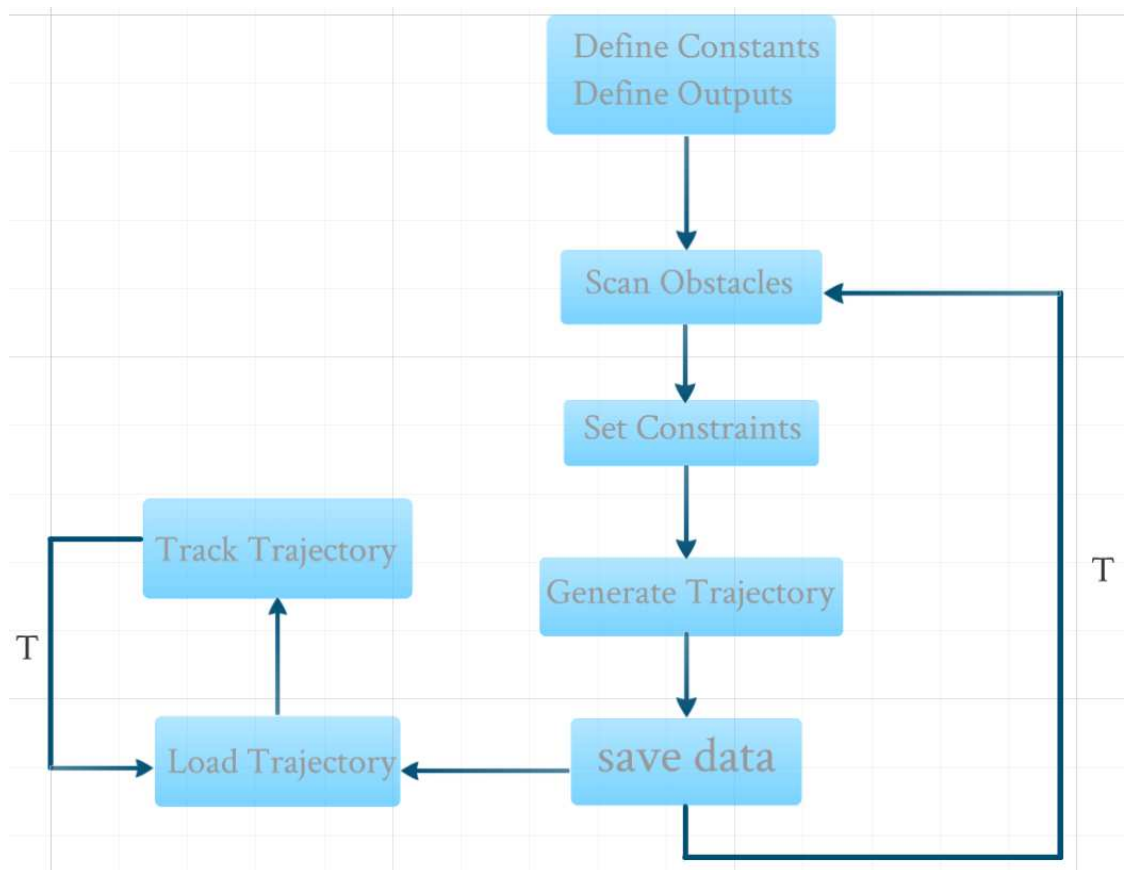

FIGURE 4.18: The controllers load the reference trajectories and nominal inputs from NTG and tracks the trajectories. Each "loading" contains reference data for time duration "T".

Figure 4.18 illustrates the tracking in the framework of trajectory generation. The controllers load the reference data and nominal inputs to generate the final inputs to the system. The reference data include the outputs z as defined in (4.10) and its derivatives. The nominal inputs that are used in the feedforward contains the values corresponding to the system inputs u as defined in (4.9). Figure 4.19 shows the data that is loaded from NTG and used to generate the inputs to the system.
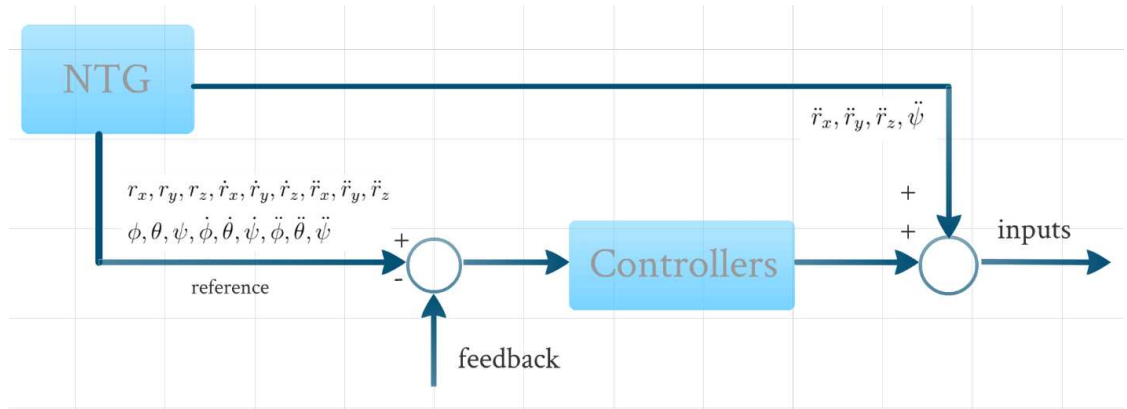
FIGURE 4.19: The reference contains the output z and its derivatives, the nominal input contains the u terms.

The obstacle avoidance "mode 5" scenario is chosen to demonstrate the tracking because it has more obstacles than other modes. Figure 4.20 shows the result. As is in mode 5, the reference starts from the initial location and passes through two waypoints before arrives at the destination at the upper right corner of the graph. Four obstacles with the shape of spheres are on the path to block the UAV. The actual start point of the UAV is different from the initial location of the reference and is located on the ground ($r_z = 0$). This designed on purpose to examine how fast the controllers can react to the large difference. The reference is displayed by the black string of dots, and the tracking trajectory is in dark green.

89
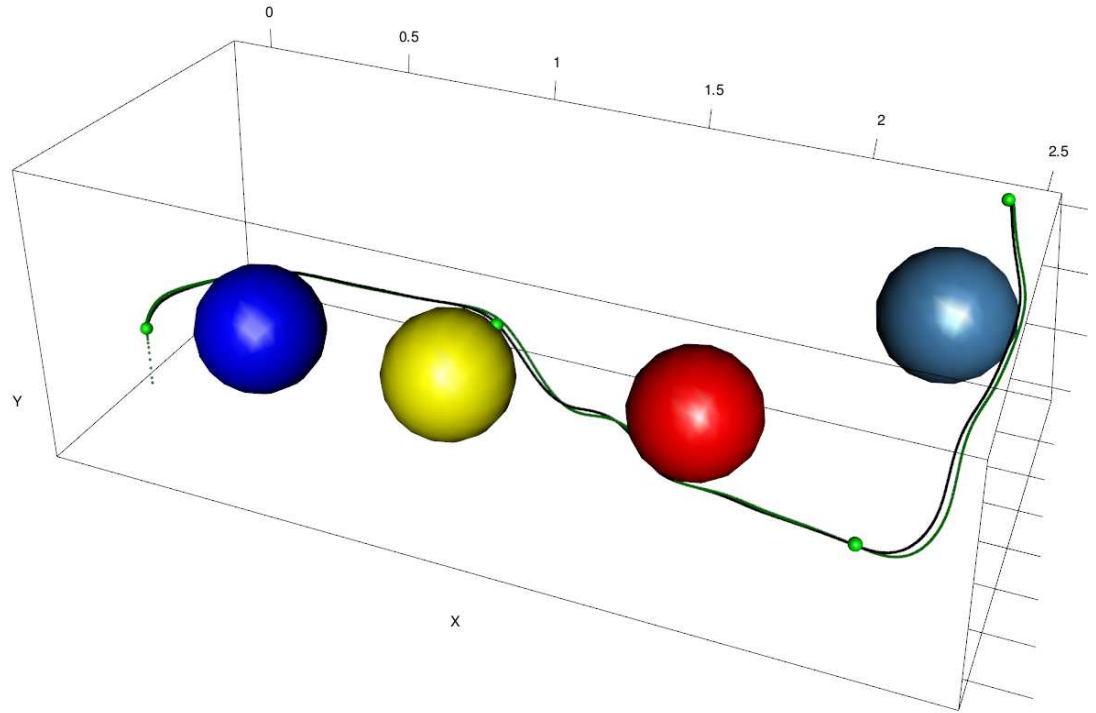
FIGURE 4.20: Tracking trajectories generated by NTG.

In Figure 4.21, the difference between the initial location of the reference and the start point of the actual UAV is shown more closely. The controller is able to "catch up" with the reference quickly. From the graph the tracking trajectory reaches the initial location of the reference (the green dot close to the viewer) within ten iterations, each iteration is designed to be 0.0056 second. Thus the controller is fast enough to even out the initial difference.
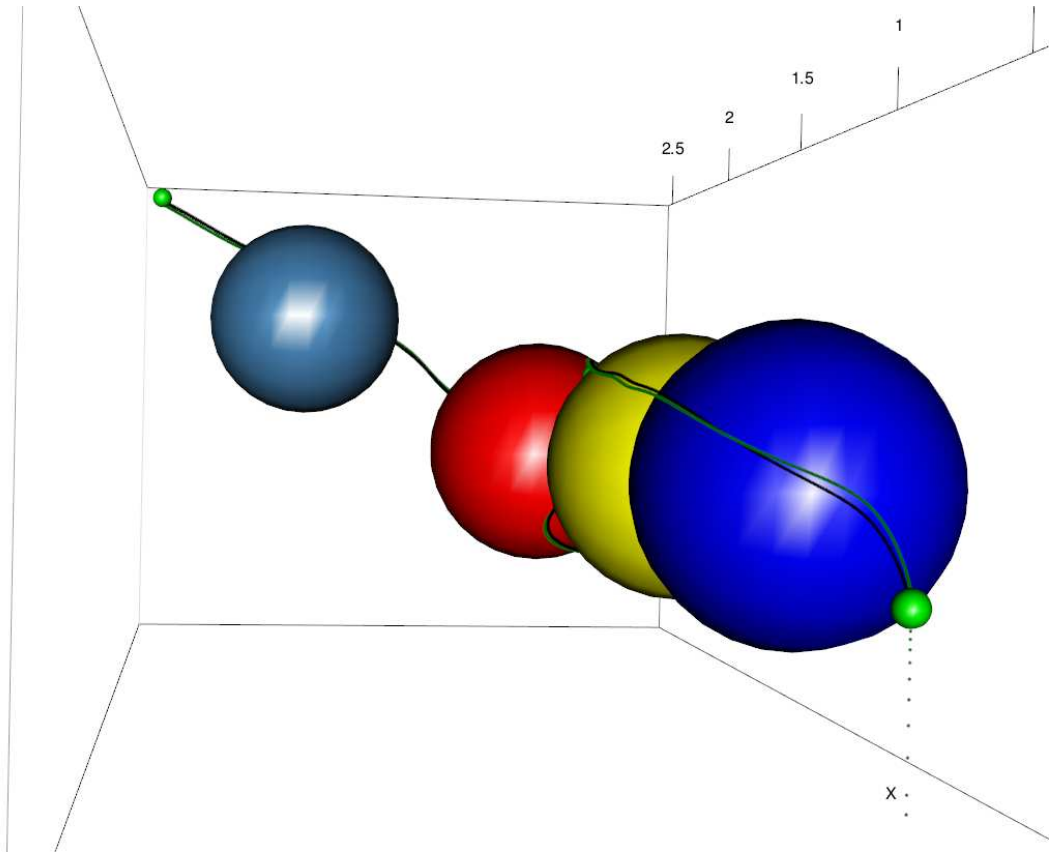
FIGURE 4.21: Trajectory tracking as seen from another perspective.

The top view as Figure 4.22 shows gives yet another perspective of the trajectory tracking.
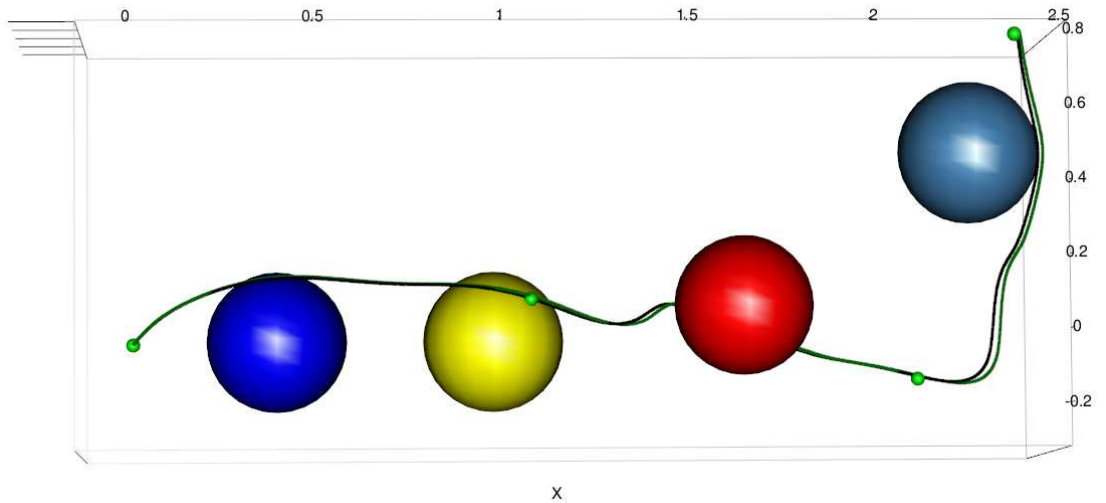
FIGURE 4.22: Trajectory tracking as seen from the top.

A close-up around the first waypoint is shown by Figure 4.23. One can observe that although there exists a steady state error between the two trajectories, they are close enough to pass through the waypoint dot, which is a small sphere with the radius of 2 centimeters.

The reference and tracking trajectories on r_x, r_y and r_z of the above tracking are plotted in Figure 4.24. The running time of this trial is 8.4 seconds. Distance on each axis is measured in meters.
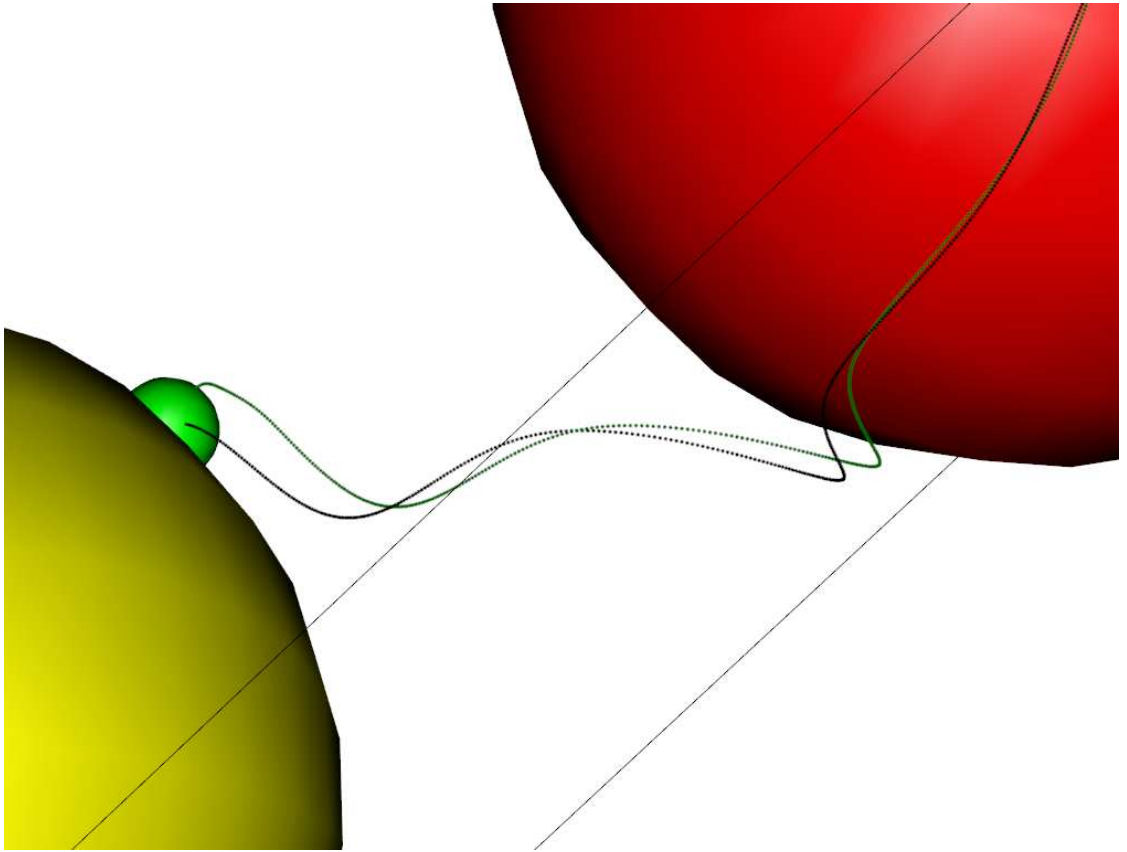
FIGURE 4.23: Close-up at the first waypoint.
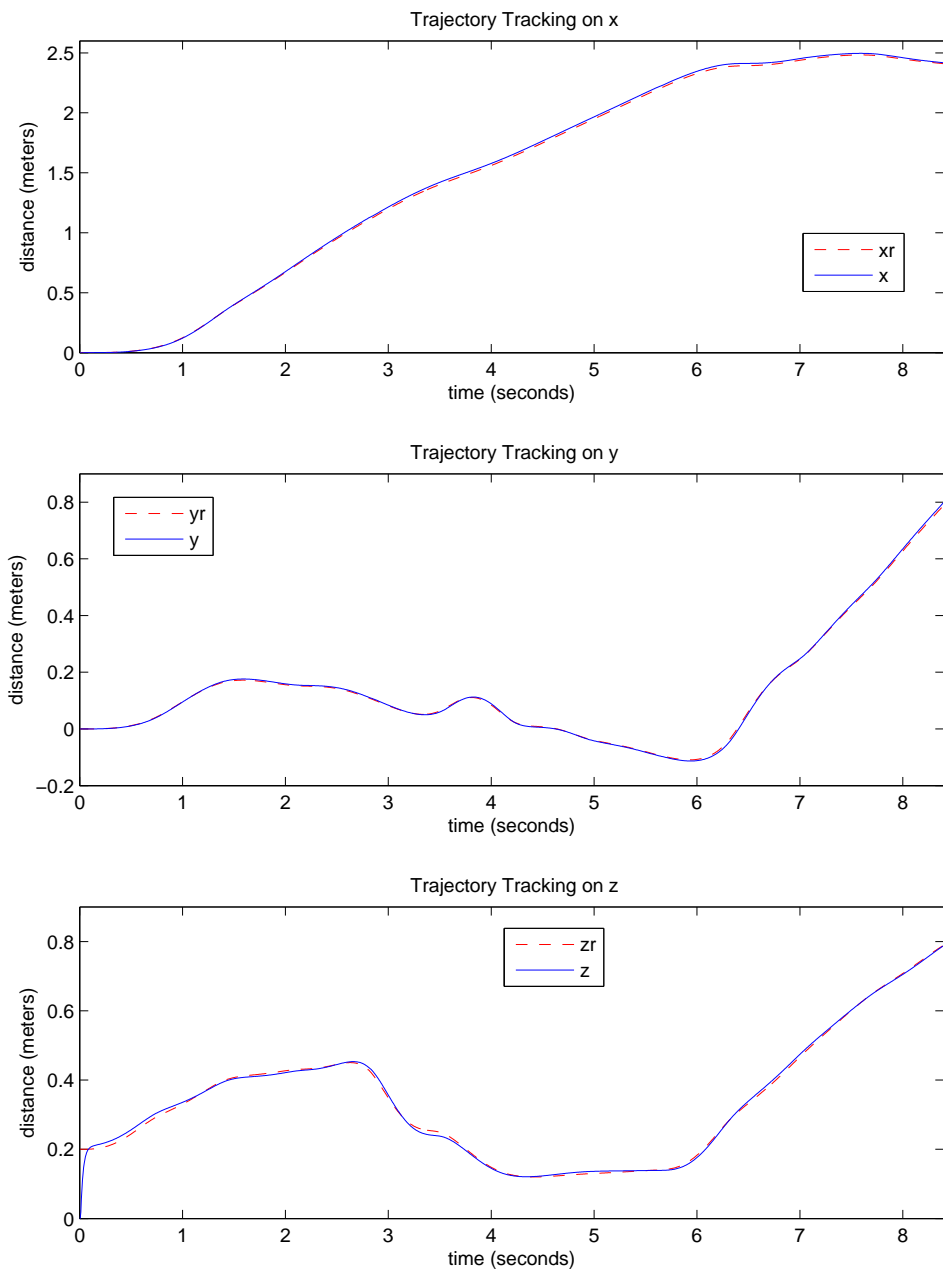
FIGURE 4.24: The reference and tracking trajectories. The tracking time is 8.4 seconds. x_r, y_r and z_r are the reference trajectories, and x, y and z are the tracking trajectories.

The errors during the tracking are plotted in Figure 4.25. The plot is generated

from 1500 sample values for each variable, and the errors are measured in meters. The initial large error of r_z reflects the difference at the initial location mentioned above. The rest of the plot generally limits the error within $\pm$ 2 cm.
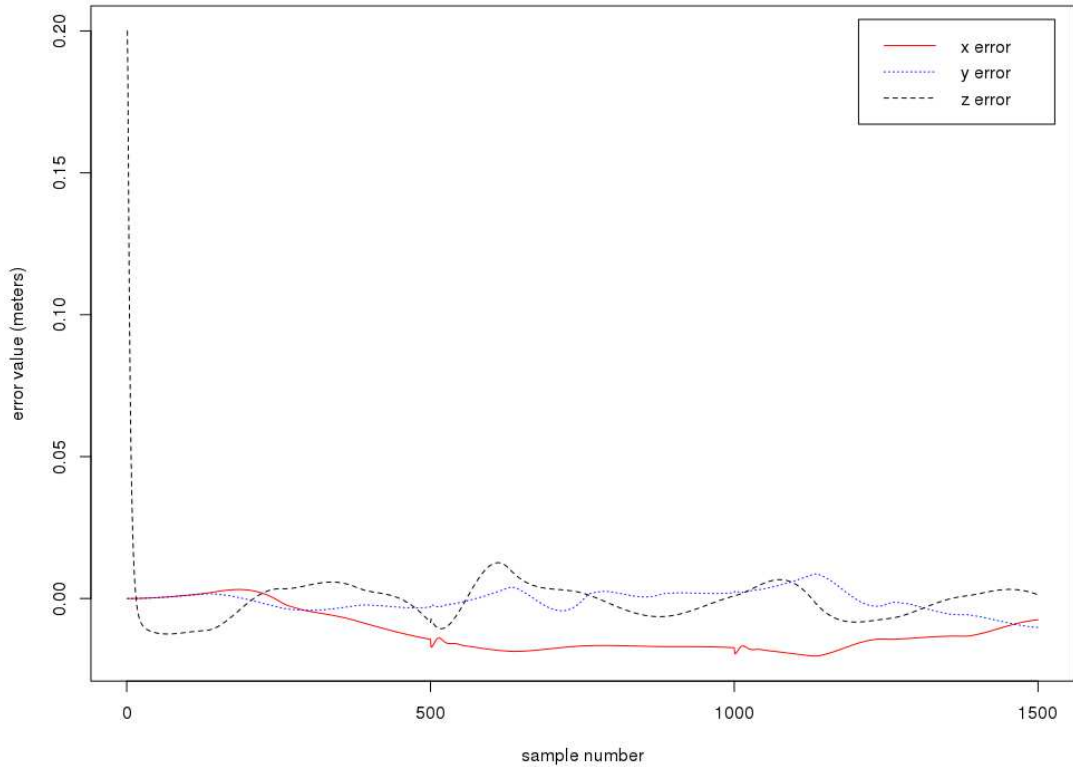
FIGURE 4.25: The errors of r_x, r_y and r_z from their respective references.

4.3   Disturbance Rejection

The previous section proves the controllers are capable of tracking the optimal reference trajectories NTG generated. In this section disturbances will be modeled and add to the system for the purpose of evaluating the controllers' robustness.

Taking the "Mode 5" obstacle avoidance that is being tracked in the previous section for example. Possible disturbance during the tracking might be a sudden gust of

wind at certain locations. The force exerted on the UAV will result in a certain amount of acceleration. The following simulation introduce a sudden acceleration to the UAV along the $-r_y$ direction when the UAV is passing through the area where $1.3 < r_x < 1.6$. Figure 4.26 - Figure 4.28 show how the controller is reacting to different magnitudes of "winds".
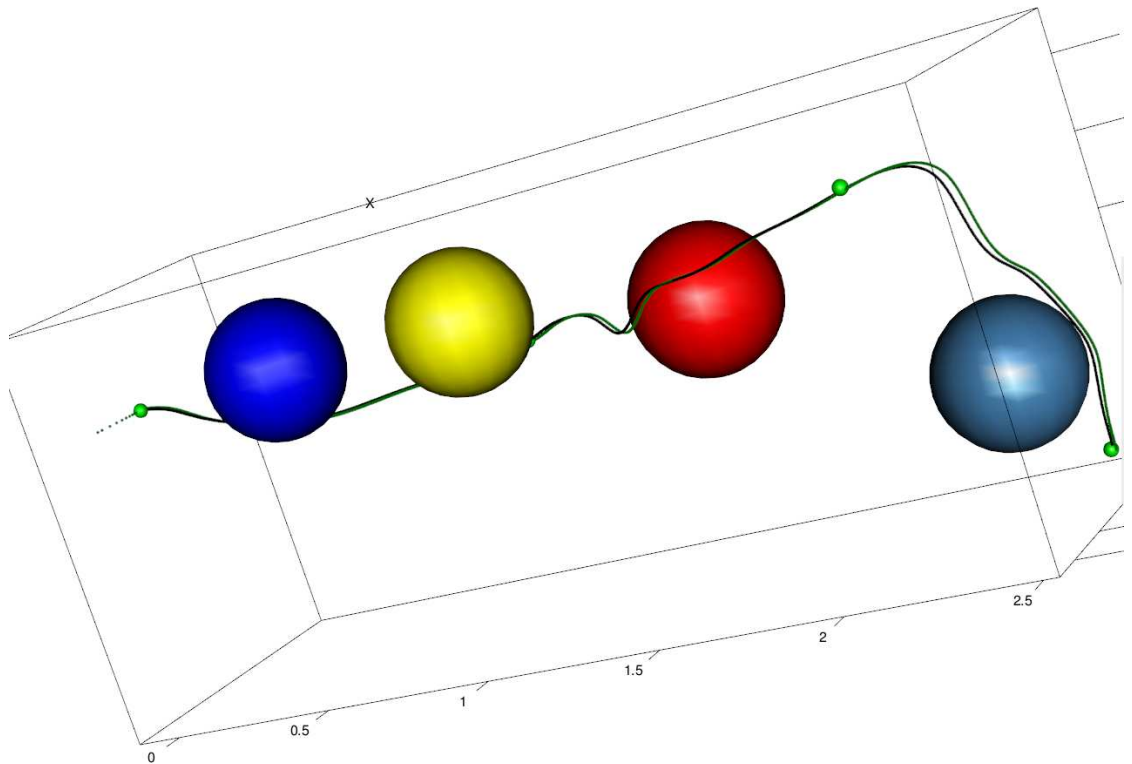
FIGURE 4.26: The original tracking without any disturbance.
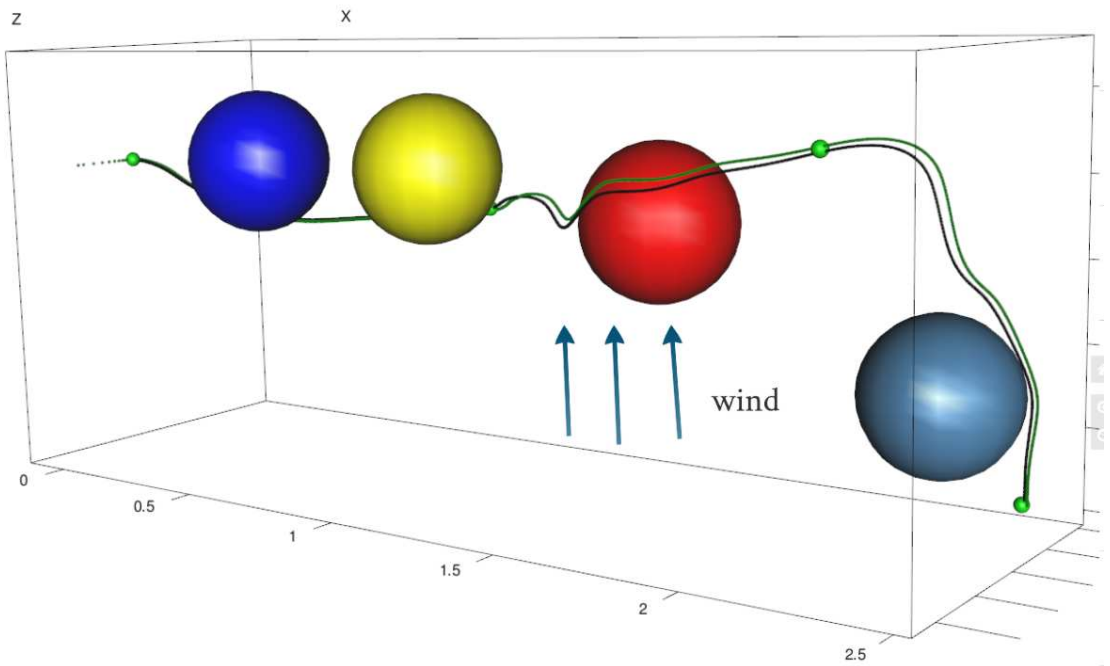
FIGURE 4.27: Small disturbance occurs at $1.3 < r_x < 1.6$. A wind blows from r_y to $-r_y$, exerting a force of 1 Newton on the mass center of the quadrotor. The controller is able to track the trajectory and arrive at waypoint 2 as planned.
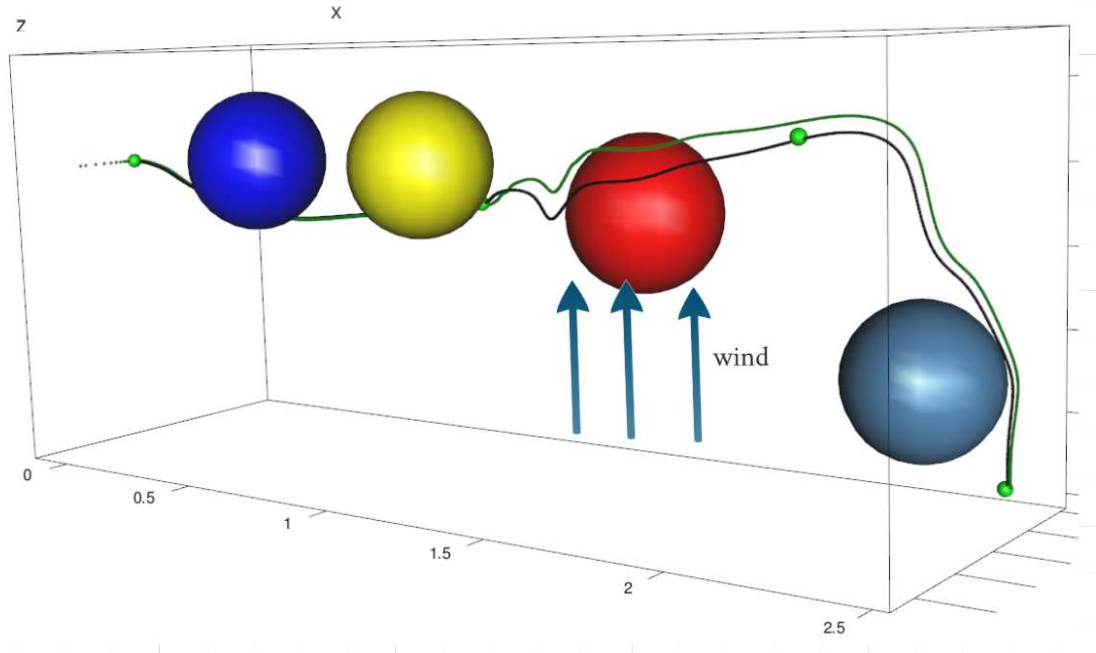
FIGURE 4.28: Large disturbance occurs at $1.3 < r_x < 1.6$. A wind blows from r_y to $-r_y$, exerting a force of 3 Newton on the mass center of the quadrotor. The tracking is seen to drift more to the $-r_y$ direction and away from the second waypoint, but the general shape of the tracking still follows the reference.

4.4 Summary

In this chapter, NTG is introduced to generate the reference trajectories for MARIT quadrotor model for trajectory tracking. First the mathematical foundation of NTG is provided. NTG is a set of software tool to solve optimal control problems by transforming the problems into nonlinear programming problems. B-splines parameterization is the method for transforming, and NPSOL is the tool NTG adopts to solve the transformed nonlinear problems. Secondly the MARIT quadrotor obstacle avoidance problems are interpreted into NTG programs to be solved numerically. The system is shown to be differentially flat so that NTG is able to map the system states and inputs to a lower dimension. The various constraints are categorized and interpreted in terms of

NTG constraints, e.g. linear initial constraints (LIC), linear trajectory constraints (LTC) and nonlinear trajectory constraints (NLTC). Thirdly NTG is examined to generate optimal reference trajectories for various situations, including multiple obstacle avoidance and real-time obstacle avoidance. Then the controllers designed in the previous chapter are constructed to work with NTG to track the reference trajectories. Results show that these two components are able to collaborate with each other to produce close trajectory tracking. Lastly different magnitudes of disturbances are modeled and introduced to the tracking system and results prove the controller is capable of resisting small disturbances like "gentle winds", and resume tracking accuracy after the disturbance disappears.

CHAPTER 5
Future Work

Fully autonomous flights The current experiments are conducted in a tethered method. The quadrotors are attached to a fixed object (ceiling or ground) by strings for safety reasons. Due to the broken parts of the Draganflyer quadrotor the motors are not running smoothly and react to constant valued commands with random accelerations. The motors sometimes start running with max speed without any input, causing injuries and frustration. The current PC/RC connector connects the R/C transmitter to the PC using usb interface, which is treated as a serial device by the computer. Find an R/C transmitter that can be connected directly to the computer can ensure better signal transmission. Upgrading hardwares could solve this problem. New dynamics modeling might be needed after the hardware upgrading, but since the quadrotor dynamics are identical in nature, the remodeling will be easy.

More completed software interface The current software interface on the controller units is written in Qt framework with C++. It provides basic functionalities such as connecting to server, starting test flight, and closing ports. A more advanced interface may include a 3D reconstructed model of the testbed. This could be used for delivering commands graphically, e.g by clicking on a desired position in the 3D model, the user is able to send the SUAV to that position. This could be realized in many ways, such as by employing the VTK toolkit to draw 3D animation in real-time, or the using traditional OpenGL technology in the Qt framework.

Adding ground vehicles MARIT is able to detect any modeled objects' motions

in real-time. This provides the potential of introducing ground vehicles into the testbed. Algorithms could be developed to command the collaboration of air vehicles and ground vehicles.

REFERENCES

[1] "Remote Piloted Aerial Vehicles : An Anthology,"

[2] Source: U.S. Government publication 'The Evolution of the Cruise Missile'.

[3] NASA-JPL Aerobot project, http://www-robotics.jpl.nasa.gov/systems/system.cfm?System=7.

[4] E. Altug, J. Ostrowski, R. Mahony, "Control of a Quadrotor Helicopter Using Visual Feedback", *IEEE International Conference on Robotics and Automation*, Washington DC, 2002.

[5] E. Altug, J. Ostrowski, C.J. Talyor, "Quadrotor Control Using Dual Camera Visual Feedback", *IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.

[6] Y. Cui, T. Inanc, "Multiple Air Robotics Testbed" *IEEE Chinese Control and Decision Conference*, Taiyuan China, 2012.

[7] TEAL Group, "World Unmanned Aerial Vehicle Systems 2011 Edition" http://www.ctie.monash.edu/hargrave

[8] Donald, David, ed. Encyclopedia of World Aircraft (Etobicoke, Ontario: Prospero Books, 1997), p.854, "Standard aircraft"

[9] Fahrney, Delmar S., RADM USN "The Birth of Guided Missiles" United States Naval Institute Proceedings December 1980 pp.5460

[10] Wagner, William: Lightning Bugs, and other Reconnaissance Drones. 1982, published by Armed Forces Journal International in cooperation with Aero Publishers, Inc.

[11] N. Michael, D. Mellinger, Q. Lindsey, V. Kumar, The GRASP Multiple Micro-UAV Testbed, *Robotics & Automation Magazine*, IEEE Volume: 17 , Issue: 3, 56 - 65, 2010.

[12] M. Valenti, B. Bethke, G. Fiore, J. How, and E. Feron, "Indoor Multi-Vehicle Flight Testbed for Fault. Detection, Isolation, and Recovery," *AIAA Guidance, Navigation, Control Conf. Exhibit,* Keystone, CO, Aug. 2006, AIAA-2006-6200.

[13] J.P. How, B. Bethke, A. Frank, D. Dale, J. Vian, "Real-time indoor autonomous vehicle test environment," *Control Systems,* IEEE Volume: 28 , Issue: 2, 51 - 64, 2008.

[14] M. Gerig, "Modeling, guidance, and control of aerobatic maneuvers of an autonomous helicopter," Ph.D. dissertation, ETH Zurich, 2008.

[15] Flying Machine Arena, http://www.idsc.ethz.ch/Research'DAndrea/FMA

[16] Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control, http://hybrid.eecs.berkeley.edu/starmac/

[17] Vanderbilt Embedded Computing Platform for Autonomous Vehicles http://www.vuse.vanderbilt.edu/ kootj/Projects/VECPAV/

[18] Vicon, "Vicon MX Systems," June 2006 [Online]. Available: http://www.vicon.com/products

[19] "Tom's RC", http://www.tti-us.com/rc/sc8000.htm

[20] "Draganfly Innovations Inc.", http://www.draganfly.com

[21] Hitec, http://www.hitecrcd.com/

[22] B.L. Stevens and F.L. Lewis, "Aircraft Control and Simulation," 2nd ed. Hoboken, NJ:Wiley, 2003.

[23] J. How, "Lecture Notes: Aircraft Stability and Control (16.333): Lectures 3 and 4" Sept. 2004 [Online]. Available: http://ocw.mit.edu/OcwWeb/Aeronautics-and-Astronautics/16-333Fall-2..4/LectureNotes/index.htm

[24] D. Gurdan, J. Strumpf, M. Achtelik, K. Doth, G. Hirzinger and D. Rus, "Energy efficient Autonomous Four rotor Flying Robot Controlled at 1KHz," *IEEE International Conference on Robotics and Automation*, Roma Italy, 2007.

[25] S. Bouabdallah, A. Noth and R. Siegwart, "PID vs LQ Control techniques Applied to an Indoor Micro Quadrotor," Autonomous Systems Laboratory Swiss Federal Institute of Technology.

[26] Y. Cui, T. Inanc, "Multiple Air Robotics Testbed" *IEEE Chinese Control and Decision Conference*, Taiyuan China, 2012.

[27] Ian D. Cowling James F., and Alastair K. Cooke. "Optimal Trajectory Planning and LQR Control for a Quadrotor UAV." [Online]. Available: http://ukacc.group.shef.ac.uk/proceedings/control2006/papers/f125.pdf

[28] S. Bouabdallah, Ph.D. thesis, "Design and control of quadrotors with application to autonomous flying", 2007.

[29] Mian, A.A.; Wang Daobo, "Nonlinear Flight Control Strategy for an Underactuated Quadrotor Aerial Robot". IEEE International Conference on Networking, Sensing and Control, 2008. Page(s): 938 - 942.

[30] Madani, T.; Benallegue, A. "Control of a Quadrotor Mini-Helicopter via Full State Backstepping Technique". Decision and Control, 2006 45th IEEE Conference on Robotics and Control Systems, page(s): 1515 - 1520.

[31] G. Hoffmann, S. Waslander, and C. Tomlin, "Quadrotor helicopter trajectory tracking control," *AAIA Guidance, Navigation and Control Conf. and Exhibit,* Honolulu, Hawaii, 2008.

[32] E. Altug, J. Ostrowski, and C. Taylor, "Control of quadrotor helicopter using dual camera visual feedback," *The Int. Journal of Robotics Research,* vol. 24, no. 5, pp. 329-341, May 2005.

[33] R. Murray "Optimization-Based Control", DRAFT v2.1a, February 15, 2010.

[34] M. B. Milam, K. Mushambi, and R. M. Murray. "A New Computational Approach to Real-Time Trajectory Generation for Constrained Mechanical Systems," Conference on Decision and Control, 2000.

[35] T. Inanc, S. C. Shadden, and J. E. Marsden. "Optimal trajectory generation in ocean flows," Proceedings of the American Control Conference, June 8-10 2005, pp. 674 - 679.

[36] T. Inanc, K. Misovec, and R. M. Murray. "Nonlinear trajectory generation for unmanned air vehicles with multiple radars," Proceedings of the 43th IEEE Conference on Decision and Control, Dec. 14 -17 2004, pp. 3817 - 3822.

[37] W. Zhang, and T. Inanc. "Opportunistic 3D Trajectory Generation for the JPL Aerobot with Nonlinear Trajectory Generation Methodology" Proceedings of 11th Int. Conf. Control, Automation, Robotics and Vision, Singapore, 2010.

[38] M. Fliess, J. Levine, P. Martin, and P. Rouchon. "Flatness and defect of non-linear systems: introductory theory and examples," *International Journal of Control*, 61(6):1327-1360, 1995.

[39] M. B. Milam, K. Mushambi, and R. M. Murray. "A New Computational Approach to Real-Time Trajectory Generation for Constrained Mechanical Systems," *Conference on Decision and Control*, 2000.

[40] M. K. Muezzinoglu and T. Inanc. "Trajectory Generation in Guided Spaces using NTG Algorithm and Artificial Neural Networks," *American Control Conference*, 2006.

[41] M. B. Milam, Ph.D. thesis, "Real-Time Optimal Trajectory Generation for Constrained Dynamical Systems", 2003.

[42] "NPSOL", http://www.sbsi-sol-optimize.com

CURRICULUM VITAE

YINAN CUI

Email: yinan.c@gmail.com

Phone: 502-852-0409

Education

Ph.D. in Electrical and Computer Engineering, University of Louisville, 2007-2013.

M.Sc. in Engineering and Management of Information Systems, Royal Institute of Technology (KTH), 2005-2007.

BEng. in Electrical Engineering, Zhejiang University, 2001-2005.

Publications

- **Y. Cui**, and T. Inanc, *Multiple Air Robotics Indoor Testbed*, 24th Chinese Control and Decision Conference (CCDC), 2012, pp 3487-3492.

- **Y. Cui**, and T. Inanc, *Controller Design for Small Air Vehicles: An Overview and Comparison*, The IEEE International Conference on Unmanned Aircraft Systems, 2013.

- **Y. Cui**, and T. Inanc, *SUAV Control with Multiple Air Robotics Indoor Testbed*, submitted to the Journal of Applied Mathematics and Computation, 2013.

- **Y. Cui**, and T. Inanc, *Optimal Trajectory Generation and Tracking for MARIT SUAVs with NTG*, submitted to the Asian Journal of Control, 2013.

Awards

- University Fellowship Stipend, University of Louisville, 2007-2009.

- Teaching Assistant Scholarship, University of Louisville, 2009-2012.

- Doctoral Dissertation Completion Award, University of Louisville, 2012-2013.

Activities

- Visiting student, Aerospace Controls Laboratory (ACL), the Massachusetts Institute of Technology, 2009.4.

- Vice chair of the robotics and controls session, 24th Chinese Control and Decision Conference (CCDC).

- Invited presentation: Y. Cui, *Multiple Air Robotics Indoor Testbed*, 24th Chinese Control and Decision Conference (CCDC), Taiyuan, China 2012.

- Invited presentation: Y. Cui, *Multiple Air Robotics Indoor Testbed Prototype*, Kentucky's National Science Foundation Experimental Program to Stimulate Competitive Research (KYNSFEPSCoR), Galt House, Louisville KY 2010.