DESIGN OF CONTROL SYSTEMS FOR AN AERIAL MANIPULATOR

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY

AHMET TURGUT BAŞARANOĞLU

A MASTER OF SCIENCE THESIS IN
THE DEPARTMENT OF MECHATRONICS ENGINEERING

SEPTEMBER 2019

Approval of the Graduate School of Natural and Applied Sciences, Atilim University.

_____

Prof. Dr. Ali KARA
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. Hulusi Bülent ERTAN
Head of Department

This is to certify that we have read the thesis "Design of Control Systems for an Aerial Manipulator" submitted by Ahmet Turgut Başaranoğlu and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____	_____
Asst. Prof. Dr. Kutluk Bilge ARIKAN Co-Supervisor	Asst.Prof.Dr. Muhammad Umer Khan Supervisor

Examining Committee Members

Asst. Prof. Dr. Hakan Tora
Electrical&Electronics Eng. Dept., Atilim University _____

Asst. Prof. Dr Muhammad Umer Khan
Mechatronics Eng. Dept, Atilim University _____

Asst. Prof. Dr. Kutluk Bilge Arıkan
Mechanical Eng. Dept, TED University _____

Asst. Prof. Dr. İbrahim Baran Uslu
Electrical&Electronics Eng., Atilim University _____

Asst. Prof. Dr. Ali Emre Turgut
Mechanical Eng. Dept, METU _____

Date: 30.09.2019

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Ahmet Turgut Başaranoğlu

Signature:

ABSTRACT

DESIGN OF CONTROL SYSTEMS FOR AN AERIAL MANIPULATOR

Başaranoğlu, Ahmet Turgut

Supervisor: Asst.Prof.Dr. Muhammad Umer Khan

Co-Supervisor: Asst.Prof.Dr. Kutluk Bilge Arıkan

September 2019, 127 Pages

Within the scope of this thesis, attitude and position control systems are designed and discussed for an aerial manipulator which is composed of a quadcopter and a single degree of freedom robotic arm. The arm is working in the pitch plane of the aerial manipulator. Several control schemes exist to control its movements, but, most of them have been tested only under simulation scenarios. The comprehensive system dynamics (i.e. the coupled quadcopter and manipulator dynamics) is controlled by the centralized control architecture. In addition, the decentralized control approach is applied by designing separate control systems for the quadcopter and the arm. The use of the arm for the purposes of disturbance rejection, regulation, agility enhancement, and the steering are discussed in the scenarios. The Active Disturbance Rejection Control (ADRC), Tracking type Linear Quadratic Regulator (LQR), and the Cascaded PID control algorithms are designed. The selected control architecture is applied on an aerial manipulator that is working indoor for the selected scenario. Ultra-wideband localization system is used to measure the position and altitude of the system. The designed control system is implemented in real-time using Raspberry Pi 3 B +, Phyton script and Matlab/Simulink. A test bench is used to tune the parameters of the attitude controller. Then, basic flight tests are utilized to tune the altitude and position controllers. The simulations and tests show that the manipulator assists the aerial manipulator to reject the disturbances, control the attitude dynamics, and to steer the system.

Keywords: Aerial Manipulator, Quadcopter, ADRC, Tracking LQR, Cascaded PID, Manipulator, Naze32, Aerial Robots, Raspberry Pi, Indoor Localization, Position Control

ÖZ

HAVA MANİPÜLATÖRÜ İÇİN FARKLI KONTROL SİSTEMLERİNİN TASARIMI

Başaranoğlu, Ahmet Turgut

Tez Yöneticisi: Dr. Öğr. Üyesi Muhammad Umer Khan

Ortak Tez Yöneticisi: Dr. Öğr. Üyesi Kutluk Bilge Arıkan

Eylül 2019, 127 Sayfa

Bu tez kapsamında, dört rotorlu bir multikopter ve tek serbestlik dereceli robot koldan oluşan bir uçan manipülatör için yönelim ve pozisyon denetimcileri tasarlanmış ve tartışılmıştır. Robotik kol uçan platformun yunuslama düzleminde çalışmaktadır. Seçilen senaryolara yönelik çeşitli denetim algoritmaları tasarlanmıştır. Bütüncül sistem dinamiğine (multikopter ve manipülatörün etkileşimli doğrusal olmayan modeli) yönelik merkezi denetimciler tasarlanmıştır. Bununla birlikte, platform ve manipülatörü ayrı ayrı denetleyen dağıtılmış kontrol sistemleri de tasarlanmıştır. Robotik kolun uçan sisteme etki eden bozucu girdinin bertaraf edilmesine, sistemin yönelim ve pozisyon kontrolüne yönelik kullanımı senaryolar çerçevesinde çalışılmıştır. ADRC, T-LQR ve ardışık PID denetim algoritmaları tasarlanmıştır. Seçilen denetim yapısı ve senaryo iç mekanda çalışan bir uçan manipülatör üzerinde uygulanmıştır. Ultra geniş bant konumlandırma sistemi pozisyon ve yükseklik ölçümü için kullanılmıştır. Raspberry Pi 3 B +, Naze 32 donanımı ile Phyton kodu ve Matlab/Simulink yazılımı kullanarak gerçek zamanlı testler gerçekleştirilmiştir. Yönelik denetimcisi parametrelerinin test düzeneği üzerinde ince ayarları gerçekleştirilmiştir. Temel uçuş testleri ile yönelim ve pozisyon kontrolcü parametreleri düzenlenmiştir. Benzetimler ve testler ile robotik manipülatörün bozucu girdileri bertaraf etmeye, yönelim ve pozisyon denetimini sağlamaya yönelik kullanımı gösterilmiştir.

Anahtar Kelimeler: Uçan Manipülatör, Kuadkopter, ADRC, Takipçi LQR, Ardışık PID, Naze 32, Raspberry Pi, İç Mekan Konumlama, Yönelim Denetimi, Pozisyon Denetimi

To My Sister, Mother, Father and Cousins

ACKNOWLEDGMENT

I express my sincere gratitude to my co-supervisor Asst. Prof. Dr. Kutluk Bilge Arıkan, for his support and guidance throughout the study. I would also like to thank my supervisor Asst.Prof.Dr. Muhammad Umer Khan for his understanding and support in the process. Also, I am grateful to the jury members for their advices. I would like to thank Anıl Sel and Orçun Altınuç for supporting me during the physical tests. Finally, I would like to thank my family patiently waiting and supporting me throughout during this time.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

F_1,F_2 - Motor Thrusts

m_1 - Mass of Quadcopter

m_2 - Mass of Manipulator

I_1 - Moment of Inertia of Quadcopter

I_2 - Moment of Inertia of Manipulator

L - Length between Center of gravity and Propeller Units

L_b - Lenght of Manipulator

h - Length between Center of Gravity of Quadcopter and Joint of Manipulator

T - Torque of Manipulator's Joint

T_{dis} - Disturbance Moment of the System

ϕ - Roll Angle

α - Manipulator's Angle

F_x, F_y - Reaction Forces on Joint of Manipulator

g - Gravity

$\ddot{x}_c, \ddot{y}_c$ - Accelerations of Quadcopter on Center of Gravity of Quadcopter

$\ddot{x}_B, \ddot{y}_B$ - Accelerations of Manipulator on Center of Gravity of Manipulator

b_P - One over Moment of Inertia of Quadcopter

LIST OF ABBREVIATIONS

COG - Center of Gravity

UWB – Ultra Wideband

LQR - Linear Quadratic Regulator

ADRC - Active Disturbance Rejection Control

DOF - Degree of Freedom

ESC - Electronic Speed Controller

I/O – Input / Output

PC - Personal Computer

PID - Proportional Integral Derivative

PWM - Pulse Width Modulation

UAV - Unmanned Aerial Vehicle

VTOL - Vertical Take-off and Landing

DC - Direct Current

IMU - Inertial Measurement Unit

RC - Radio Control

CAD - Computer Aided Drawing

LI-PO - Lithium Polymer

CHAPTER 1

1. INTRODUCTION

Flying robots have been evolved promptly from being sensing agents which merely monitor the environment to being flying manipulators during the previous few years. The aerial manipulation includes a flying platform with a manipulator to perform grasping, transporting, picking and placing, pulling/pushing some objects in order to fulfil particular missions. The aerial manipulation is a form of mobile manipulation. However, it has diverse challenges with respect to the regularly focused ground robots [1].

Manipulation with the quadcopters is the most common implementation in the aerial manipulation [1-25]. Several control methods have been applied to the aerial manipulators. Two main approaches are encountered in the modeling and control of the aerial manipulators [2]. In the first approach, the manipulator and the flying platform are modeled as a single dynamical system [3]. The relevant control systems are designed accordingly. This approach reveals a kind of centralized control for flying platform and the manipulator [3]. In the second method, the flying platform is modelled as if it were not equipped with a manipulator. The dynamical coupling due the manipulator motion and the environmental interaction are assumed as an external disturbance. This technique presents a decentralized control approach for the flying platform and the manipulator separately [4, 5]. The former method requires a complex model. On the other hand, it reveals the complete coupled and nonlinear dynamics of the aerial manipulator. This gives the opportunity of designing advanced nonlinear control systems and estimators. The decentralized method provides relatively simple models and control systems. However, this may degrade the real performance.

As the control algorithm, numerous studies show that PID-type controllers have been used in quadcopter control [6, 9]. In a typical decentralized type of control, the adaptive sliding mode controller is used to manipulate the robotic arm of the aerial manipulator [8]. Another decentralized control system uses back stepping control for the flying platform and PID controller for the robotic arm [9]. In another study, bounded quaternion-based feedback is used to reject arm movements. With the controller in use, the system appears to be stable in the hovering mode despite the robot arm movements [10].

The tools of intelligent control are utilized for the aerial manipulation, as well. In [7] and [13], fuzzy logic is employed in the control system. The performance of fuzzy controller is compared with that of a feedback linearized control system in [7]. In [13], the controller is designed by combining Robust Internal-Loop Compensator (RIC) and Fuzzy Model Reference Learning Control (FMRLC). The designed system is able to track the defined trajectory while the arm under the system performs the tasks of holding and releasing.

When the studies are examined, the most obvious common point is to use the single loop [15, 18, 22, 24, 25] or cascaded type [14, 21, 23] PID algorithms to control the systems. Because of its simple and functional structure, this control structure is frequently utilized in the studies where the physical implementations are performed. However, since it is difficult and complex to adjust for multiple input systems, different control methods are also seen in the literature. Adaptive control structures, Lyapunov-based stability analysis, Augmented Positivity-based control, Operational Space control, Cartesian impedance control approaches are used in the literature [15, 16, 20, 23, 25].

Different kinds of manipulators are utilized in aerial manipulation. [4] Presents the coupled dynamics of a hexacopter and a 2-DOF serial robot arm. Linear active disturbance rejection control is utilized while the arm is active. The performance is compared with that of the cascaded type of PID controllers. [5] Uses a delta robot, i.e. a parallel manipulator that is coupled to a quadcopter. The control system is the combination of back stepping type control and an extended state observer. The flying platform with a three-DOF robotic arm is modeled and controlled in study [20]. A Cartesian impedance controller is designed to stabilize system while external forces

are affecting the system as disturbances. The selected control technique can realize dynamical relationship between the platform, manipulator, and external forces. Another three-DOF manipulator is utilized for carrying a payload autonomously by using motion capture system in [21]. Designed controllers are able to reduce the oscillations due to the payload.

For the performance of the aerial manipulation, the disturbance rejection for the combined system is an important issue. Depending on the preferred methodology, the disturbance includes the movement and interaction of the manipulator, external disturbances such as wind gusts, and the parametric uncertainties. Estimation of the disturbances provides the rejection ability and enhancement of the control performance [14, 16, 18, 24].

In this thesis, the nonlinear coupled dynamics of the aerial manipulator with single DOF robotic arm, i.e. the manipulator is modelled in two-dimensional space. The aerial manipulator is composed of a quadcopter and a single degree of freedom robotic arm. The arm maneuvers in the pitch plane of the aerial manipulator. The centralized and decentralized control architectures are implemented on the nonlinear model in the first part of the thesis. The implementation is performed on the several scenarios. It is aimed to present and discuss the alternative control topologies for aerial manipulators. The uses of the manipulator for the attitude regulation, disturbance rejection, and the position control are revealed. The designed control systems are in the form of Linear Quadratic Regulator (LQR), Tracking LQR (T-LQR), Active Disturbance Rejection Control (ADRC), the combination of ADRC and T-LQR algorithms, and the cascaded type of PID. Selected topology with the selected scenarios are implemented on the real system, as well. The decentralized control of the manipulator for attitude regulation and position control by using the cascaded type of PID controller are implemented in real time. The indoor localization and position feedback are achieved by using an Ultra-Wide Band (UWB) Localization system. The control system is run in real time using Raspbery Pi 3B+, Naze 32 Flight Controller and MatLab/Simulink.

CHAPTER 2

2. MATHEMATICAL MODELING

The equations of motion are derived using Newton's equations. The state equations are obtained based on the equations of motion. The dynamical equations and the corresponding model are developed in 2 dimensional space to scale down the modeling and computational effort.

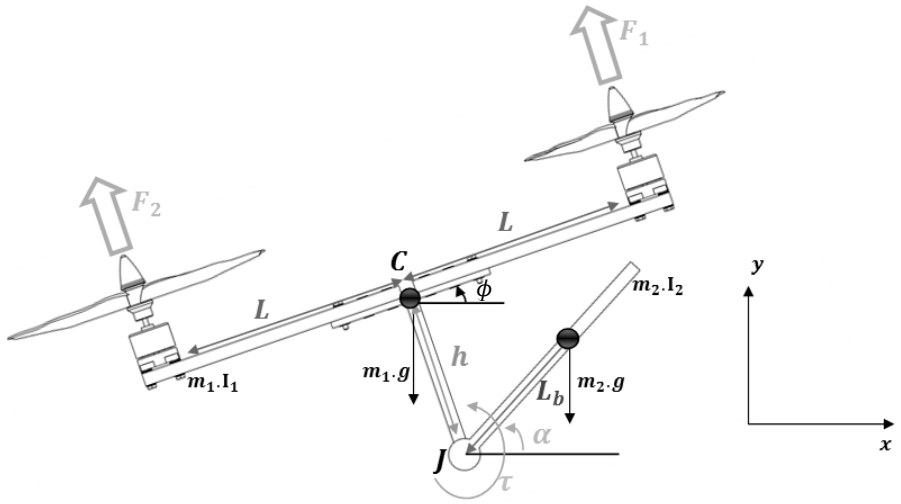


Figure 2.1 Schematic of the System

In the Figure 2.1, F_1 and F_2 are the thrust forces generated by the propeller units, m_1 is mass of quadcopter, m_2 is mass of link, I_1 is moment of inertia of quadcopter, I_2 is moment of inertia of link, L is distance from the rotor axis to the center of gravity of quadcopter(C), L_b is distance between joint (J) and center of gravity of link. h is distance between the center of gravity of quadcopter and joint, J, T is torque

generated by the motor at joint J, ϕ is the roll angle, α is the rotational angle of the link and F_x, F_y are the reaction forces at joint J.

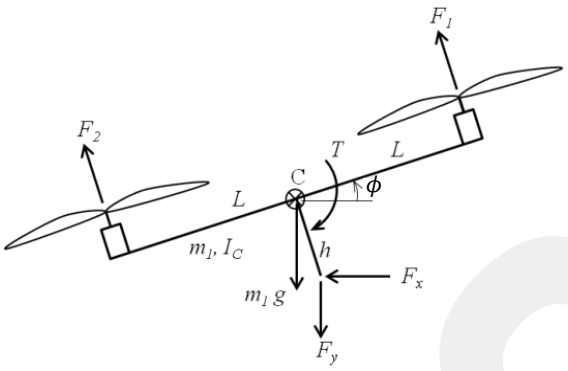


Figure 2.2. Free Body Diagram of the Quadcopter

Using the free body diagrams in Figures 2.2 and 2.3, the following equations of motions are obtained.

$$m_1.\ddot{x}_c = -(F_1 + F_2).\sin(\phi) - F_x \quad (1)$$

$$m_1.\ddot{y}_c = (F_1 + F_2).\cos(\phi) - F_y - m_1.g \quad (2)$$

$$I_1.\ddot{\phi} + T_{dis} = (F_1 - F_2).L - T - F_x.h.\cos(\phi) - F_y.h.\sin(\phi) \quad (3)$$

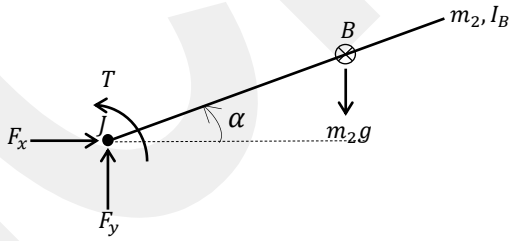


Figure 2.3 Free Body Diagram of Manipulator

$$m_2\ddot{x}_B = F_x \quad (4)$$

$$m_2\ddot{y}_B = F_y - m_2 g \quad (5)$$

$$I_2.\ddot{\alpha} = T + F_x L_b \sin(\alpha) - F_y L_b \cos(\alpha) \quad (6)$$

For determining the non-linear model, equations of $\ddot{x}_c$, $\ddot{y}_c$, $\ddot{\phi}$, $\ddot{\alpha}$ must be determined.

Using the following kinematic Equations acceleration terms are obtained.

$$x_B = x_c + h.\sin(\phi) + L_b\cos(\alpha) \quad (7)$$

$$y_B = y_c - h.\cos(\phi) + L_b\sin(\alpha) \quad (8)$$

$$\frac{d^2x_B}{dt^2} = \frac{d^2\,(x_c + h.\sin(\phi) + L_b\cos(\alpha))}{dt^2}$$

$$\frac{d^2y_B}{dt^2} = \frac{d^2(y_c - h.\cos(\phi) + L_b\sin(\alpha))}{dt^2}$$

Thus,

$$\ddot{x}_B = \ddot{x}_c + h\ddot{\phi}\cos(\phi) - h\dot{\phi}^2\sin(\phi) - L_b\ddot{\alpha}\sin(\alpha) - L_b\dot{\alpha}^2\cos(\alpha) \quad (9)$$

$$\ddot{y}_B = \ddot{y}_c + h\ddot{\phi}\sin(\phi) + h\dot{\theta}^2\cos(\phi) + L_b\ddot{\alpha}\cos(\alpha) - L_b\dot{\alpha}^2\sin(\alpha) \quad (10)$$

The dimensions and working configurations of the 2 frames used for system tests are given. For both frames, the motors were fitted accordingly using the plus configuration. The figure with the configuration is given in Figure 2.4, with the length information of the frames.

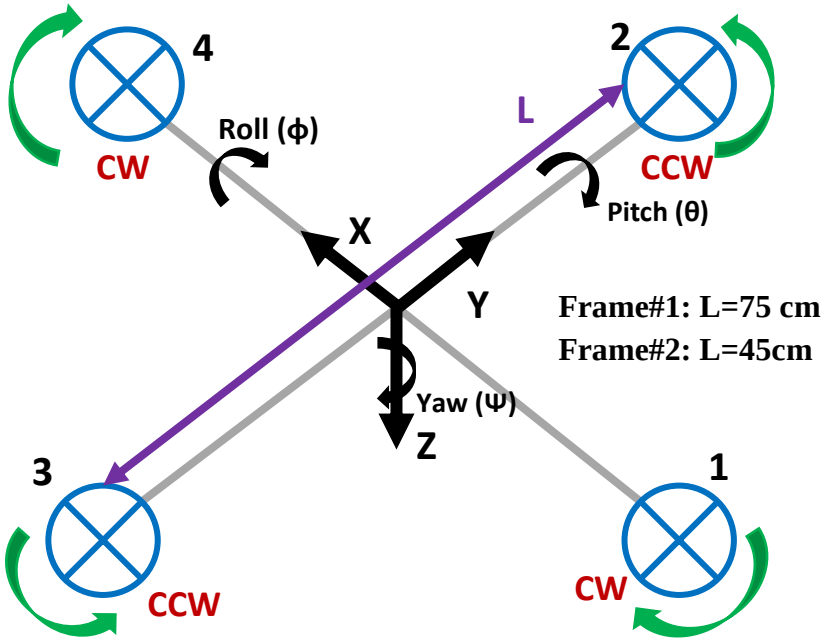


Figure 2.4 Frame Configuration

Attitude of the quadrotor is described using roll, pitch and yaw angles; namely ϕ, θ, and ψ where rotations are about the x, y, and z-axes, respectively. The Matlab m-file code, in which the state-space modeling is performed according to the dynamic and kinematic equations 1-10, is in Appendix A.

CHAPTER 3

3. CONTROL SYSTEM TOPOLOGIES AND SCENARIOS

It is possible to control the aerial manipulator by various control topologies and algorithms. The controller design and applications are simulated in various scenarios.

3.1 Design of a Centralized Control System to Guide the Complete System

In the centralized control topology, there is a single controller for both of the propeller units and the manipulator. In this thesis, Tracking Linear Quadratic Regulator (T-LQR) is designed as the centralized control system. The control schematic of the system is given in the Figure 3.1.

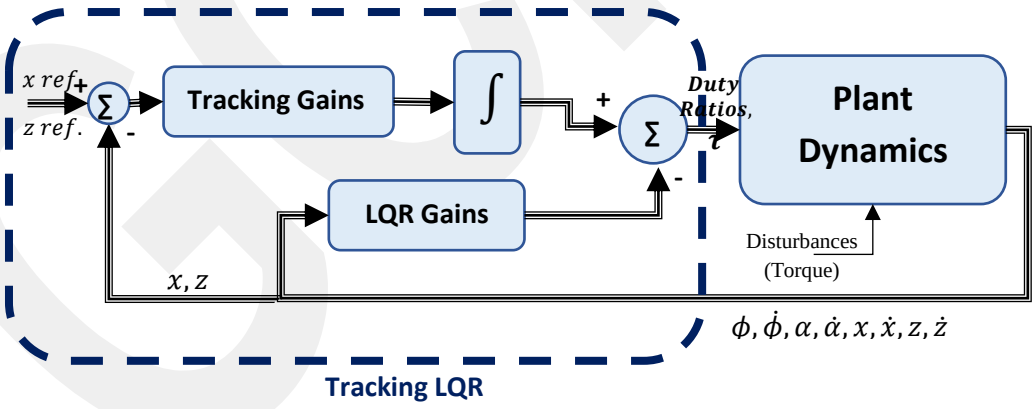


Figure 3.1 Centralized Control System Structure

T-LQR design concerns the movement limits of the manipulator when normalization is done. The Q and R matrices are designed with the most appropriate limits for the states.

To test the performance of this control algorithm, the system is simulated based on different scenarios. The scenarios tackle the following points.

- Performance of both the propeller units and the manipulator can be examined by giving random torque disturbances on the system while keeping the system at fixed point (for position and altitude). In this scenario, the performance of the controller is examined when the system is given random torque disturbances.
- The system has a fixed position and altitude while the manipulator is performing a defined task. In the meantime, the system rejects the internal disturbances (caused by the manipulator's movement, counter torque) and the external disturbances (wind, force or similar disturbances that may affect the system as external factors). The manipulator is only doing given task while the propeller units regulate the system.
- When the system is tracking the given position, the manipulator assists the system in order for the system to correctly track the given reference, while the integrated system rejects the disturbances.
- While the system is tracking a trajectory and changing altitude, the manipulator remains fixed and does not assist the system. After the system has reached the desired position, the manipulator performs the required task (ie. door handle interaction). This task can include keeping the manipulator's angle fixed or swinging. The internal disturbances caused the manipulator during the performance of task are rejected by the propeller units.

The abovementioned concerns are appropriately realmed in two main scenarios. All system simulations are performed by using Matlab/Simulink.

3.1.1 Hover Control under Disturbances for Centralized Control

The system is simulated in hovering condition to evaluate the performance of the controller under different disturbances. At first, the performance is simulated under uniform random disturbances with 0 mean, 1 variance and 0.1 sample time. In addition to random disturbances, the system performance is tested by disturbing it with the moment that is required for turning the door handle. The scenario of the door handle opening test and disturbance rejection while hovering are described in the following illustrations in Figure 3.2.

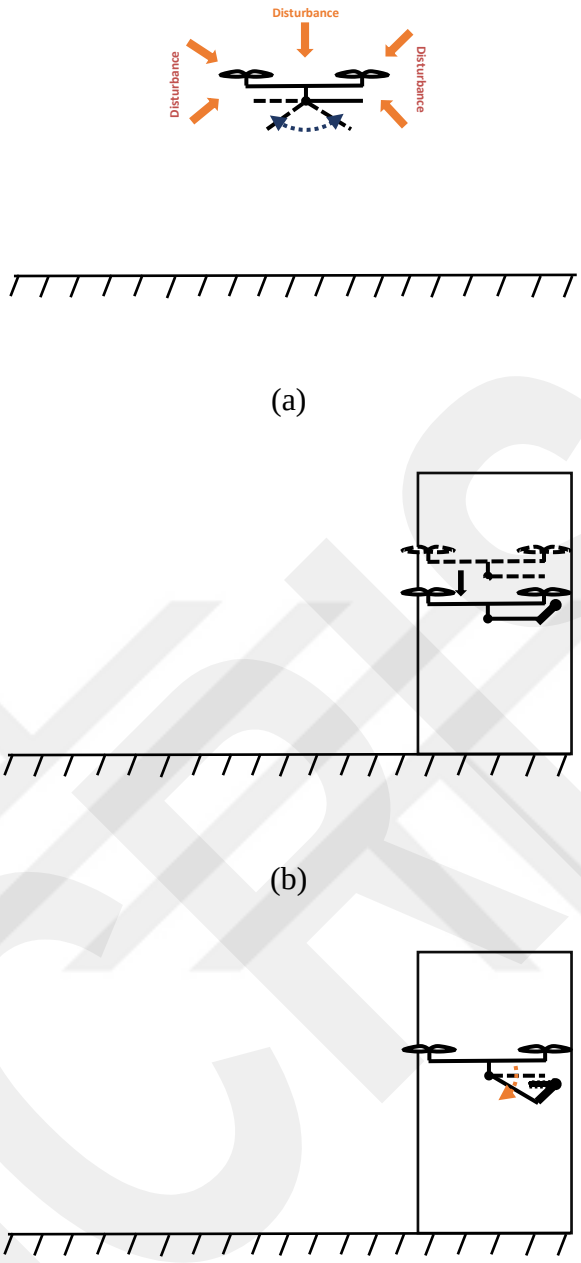
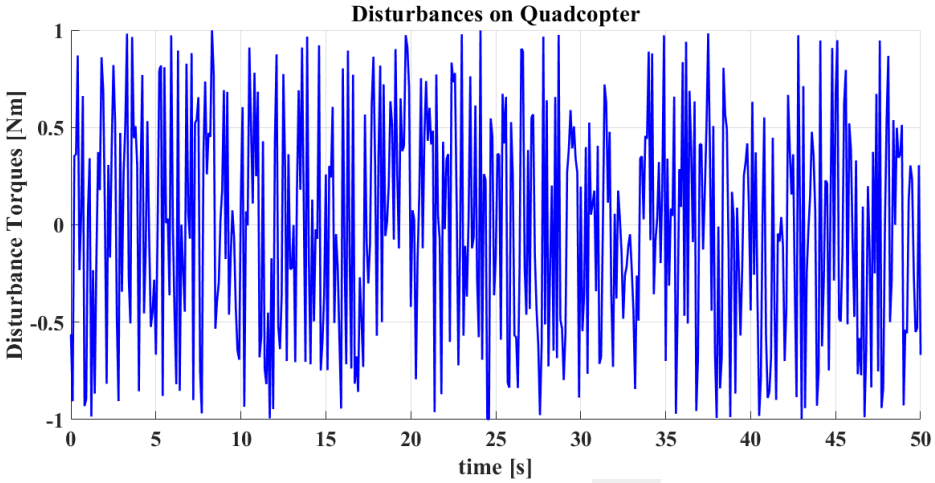


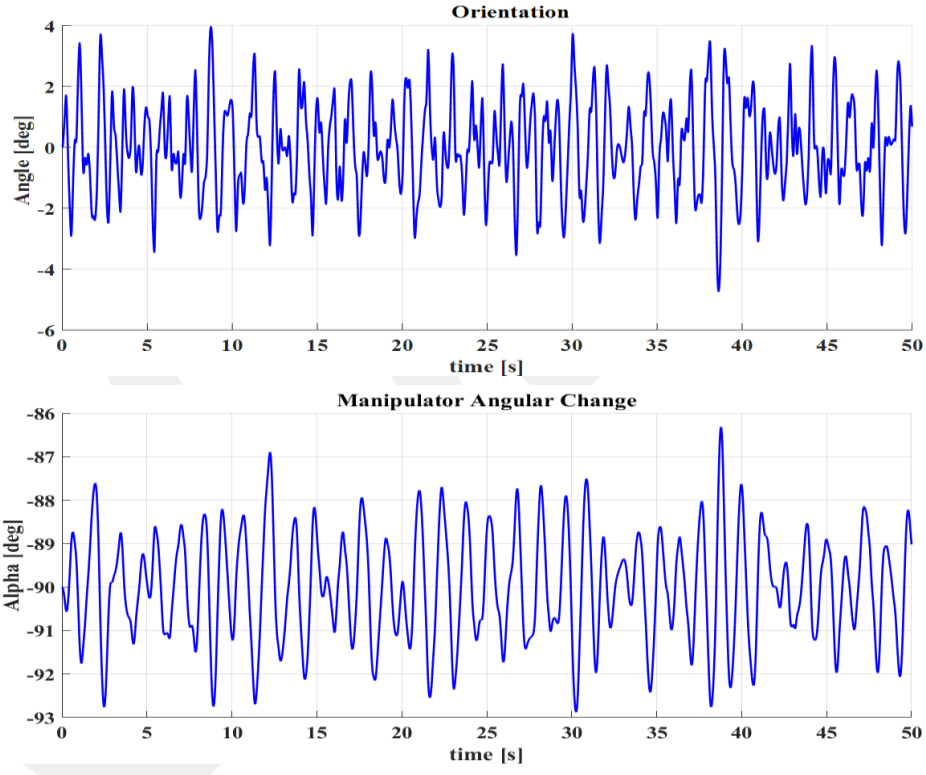
Figure 3.2 (a) Illustrations of Disturbance Rejection while Hovering (b) Door Handle Opening Test Fixed Manipulator, Altitude Change(c) Fixed Altitude, Manipulator Angular Change

The system response due to the random disturbances are given in Figure 3.3.

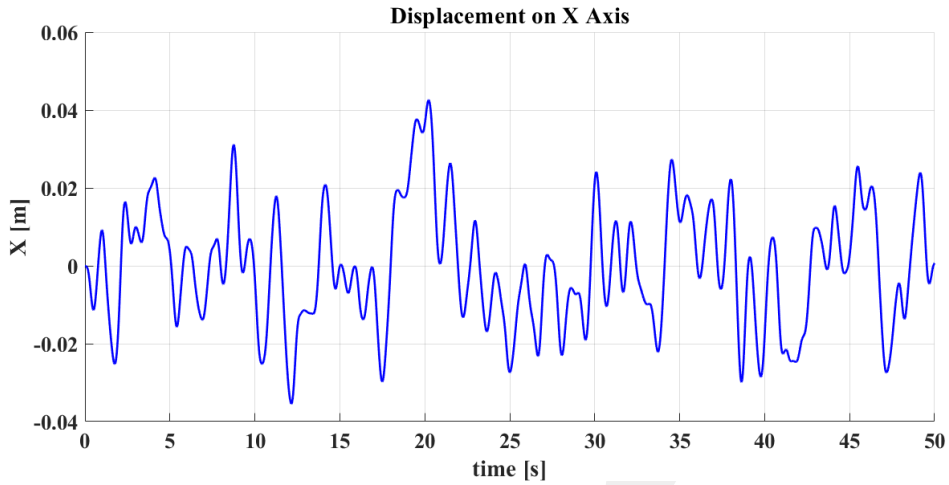

Disturbances on Quadcopter
Disturbance Torques [Nm]
time [s]

(a)

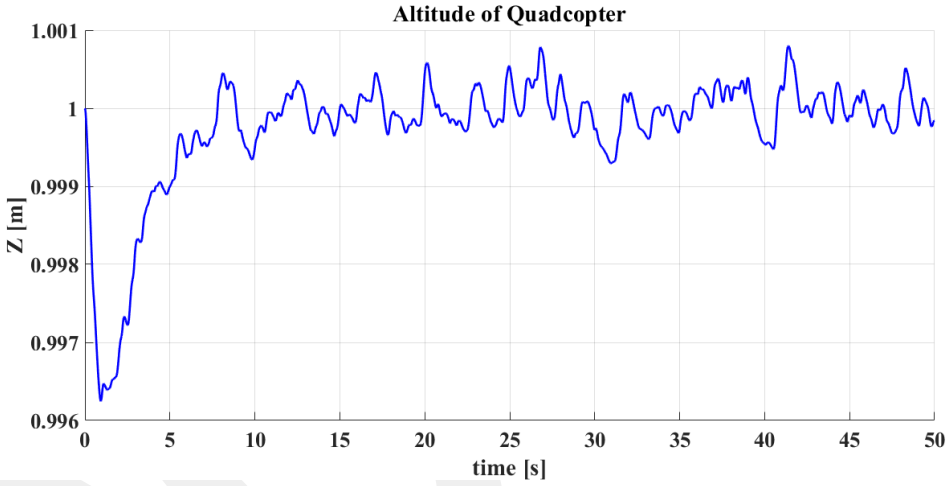
5


Orientation
Angle [deg]
time [s]
Manipulator Angular Change
Alpha [deg]
time [s]

(b)



(c)

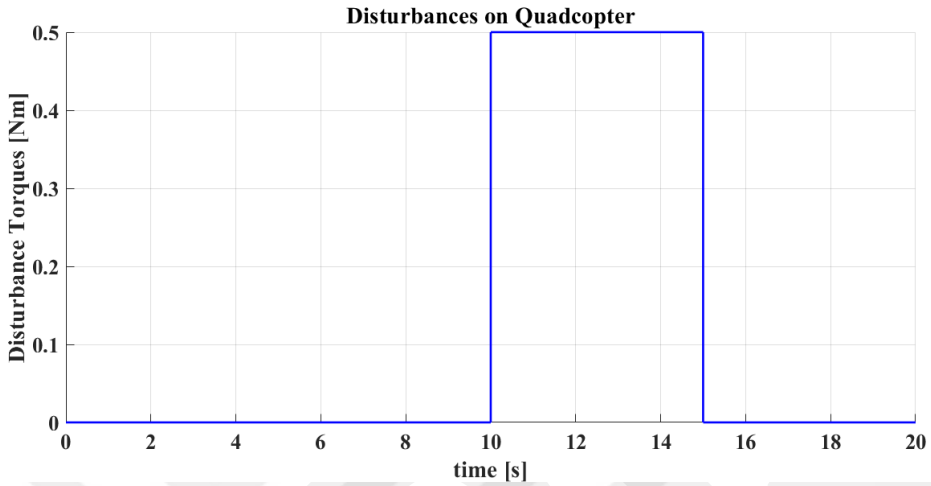


(d)

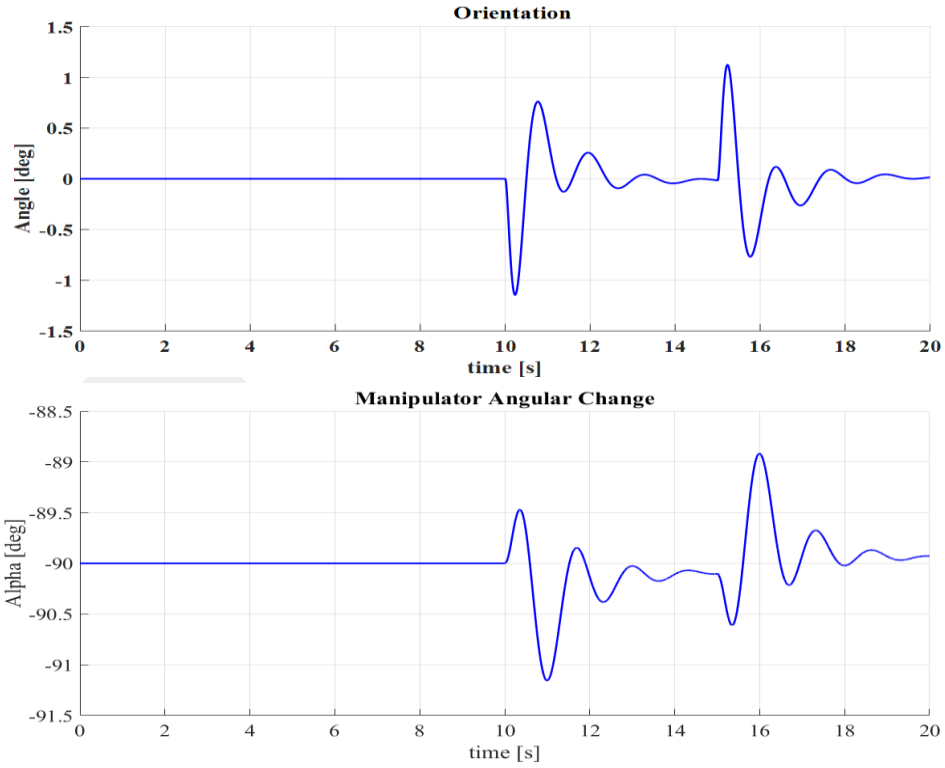
Figure 3.3 System Responses under Random Disturbances (a) Given Random Disturbance (b) α and θ Changes (c) Displacement on x Axis (d) Displacement on y Axis

The results show that, although the centralized control system is designed for more intensive use of the normalized manipulator, it does not provide the required changes in α to stabilize θ. In spite of the necessary disturbances varying between 0 and + -1 Nm, the manipulator produces no more than 0.1 Nm to reject it. However, the results show that T-LQR works well in position control.

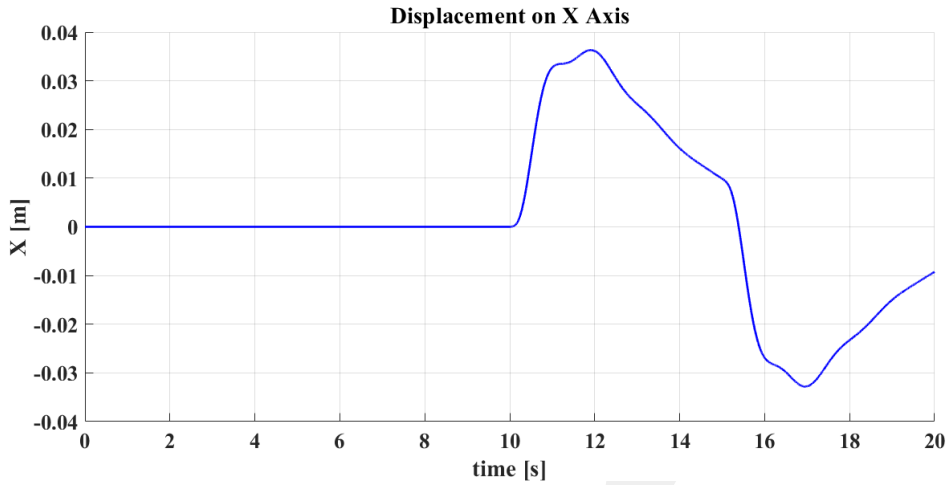
In addition, the system should be simulated and evaluated by performing a door handle test. The required torque disturbance for the door handle is applied to the tip of the manipulator. The results of the system simulated to turn the door handle are given in Figure 3.4



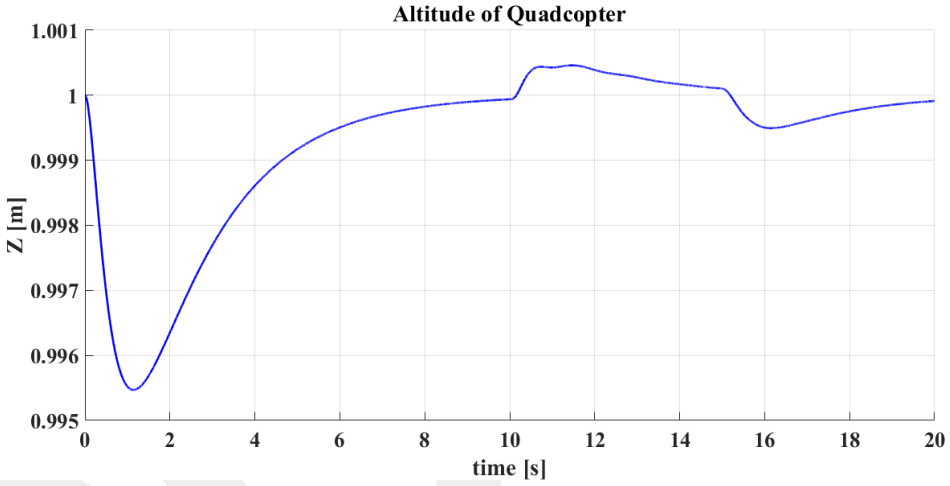
(a)



(b)



(c)



(d)

Figure 3.4 System Responses under Disturbances Calculated for Door Handle (a) Given Random Disturbance (b) α and θ Changes (c) Displacement on x Axis (d) Displacement on y Axis

In these results, it is seen that the system performance is satisfactory by the centralized control.

3.1.2 Position Control at Constant Altitude under Disturbances by the Centralized Controller

When the manipulator is kept fix at a given angle, the displacement of the system and its interaction with the door handle are performed as given in the Figure 3.5.

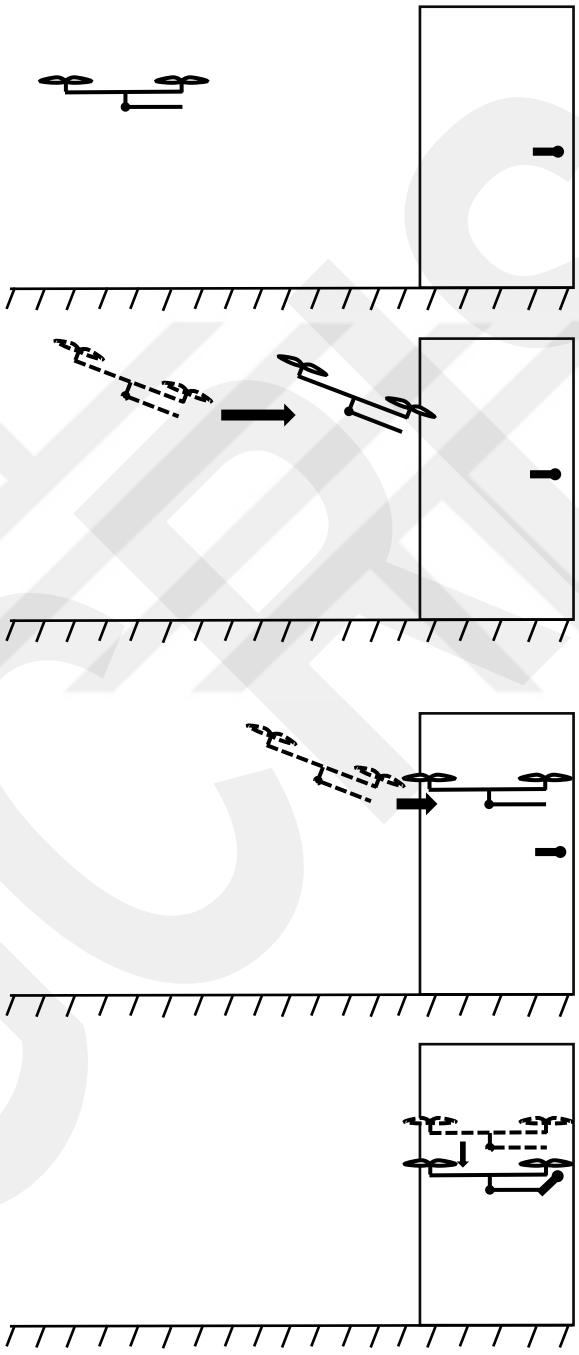
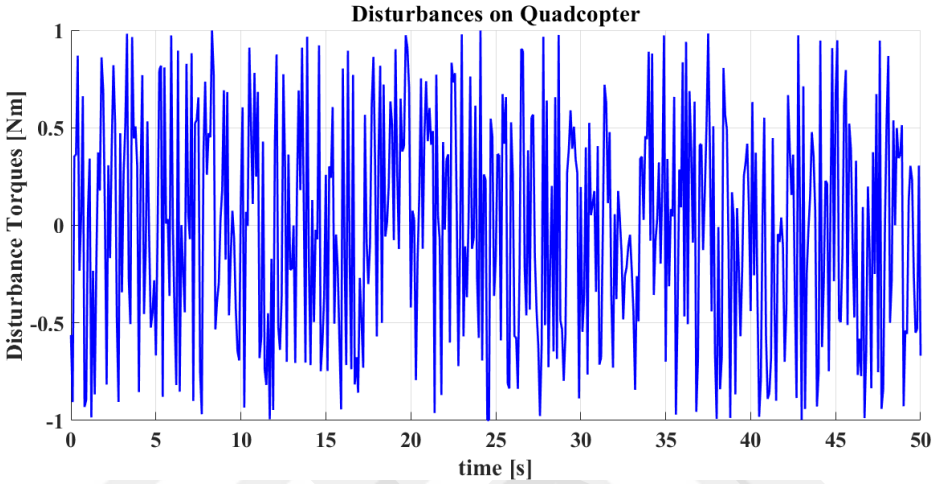
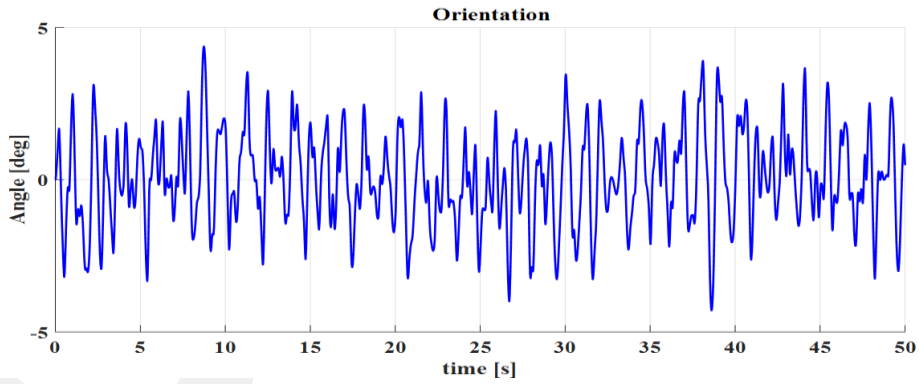


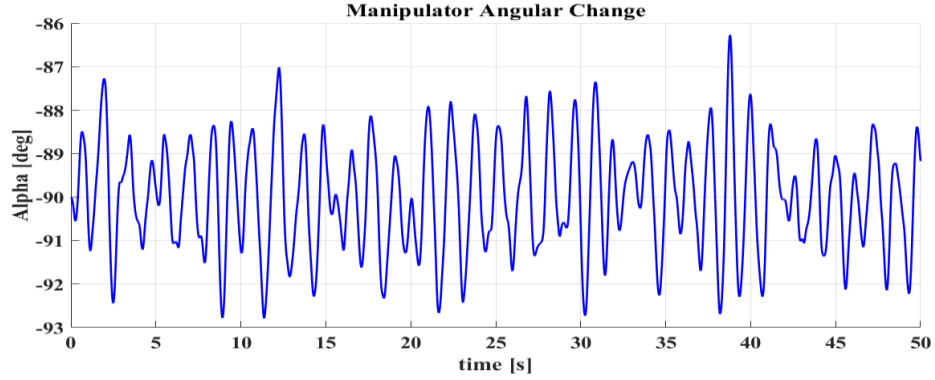
Figure 3.5 Door Handle Interaction Scenario Illustration

In this scenario, the disturbances are applied to the system according to the random disturbances in the same way as the previous simulations but while the system is changing position. The disturbances are applied to the system by using the Matlab Simulink's Random Disturbance, White Noise and Pulse Generator (for the door handle) blocks. The simulations with random disturbances are presented in Figure 3.6.

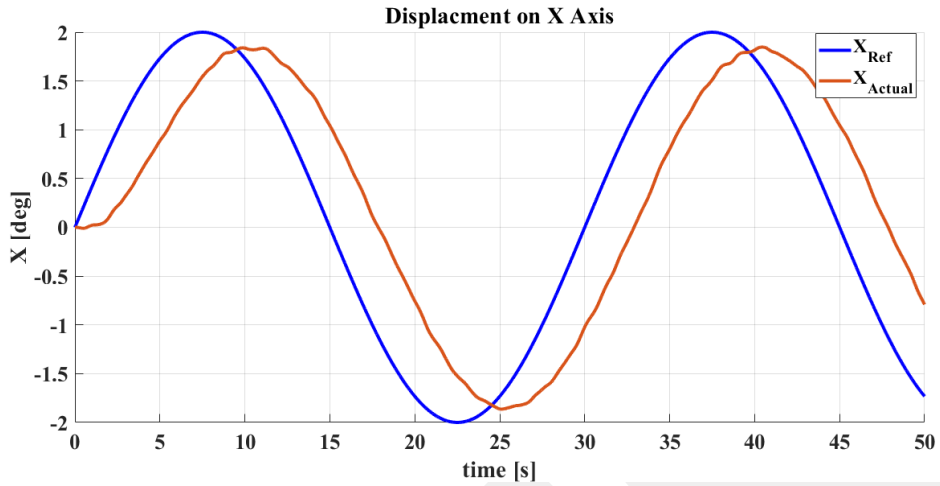


(a)

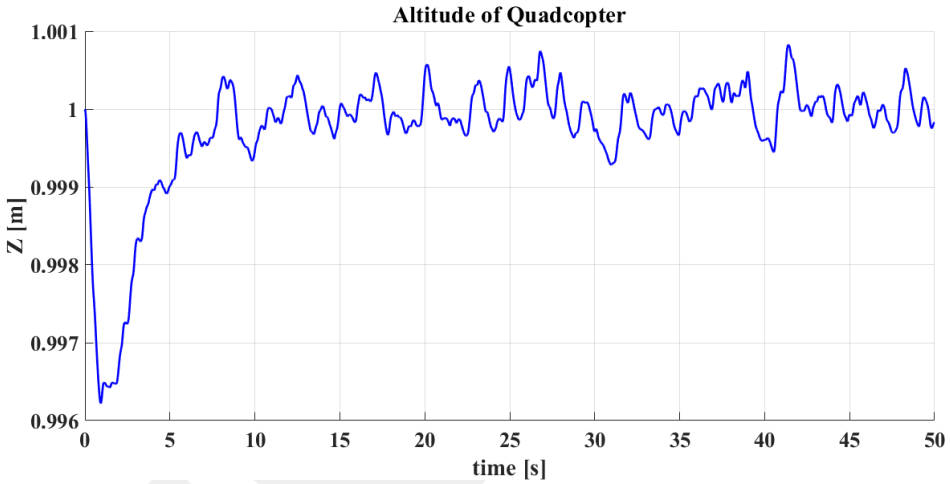




(b)



(c)



(d)

Figure 3.6 System Responses under Random Disturbances during Displacement (a) Given Random Disturbance (b) α and θ Changes (c) Displacement on x Axis (d) Displacement on y Axis

When the results are examined, it is seen that the system follows the position reference while the random disturbances are rejected. In addition, the latency in position follow-up is noticeable. To prevent this latency, the control algorithm needs to be made more stringent. But, if the control algorithm is changed in this way, the

system structure will be made more agile, and the resistance of the system to the external disturbances will be reduced.

3.2 Design of Decentralized Control Systems for the Propulsion Units and the Manipulator

In the decentralized control topology, two different controllers are working separately. The first controller (Propulsion Unit Controller, C_1: manipulates the dc motor and propeller units), can track the given reference values for the quadcopter's position and orientation. In addition, the controller is stabilizing the system at the given hovering references. Cascaded PID type of controller is applied on the hover controller.

The second controller (Manipulator Control, C_2), uses the attitude and attitude rate of the quadcopter and the manipulator to reject disturbances and to stabilize the quadcopter. A combined controller that is designed by using Active Disturbance Rejection Control (ADRC) and Tracing Linear Quadratic Regulator (T-LQR) is used for the Manipulator Controller.

System model consists of 2 different subsystems with 2 different controllers (C_1, C_2). Both controllers are used for disturbance rejection, and angular stabilization. System structure shown in Figure 3.7.

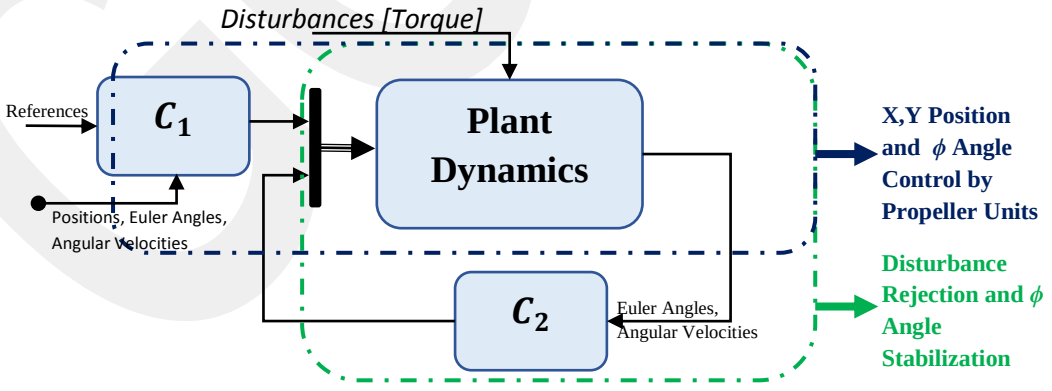


Figure 3.7 System with Decentralized Control Schematic

Obtaining the optimum control gains for the Tracking LQR, normalization is used to calculate Q and R matrixes. α and $\emptyset$ are the states of tracking controller and $\alpha, \dot{\alpha}, \emptyset, \dot{\emptyset}$

are the states for LQR controller. In pursuant of ADRC controller; mathematical design shown below.

$$I_1.\ddot{\phi} = (F_1 + F_2).L - T - F_x.h.\cos(\phi) - F_y.h.\sin(\phi) \tag{11}$$

If $x_1 = \phi$, $x_2 = \dot{\phi}$, $b_P = \frac{1}{I_1}$ and $T = u$;

$$\dot{x}_1 = x_2 \tag{12}$$

$$\dot{x}_2 = b_P \cdot ((F_1 - F_2).L - F_x.h.\cos(\phi) - F_y.h.\sin(\phi) - u) = x_3 - b_P \cdot u \tag{13}$$

$$\dot{x}_3 = h_{dis} \approx 0 \tag{14}$$

ADRC matrix equation of $\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ h_{dis} \end{bmatrix} + \begin{bmatrix} 0 \\ -b_P \\ 0 \end{bmatrix} \cdot u$, can be rewritten as;

$$\begin{bmatrix} \dot{\hat{x}}_1 \\ \dot{\hat{x}}_2 \\ \dot{\hat{x}}_3 \end{bmatrix} = \begin{bmatrix} \hat{x}_2 \\ \hat{x}_3 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -b_P \\ 0 \end{bmatrix} \cdot u + L_P \cdot (x_1 - \hat{x}_1) \tag{15}$$

Where;

$$L_P = [3\omega_0 \quad 3\omega_0^2 \quad \omega_0^3] \tag{16}$$

When we examine the entire system structurally, the disturbances affect as an external moment to the system dynamics. To ensure this, the input is added to the system's equation of motion as follows.

$$I_1.\ddot{\phi} + T_{dis} = (F_1 + F_2).L - T - F_x.h.\cos(\phi) - F_y.h.\sin(\phi) \tag{17}$$

Disturbance effect on system dynamics is shown in Figure 3.8.

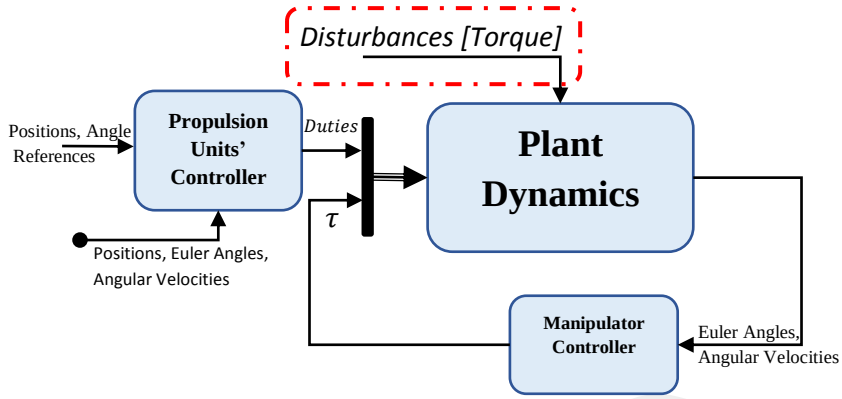


Figure 3.8 Schematic representation of the effect of the disturbance on the system

3.2.1 Hovering Control under Disturbances with Decentralized Control

While system is hovering, the manipulator controller schematic is shown in Figure 3.7. For controller, different types of controllers are applied on the system to stabilize quadcopter by using manipulator. Those controllers are; cascaded PID (3 level), Tracking LQR, Active Disturbance Rejection Control (ADRC), and combined controller (ADRC with T-LQR) for manipulator controller. For propulsion units' controller just 2 level cascaded PID is applied.

According to system responses, combined controller is chosen for system control. For each stabilization and tracking tasks, responses are more satisfactory than other controllers. Combined controller structure for stabilization while hovering is shown in Figure 3.9.

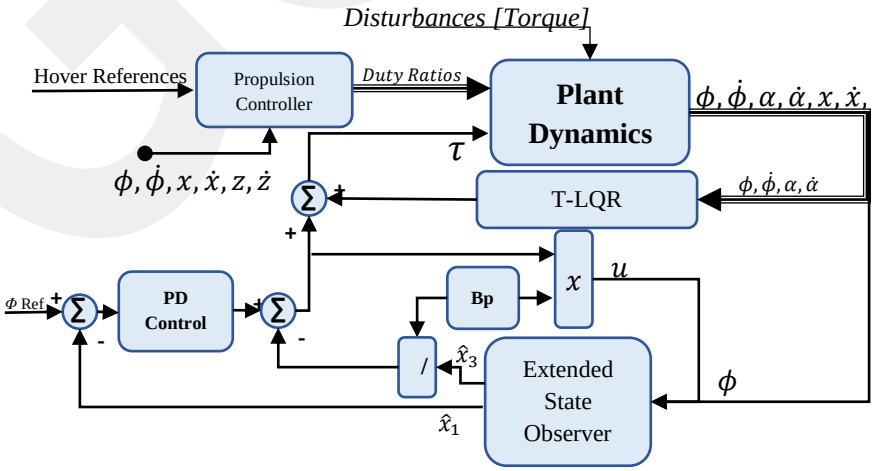
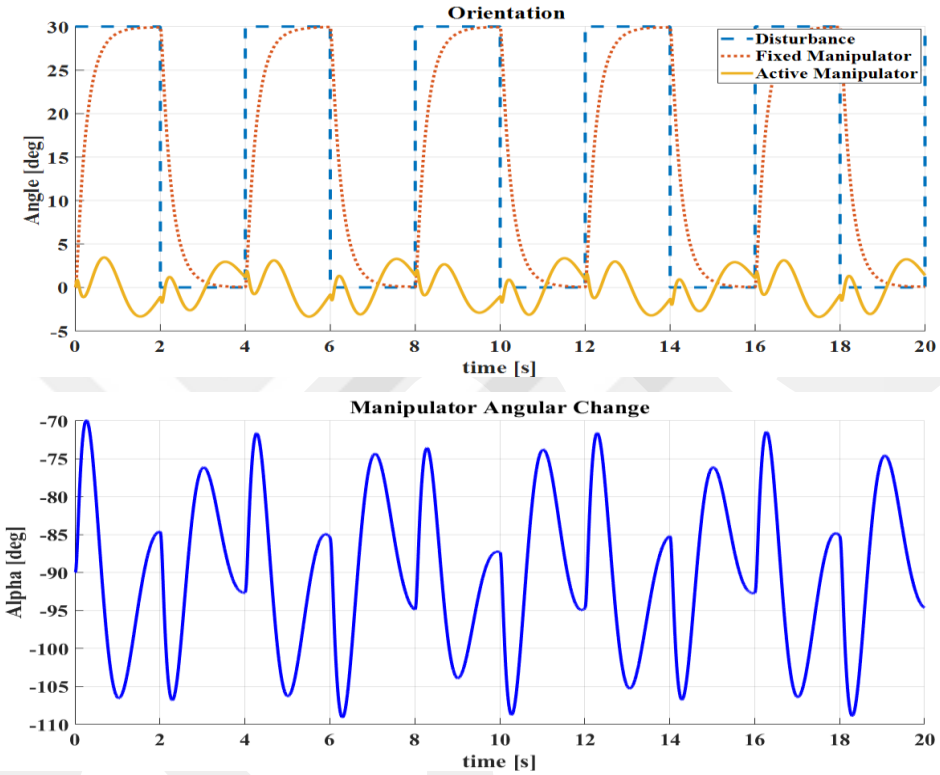
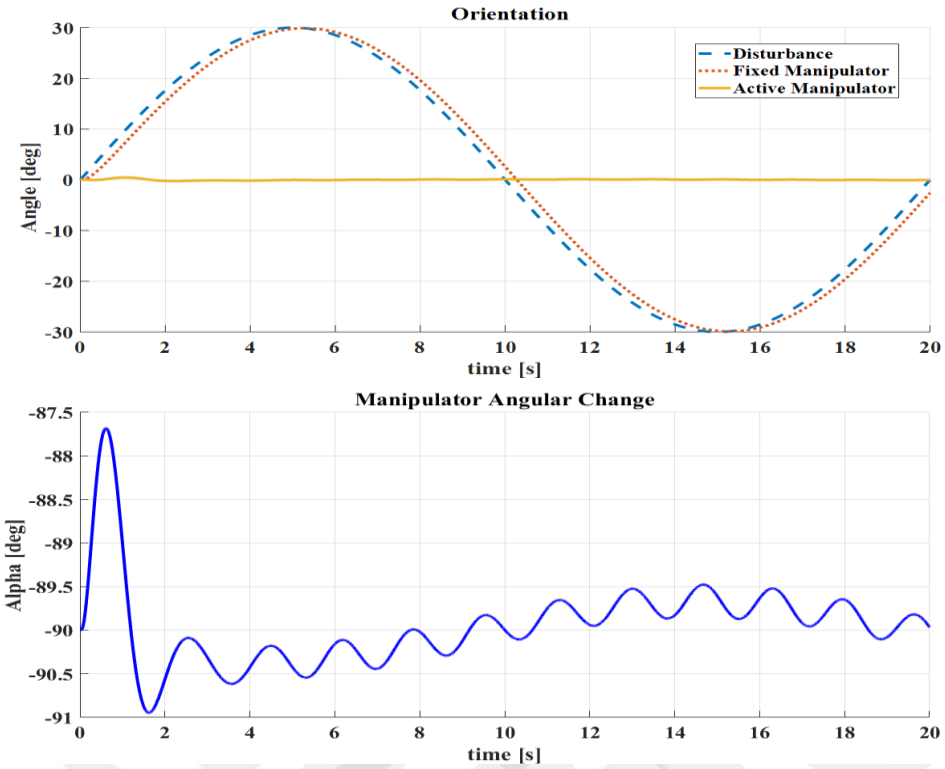


Figure 3.9 Decentralized Controller Structure

In order to analyze the system performance, different types of disturbances are applied to the system dynamics externally. To present the performance of the decentralized control system with the active manipulator, the response of the quadcopter with a fixed manipulator is employed, as well. Furthermore, the torque disturbances given to the system are reflected as the disturbance on the attitude angle in order to simplify the analysis. All results are shown in Figure 3.10.



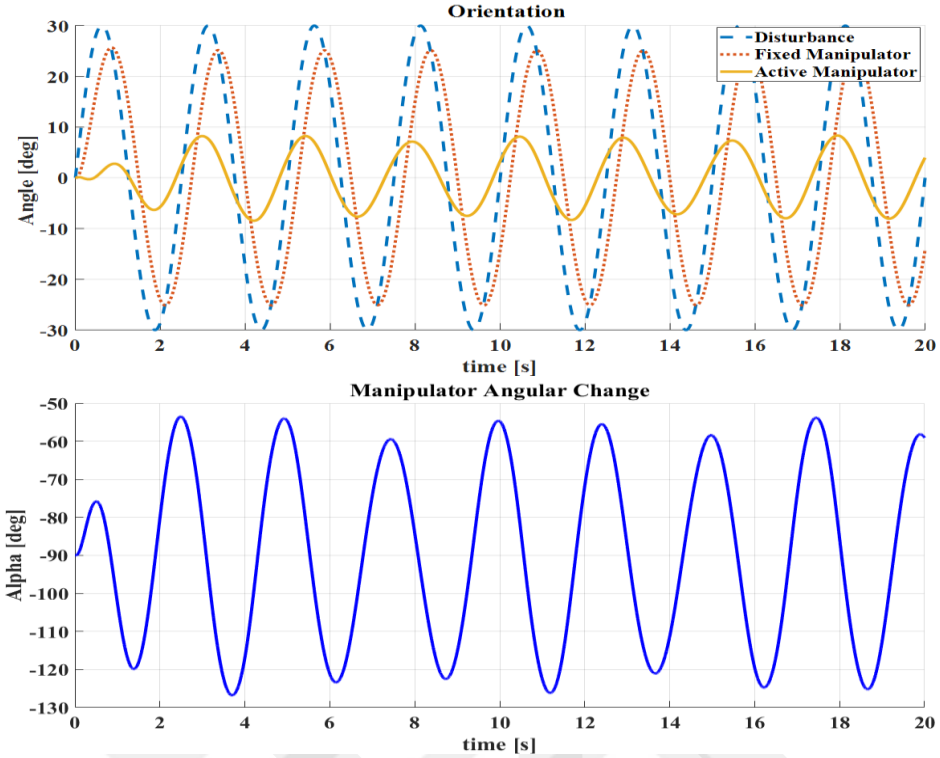
(a)



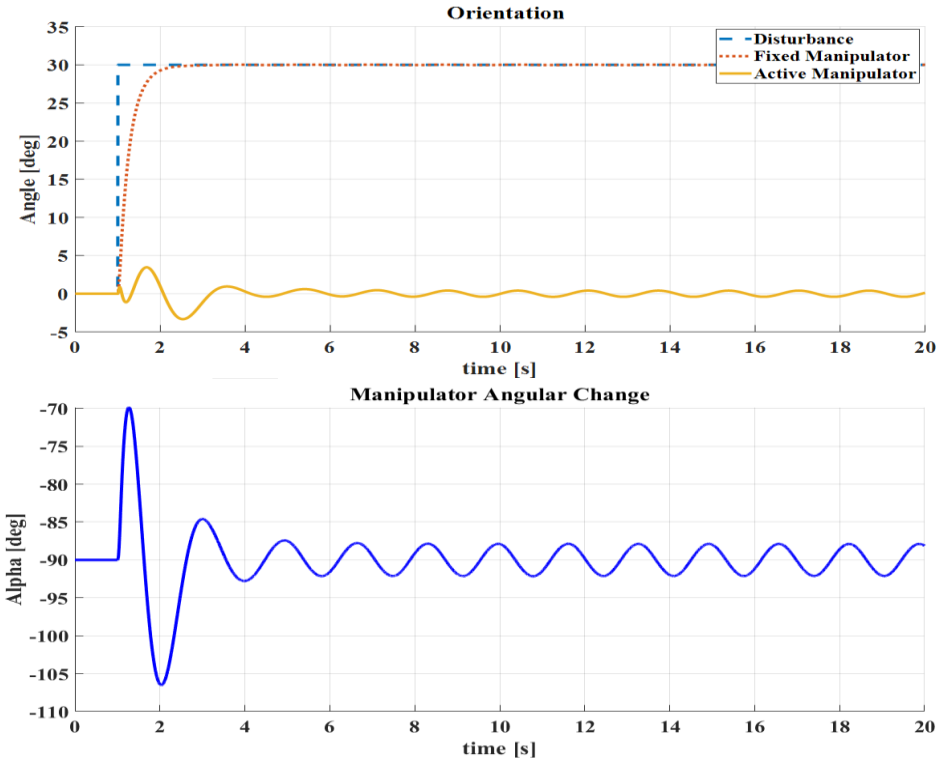
(b)

Figure 3.10 System Responses under Step and Low Frequency Sine Disturbances (a) Pulse Disturbance (b) Low Frequency Sine Disturbance

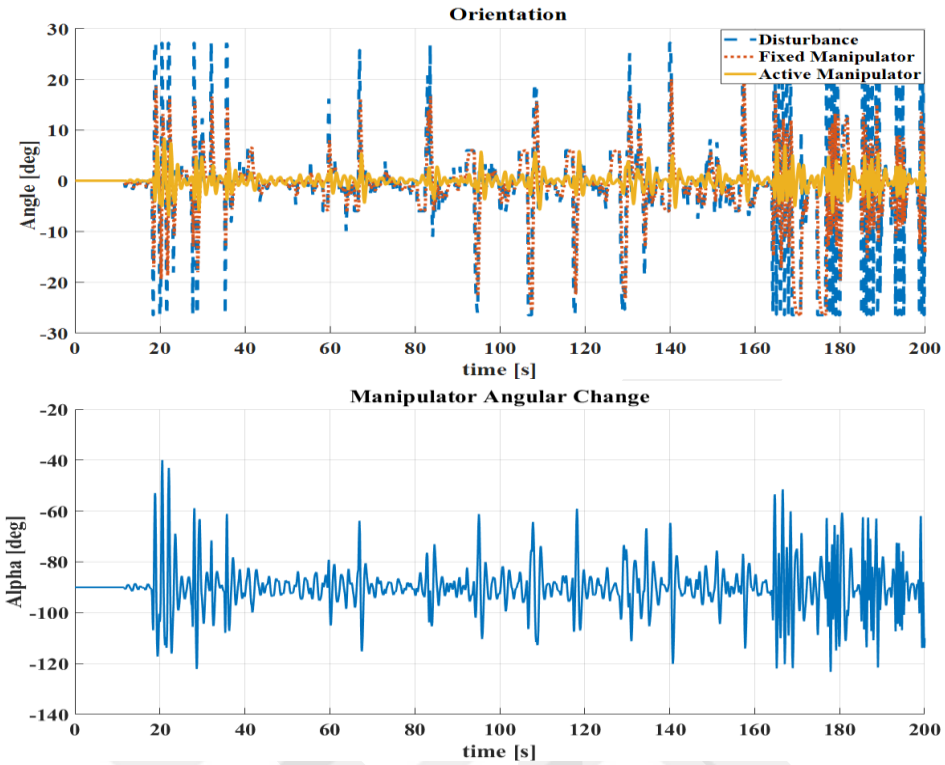
In the graphics given in Figure 3.10, the systems are shown to respond to the disturbances given as the pulse and the 0.5 Hz sinusoidal signals. In addition, the disturbances are applied to the system as 0.8 Hz sinusoidal, step and random signals. The results of these simulations are as follows in Figure 3.11.


Orientation
Disturbance
Fixed Manipulator
Active Manipulator
Angle [deg]
time [s]
Manipulator Angular Change
Alpha [deg]
time [s]

(a)


Orientation
Disturbance
Fixed Manipulator
Active Manipulator
Angle [deg]
time [s]
Manipulator Angular Change
Alpha [deg]
time [s]

(b)



(c)

Figure 3.11 System Responses for disturbances as (a)0.8 Hz Sine Disturbance (b) Step Disturbance (c) Random Disturbances

In particular, the manipulator increases the resistance of the system against the disturbances and satisfies expectations.

3.2.2 Position Control at Constant Altitude under Disturbances with the Decentralized Control

In order to accomplish the navigation of the underactuated system in the X axis, a cascaded type of position control loop is added to manipulate the angle θ and control the position in the determined axis. In this case, it allows us to perform additional position control using the previous attitude control structure. The schematic of the control system is shown in the Figure 3.12.

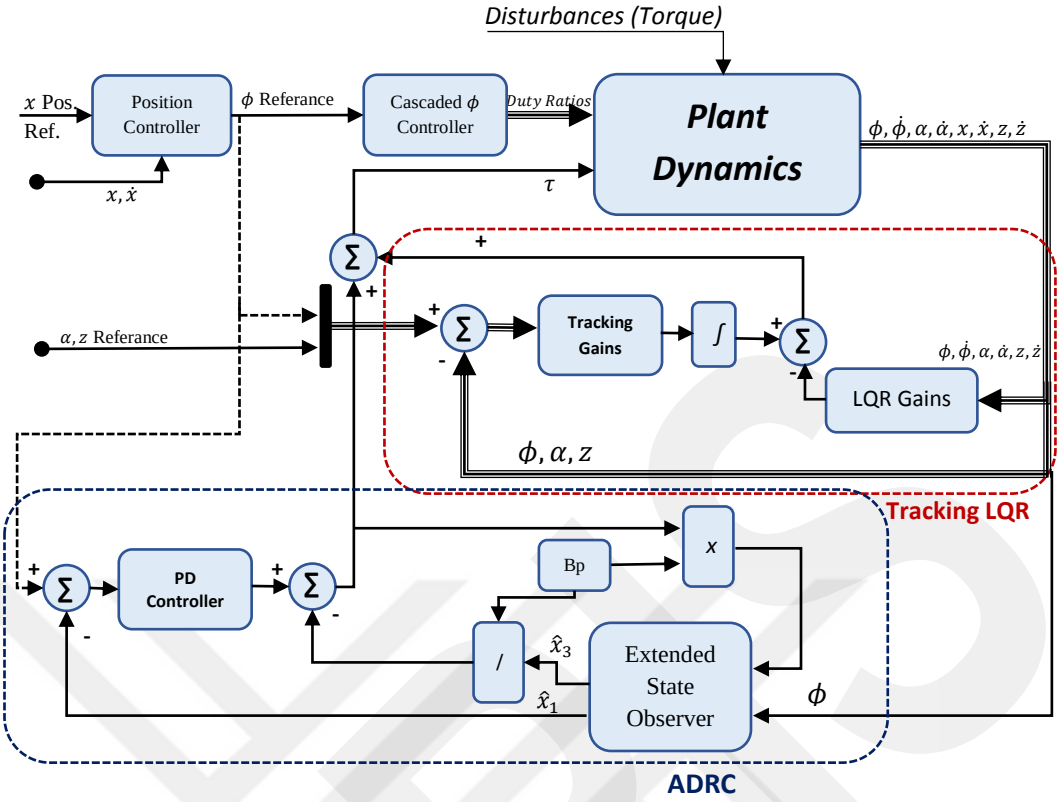
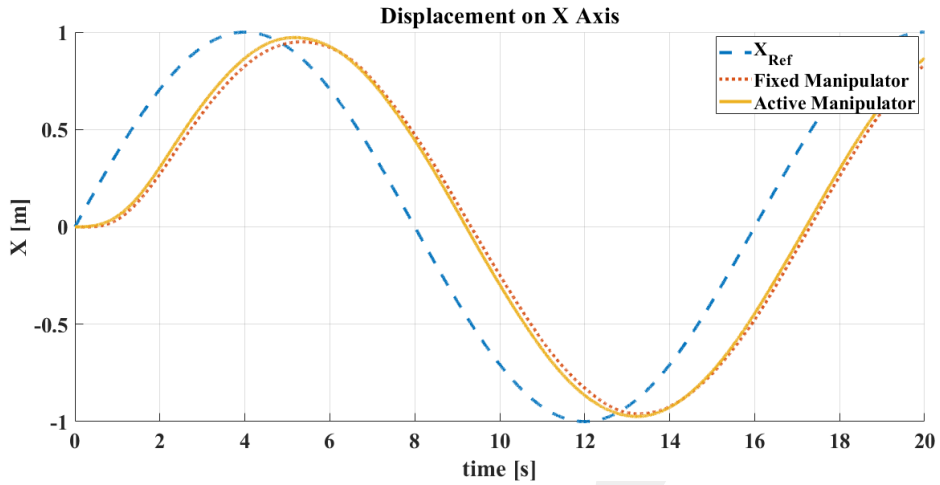


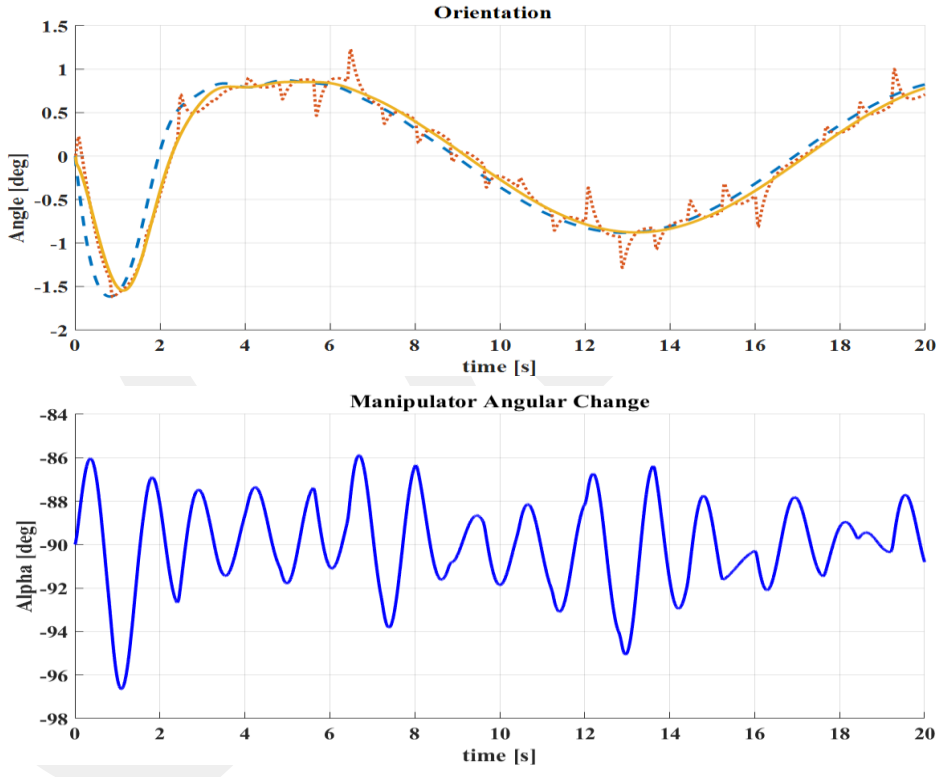
Figure 3.12 Decentralized Control Detailed Schematic for Case 1

System simulations are made on two different scenarios. In the first case, the position control in all axes of the system is controlled by the propeller units while the manipulator is rejecting disturbances on x axis. In this way, the system is intended to provide high accuracy while performing the tasks. In the second case, the propeller unit of the system provides only altitude control, while the manipulator is responsible for the position and angular control in the x axis. Thus, the system can be controlled by the manipulator in case of any failure or similar situations.

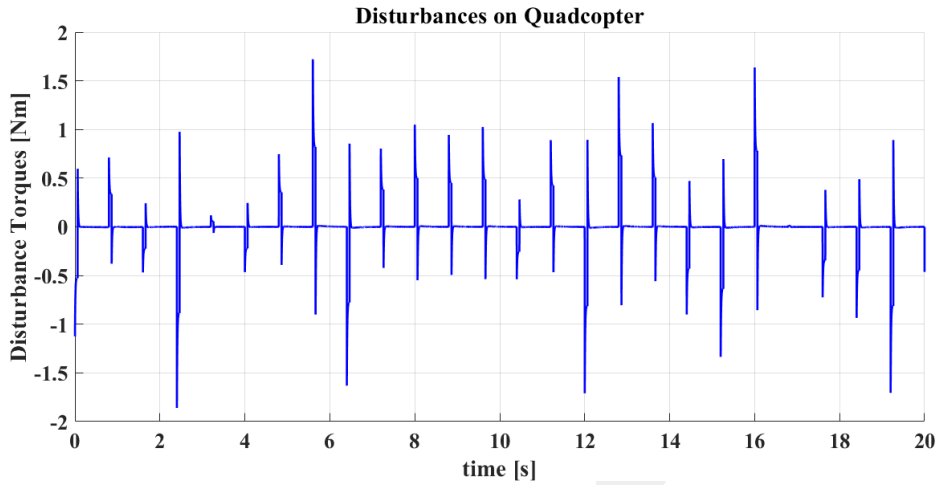
The simulation of the first scenario, i.e. the position control under 1.25 Hz random impulsive disturbances (mean=0, variance=2) at the constant altitude, is given in the Figure 3.13.


Displacement on X Axis
X [m]
time [s]
X_Ref
Fixed Manipulator
Active Manipulator

(a)


Orientation
Angle [deg]
time [s]
Manipulator Angular Change
Alpha [deg]
time [s]

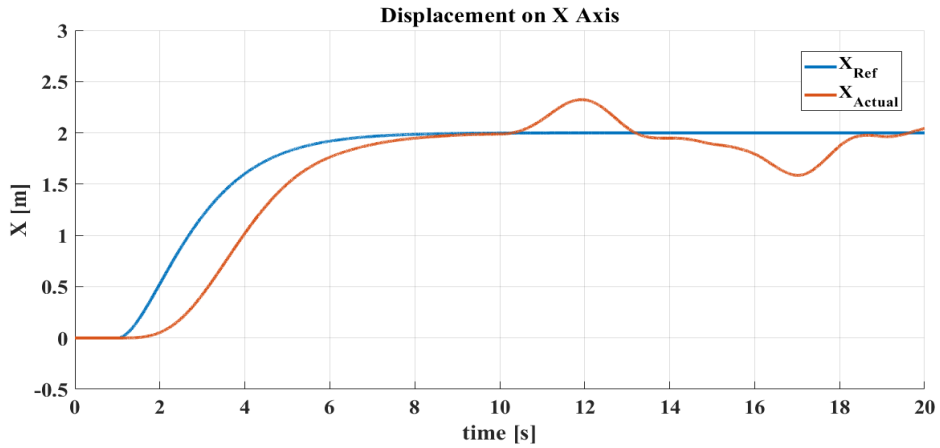
(b)



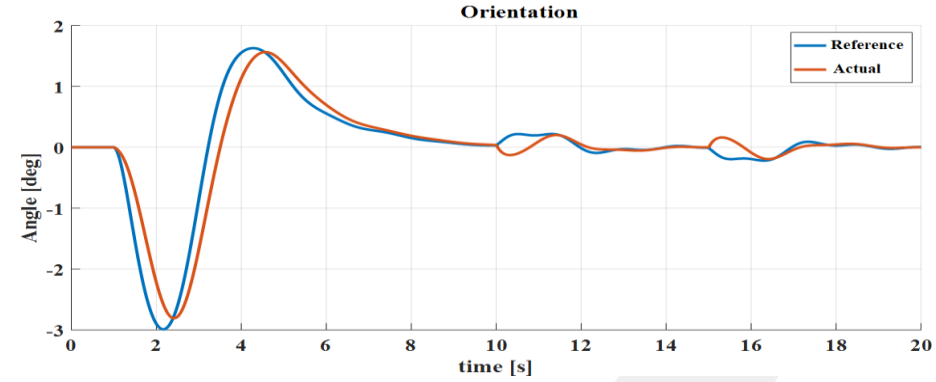
(c)

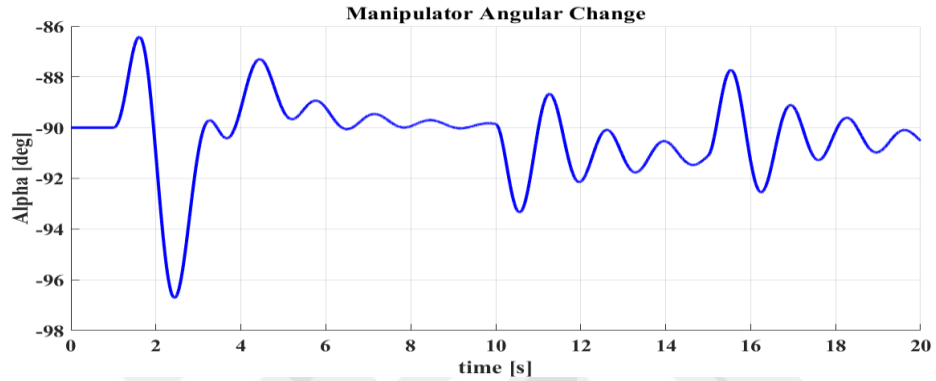
Figure 3.13 System Position Control Results Under Random Disturbances (a) System with and without Manipulator Position Responses (b) System with and without Manipulator Angular Changes (c) 1.25 Hz Random Impulsive Torque Disturbance on System

The simulations depict that the quadcopter with the fixed manipulator (or no manipulator) show oscillatilatory response during the position tracking in X axis. On the other hand, the manipulator acts for better position tracking by rejecting disturbances. In addition to the given random disturbance, in order to simulate the interaction of the system with the environment, the necessary disturbance torque input for the door handle opening test is applied to the system. Obtained responses of the system given in Figure 3.14.

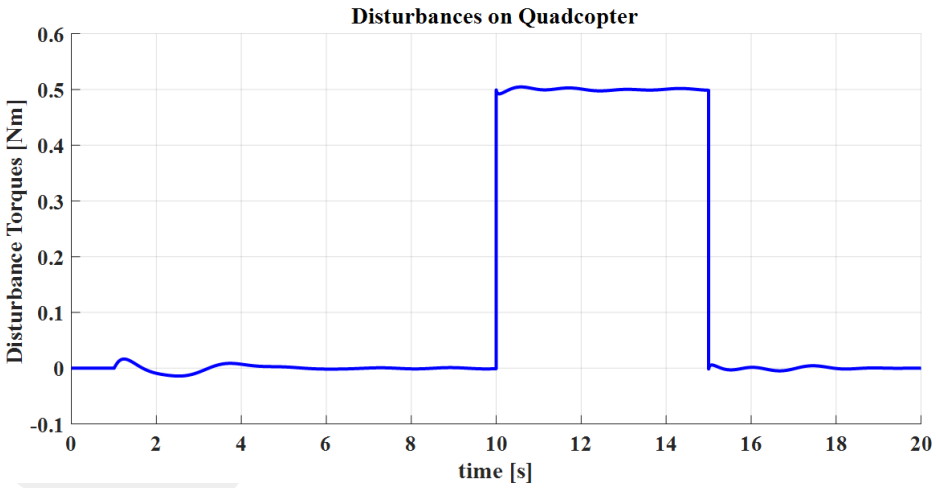


(a)





(b)

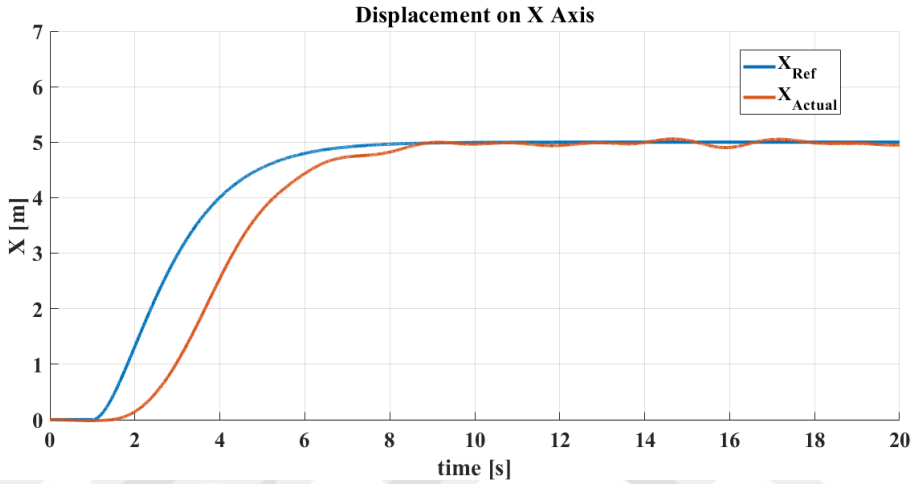


(c)

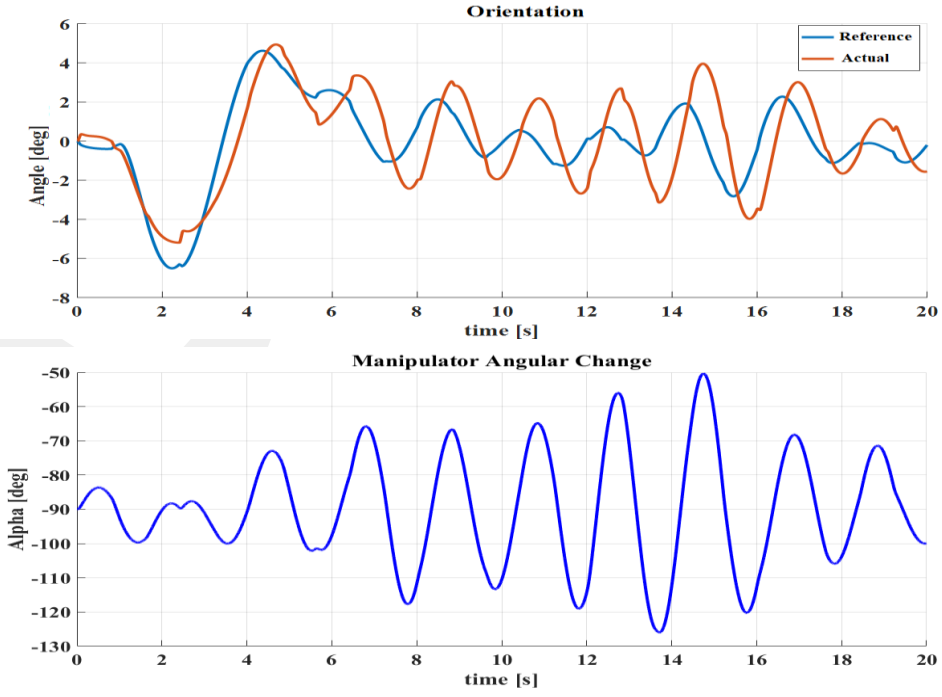
Figure 3.14 System Responses under Disturbances Calculated for Door Handle during Displacement (a) Position Change on x axis (b) Angular Change (c) Applied Door Handle Disturbance

The results show that the manipulator successfully rejects the effects of disturbances on the system. The system changes the position to approach the door handle as required and it can successfully accomplish the opening of the door by creating the necessary counter-torque by manipulator.

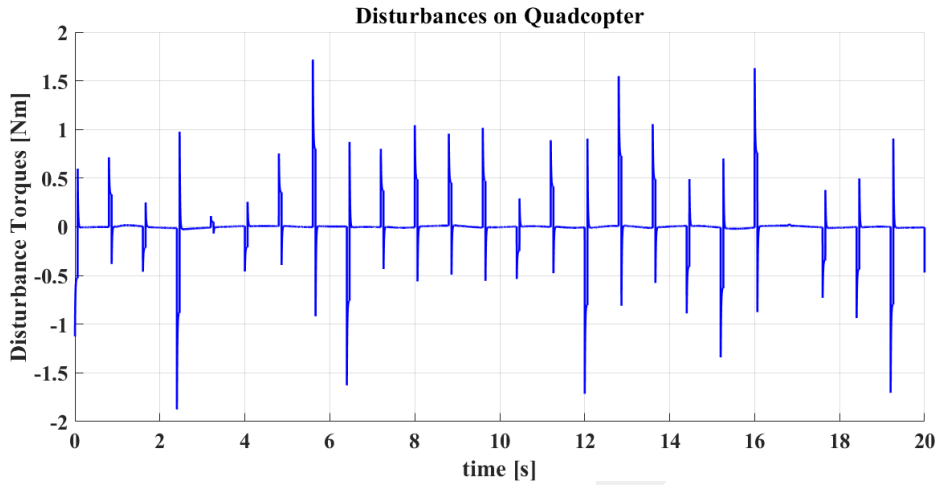
The simulation of the second scenario, i.e. the position control under random disturbances at the constant altitude, is given in the Figure 3.15. In this case, the attitude is manipulated only by the action of the manipulator.



(a)



(b)

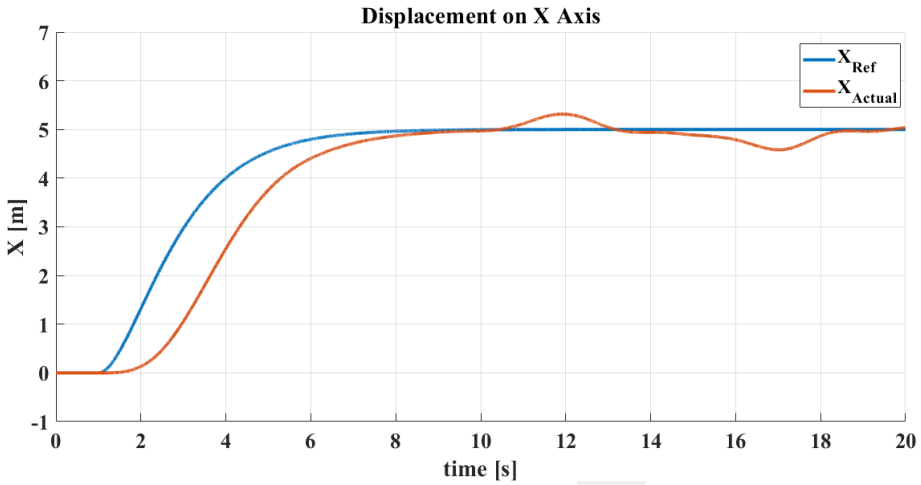


(c)

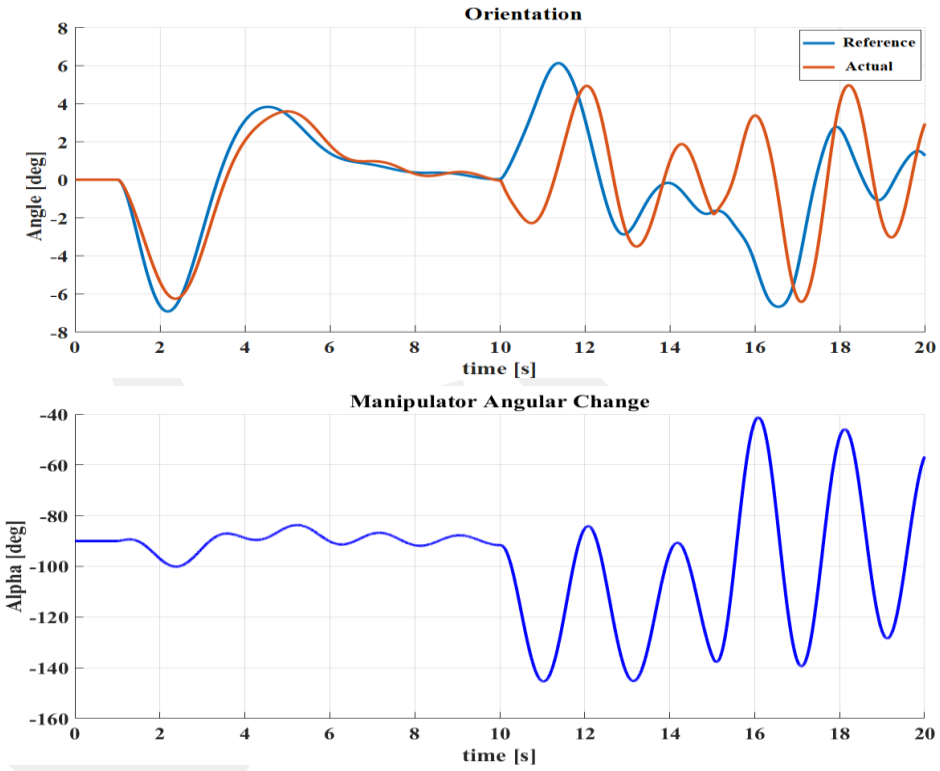
Figure 3.15 System Position Control Results Under Random Disturbances for Second Scenario (a) System with and without Manipulator Position Responses (b) System with and without Manipulator Angular Changes (c) 1.25 Hz Random Impulsive Torque Disturbance on System

The results obtained shows that the manipulator performs the position and angular control of the system with better performance than expected. However, because of the system also works in the rejecting disturbances, amplitude of angular change of the manipulator is much larger than normal. The latency in tracking the position reference can be corrected by making the control algorithm tighter. However, this change decreases the resistance of the system to disturbances and decreases performance in orientation stabilization.

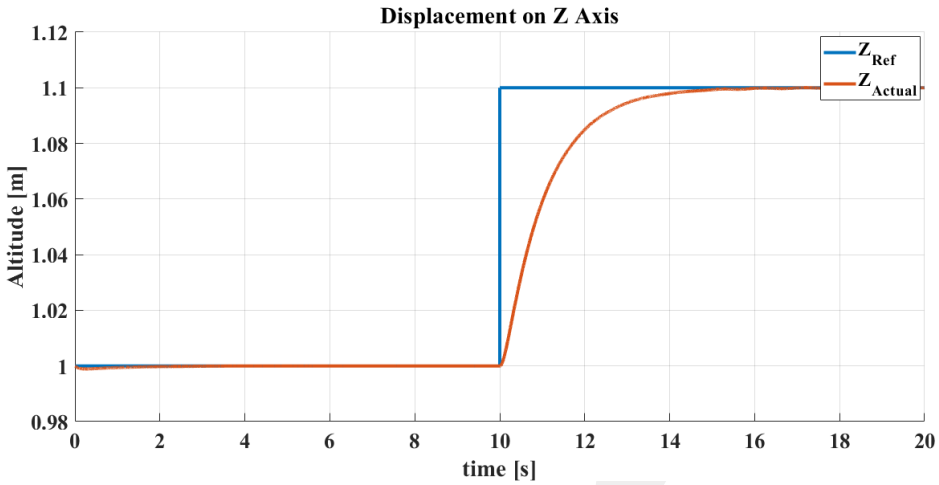
In addition, the door handle opening test like all other simulations is also applied for this scenario as well. System responses given in Figure 3.16.


Displacement on X Axis
X [m]
time [s]
X Ref
X Actual

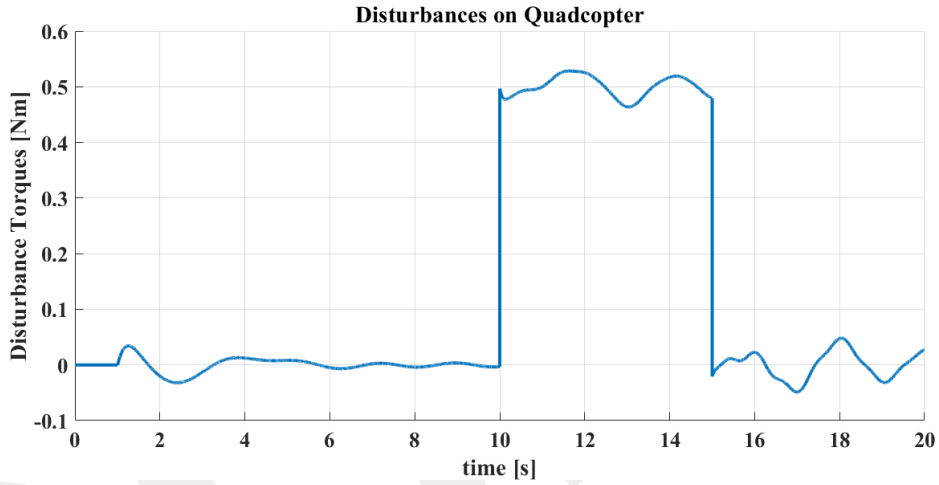
(a)


Orientation
Angle [deg]
time [s]
Reference
Actual
Manipulator Angular Change
Alpha [deg]
time [s]

(b)



(c)



(d)

Figure 3.16 System Responses Under Disturbances Calculated for Door Handle during Displacement for Second Scenario (a) Position Change on x axis (b)Angular Change (c) Altitude (d) Applied Door Handle Disturbance

In order the scenario more flexible control required, the control structure is normalized again to accomplish position tracking by the manipulator. Therefore, the results obtained from the scenario cause the system to be more sensitive to disturbances and increase the magnitudes of angular change of the manipulator.

CHAPTER 4

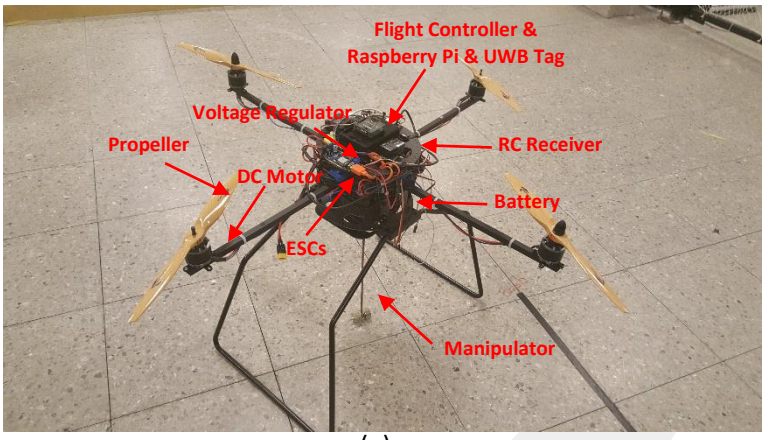
4. PHYSICAL SYSTEM AND SYSTEM PERFORMANCE TESTS

4.1 Physical System Topology

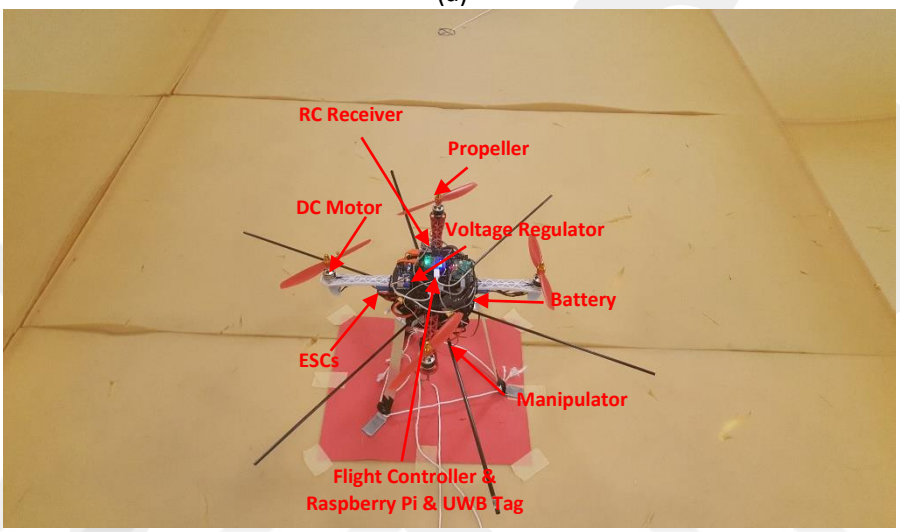
In this section, 2 different Quadcopters are used to test the control structures designed in the simulations. The reason for this is to see the effect of the designed manipulator on different quadcopters. The first one (Frame 1) has a large frame capable of generating high thrusts, while the second one (Frame 2) has a smaller frame producing lower thrusts. Other than the chases, propulsion units, and the battery packs, all the remaining subsystems are the same for both of the frames. The Quadcopter platforms simply include the following parts:

- Flight Controller (NAZE32)
- Inertial measurement unit (MPU6050)
- Electronic Speed Controllers
- Brushless DC motors
- Propellers
- Battery
- RC Transmitter
- Raspberry Pi 3B
- Power Distribution Board
- Manipulator
- Indoor Localization Tag

Figure 4.1 shows the Frame 1 and 2's physical structure.



(a)



(b)

Figure 4.1 Physical Structures of Quadcopter Frames (a)Frame 1 (b)Frame 2

The flight controller drives the brushless DC motors with the help of drivers using I/O ports. The control algorithm was tuned by the interface called Multiwii, which is the embedded software of the flight controller. With this software, the necessary mode changes can be made with the RC controller during the flight. In addition, the flight controller board is connected to Raspberry Pi via USB connection. Using the Multiwii's own communication protocols, the required sensor data, RC inputs, and the PWM inputs to the motors can be acquired.

The manipulator is controlled in real time with the Raspberry Pi. The control algorithms designed on Matlab/Simulink and a Python Script are applied to the manipulator in real time. Data logging and analysis can be performed in real time using the same method. Communication between the computer and the Raspberry Pi is provided wirelessly over a rooter. The 14.8 volts li-po battery provides the necessary power to the whole system. The 5 volts low voltage required for the cards

and the servo motor of the manipulator is obtained by regulating 14.8 volts. A simple schematics representation of the relation among the components is given in Figure 4.2.

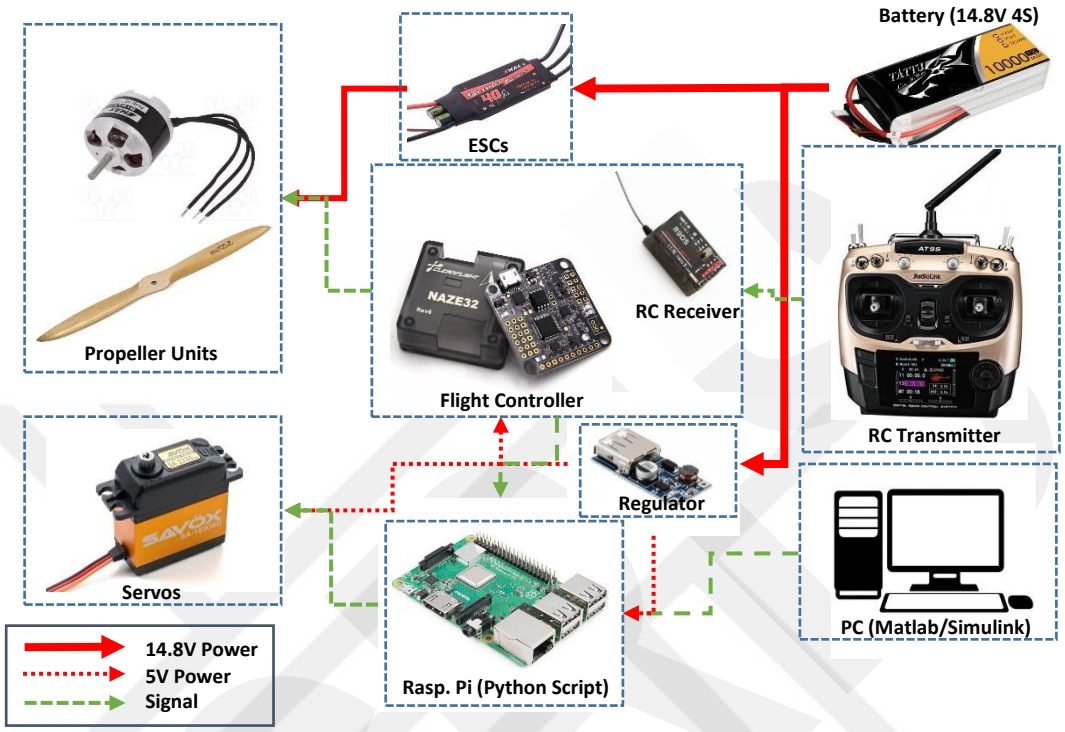


Figure 4.2 Physical System Schematic

4.2 Mechanical Structure of the System

The simulated system model should be realistic and ready for real field applications. Therefore, it is very important to obtain the unknown parameters in the system dynamics correctly. The Bifilar Pendulum experiment is used to calculate the moment of inertia values about body x, y, and the z axes. By allowing the system to oscillate on the specified axis, the period of oscillation is calculated. The moment of inertia can be calculated by using the period of the oscillation and the lengths which shown in Figure 4.3.

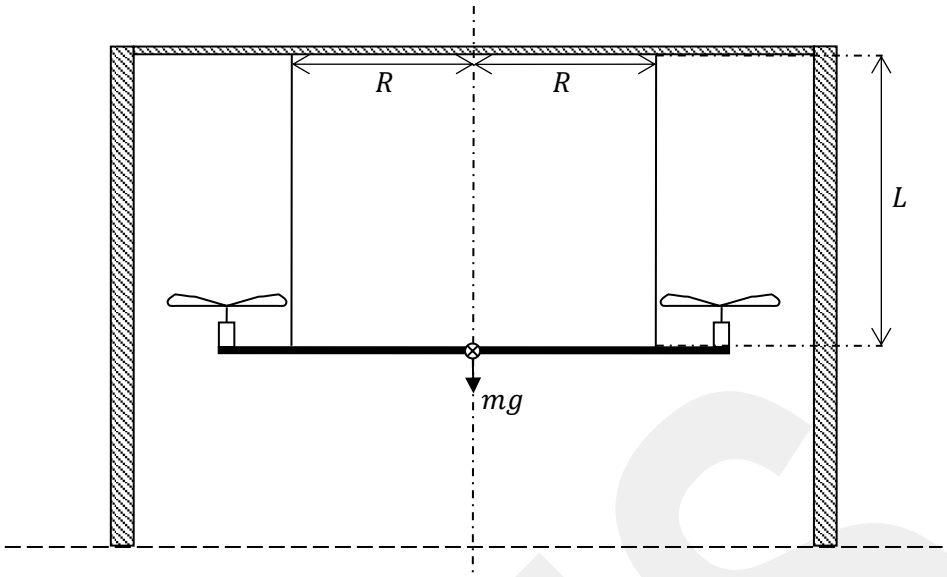


Figure 4.3 Bifilar Pendulum Experiment Structure

After the period of an oscillation is found, the found period value and lengths are written in the equation given in equation 18.

$$J = \left[\frac{T_n}{2\pi}\right]^2 \frac{mgR^2}{L} \tag{18}$$

Where T_n the period of an oscillation is, R is the length between center and the rope, L is the length of the rope, m is the mass of the system and J is the moment of inertia. The experiment was done for each axis using the structure given in Figure 4.3. The images of the experiment of Frame 1 are given in Figure 4.4.

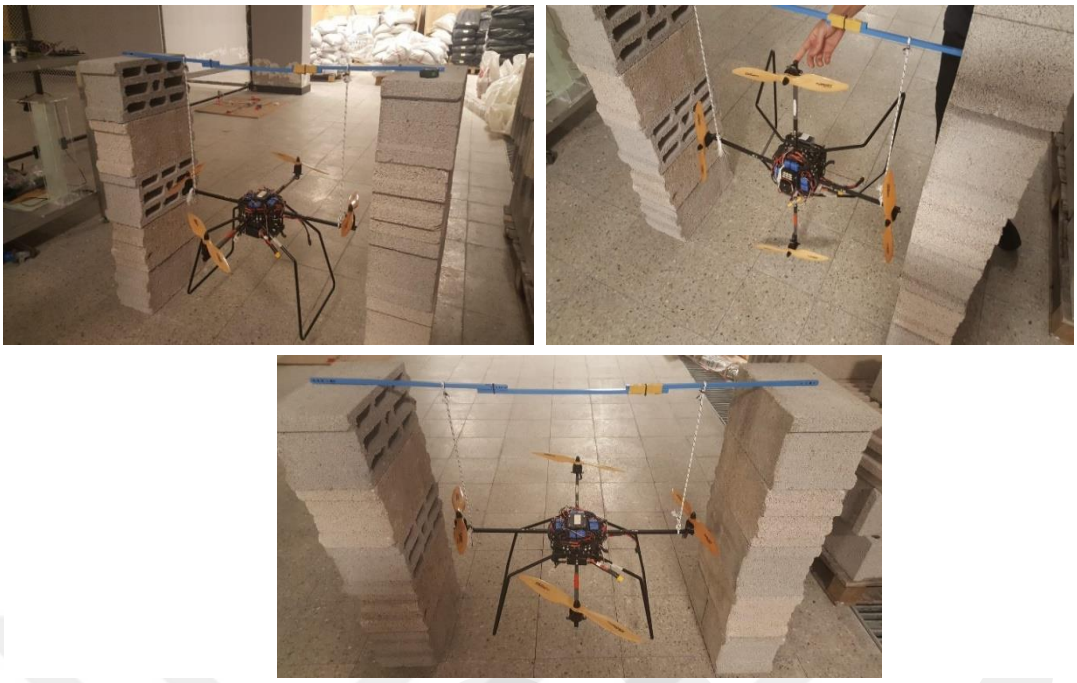
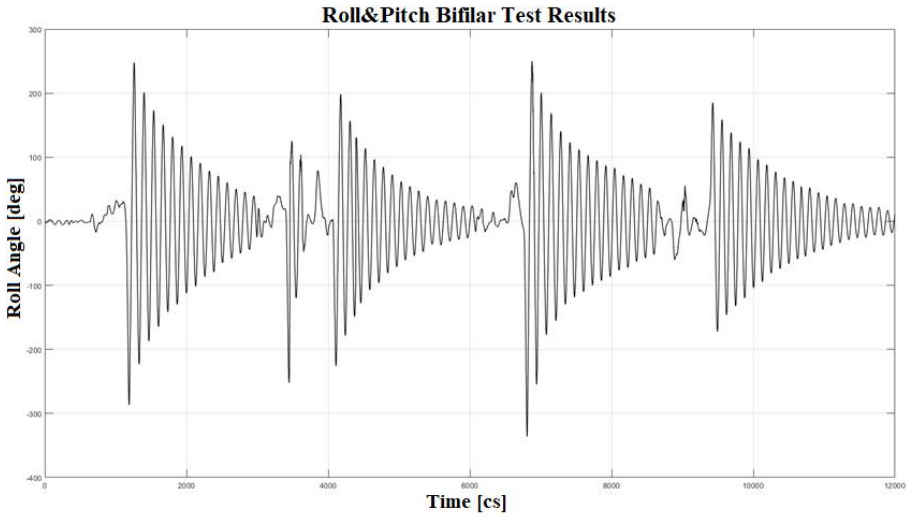
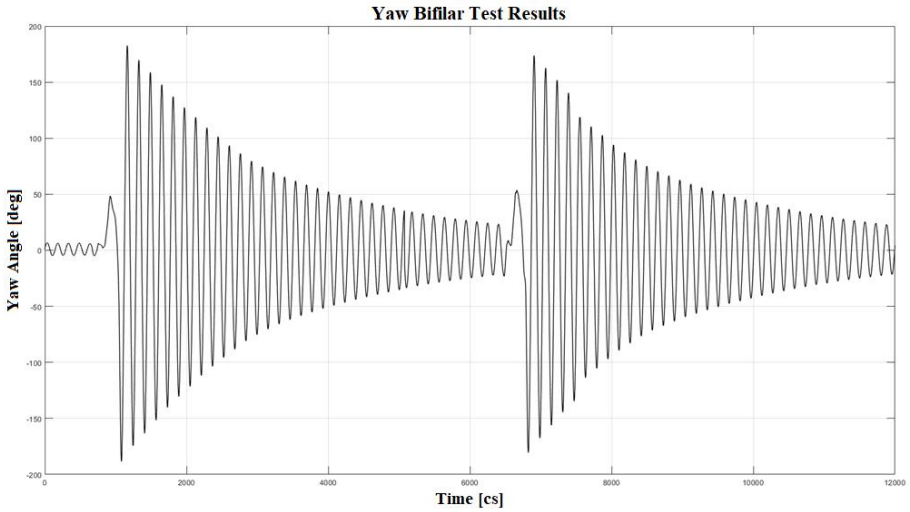


Figure 4.4 Bifilar Pendulum Test of Frame 1

After the system was prepared as in the figure, an external force was applied to ensure oscillation. Angular changes during testing were recorded using IMU on Pixhawk v2 for Frame 1. The graphs of the data are given in figure 4.5.



(a)



(b)

Figure 4.5 Angular Change Graphs for Roll (a) and Yaw (b) of Frame 1

When using the data and graphics we have, we find period 0.91 second for yaw and period 0.75 second for roll. Since the system is symmetrical, it is assumed that the system will give the same result for roll and pitch. When the period values found are written to equation 18, the moment of inertia of the Frame 1 for roll and pitch is calculated as 0.0676 kgm^2 and 0.0995 kgm^2 for yaw.

The same tests were repeated in Frame 2 to determine the initial inertia of it. Experiment pictures of Frame 2 are given in Figure 4.6.

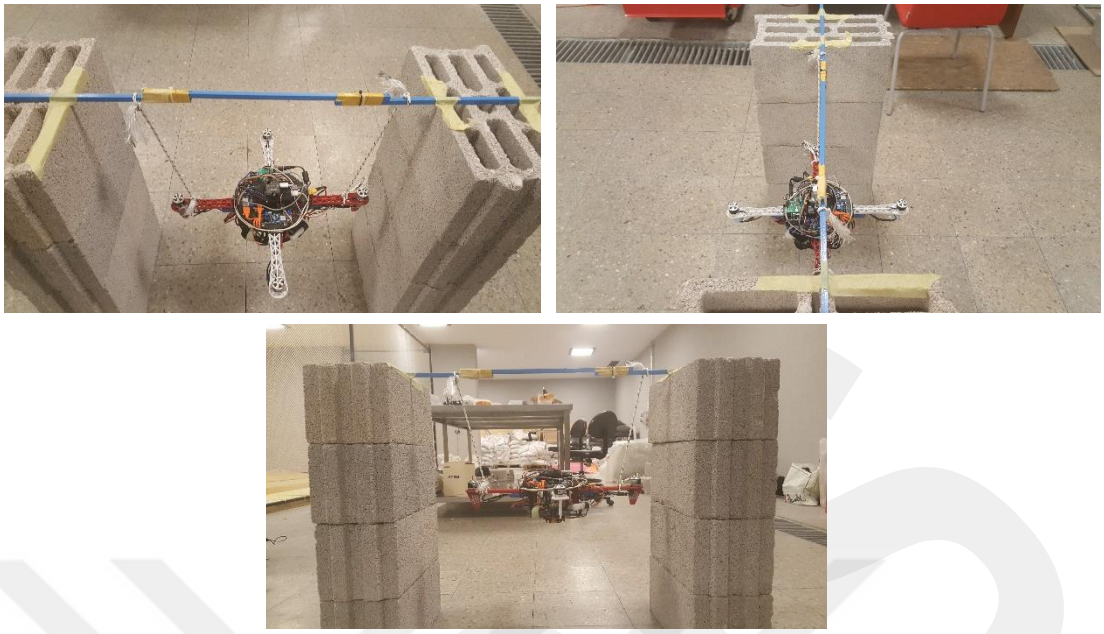
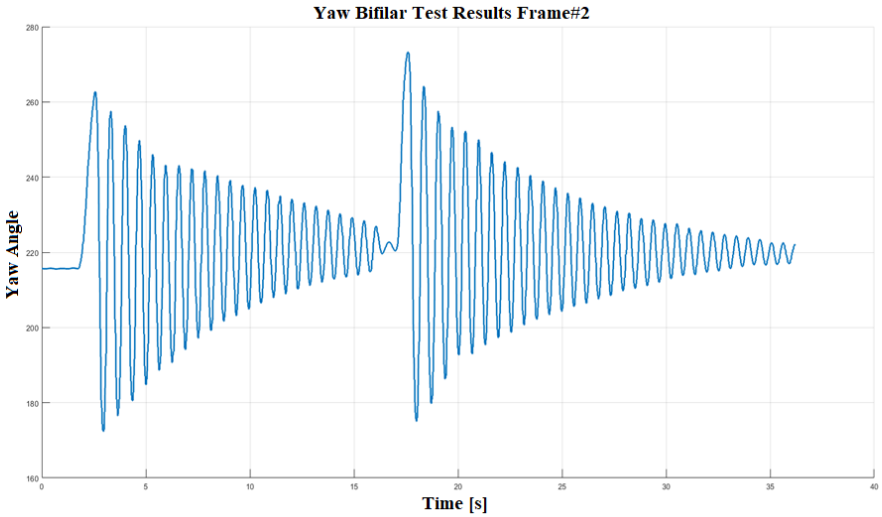
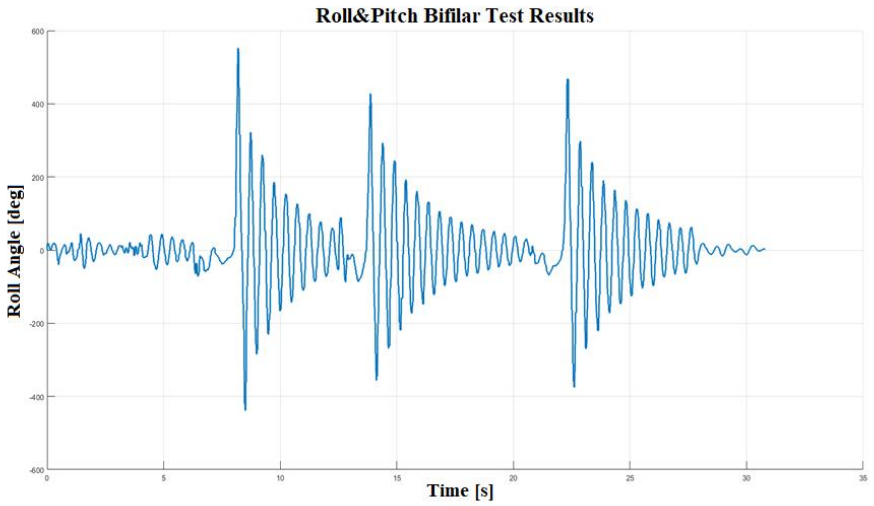


Figure 4.6 Bifilar Pendulum Test of Frame 2

When the same test was applied to Frame 2, the Euler angle values were taken from the IMUs above the Pozyx indoor localization tag and Naze32. The obtained angular change graphs are given in Figure 4.7.



(a)



(b)

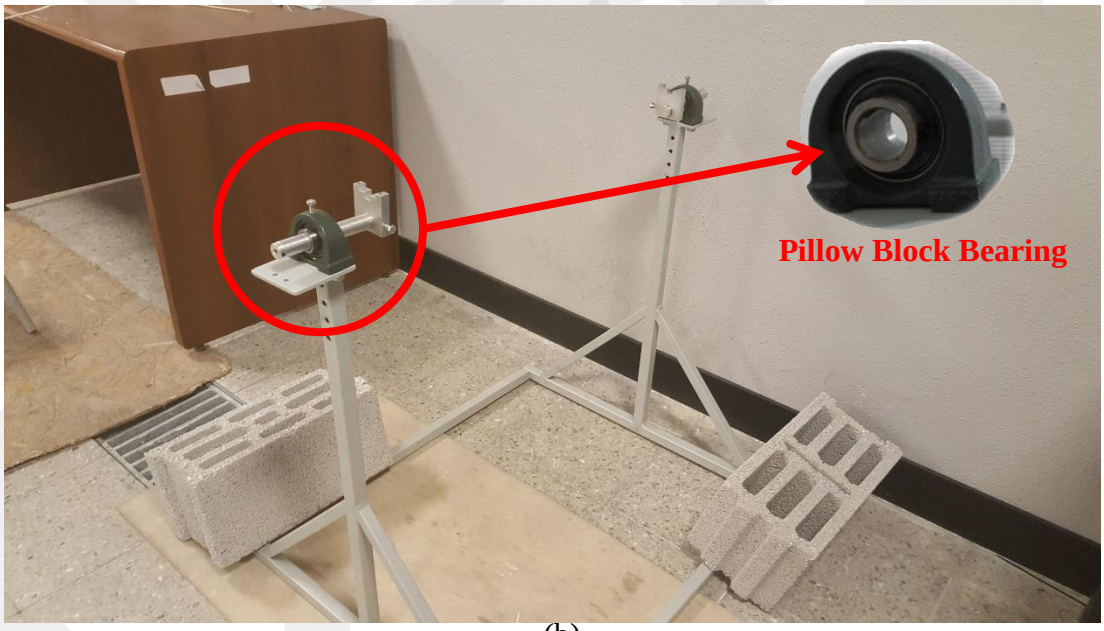
Figure 4.7Angular Change Graphs for Roll (b) and Yaw (a) of Frame 2

When the plots are examined, period of oscillations on yaw is 0.715 seconds and period of oscillations on roll is 0.476 seconds. Since Frame 1 is symmetrical as like Frame 1; the moment of inertia of the Frame 2 for roll and pitch is calculated as 0.0058 kgm^2 and 0.0130 kgm^2 for yaw.

4.2.1 Test Setup

A test setup with 2 pillow block bearings is used to start the system's physical tests more safely and with precautionary measures, and tune the controllers easily. The test platform, combined with aluminum profiles, is of great importance in understanding the attitude dynamics of the system and tuning the attitude controllers. It prevents accidents caused by unstable controls during flights. The pictures and the CAD model of the test platform are given in the Figure 4.8.

(a)



(b)

Figure 4.8 Test Platform (a) CAD Model (b) Real Structure

4.3 Manipulator Structure

The manipulator mounted on the pitch axis of the quadcopter consist of; servo motor, 'C' type servo bracket, link and end effector as weight. Although the weight of the manipulator was the same in the both frames, the link length was 0.2 meters for the smaller Frame 2, and 0.3 meters for the large Frame 1. The CAD model of the manipulator is given in Figure 4.9.

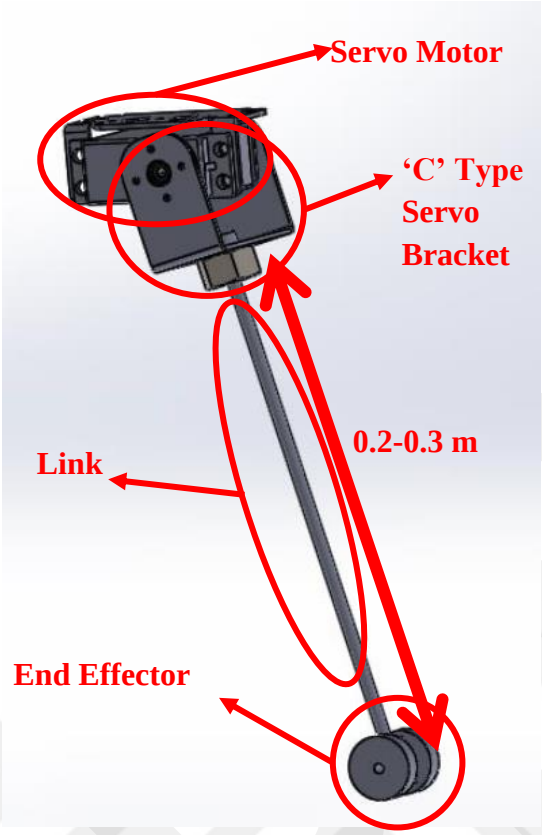


Figure 4.9 Manipulator's CAD Model

Savöx SH-1290MG servo motor was used for actuating the manipulator. Just as all the conventional servo motors, it can provide angular position control with 1-2 ms PWM inputs. The characteristics of the servo motor are given in Table 4.1.

Table 4.1 Servo Motor Specification of Manipulator

FEATURE	UNIT
OPERATING VOLTAGE	4.8-6 V
OPERATING SPEED (6V)	0.05 ses/60^{o} at no load
OPERATING ANGLE	100 deg
TORQUE (6V)	0.049 kg.m
CURRENT DRAIN (6V)	Idle Current: 5mA, Stall Current : 3A
DIMENSIONS	40.3x20.2x37.2 mm
WEIGHT	56.1 g

4.4 Flight Controller

In this study, Naze 32 rev6 10 DOF flight controller is used. This version of Naze32 uses a 32-bit STM32F103 processor instead of the 8-bit processor available on standard MultiWii or KK series flight controller. In this way, many advanced PID control algorithms, GPS "return to home" feature, different flight modes, built-in flight data logging features are applicable. Naze32 is shown in Figure 4.10 with its pin diagram.

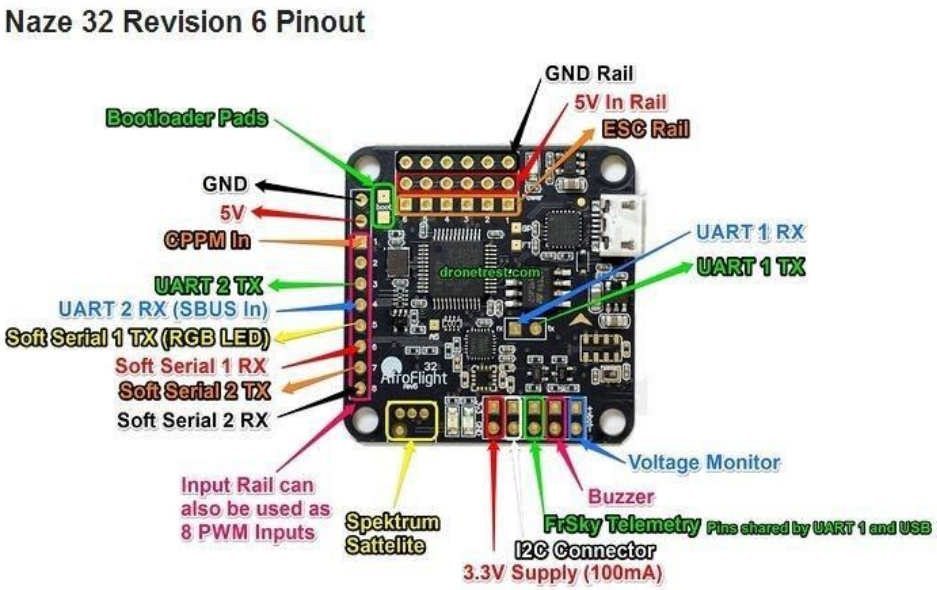


Figure 4.10 Naze32 Flight Controller and Pin Layout [26]

Although the flight controller designed for small and medium sized (180-250 class quadcopter frames) multicopters, it can be also used for multicopters up to 550 class. Since it is a 10 DOF version, it has gyroscope, accelerometer, compass and barometer sensors. Technical specifications of the flight controller:

- Supports PWM / (C) PPM / SBUS / Spectrum Satellite receiver,
- 6 motor / servo outputs (supports tricopter, quadcopter and hexacopter)
- Micro USB type B connector
- MPU6500 accelerometer / gyroscope sensor
- BMP280 barometric pressure sensor
- HMC5983 magnetometer (compass) sensor
- 3.3V standard header pins for I2C connection

- Battery voltage monitor, buzzer, telemetry outputs
- Inverter circuits for SBUS
- Connection pins for sonar sensor

In addition, sensor outputs or other outputs can be accessed with the designated communication protocol via the USB or serial pin connections.

4.5 Power Distribution

There is a 14.8 volts 4S Li-Po battery on the system. The voltage from the battery can be distributed to ESCs by parallel distribution, maintaining the 14.8 volts value without changing. In addition, the system has a flight control card, Raspberry Pi and Servo motor. They are supplied with 5 volts using an integrated voltage regulator. Power distribution diagram is given in Figure 4.11.

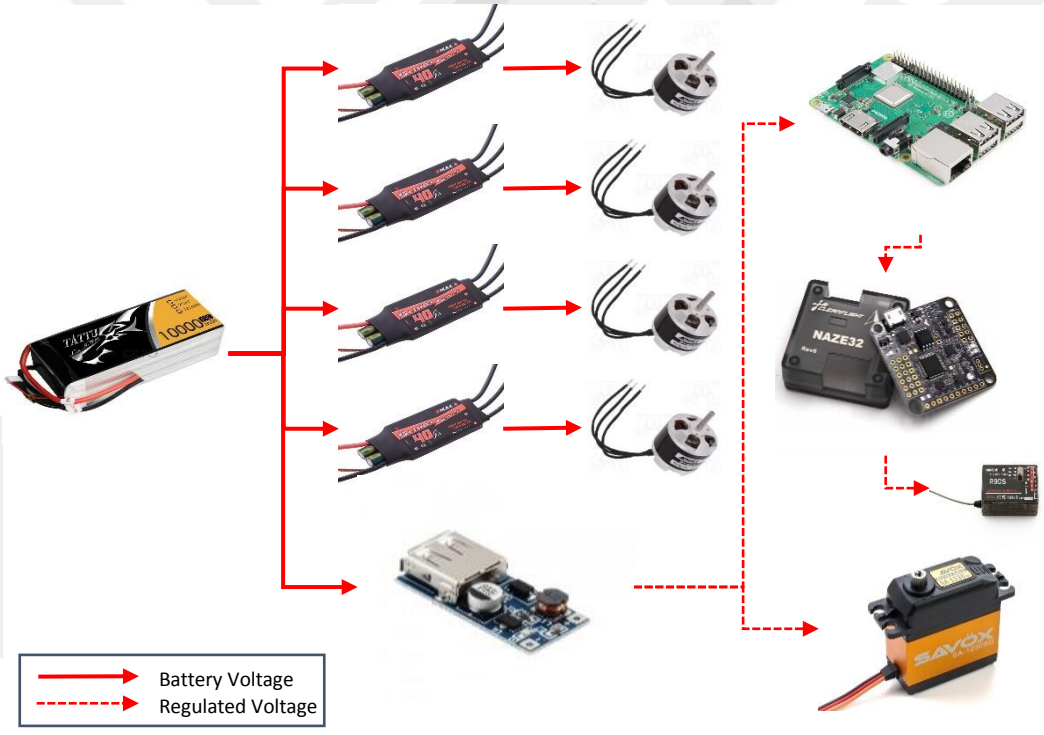


Figure 4.11 Power Distribution Diagram

The voltage regulator can reduce the received 14.8 volts to 5 volts and distribute it via USB and the additional connections on it. While the Raspberry Pi is supplied at 5 volts from the USB connection on the regulator, the flight card is powered via Raspberry Pi. The servo motor reaches 5 volts from the additional connections on the regulator.

4.6 Controller Board (Raspberry Pi 3 B+)

Raspberry Pi is used as the main control equipment in this study. Raspberry pi is a single board computer (Dimensions: 85 x 56 x 17 mm) that has its own Python-based operating system, which can be accessed via a display or a Linux-based terminal display. With its features and software support, it is very suitable for fast and real-time control applications. In addition to being Python based, it is also supported by MatLab / Simulink and is capable of carrying out many new studies. In addition, it can easily communicate with many devices with the connections and hardware support on it. It is based on Linux operating system and provides a wide working area to the user. Hardware structure of Raspberry Pi 3 B+ in Figure 4.12.

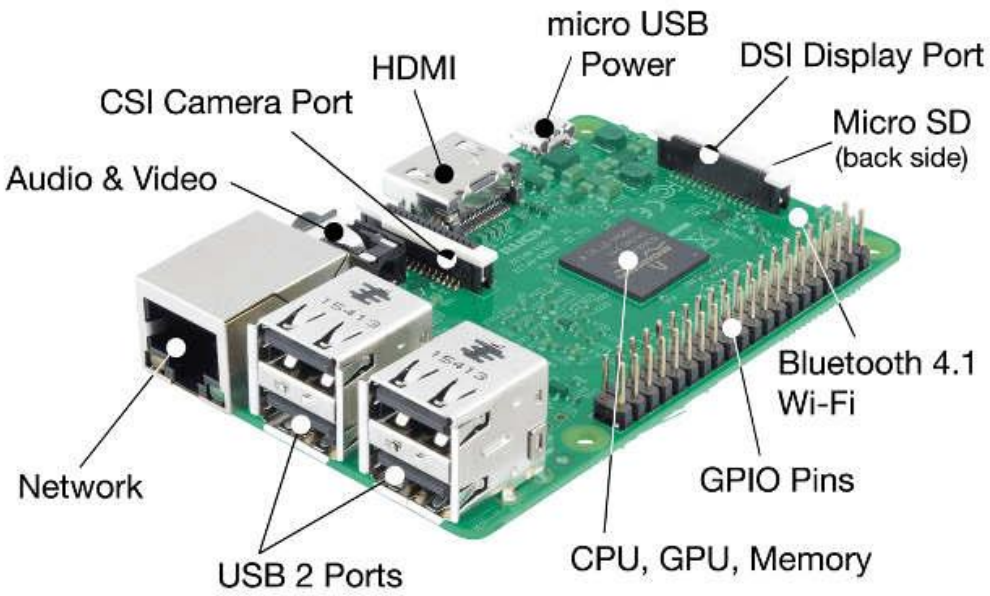


Figure 4.12 Raspberry Pi 3 B+ Hardware [30]

In addition, it provides data exchange with 40 other input-output pins. These pins have different characteristics due to the hardware structure and can be used for different purposes. The pin diagram of Raspberry Pi 3B + is given in Figure 4.13.

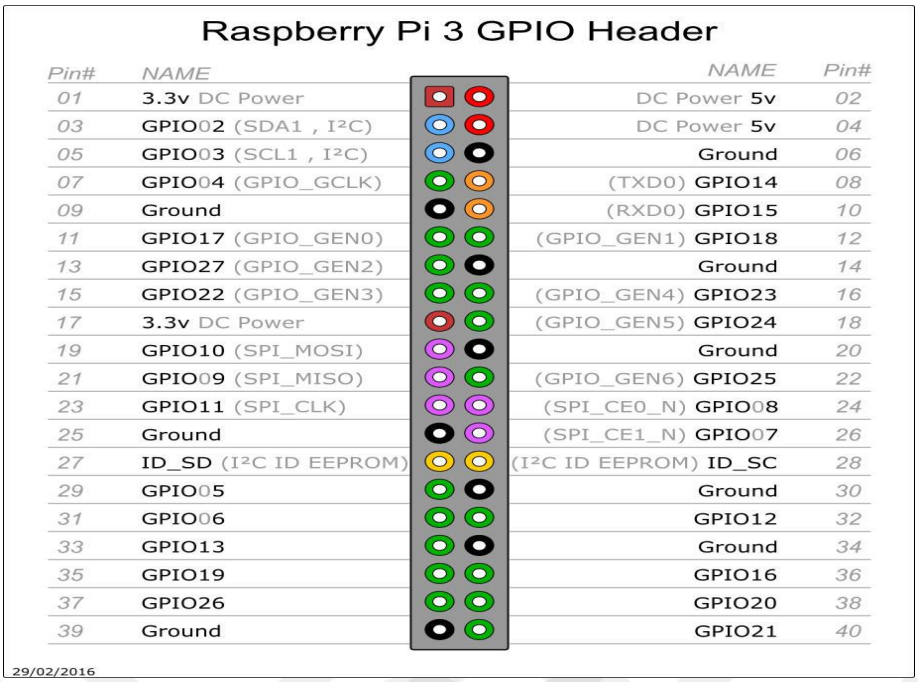


Figure 4.13 Pin Layout of Raspberry Pi 3B+ [31]

Other technical specifications of the control board are listed;

- Broadcom BCM2837 SoC
- 1.2 GHz 4-core 64-bit ARM Cortex-A53 processor
- 2-core Videocore IV® Multimedia processor
- 1GB LPDDR2 memory
- WiFi with built-in 802.11b / g / n support
- Bluetooth 4.1, low-energy supported
- Ethernet port with 10/100 Mbit support
- HDMI port (HDMI 1.4 supported)
- 3.5mm TRRS (4-pin) connector for composite video and audio output
- 4 USB2.0 ports
- 40 GPIO pins, compatible with previous Raspberry Pi models
- Built-in chip antenna for WiFi / Bluetooth
- CSI (camera) and DSI (display) connectors
- Micro SD card slot
- Supports all Raspberry Pi compatible Linux distributions and Windows 10 IoT Core operating system

4.7 Details of Subsystems

4.7.1 Battery

Since different propeller units are used in the frames and there was a large thrust difference between them, different batteries were used for the frames. For Frame 1 1000 mAh 14.8V 4S Li-Po battery is used. Details of the battery are in the table given in Table 4.2.

Table 4.2 Frame 1 Battery Features

FEATURE	UNIT
CAPACITY	10000 mAh
VOLTAGE	14.8 V (4S)
MAX CONTINUOUS DISCHARGE	25C (250A)
MAX BURST DISCHARGE	50C (500A)
WEIGHT	942 g
DIMENSIONS	166x64x41 mm
CHARGE RATE	1-5C

As the Frame 1 can produce more thrust and the DC motors are suitable for 14.8 volts, this battery with a high capacity and close to 1 kilogram could easily be used. However, Frame 2 is able to produce much less thrust and DC motors are suitable for 11.1 volts. Therefore a lighter and lower capacity battery is used. Details of the battery 2 are in the table given in Table 4.3.

Table 4.3 Frame 2 Battery Features

FEATURE	UNIT
CAPACITY	3300 mAh
VOLTAGE	11.1 V (3S)
MAX CONTINUOUS DISCHARGE	25C (250A)
MAX BURST DISCHARGE	50C (500A)
WEIGHT	240 g

DIMENSIONS	135x44x19 mm
CHARGE RATE	1-3C

In addition to the features given in Table 4.3, the pictures of the batteries used are given in Figure 4.14.

(a) (b)

Figure 4.14 Used Batteries (a) Frame 1 (b) Frame 2

4.7.2 ESCs

In order to drive brushless motors, Electronic Speed Controller (ESC) is used. In this system, for both frames TurkUAV 40A brushless ESC is used (Figure 4.15). Brushless ESC technical specs are given in Table 4.4.

Table 4.4 Feature of ESCs

FEATURE	**UNIT**
INPUT VOLTAGE	2-4S (7.4V-14.8V)
INPUT TAE RAGE	50-400Hz
OUTPUT CURRENT	18.9A
MAX WATTS	40A Continuous,50A Burst(10 Sec)
BEC OUTPUT	5V/3A
WEIGHT	38g
DIMENTIONS	55x26x13 mm

Figure 4.15 ESC of Quadcopter

4.7.3 Brushless DC Motors and Propellers (Propeller Units)

For both frames, completely different propeller units were used. This is due to the suitability of the batteries used for the DC brushless motors and the weight capacities of the frames**.** For the heavier Frame #1, the 750 kV brushless motor and the wooden 12-inch propeller were used to produce more thrust. However, for the lighter Frame #2, which was intended to produce less thrust, a 980 kV brushless motors and 10 inch plastic propellers were used. The details of the motors used are given in the tables 4.5 and 4.6.

Table 4.5 Motor Properties of Frame 1

FEATURE	UNIT
KV VALUE	750
INTERNAL RESISTANCE	0.0063 mΩ
NO-LOAD CURRENT	0.65A
MAX CURRENT	26.75A
CONTINIOUS CURRENT	18.9A
MAX WATTS	375 W
LI-PO CELLS	3-4S
WEIGHT	94 g
DIMENTIONS	35x36 mm
SHAFT DIAMETER	4 mm

Table 4.6 Motor Properties of Frame 2

FEATURE	UNIT
KV VALUE	980
CONTINIOUS CURRENT	15A
LI-PO CELLS	2-3S
WEIGHT	49 g
DIMENTIONS	27.9x43.16 mm
SHAFT DIAMETER	3 mm

The DC brushless motor characteristics according to the propellers is given in Table 4.7.

Table 4.7 Propeller Properties of Frames

MOTORS	VOLTAGE	PROPELLER	CURRENT (A)	THRUST (G)	POWER (W)	EFFICIENCY (G/W)	RPM
FRAME 1 750KV	14.8	12”5	24.76	1850	380	-	8427
FRAME 2 980 KV	11.1	10”4.7	15.1	880	181.2	4.9	6860

The images of the motors used are given in Figure 4.16.

(a) (b)

Figure 4.16 Brushless Motors of the System (a) Frame 1 (b) Frame 2

4.7.4 Voltage Regulator

As mentioned in the previous sections, there are many subsystems on the system that operate at different voltage and amperage values. However, the whole system is powered from a single power source for efficiency. For these reasons, the voltage coming from a single source should be regulated and given where necessary. Voltage regulator board with USB ports is used for this purpose. The regulator used is shown in Figure 4.17.

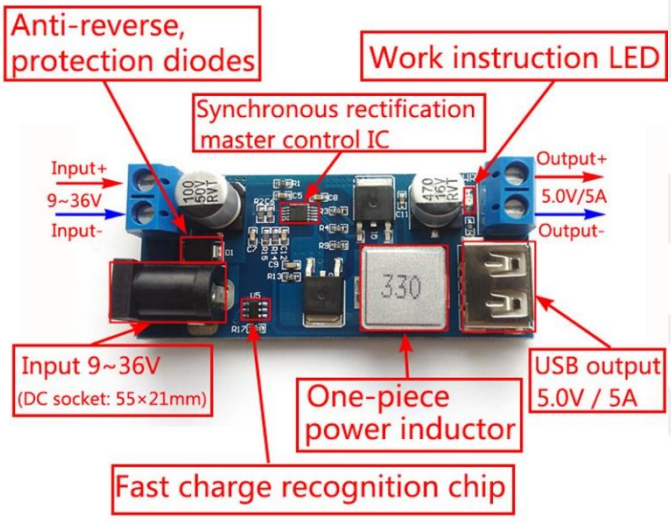


Figure 4.17 DC Regulator Board [27]

The characteristics of the card are given in the table in Table 4.9.

Table 4.8 XY-3606 Regulator Module Specifications

FEATURE	UNIT
INPUT VOLTAGE	DC 9-36V
OUTPUT VOLTAGE	5.2V / 5A
OUTPUT CAPABILITY	5.2V / 6A
WEIGHT	22 g
DIMENTIONS	63x27x10 mm

4.7.2 RC Receiver and Transmitter

The RC receiver and transmitter must be used for manual control and testing of the system. Since the flight controller used supports the SBUS and PPM RC communication protocols, the corresponding RC receiver and transmitter was used. For this, Radiolink AT9S model RC controller (transmitter) and R9DS model receiver of the same brand were used. Features and pictures of the receiver and transmitter are given in Table 4.9 and Figure 4.18.

Table 4.9 RC Receiver and Transmitter Properties

FEATURE	TRANSMITTER	RECEIVER
FREQUENCY	2.4GHz	=
CHANNLE BANDWITH	5MHz	=
INPUT VOLTAGE	18V	5V
RANGE	900x1500 m	=
NUMBER OF CHANNELS	10	=
COMMUNICAITON RATE	3 ms	=
COMMUNICAITON PROTOCOLS	SBUS-PPM-PWM	=
WEIGHT	880 g	20g
DIMENTIONS	183x193x100 mm	41x23x14 mm

Figure 4.18 RC Transmitter and Receiver [28]

4.8 Flight Controller Software

Flight control card uses 3 different software as interface. The oldest one is Baseflight. Since 2014 it is no longer supported by Multiwii, thus, several experimental studies have been carried out in the software's source code because of that software contains many errors. The other software is Betaflight. Although this software is the most recommended software for Multiwii based flight controller users, it does not support Naze32 flight controller. The last and the interface software is used in the study is CleanFlight. This interface has been updated many times and includes different flight modes, control structures and many user-friendly applications. The home page of the interface is given in Figure 4.19.

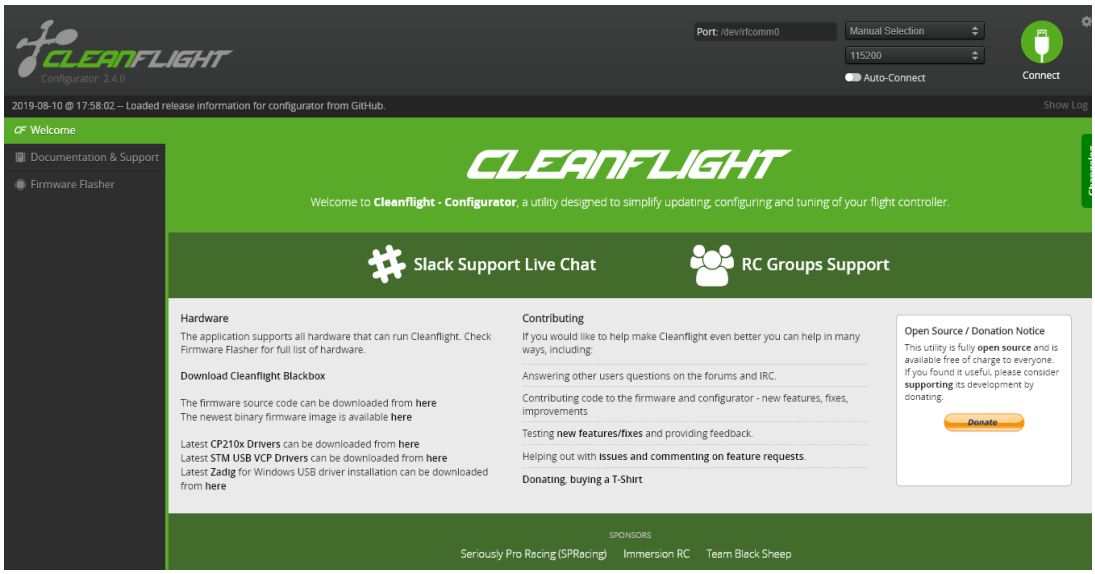


Figure 4.19 Cleanflight User Interface Home Page

If the required firmware is not installed on the flight card, the required firmware must be installed using the Firmware Flasher section of the software. At this part, the "BOOT" pins on the controller must be shorted before connecting the controller to the PC. The flight controller connected in "Boot" mode will be ready for firmware installation. Then, the Firmware Flasher section of the interface should be opened, and the properties of the flight controller should be selected, and the desired firmware version should be installed on the flight controller. The flight controller will then be available to connect CleanFlight interface.

The serial communication settings and pins of the flight controller whose firmware installation is completed, the communication protocol of the RC controller to be used, the shape of the UAV and the magnitudes of the control commands of RC receiver must be determined.

Once all the basic settings have been set, the 2-DOF cascade PID controller coded in the firmware must be tuned in the "PID Tuning" section of the interface (Figure 4.20). At this stage, the coefficients can be changed in real time while the system is flying. In this way, necessary adjustments were made by connecting the system to the test platform. In addition, there are some modes in the firmware. These modes provide additional control by closing the control structure from the outside. With the help of this interface, coefficients of these modes can also be tuned.

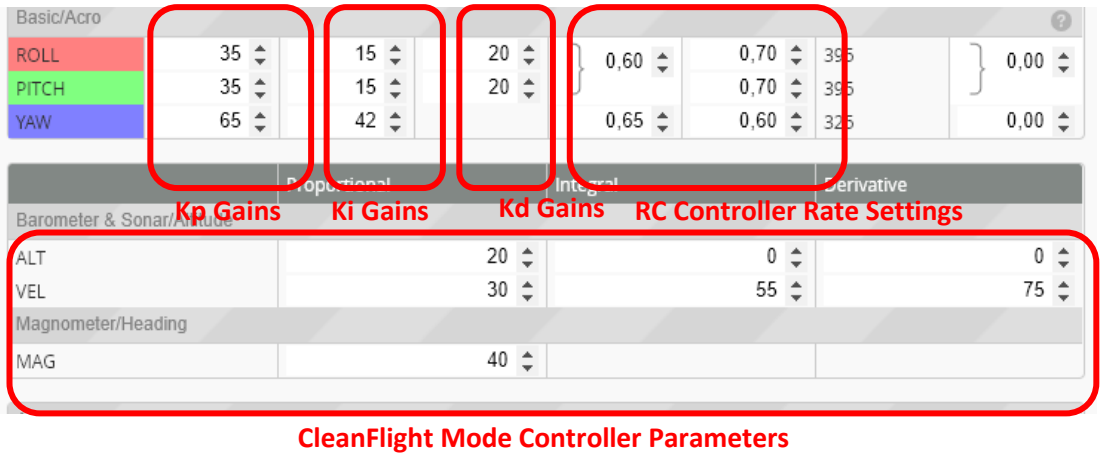


Figure 4.20 PID Tuning Page

In addition to all these interface settings, advanced settings can be made in the CLI section of the interface. For example, relocating the pins on the controller, or the percentages of the control input of the motors. However, if further adjustment is required, no settings are made in the interface. For advanced adjustments, required firmware should be downloaded as a C++ code from Github that Multiwii shared with a desired Python Shell. The downloaded firmware code can then be opened with any editor (ie. Notepad ++) to make the desired changes. After the changes, all the codes of firmware are saved as Hex File by writing “make TARGET=NAZE” in firmware file direction on Python Shell. In this way, it can be changed from control algorithm to communication protocols or algorithms of gimbal controllers. The Hex file can be uploaded to the flight controller using the “Firmware Flasher” on the Cleanflight interface.

4.9 Raspberry Pi - Flight Controller Communication

In order to communicate the Raspberry Pi and the Naze32 flight controller a Python code script, based on the communication of the flight controller with the interface on the PC, was used. This method is a communication protocol used by the software called Cleanflight or BetaFlight, which is the interface of the flight controller, to read the data on the Naze32 and write the necessary operations to the register of the Naze32. The received data can be processed data of any sensor such as accelerometer, gyroscope, magnetometer or barometer on the Naze32 or channels data of the RC controller transferred to the card. The communication protocol is called "Multiwii Serial Protocol" as MSP. This communication protocol can be used

via the USB port of the flight control card or serial communication pins on it. When the general message format specified by Multiwii is followed and transmitted data is correctly packaged or the received data is correctly separated from the package, the required information can be easily accessed. However, the specified message format must be created in hexadecimal format. This makes it impossible to use Matlab / Simulink. According to the format specified, the message should be prepared as follows; preamble, direction, size, command, data.

- Preamble: This is the ASCII character as '$M' to start the message.
- Direction: This is the direction of the data. According to flight controller to a receiver or from a transmitter to flight controller it is the ASCII character '<' for receiving or '>' for transmitting.
- Size: It is the number of data bytes as binary.
- Command: This is the message id number of the requested or sent data. These ID numbers can be reached Multiwii's ID tables.
- Data: This is requested or sent data. However, the data must be in the specified format. For example, if the attitude data is requested from the flight controller, the received data format will be INT16.

Before the communication protocol is applied, the necessary configuration of the communication port of the flight controller should be specified in the python script and serial port settings must be set. Therefore, for the USB port on Raspberry Pi should be set according to the flight controller USB connection that connected the specific port of Raspberry Pi. For example for USB0 port should be set as ser.port="/dev/ttyS0"; baudrate of the port must be set as 115200; byte size of port must be 8 bits; parity must be none. The Flow Chart of the communication between the flight controller and Raspberry Pi as is given in Figure 4.21.

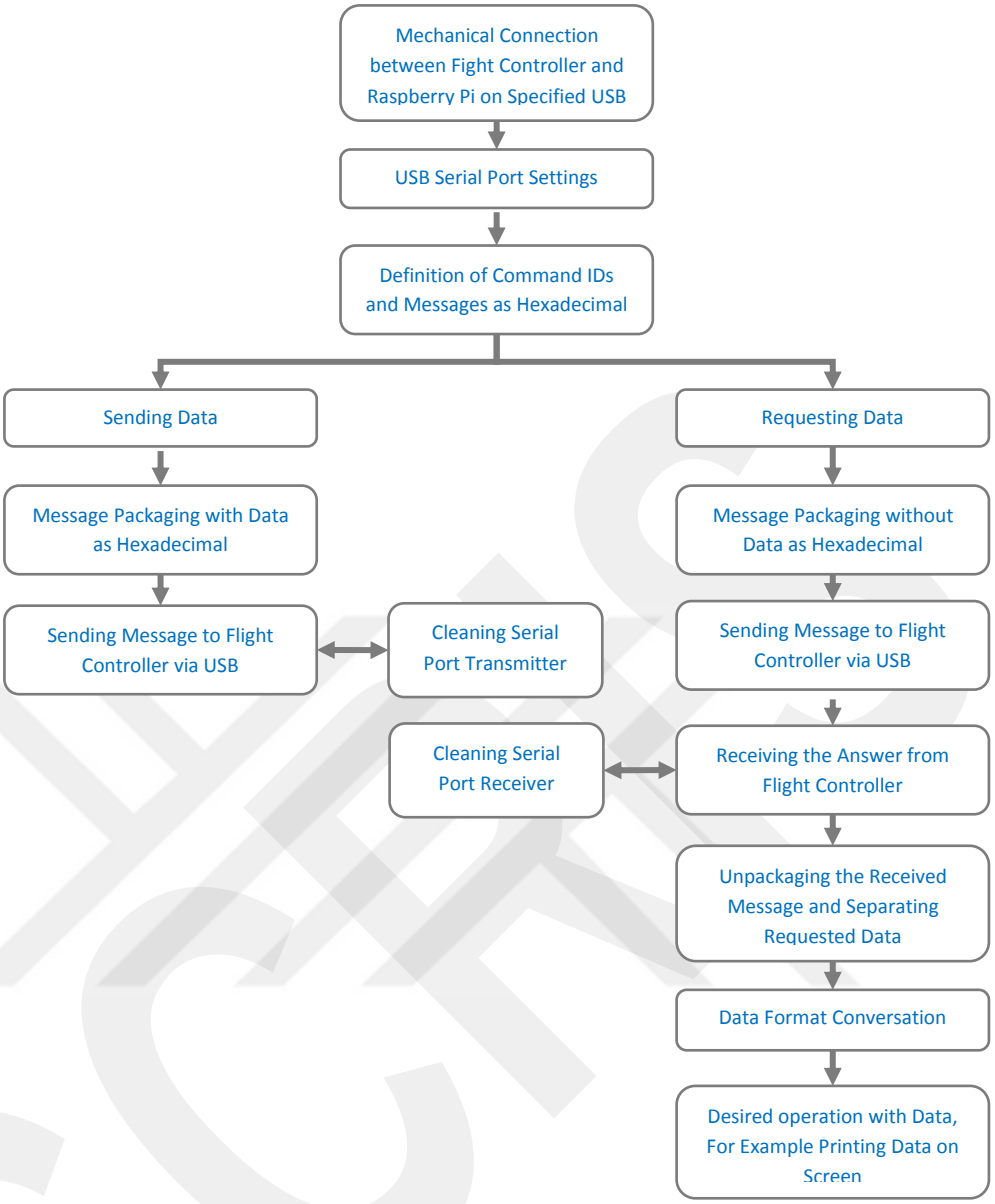


Figure 4.21 Raspberry Pi-Flight Controller Communication Flow Chart

As mentioned above in this section, the communication protocol was created as a Python code. The protocol was executed in real time after the flight controller was connected to one of the USB ports on the Raspberry Pi. In this way, the desired data from the flight controller can be obtained in the frequency range of 200-300 Hz. The Python script in which the attitude data is requested from the flight controller at 200 Hz and printed to the screen is given in Appendix B.

4.10 Raspberry Pi - PC - Matlab Communication

Raspberry Pi is a control card that has a powerful processor and has its own Linux operating system. This allows you to run more than one code script at the same time.

In addition, the serial communication pins provided thereon allow communication in accordance with different communication protocols. It has also hardware support by Matlab / Simulink. This provides the use of I / O and communication pins, remote wireless communication devices with Simulink blocks prepared for Matlab / Simulink. However, Maltab / Simulink contains some limits. One of these is that Simulink does not generally support hexadecimal. Another of these is that, no changes can be made in the content of the Raspberry Pi Simulink Blocks. In this case, it limits the user. Therefore, it is necessary to find the most appropriate solutions within the limits. To solve the lack of hexadecimal support, which is one of the problems, the simplest solution is not to send hexadecimal data to Matlab/Simulink. For this purpose, before each hexadecimal data was sent to Matlab, it was converted by the generated Python code script to "double" that is the format supported by Matlab. The problem of not making any changes in the Simulink blocks was solved by carrying out studies by remaining within the limits of the blocks.

The most challenging problem is the communication between Matlab / Simulink and the code running in Raspberry Pi. Since it is one of the most important issues in this study, it must be solved without error. This problem was solved by using a very simple method. The Raspberry Pi is included with a serial communication port for receiving and transmitting. Using this port, data exchange between Python Code running behind and Matlab / Simulink is provided. This method is done as follows; Python code, which communicates with flight control card via USB port, converts incoming data from hexadecimal to "double" format which can be easily distinguished by Matlab / Simulink. Each translated data is then packed into a vector. The packaged data is transmitted to the transmitter via serial communication pins located on the Raspberry pi. The data written on the transmitter is transmitted to the receiver by means of the serial communication pins of the receiver and transmitter that are connected to each other on hardware. Then the data written on the transmitter can be read by the receiver using the receiver block from the blocks prepared for serial communication pins from Raspberry Pi support blocks prepared by Matlab / Simulink. The data read from Matlab/Simulink can be opened and used in the desired process as packaged.

However, this method has a problem. Since the Raspberry Pi runs two codes (Python and Matlab/Simulink) at the same time and have their own loop frequencies, it is

impossible for them to work in a fully synchronized manner. Since the transmitted and read data cannot operate at the same frequencies, breaks occur in the data stream. This causes errors in the data that was reading on Maltab/Simulink. To solve this problem, the data was read, when the frequencies overlap. For this purpose, a marker was added to the packaged data vector. In the data package read via Matlab / Simulink, first check whether the marker is received or not. If the marker is read, then the data package is processed and used accordingly. The flow diagram of this process is given in Figure 4.22.

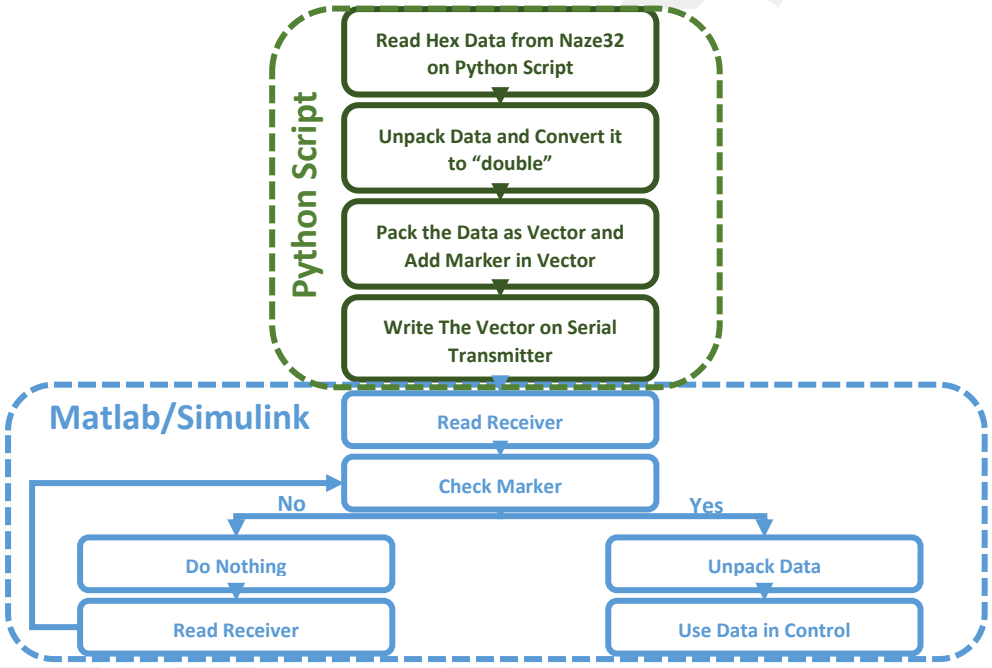


Figure 4.22 Data Transmitting-Receiving Flow Chart

In addition to the data exchange communication described in Figure 4.22, communication between the operating system in the Raspberry Pi and the PC is also required. There are 2 methods for this communication and the two methods are the same protocols. The first method of communication is to provide an ethernet connection between the PC and the home Raspberry Pi. After the connection is established, the net address of the ethernet connection can be accessed by typing "netstat -a" on the command screen of the PC. The link is also called Raspberry Pi and is easy to find. An example of the command Windows's response to the given command "netstat -a" in Figure 4.23.

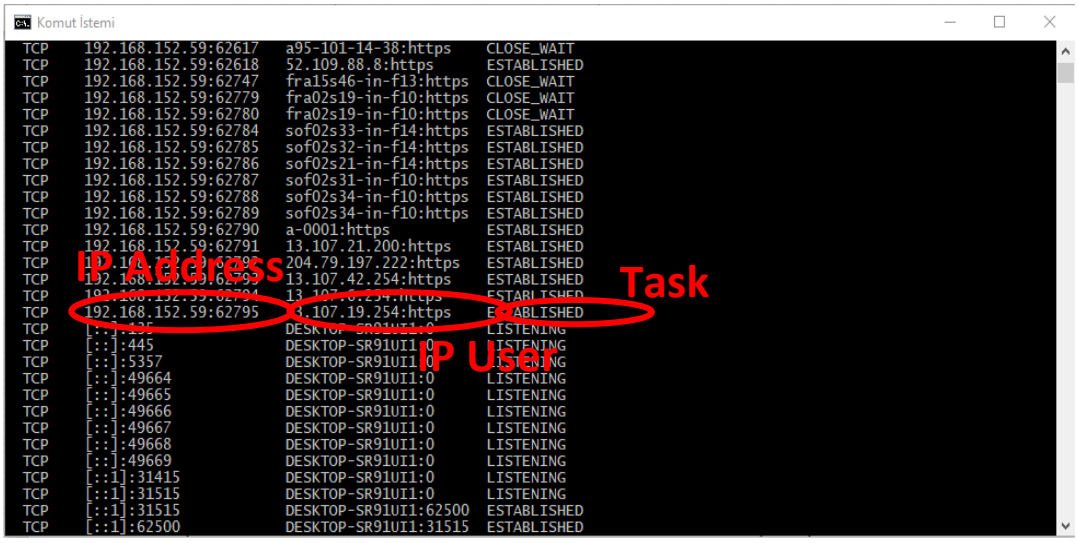


Figure 4.23 PC Command Windows "netstat -a" Answer

Once the IP address has been defined, there are 2 ways to access Raspberry Pi. The first method provides access to the Linux Command Window, which can be used for Linux Shell programs such as Putty or Solar Putty. If the connection is not broken after entering the specified IP address, it can be connected to Raspberry Pi by using shell software. The other method is an image-based method. In this method, VNC type communication can be used to reach Raspberry Pi. It can be connected to Raspberry Pi via operating system screen as if creating a virtual computer. For this method, VNC Connect software is used. In this software, you only need to enter the specified IP address correctly.

However, it was not possible to make a cable connection because the system was a quadcopter. Therefore, two communication methods, wireless communication, were used for communication between PC and Raspberry Pi. A rooter was used for this method. After the PC and Raspberry Pi are connected wirelessly to rooter, to reach advance setting of the rooter a specific IP address can be used. The IP address that Raspberry Pi took on the rooter can be reachable from the roooter advanced settings page. Using this IP address, the shell connection or image-based connection described above can be easily achieved. The IP address that Raspberry Pi leased over a rooter is shown in Figure 4.24.

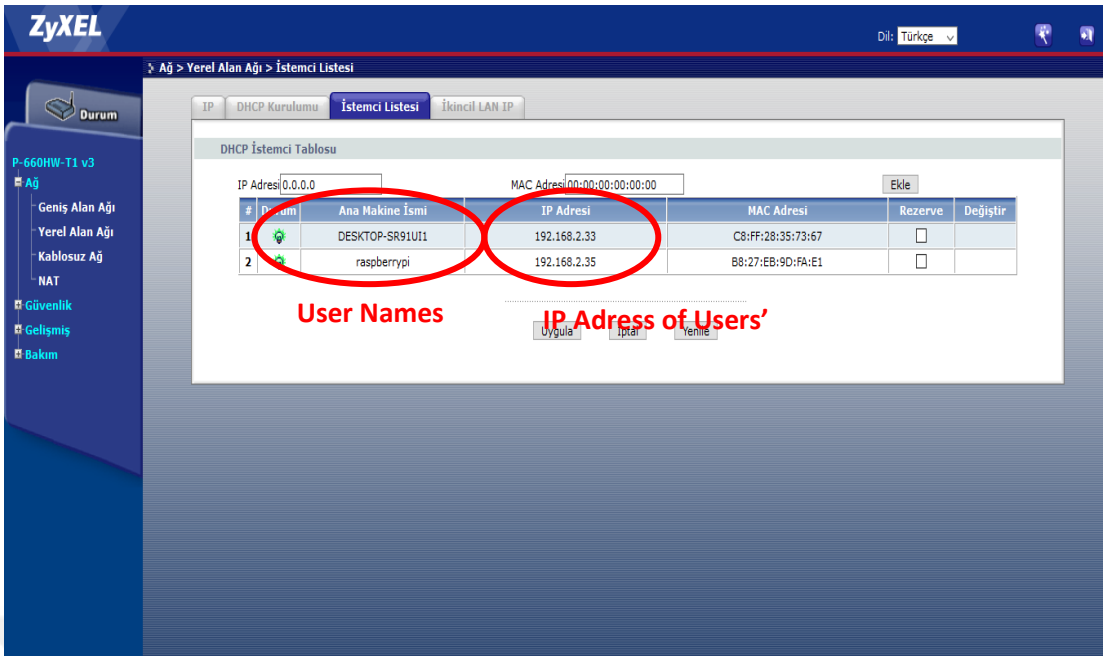


Figure 4.24 Rooter Advanced Settings Page

The following diagram illustrates all communication processes described above is shown in Figure 4.25.

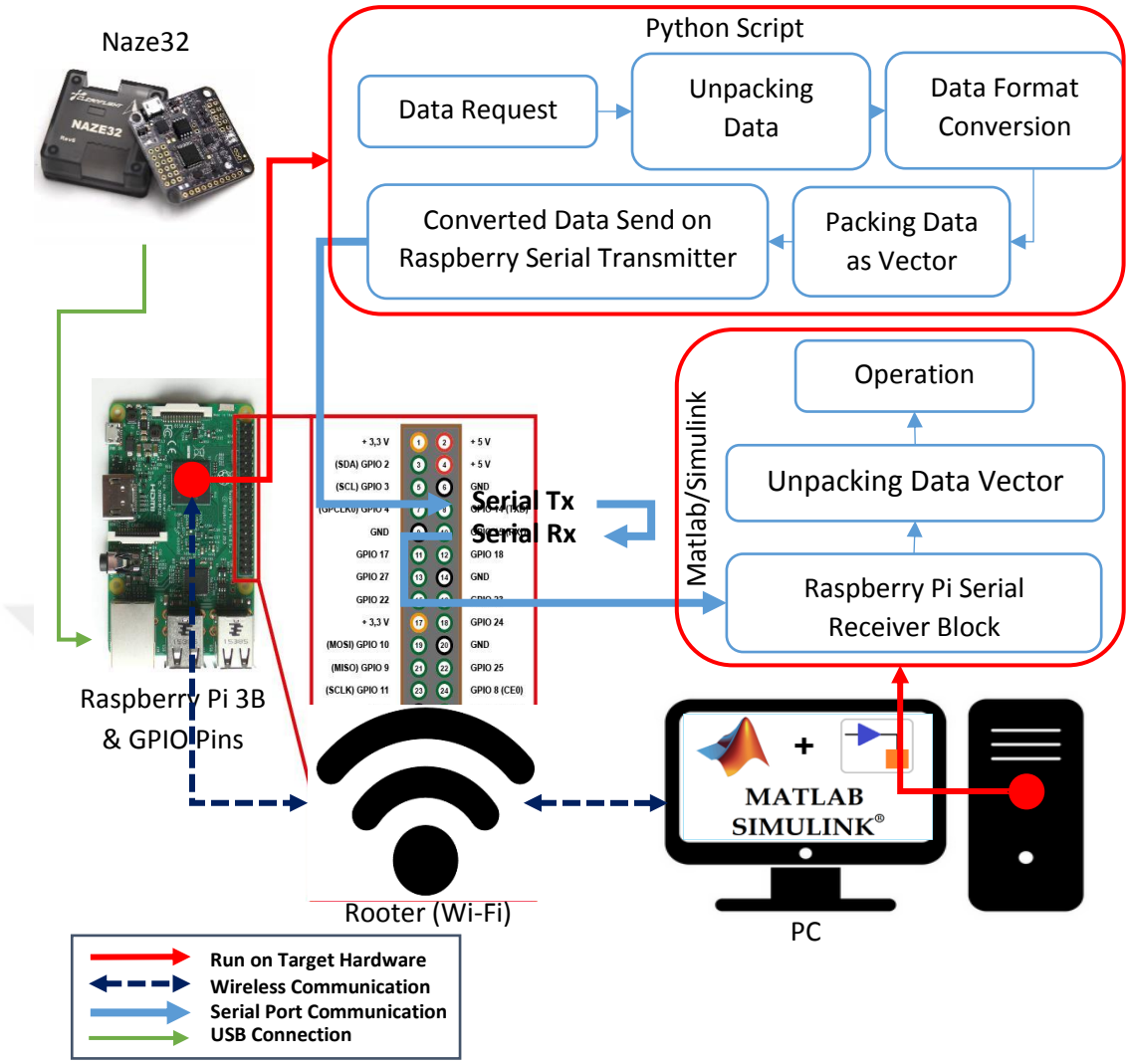
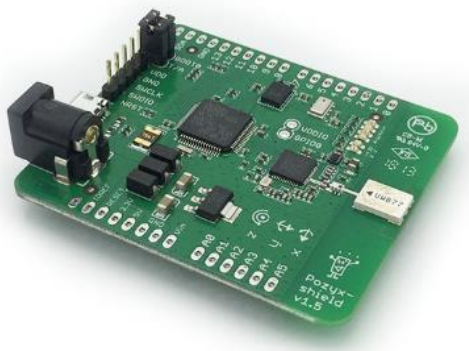


Figure 4.25 Raspberry Pi-Matlab/Simulink Communication

4.11 Indoor Localization Hardware (Pozyx) and Data Acquisition

The use of indoor locating system is of great importance for advanced tasks. Position control allows the system to operate completely autonomously and to complete more sensitive tasks. There are several methods for locating a quadcopter for indoor and outdoor uses. Since the tasks performed in this study are designed for indoor applications, the position control solution is based on the indoor localization system. The indoor localization hardware used in the study is a system called Pozyx using Ultra-Wideband (UWB) technology with filtering and machine learning methods. In this system, 3-dimensional position data can be collected with 10 cm error.

In order to obtain position data, 4 anchors are mounted on the walls to cover the flight area. Using the UWB technology, the system communicates with the anchors mounted with the tag on the quadcopter. Thus, the position data of the tag and quadcopter can be obtained with a 10 cm error. Photos of the tag and anchor are given in Figure 4.26.



(a)

(b)

Figure 4.26 Pozyx Indoor Localization System (a) Pozyx Tag (b) Pozyx Anchor [29]

One of the most important reasons for using this system in the study is that the system supports Python based hardware. This allows communication with a Python code written in the Raspberry Pi with the Pozyx Tags' via USB connection. The manufacturer's Python library makes easy to data exchange from sensors and anchors on the localization system. At the same time, Anchor positions and other device features can be easily integrated into the system using a library. In simple steps, the use of the indoor localization system with Raspberry Pi is as follows;

- Once all the anchors have been fixed on walls, Pozyx tag is connected to the Raspberry Pi via USB connection.
- The USB connection should be available to Linux users with all privileges. Therefore, the permission command for the USB port to which Tag is connected is written to the Linux terminal as “sudo chmod 666 /dev/ttyACM0”. This command opens all permissions for USB.
- Then the necessary libraries should be loaded. To do this, the following is written on the terminal screen;

✓ pip install pypozyx
✓ pip3 install pypozyx
✓ git clone https://github.com/pozyxLabs/Pozyx-Python-library
✓ cd <PozyxFolderAdress>
✓ python setup.py install

- If the USB port to which the Tag is connected is not known, the port can be learned by running following code as a Python Script;

```
from pypozyx import PozyxSerial, get_first_pozyx_serial_port
serial_port = get_first_pozyx_serial_port()
    if serial_port is not None:
pozyx = PozyxSerial(serial_port)
print("Connection success!")
else:
print("No Pozyx port was found")
```

If “serialport” is printed on the screen, it will show the connected port.

- Then, the required data can be received at 1Khz speed via the sensors on the Tag. For example, to learn angular changes;

```
euler_angles = EulerAngles()
pozyx.getEulerAngles_deg(euler_angles)
```

can be written.

- In order to receive position data, additional hardware features and position data of the anchors must be saved in the system. The following codes can be used for this operation;

- from pypozyx import ..., Coordinates, DeviceCoordinates
  anchor = DeviceCoordinates(AnchorID), 0, Coordinates(X, Y, Z))
  pozyx.addDevice(anchor)
  pozyx.saveNetwork()

➢ Once the necessary hardware settings have been made, the system is ready to transmit the position data. To do so, use the following code;

- position = Coordinates()
  pozyx.doPositioning(position)

  When the above steps are followed correctly, by connecting the Pozyx Tag hardware to Raspberry Pi via USB, data like angular velocity, acceleration and Euler angles as well as the position data of the system can be easily obtained.

  In addition, the coordinates of other anchors must be saved on the anchor. Although it does not contain as many operations as Tag settings mentioned in this section, but it is a must be done. After each anchor is connected to Raspberry Pi via USB, it must be saved on the coordinates of the other anchors. The following codes are used for this,

➢ For delete the all saved devices before studies;

- pozyx.clearDevices()

➢ For saving other 3 anchors' coordinates;

- from pypozyx import ..., Coordinates, DeviceCoordinates
  anchor1 = DeviceCoordinates(AnchorID), 0, Coordinates(X, Y, Z))
  pozyx.addDevice(anchor1)
  pozyx.saveNetwork()

  anchor2 = DeviceCoordinates(AnchorID), 0, Coordinates(X, Y, Z))
  pozyx.addDevice(anchor2)
  pozyx.saveNetwork()

  anchor3 = DeviceCoordinates(AnchorID), 0, Coordinates(X, Y, Z))
  pozyx.addDevice(anchor3)
  pozyx.saveNetwork()

After this operation, all the anchors would be ready for correct positioning. However, communication speed, data accuracy, data filtering should be set on each anchors and tags. These settings are of paramount importance for the indoor localization system positioning with the UWB method. If it is done incorrectly or differently for each anchor/tag, problems such as breaks in incoming data, incorrect values or no data can be encountered. These settings do not require a Raspberry Pi connection or any Python script. All the settings can be made with a simpler method. For this purpose, "Pozyx Device Configurator", which is the hardware interface of the UWB localization manufacturer, can be used. The interface can be set up by connecting the devices to a PC with USB. The area of the positioning has a great importance to making these settings. The drawing of the area used in this study is given in Figure 4.27.

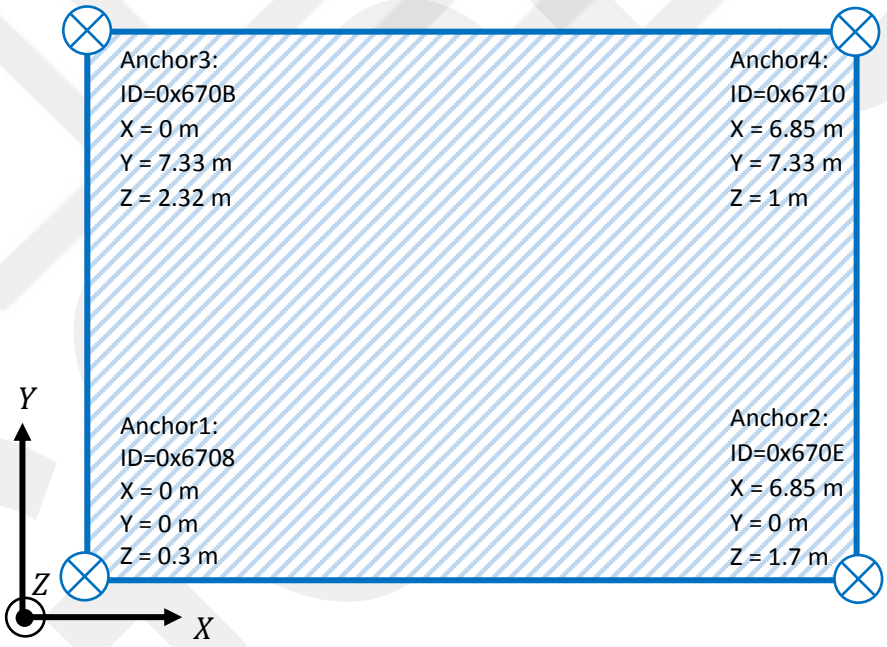


Figure 4.27 Flight Area and Anchor Locations

For the Pozyx UWB localization system, which we use on an average area of 50 square meters as in the study, all components are communicated via channel 2. This is the recommended channel by the manufacturer for small areas. The whole system should operate at 64 preamble, 865 Kbps data rate, 64 PRF and 11.5 dB. Apart from all these settings, there is a setting for the filtering of all read data from the tag. Filtering options such as FIR, Average, and Median are among the options for the tag. However, due to the nature of the filters, there should be a delay in the incoming

data. The ideal settings for the area in operation on the Pozyx interface is given in Figure 4.28.

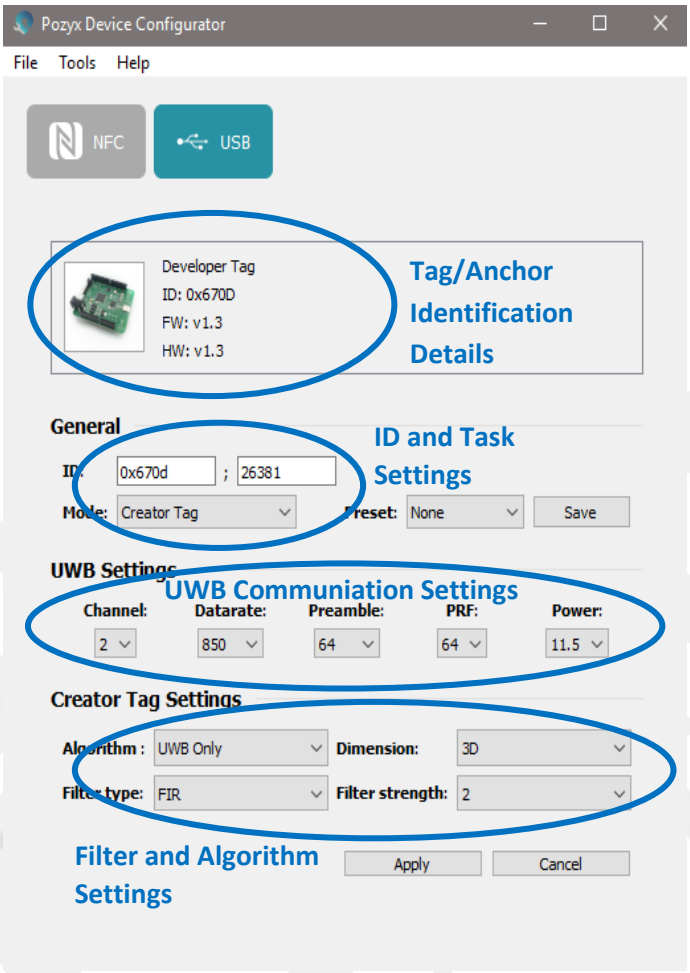


Figure 4.28 Ideal Pozyx Setting on Interface

4.12 Raspberry Pi Manipulator Control

The simulations described in the earlier sections of the study contain details of several advanced control algorithms for the manipulator. One of the important part of the study is that these control algorithms designed in the simulation can be used in the real system tests. However, such advanced control algorithms contain many complex equations and multi-layered control structures. This makes it difficult to use these control algorithms in real systems. However, the Matlab/Simulink environment, in which simulations are performed, allows such control structures to be designed much more easily. In this way, these control algorithms can be designed without being written in C, C++ or Python code form.

For these reasons and for many of the aforementioned reasons, Raspberry Pi was used as the control card of the system. Thanks to the Raspberry Pi, which is supported by MatLab/Simulink, the control structures used in simulations can be easily embedded in the system and run in real time. As described in the previous sections, the required sensor data can be obtained and used in real time in the Simulink environment. In addition, the control structures designed in the simulation can be used in Simulink Model as well. In addition, the servo motor used for the manipulator must be controlled with the Raspberry Pi. The servo motor used in the study uses PWM duty signal for control.

All digital pins on the Raspberry Pi are able to generate PWM signals in real time and in the specified frequency range. The percentage range of the generated signal is set in the range of 0-1. In other words, it generates 0% duty when 0 signal is sent and 100% duty when it is sent 1 as signal from specified digital pin.

Like most servo and other motors on the market, the servo motor used in operation is controlled by PWM signals in the range of 1 millisecond (%0 Duty) and 2 milliseconds (%100 duty). Therefore, the output of the control structure must be adjusted according to the frequency of the PWM signal sent. An example of the Matlab / Simulink model used is given in Figure 4.29.

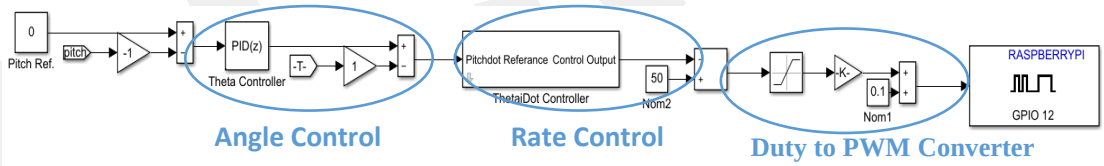


Figure 4.29 Manipulator Control Structure in Matlab/Simuklink

4.13 Autonomous Position Control and Design Details

In this section, the quadcopter is tested by the autonomous flight tests using the indoor localization system. Before starting these tests, some necessary procedures must be followed and the algorithm designed accordingly. The most important of these is how to use the reference values generated by the control algorithm. Because the control structure designed in MatLab/Simulink environment, control references should be sent to the flight controller. To solve this problem, using the RC controller references of the flight controller is the simplest solution. As with most flight

controllers, the Naze32 card used in the study can read many different RC controller references. These include classical references such as SBUS, IBUS and PPM, as well as specific communication methods of more specific RC controllers. The simplest of the methods is to read the PWM signal from different channels for each reference (Roll, Pitch, Yaw, and Throttle). The pins where these channels are read on the controller card are given in Figure 4.30.

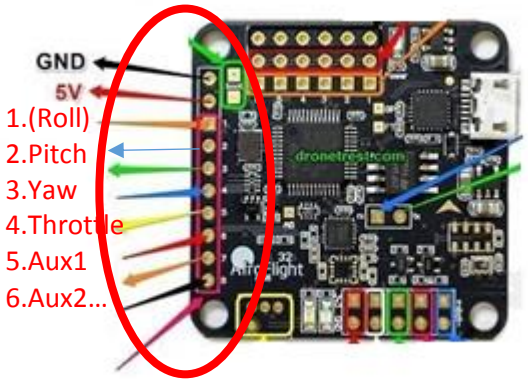


Figure 4.30 RC Command Referance Pins of Naze3[26]

In this way, the references generated from the controllers can be easily converted to PWM and sent to the flight controller via Raspberry Pi. This includes the reference required to arm the system. This reference can be sent as PWM to one of the AUX channels in the same pin layout. However, this must already be set to an AUX channel on Cleanflight Mode Settings.

In addition, the throttle reference required for the hover of quadcopter should not be given as a step reference. This is because the flight controller cannot follow the sudden step reference. Therefore the given reference should be increased slowly by a transfer function. The reference graph formed by the transfer function used for this purpose in the algorithm is given in Figure 4.31.

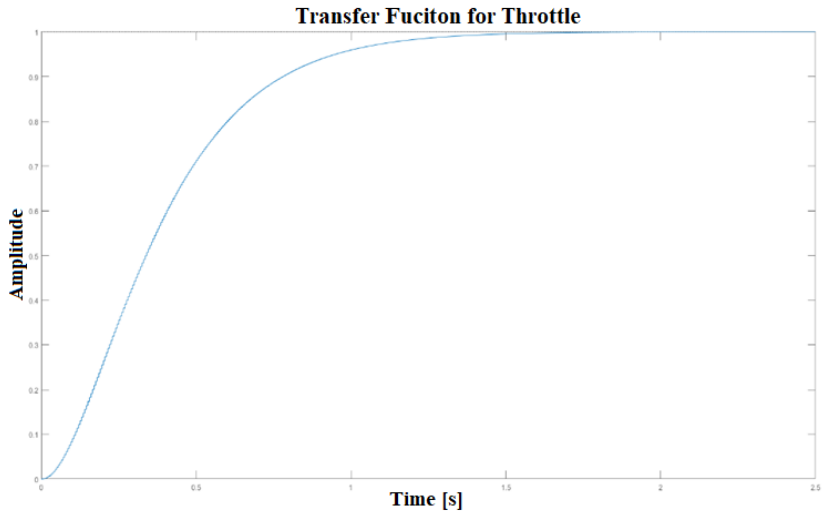


Figure 4.31 Transfer Function Graph of Levitate

Another important procedure that must be followed before an autonomous flight is that the controllers should not read data from the sensors until the system is armed. This counts away from possible accumulation of errors. This precaution should be taken in order to prevent the first hovering of system that is already difficult with the accumulated errors. In order to achieve this, the algorithm is designed using a simple switching method. This gives the controllers "0" instead of sensor data until the system is armed. The designed switching structure is given in Figure 4.32.

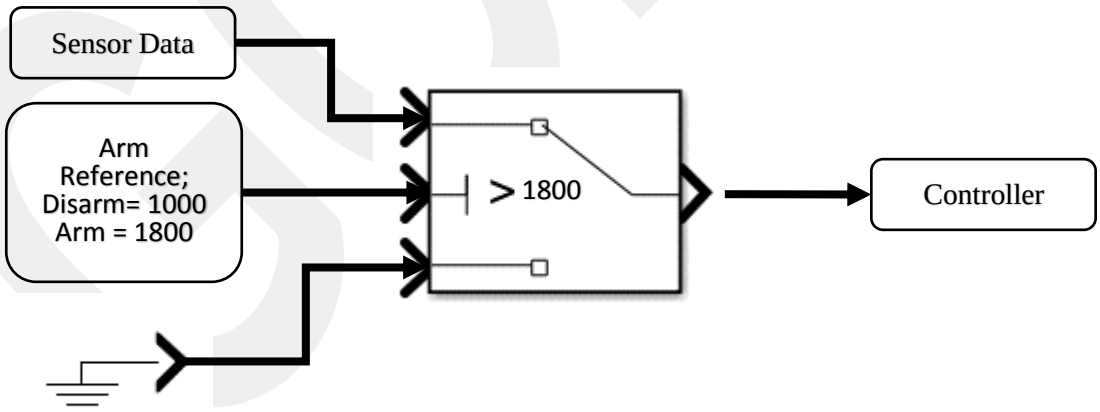


Figure 4.32 Arm Sensor Switching Structure Design

In addition, bias in the position data must be corrected. This may be negligible for the X and Y position, but must be done for the Z plane. The height of the quadcopter itself, the height of the platform on which it is located (eg. sponge) must be

subtracted from the measured value to avoid any initial error. For this, 3 seconds are not taken off and the incoming sensor values are read. The readings are averaged and subtracted from the sensor data. The structure of the algorithm is given in Figure 4.33.

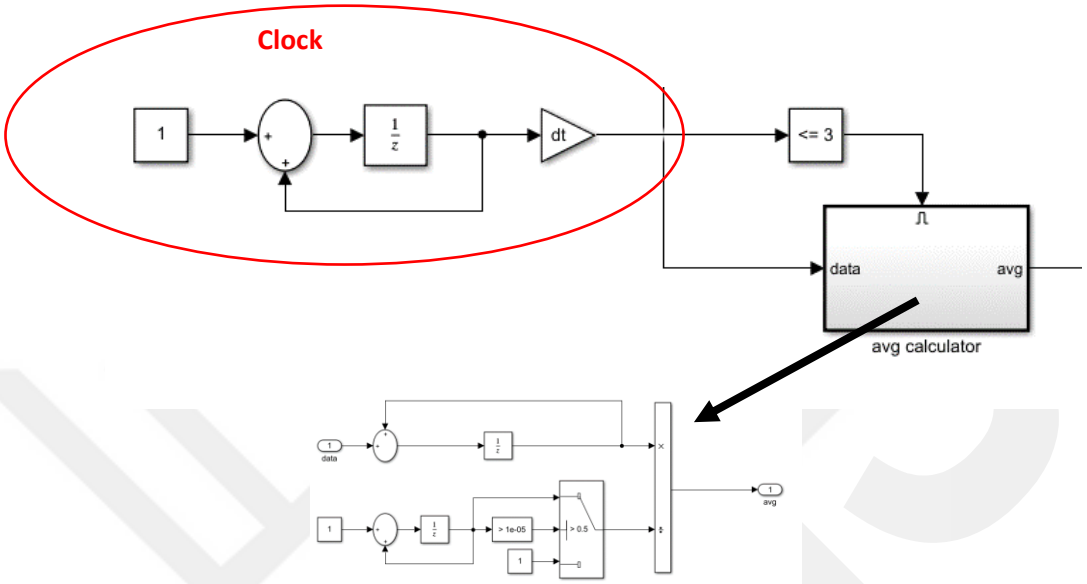


Figure 4.33 Bias Average Model Structure

Finally, the position references to the controllers must be in accordance with the system dynamics. This is directly proportional to how long the system will travel to the position it will go to. The position reference to be given is given to the controllers using a transfer function determined by considering the system dynamics. A reference is generated made to make the system travel with the velocity of approximately 30 centimeters per second. The given reference can be started at any time while flight.

The cascade PID type controller was designed to perform 3D autonomous position control according to the position references. The cascaded structure consists of 3 loops. The outer loop takes the position reference given to the system and compares it with the actual position data to form a reference angle for inner loop. In the middle loop, it compares the desired reference angle with the Euler angles of the actual system, and provides the angular velocity reference required for control. In the inner loop, the obtained angular speed reference is compared with the actual angular speed values of the system and generates the PWM values required for the motors. Through this control structure, the system can go to the desired X, Y and Z references from

where it is located, as designed. XY position controller structure is given in Figure 4.34.

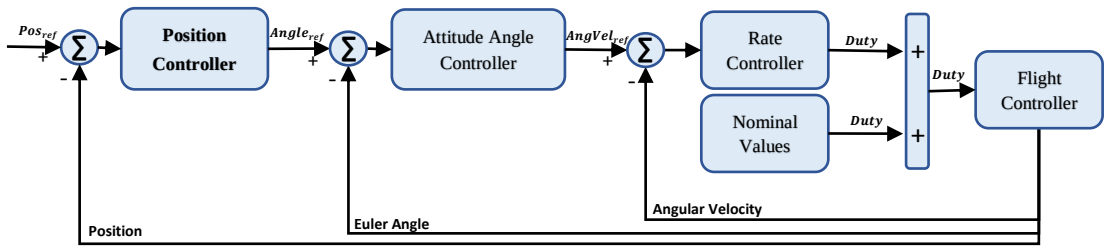


Figure 4.34 Position Controller Cascaded Structure

However, the Z position control is completely different from the XY control structure. Z control is only performed with a single PID controller via the received altitude value from the indoor localization system. The most important point here is to give the nominal PWM/Duty that the system is hovering accurately and slowly. The system generates more thrust with the boost from ground when taking off. Therefore, if the nominal value required to hover higher is given while takin off, it will exceed the desired height and overshoot. Therefore, it makes large oscillations up to the desired altitude. To solve this problem, the nominal value given to the system for hover should be given by increasing the system after taking off. Altitude control structure is given in Figure 4.35.

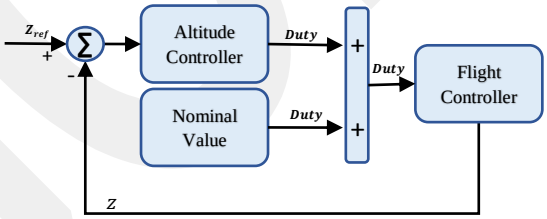


Figure 4.35 Altitude Controller Structure

CHAPTER 5

5. EXPERIMENTS

This section provides performance examinations while connected to the test platform of the system and adhering to the scenarios identified during the flights. The tests are carried out on the Quadcopter structure described in Chapter 4. The images of the test system of Frame 1, which are connected to the platform and taken during flight, are given Figure 5.1.

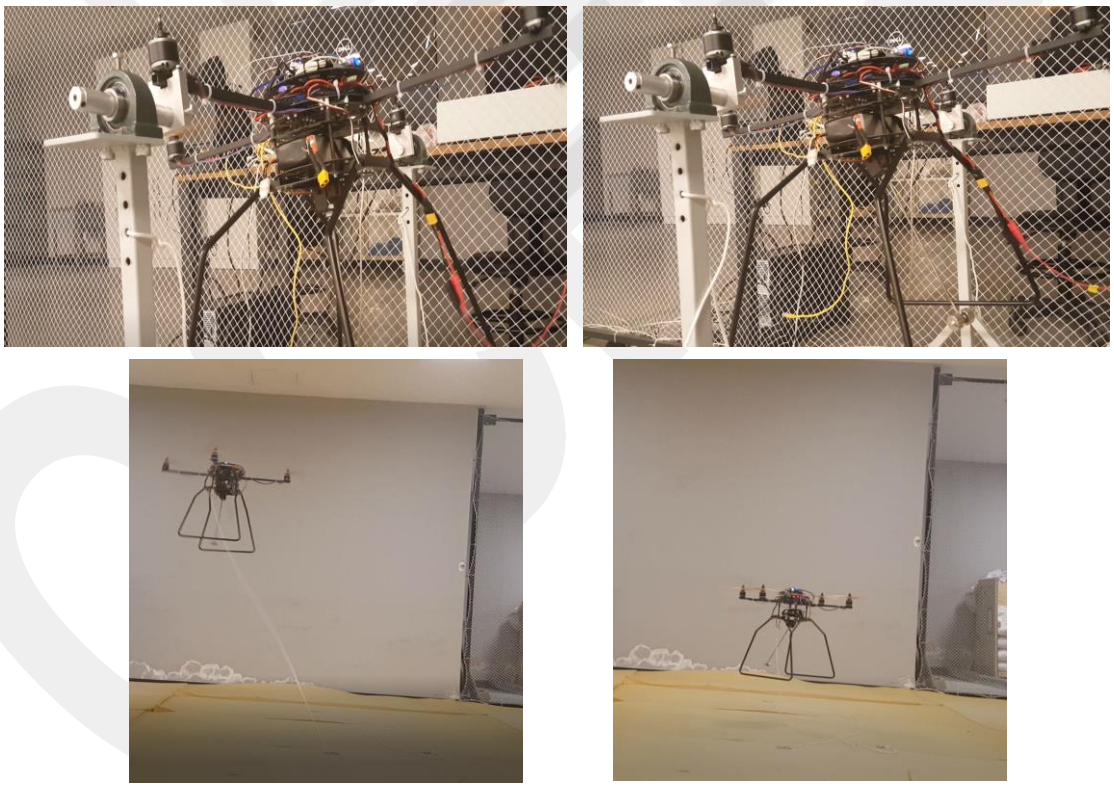


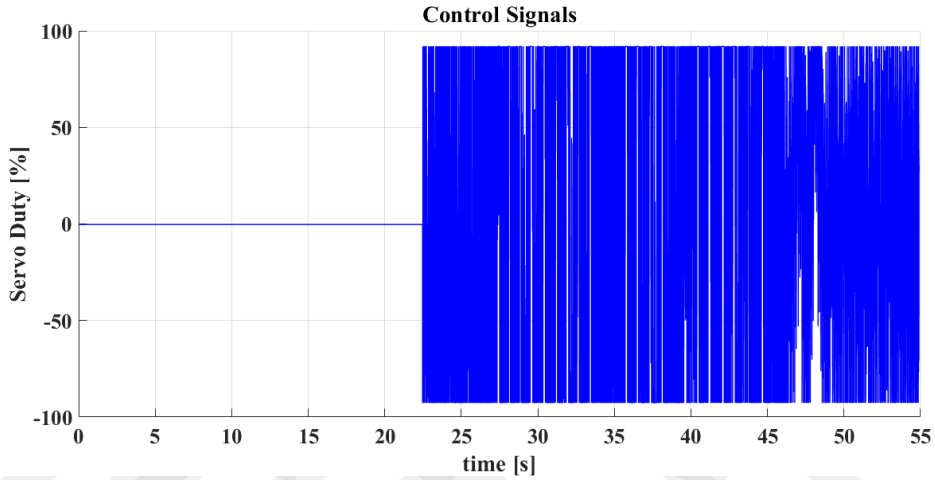
Figure 5.1 System during Tests

The control algorithms applied in the simulationss are tested while the system is connected to the test platform and during the flight. The tests are conducted in accordance with the selected scenarios identified. The tests are described in the following sections.

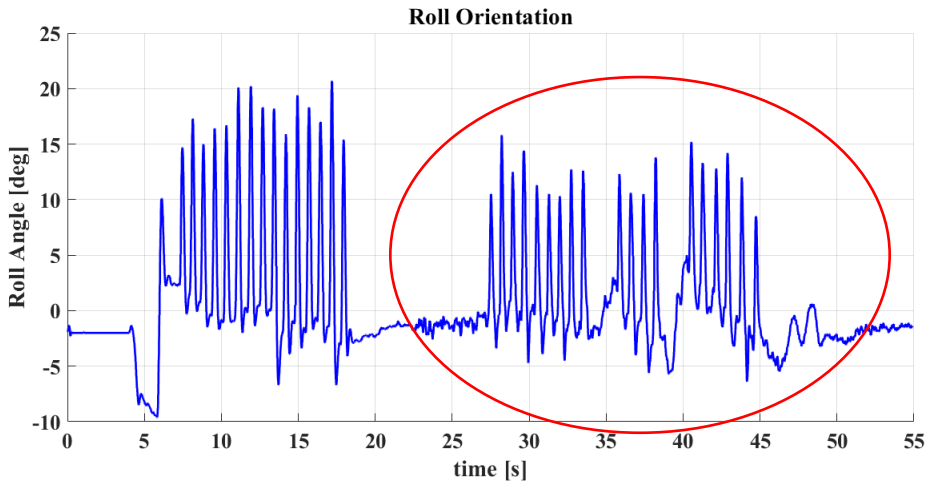
5.1 Flight Assistant Scenario

5.1.1 Disturbance Rejection Test while Quadcopter Connected to Test Bench

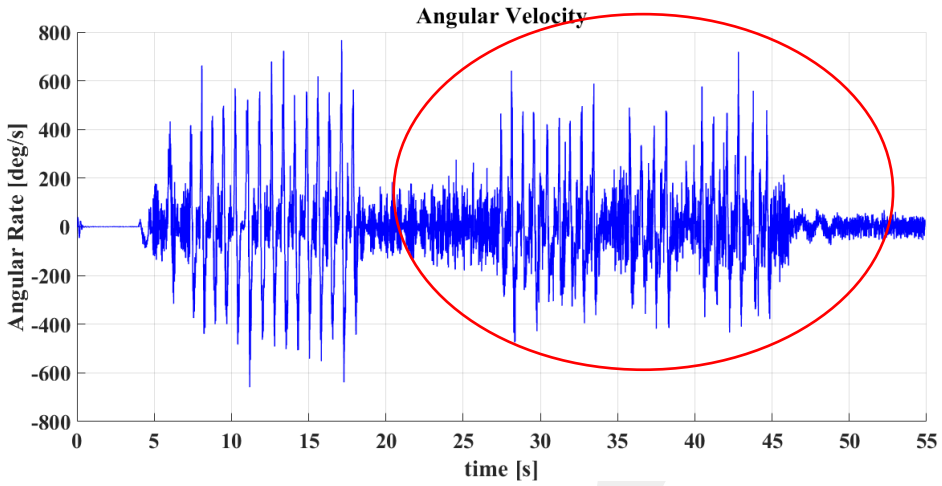
In this section, the simple cascade PID control structure designed for manipulator is tested. Angular velocity and orientation data were obtained in real time from the flight control card with Raspberry on MatLab. The performance graphs obtained while connected to the test platform of the system are given in Figure 5.2.



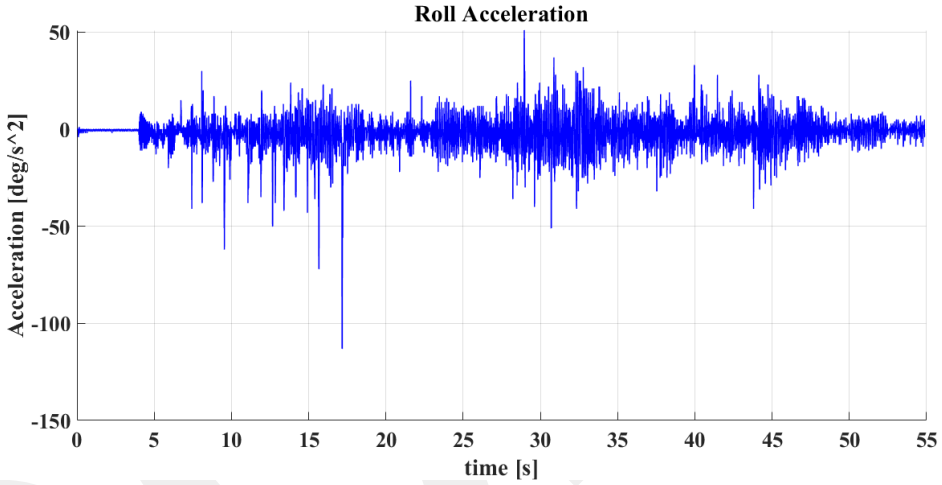
(a)



(b)



(c)



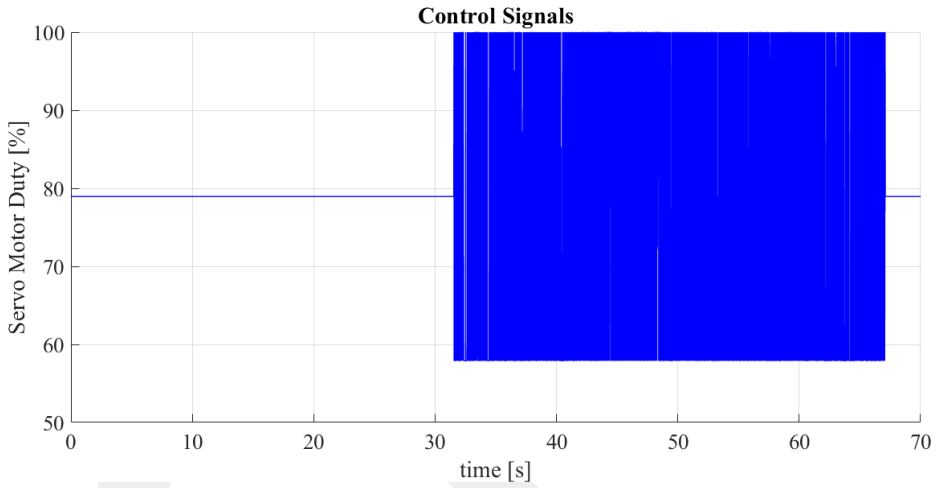
(d)

Figure 5.2 System Performance with Casceded PID on Test Platform ((a)Servo Control Duty (b)System Orientation (c) Angular Velocity (d) Acceleration)

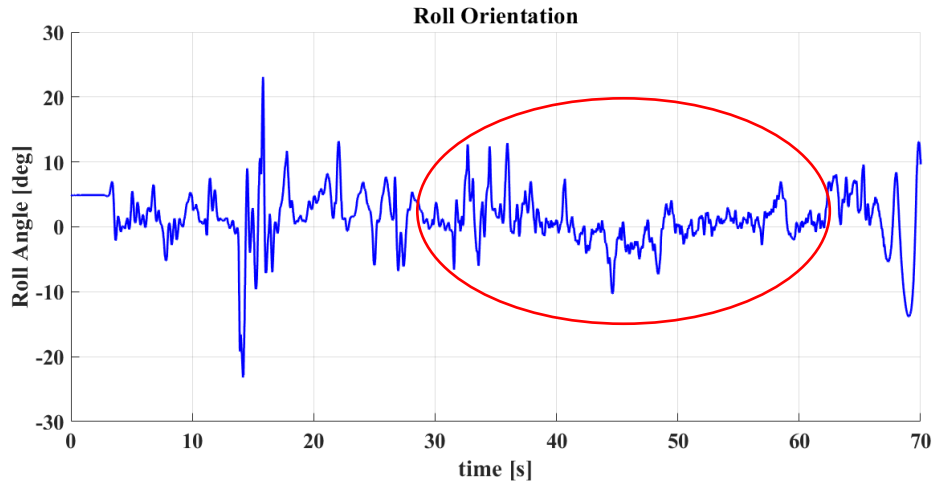
The results obtained while connected to the test platform show that there has been a decrease in angular velocity and orientation after the link has been activated. In the graphs, the red circles indicate the region where the manipulator is active. There was a 5-6 degree decrease in orientation. It is seen in the decrease in the angular velocity of the system. In this case, it shows that the disturbances acting on the system can be rejected more easily with the help of the manipulator.

5.1.2 Disturbance Rejection Test while Flight

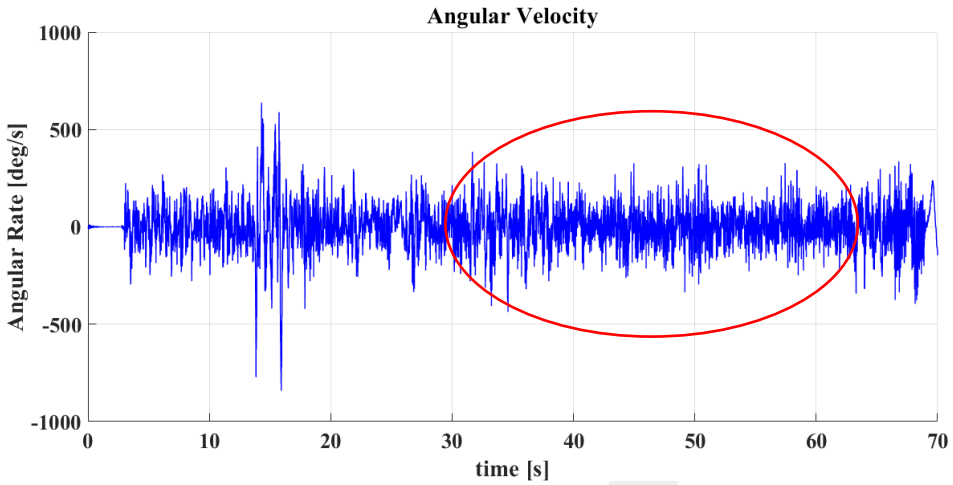
In this section, also the simple cascaded PID control structure designed for manipulator is tested as in the previous section. Angular velocity and orientation data are obtained in real time from the flight control card with Raspberry on MatLab, as well. Considering the resistance generated by the test platform in previous section, the effect of the manipulator is too much in disturbance rejection. This effect is seen more clearly during the flight. The performance graphs obtained from flight tests are given in Figure 5.3.



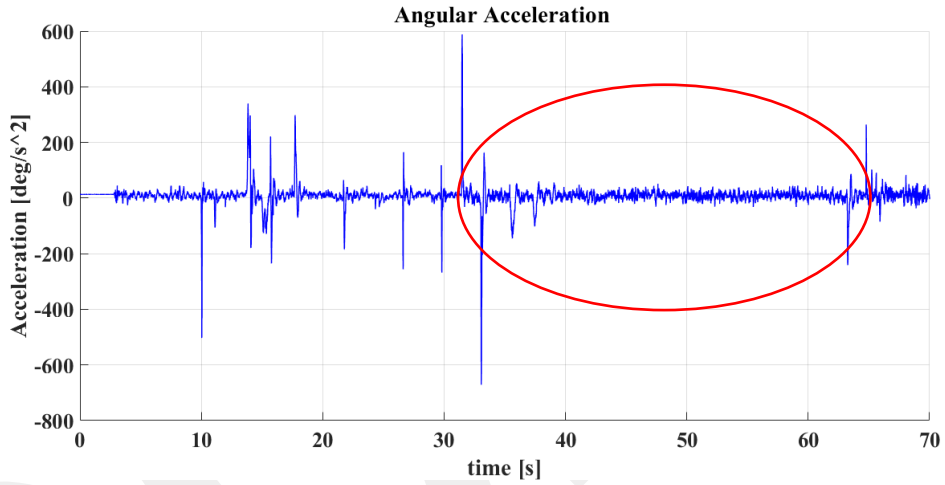
(a)



(b)



(c)



(d)

Figure 5.3 System Performance with Casceded PID during Flight ((a)Servo Control Duty (b)System Orientation (c) Angular Velocity (d) Acceleration)

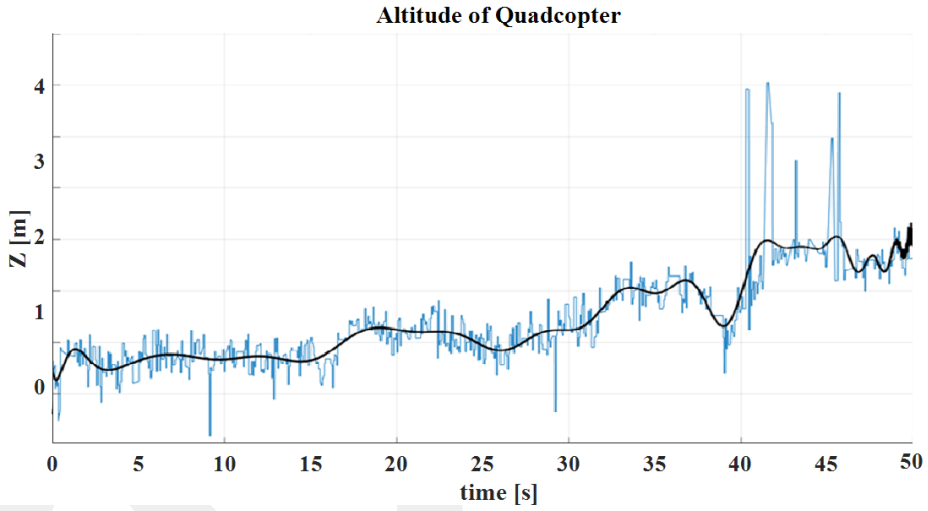
The performance evaluation from the test results indicates that the system is more stable when the manipulator is active. The sections indicated by the red circle in the graphs represent the time interval at which the manipulator is active. The results showed that the system is less affected by disturbances. The magnitudes of angular velocity changes decreased and the number of impulsive reactions in the accelerations decreased. These results showed that the system is more resistant to disturbances when the manipulator is active.

5.2 Autonomous Position Control Tests

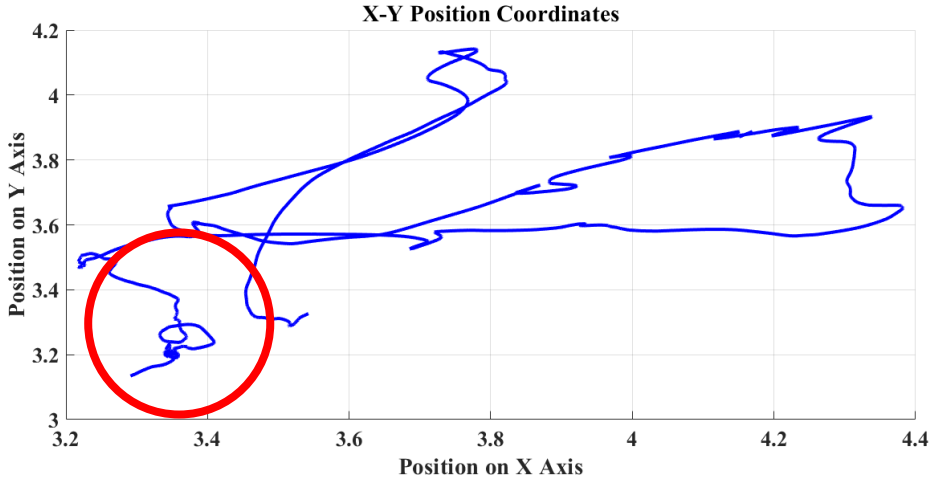
Autonomous position control tests and their results will be discussed in this section. For the autonomous flight, two important tests were performed. The first one was referenced to maintain its position after take-off it means hover test. The other was refer to the position reference given after take-off.

5.2.1 Hover Tests

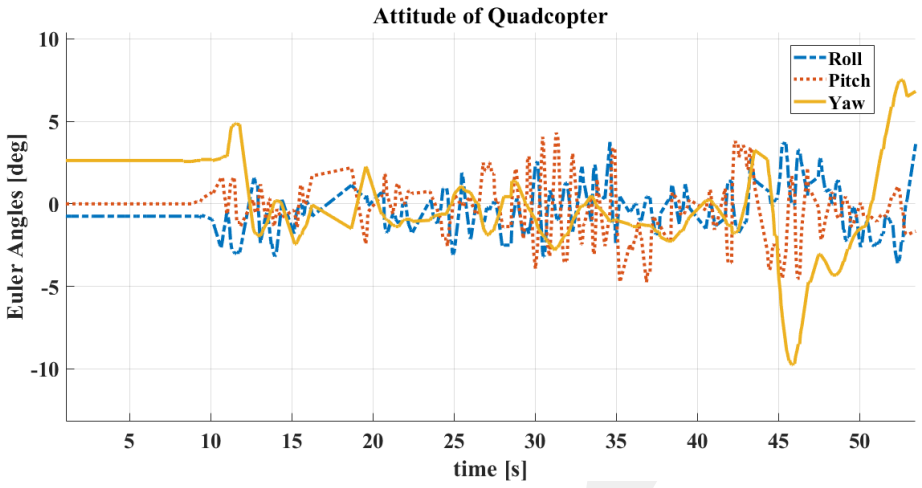
The Hover test 1 is referred to as a gradual increase in height after the quadcopter is taken off from ground. It was expected to stand in the air hover position at the desired height. The test results are as follows.



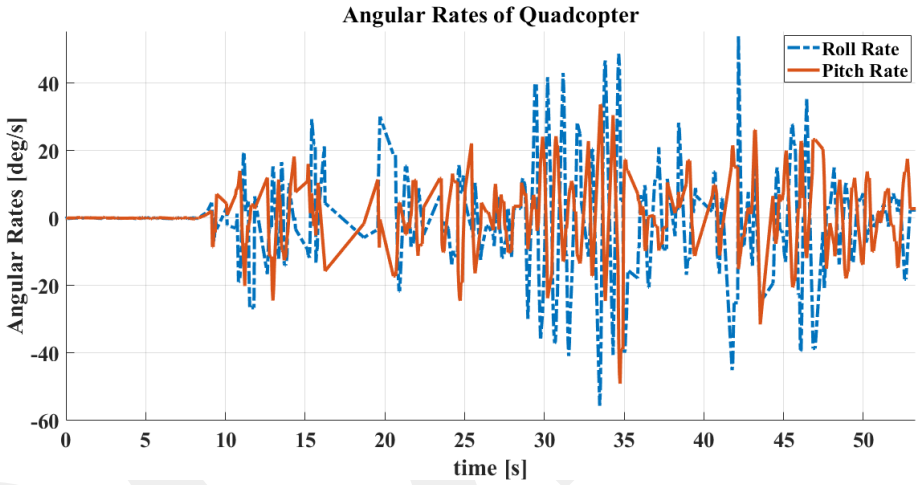
(a)



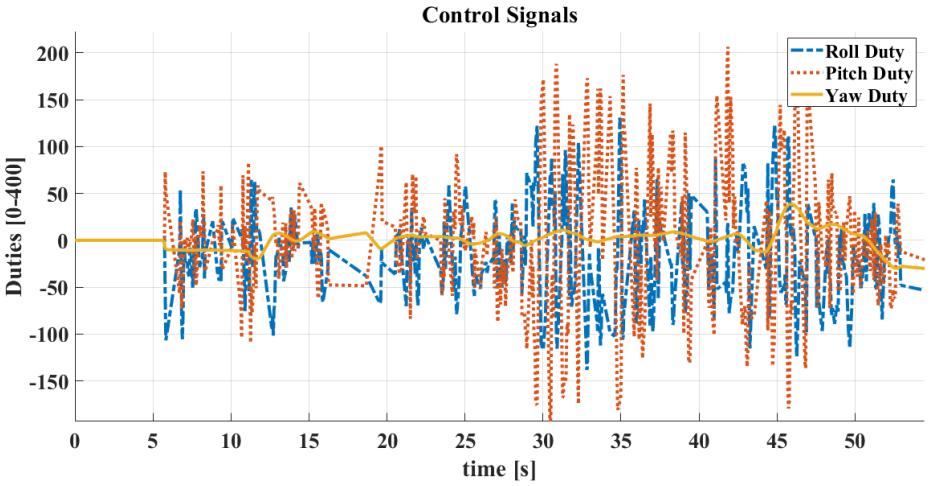
(b)


Attitude of Quadcopter
Roll
Pitch
Yaw
Euler Angles [deg]
time [s]

(c)


Angular Rates of Quadcopter
Roll Rate
Pitch Rate
Angular Rates [deg/s]
time [s]

(d)

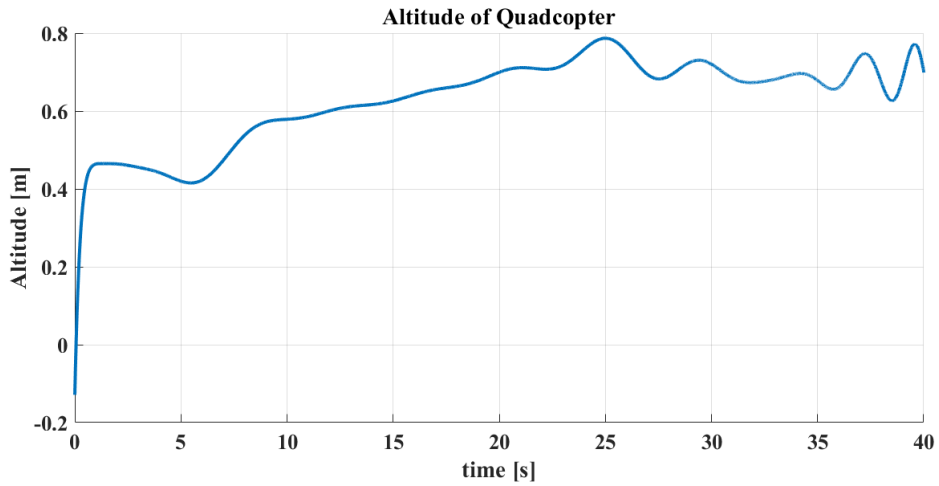

Control Signals
Roll Duty
Pitch Duty
Yaw Duty
Duties [0-400]
time [s]

(e)

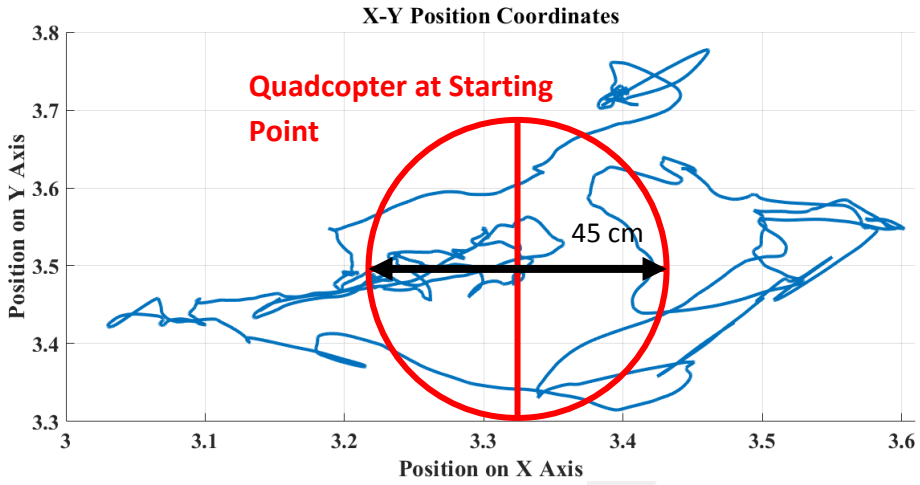
Figure 5.4 Hover Test 1 Results (a)Altitude (b) X-Y Trajectory (c) Attitude (d) Attitude Rates (e) Control Outputs

The results in Figure 5.4 show that the system remained within 0-5 degrees with slow movements with a movement of 0-30 deg/s. In this way, quadcopter tried to maintain the given starting position as desired. On the other hand, it has been kept at constant heights for a certain period of time by increasing it steadily. However, when X-Y trajectory is observed, deviations from the desired position can be seen. These deviations are caused by disturbances that occur during the first take-off from the ground. The friction in the ground causes the system to degrade 5-10 degrees on the yaw axis. In this case, the system leads to the wrong position after takeoff. However, quadcopter can easily return to the point where it started and completed his flight with a 10-20 cm error from the desired reference. The red circle on the X-Y trajectory graph indicates the desired reference point.

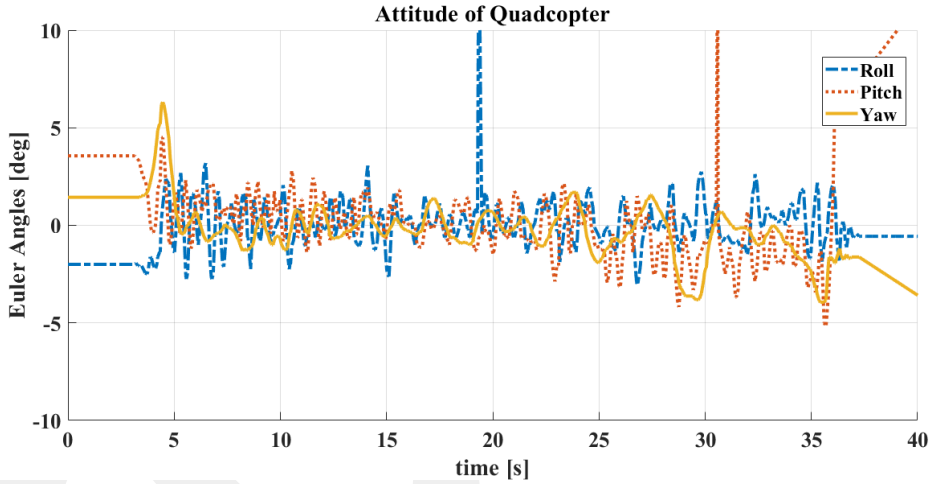
To solve the take-off problems in Test 1, the rate loop of the cascade controller was designed agile. This will increase the resistance of the system to disturbances like takeoff of landing. One of the problems of this method is that more control outputs will be produced by the controller. This causes the system battery to run out more quickly. However, the problem can be ignored since the lost flight time is not very much. System responses obtained from the test with the adjusted controller are given in Figure 5.5.



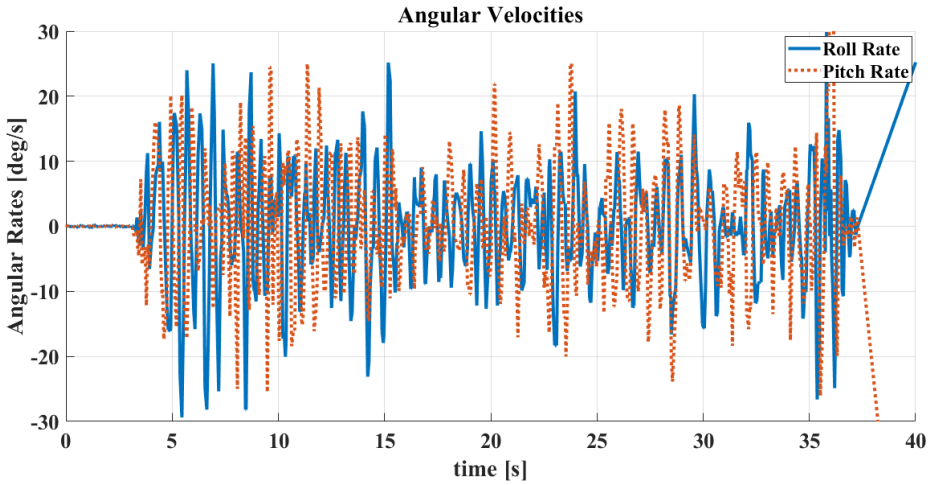
(a)


X-Y Position Coordinates
Quadcopter at Starting Point
45 cm
Position on Y Axis
Position on X Axis
3.8
3.7
3.6
3.5
3.4
3.3
3
3.1
3.2
3.3
3.4
3.5
3.6

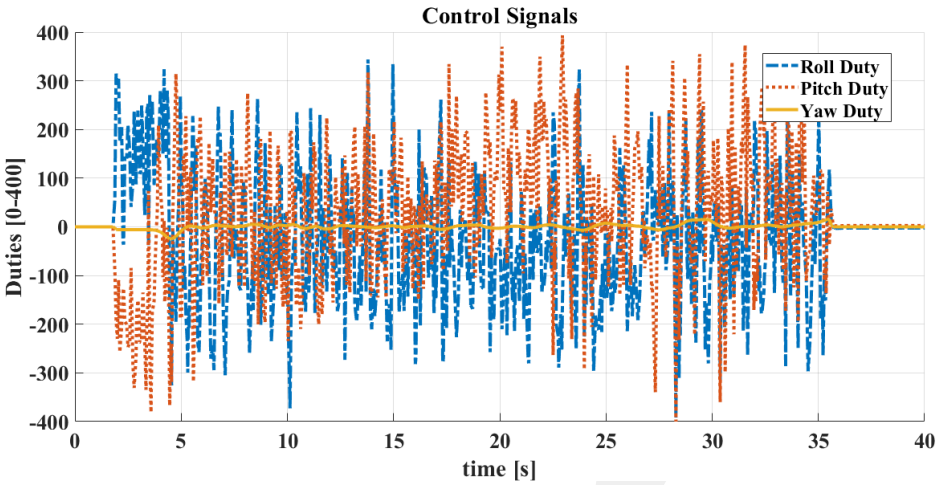
(b)


Attitude of Quadcopter
Roll
Pitch
Yaw
Euler Angles [deg]
time [s]
10
5
0
-5
-10
0
5
10
15
20
25
30
35
40

(c)


Angular Velocities
Roll Rate
Pitch Rate
Angular Rates [deg/s]
time [s]
30
20
10
0
-10
-20
-30
0
5
10
15
20
25
30
35
40

(d)



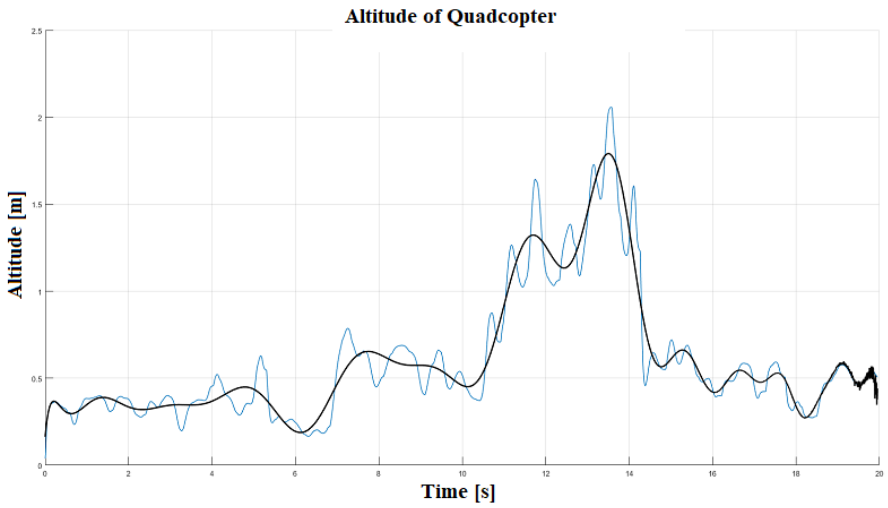
(e)

Figure 5.5 Hover Test 2 Results (a)Altitude (b) X-Y Trajectory (c) Attitude (d) Attitude Rates (e) Control Outputs

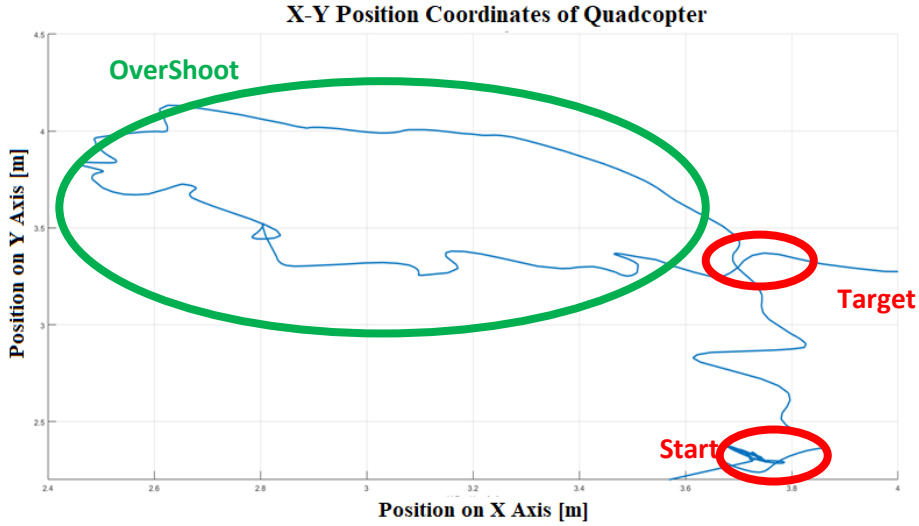
When the data is examined, it is seen that the system disturbances resist more and suppress the smallest movement. This is easily understood when comparing rate graphs. Although Test 2 had a higher number of rates, it was reduced in amplitudes.

5.2.2 Trajectory Tracking Tests

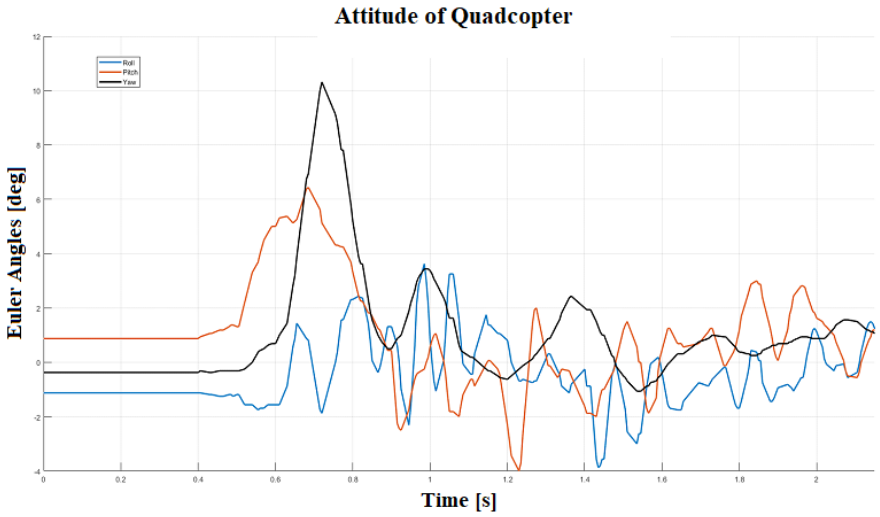
In these tests, the system is expected to return specified target. In order to move from the starting point to the target, the reference must be given in a non-step reference. In this way, the system is prevented from passing over the target. In the first test, a reference is given to the system as a step. This causes the system to miss the target. The test results are as follows in Figure 5.6.


Altitude of Quadcopter
Altitude [m]
Time [s]

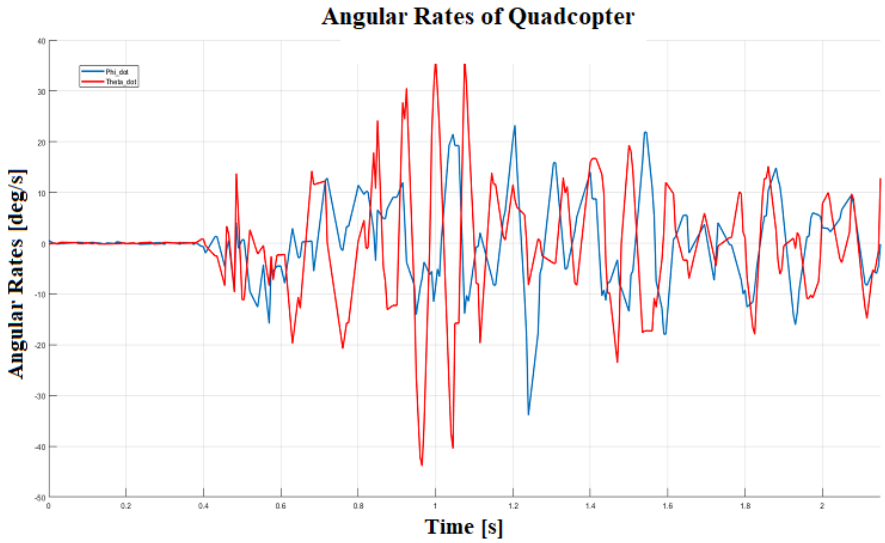
(a)


X-Y Position Coordinates of Quadcopter
OverShoot
Target
Start
Position on Y Axis [m]
Position on X Axis [m]

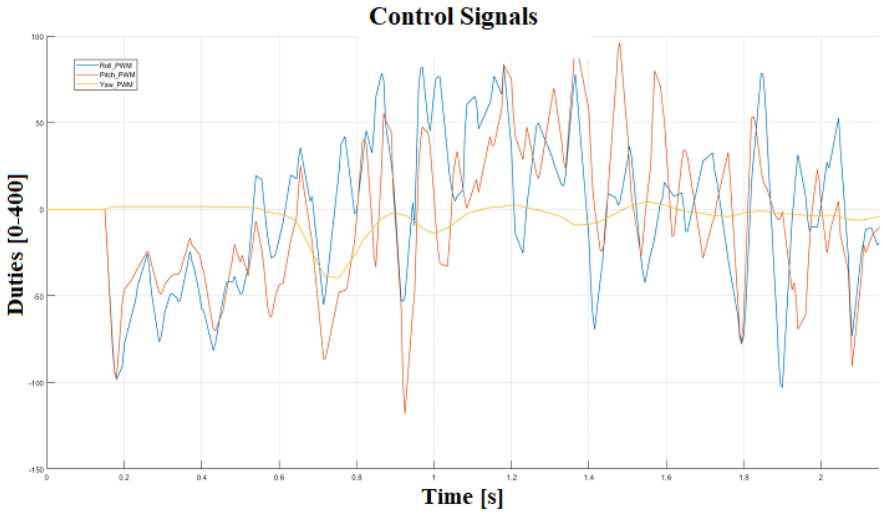
(b)


Attitude of Quadcopter
Euler Angles [deg]
Time [s]

(c)


Angular Rates of Quadcopter
Angular Rates [deg/s]
Time [s]

(d)

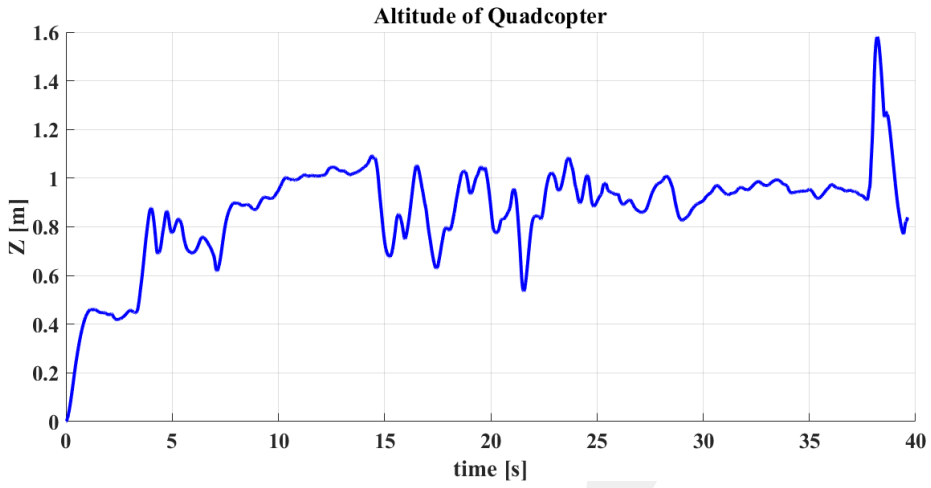


(e)

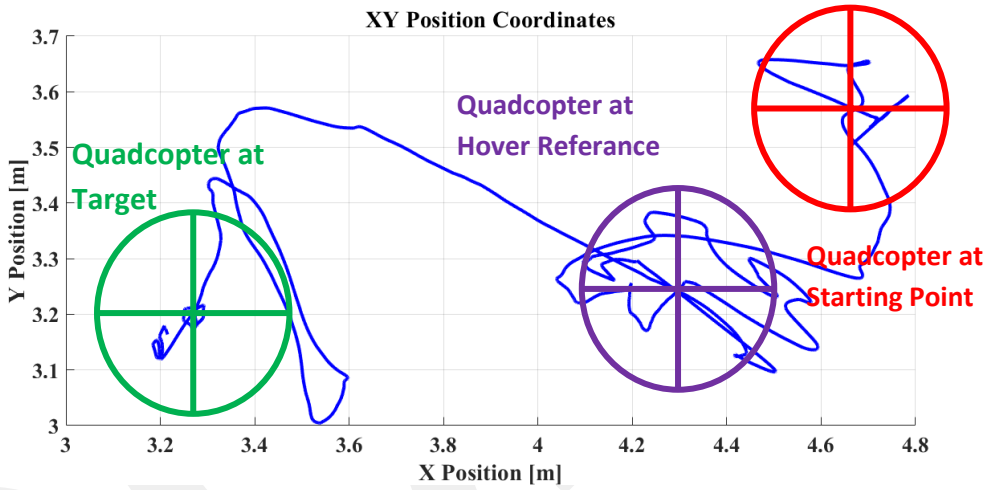
Figure 5.6 Position Change Test 1 Results (a) Altitude (b) X-Y Trajectory (c) Attitude (d) Attitude Rates (e) Control Outputs

The step reference given while the system is taking off causes the integral actions in the controllers to collect the error from the beginning. This causes the system to continue its path even if the target has reached. As shown in Figure 56 (b), the red circles show the start and the target. However, the system deviated from the desired point 1 meter in X axis, 0.5 m in Y axis (Green Circle). Although the system returns to the desired location, the error accumulated in the beginning caused the deviation from the first encounter with target.

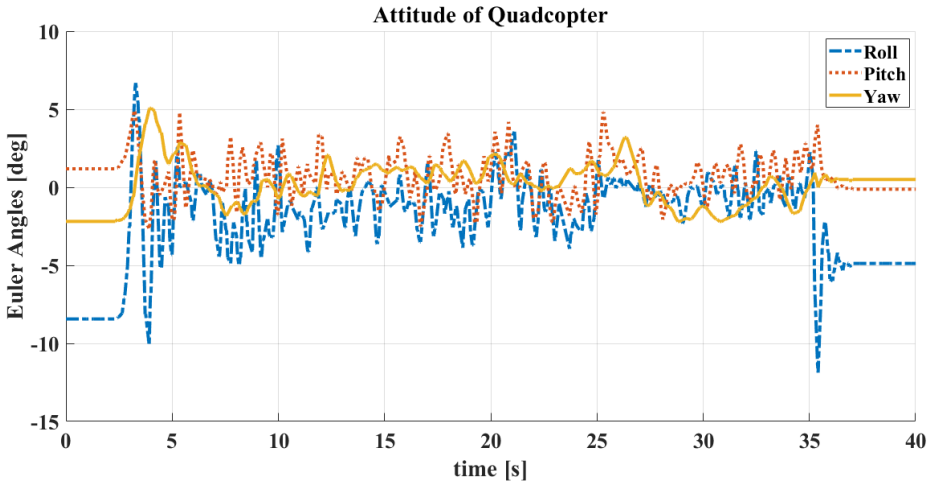
However, if the position reference to be given is increased slowly, the system can easily move to the desired point without overshoot. It is possible to give this reference to the system using a second-order transfer function, with more rise time. In this way, the system reaches the position reference slowly and without error. The results of a test where a system with a position reference is given slowly to the desired position is given in Figure 5.7.


Altitude of Quadcopter
Z [m]
time [s]

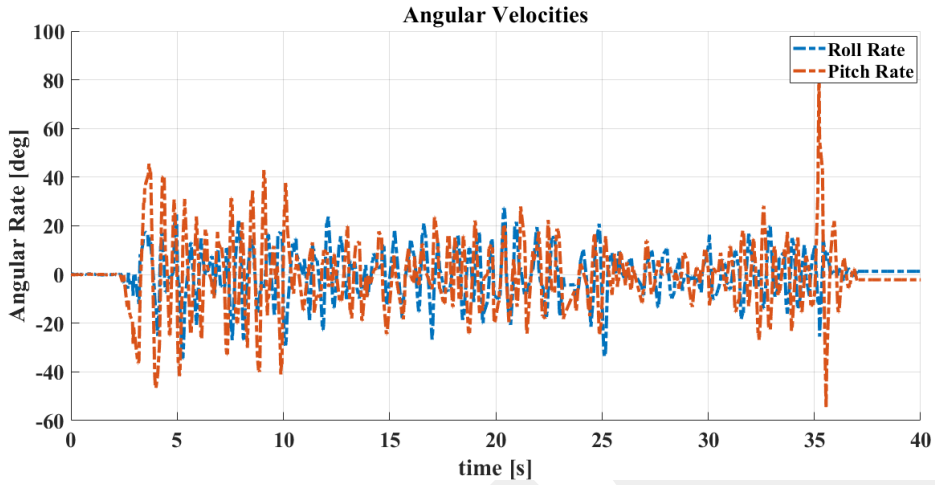
(a)


XY Position Coordinates
Quadcopter at
Target
Quadcopter at
Hover Referance
Quadcopter at
Starting Point
Y Position [m]
X Position [m]

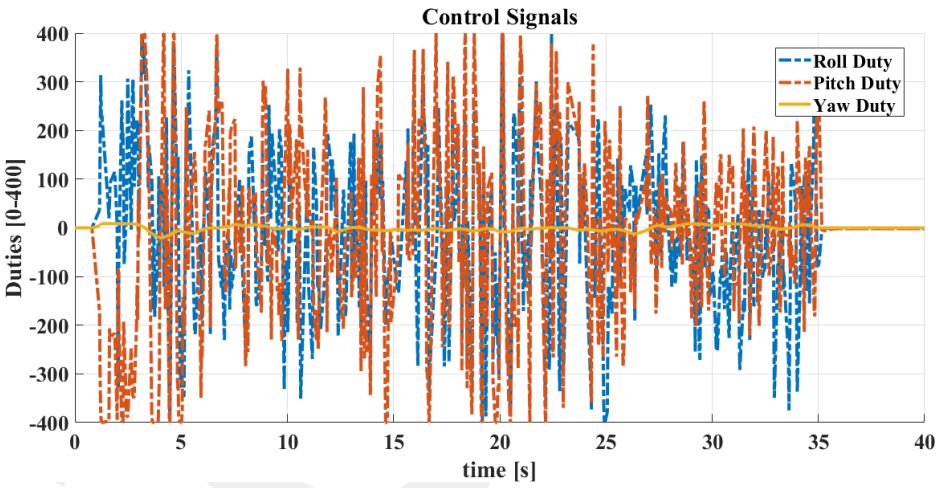
(b)


Attitude of Quadcopter
Roll
Pitch
Yaw
Euler Angles [deg]
time [s]

(c)



(d)



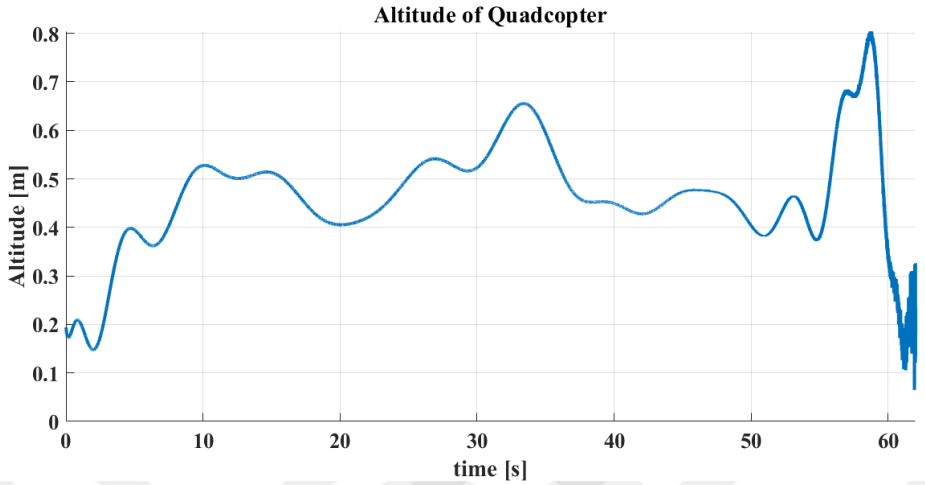
(e)

Figure 5.7 Position Change Test 2 Results (a) Altitude (b) X-Y Trajectory (c) Attitude (d) Attitude Rates (e) Control Outputs

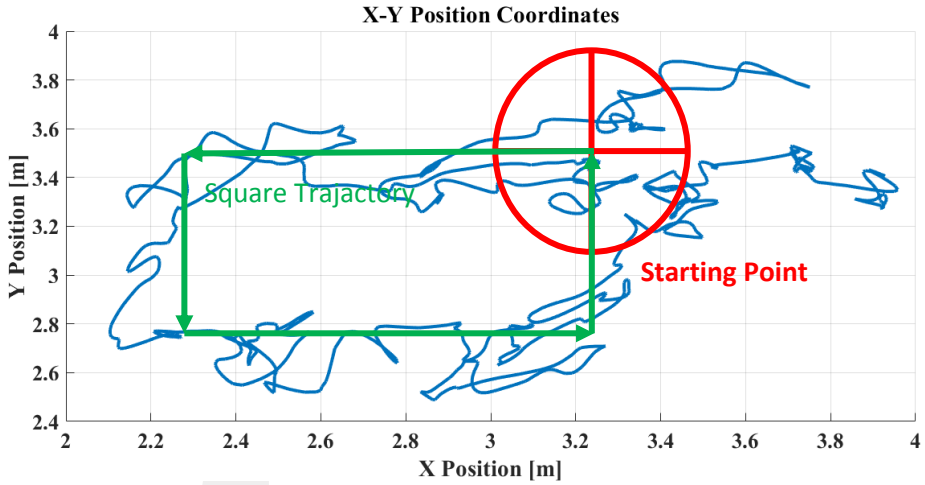
Position change test 2 results show that if the desired reference is given slowly, the accumulation in the integral within the controllers is prevented. In this way, the overshoot problem is solved.

Trajectory-Tracking tests are satisfactory with accurate references and low angular velocities of the system. However, in order to understand the full performance of trajectory tracking, a more complex reference should be sent to the system and the behavior should be examined. For this purpose, the system was given a position

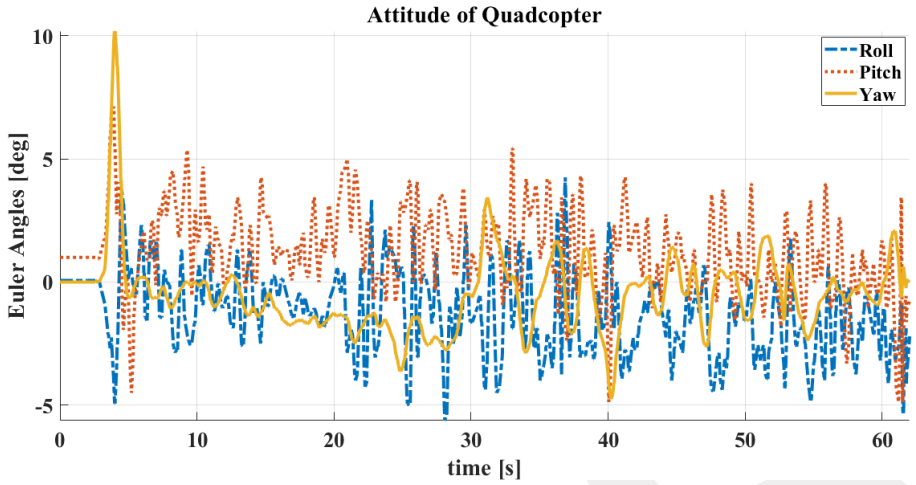
reference to create a square within the flight area. The results are given in the graphs in Figure 5.8.



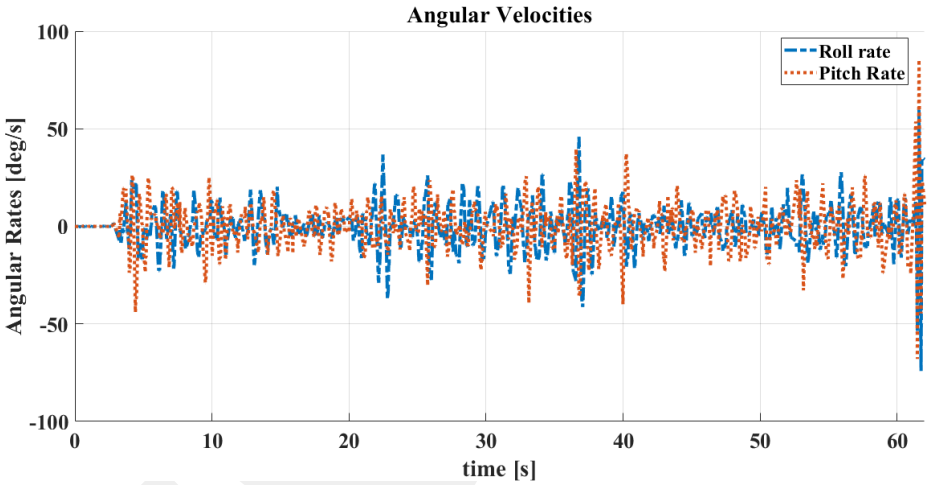
(a)



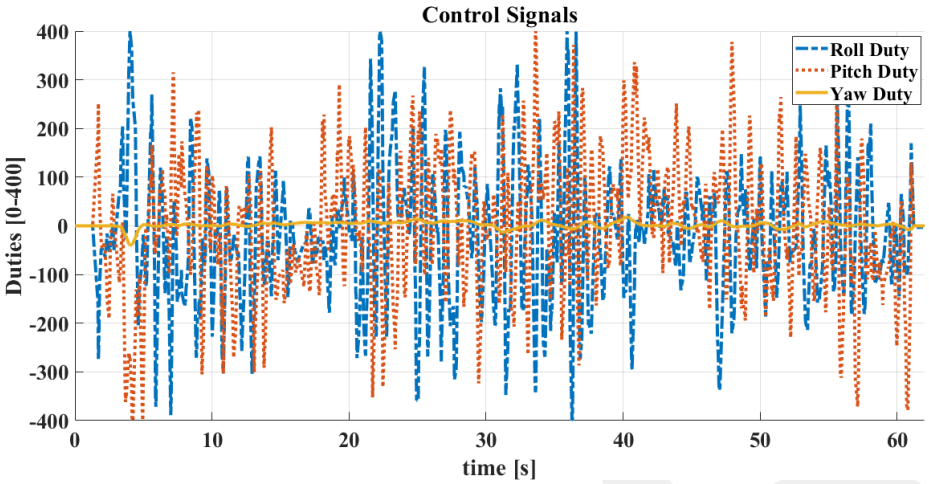
(b)


Attitude of Quadcopter
Roll
Pitch
Yaw
Euler Angles [deg]
time [s]
10
5
0
-5
0
10
20
30
40
50
60

(c)


Angular Velocities
Roll rate
Pitch Rate
Angular Rates [deg/s]
time [s]
100
50
0
-50
-100
0
10
20
30
40
50
60

(d)



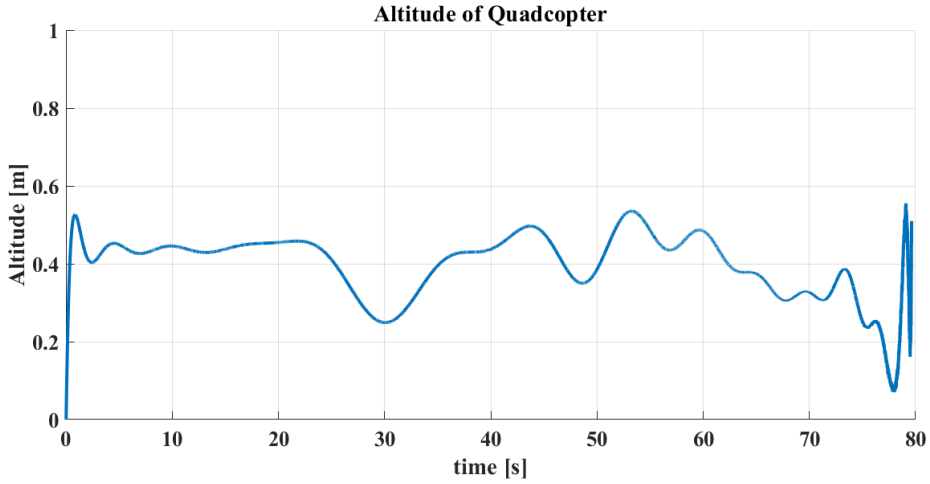
(e)

Figure 5.8 Square Reference Tracking Test Results (a) Altitude (b) X-Y Trajectory (c) Attitude (d) Attitude Rates (e) Control Outputs

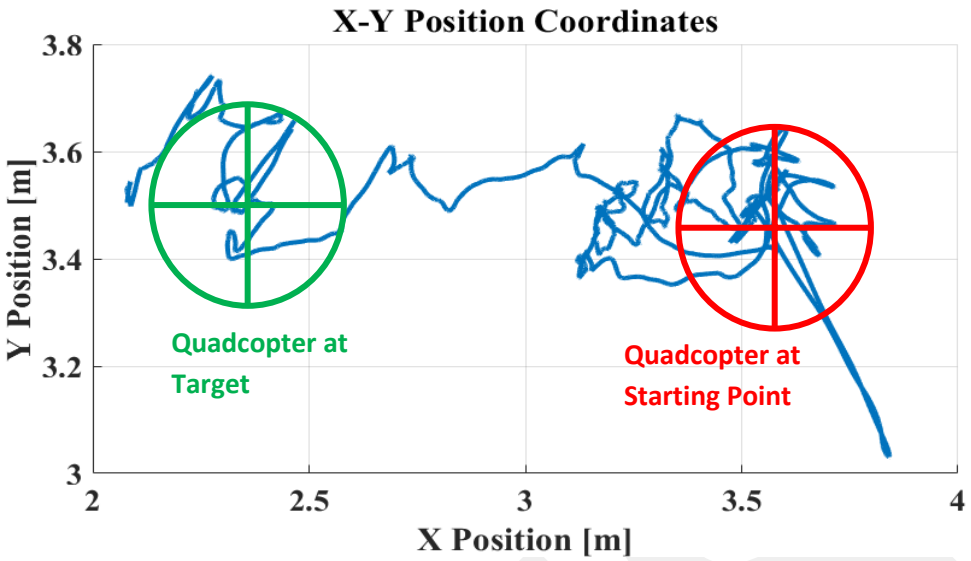
The results show that the system can follow the reference given with + -10 cm error.

5.2.3 Hover & Trajectory Tracking Tests

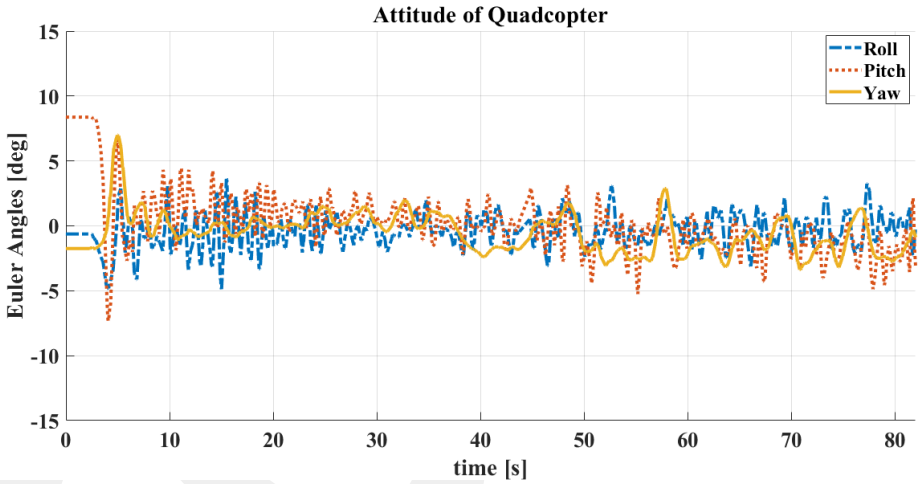
The autonomous position control performance of the system was satisfied based on hover and displacement terms. Therefore, another test was performed by combining the two. In this test, the system was initially hover at a certain points a certain period of time it was tracking the reference position. The test results are as follows in Figure 5.9.



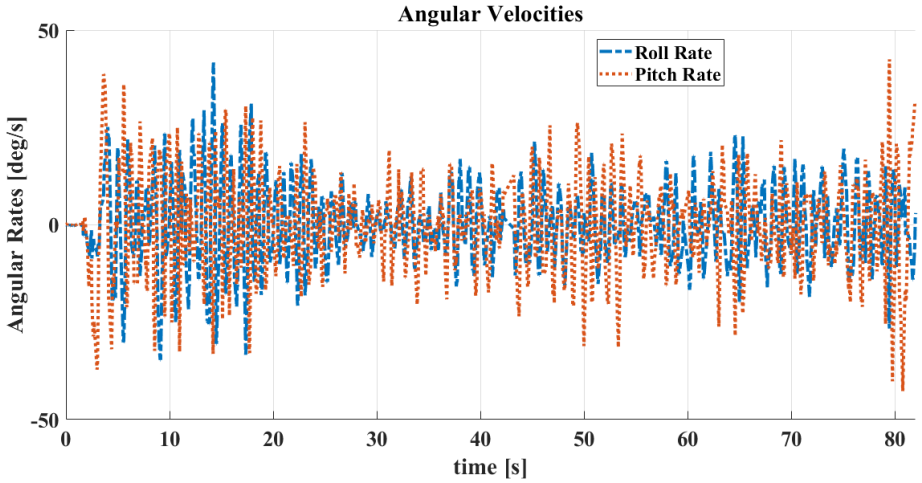
(a)


X-Y Position Coordinates
Y Position [m]
X Position [m]
3.8
3.6
3.4
3.2
3
2
2.5
3
3.5
4
Quadcopter at
Target
Quadcopter at
Starting Point

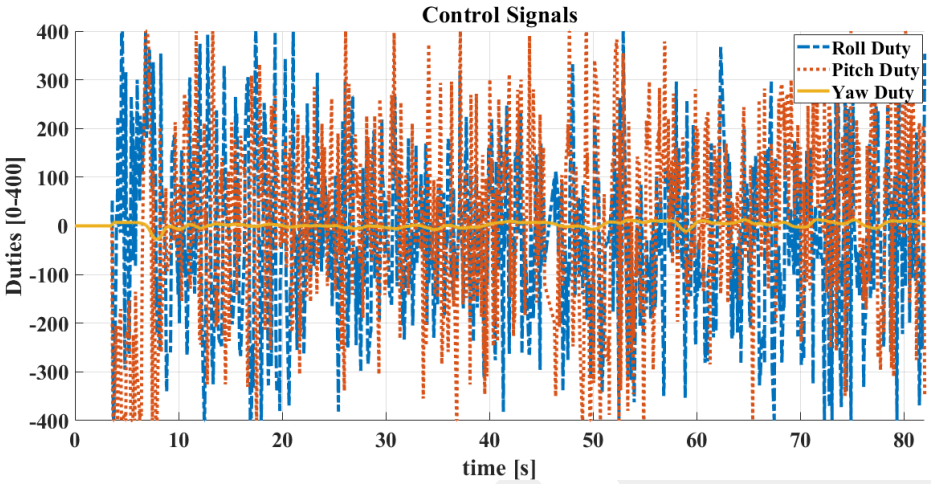
(b)


Attitude of Quadcopter
Euler Angles [deg]
time [s]
Roll
Pitch
Yaw
15
10
5
0
-5
-10
-15
0
10
20
30
40
50
60
70
80

(c)


Angular Velocities
Angular Rates [deg/s]
time [s]
Roll Rate
Pitch Rate
50
0
-50
0
10
20
30
40
50
60
70
80

(d)



(e)

Figure 5.9 Combined Position Change Test Results (a) Altitude (b) X-Y Trajectory (c) Attitude (d) Attitude Rates (e) Control Outputs

Autonomous position control test results showed that when the angular velocities of the system are in the range of 0-40 deg / s and an agile control is designed, the system can go to the desired position with 10-20 cm error margin and hover with 0-10 cm error at the desired point.

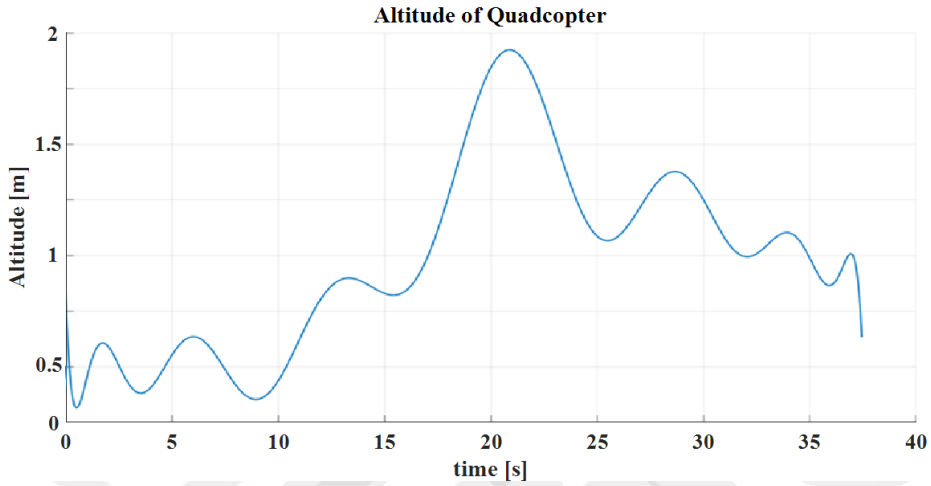
5.2.4 Hover Test under Disturbance

5.3 Autonomous Positioning with Manipulator Tests

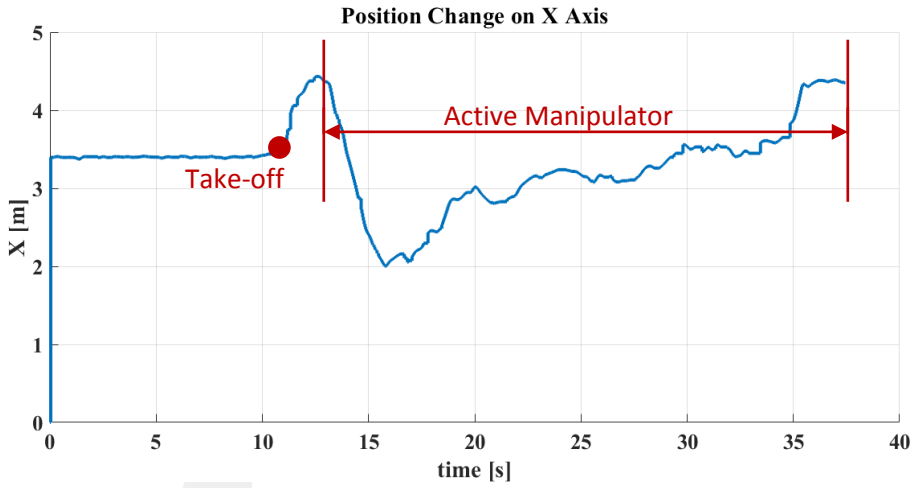
In this section, by activating the manipulator during the flight of the autonomous quadcopter, the pitch axis which the arm is connected is controlled only by the manipulator. For the pitch axis, only the altitude controller outputs are sent. For this purpose, two different tests were performed. First, the arm activated during the flight was asked to maintain the hover position. In the second test, after activating the manipulator, a reference is given to keep the hover position for a while and then to move to a given position reference.

5.3.1 Hover Test with the Active Manipulator

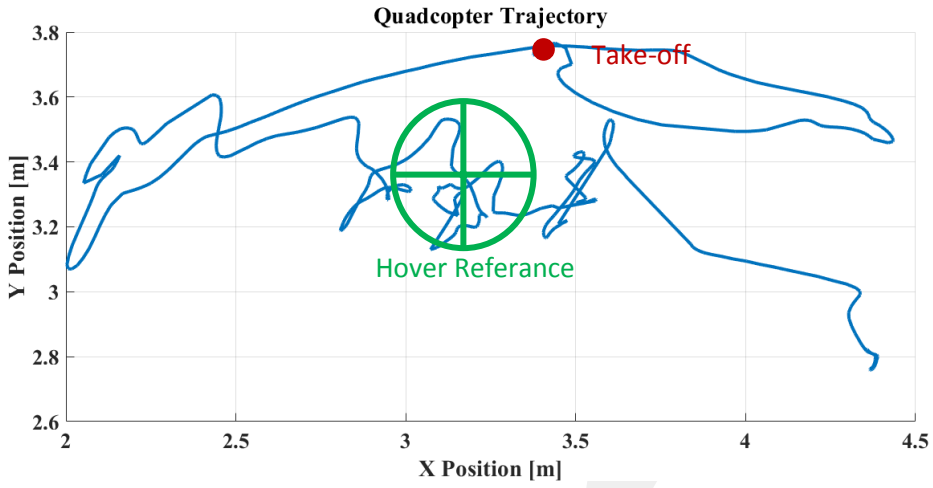
In this test, the system is expected to maintain the hover position as indicated. In this process, the manipulator was activated and the system was tested after the system took off. The test results are given in Figure 5.10.



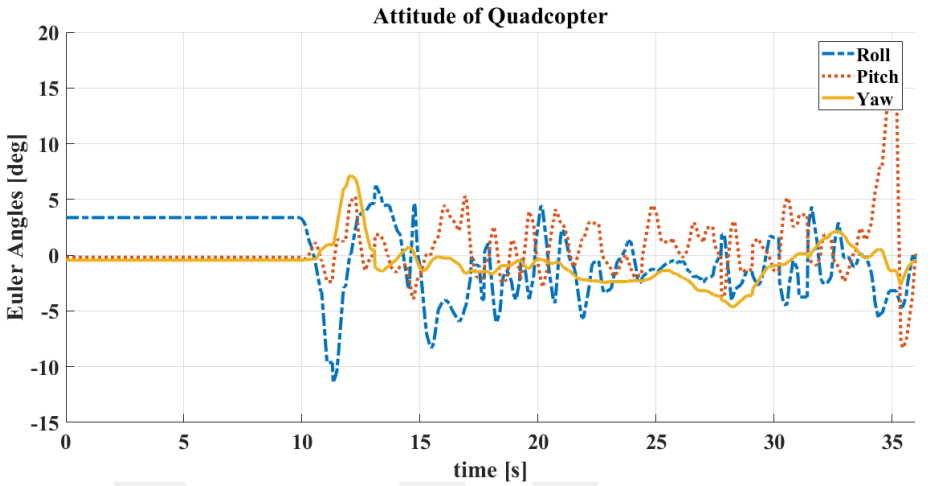
(a)



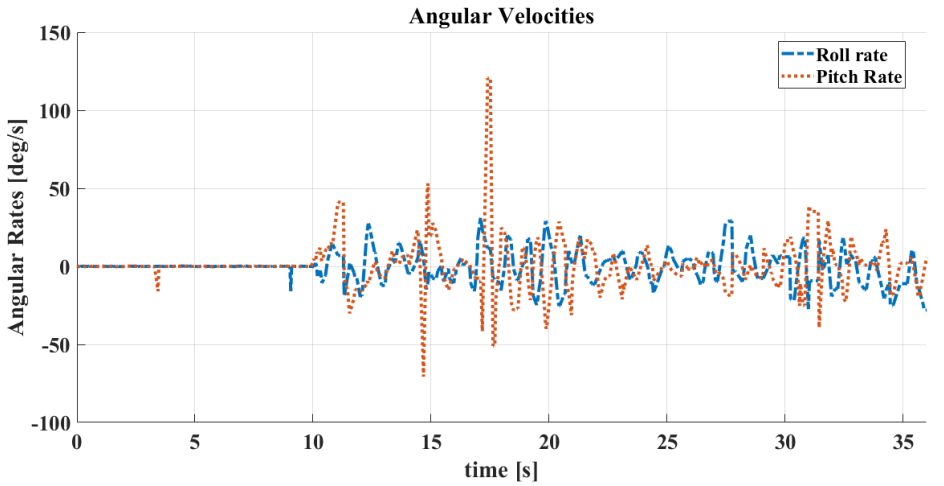
(b)


Quadcopter Trajectory
Take-off
Hover Referance
Y Position [m]
X Position [m]
3.8
3.6
3.4
3.2
3
2.8
2.6
2
2.5
3
3.5
4
4.5

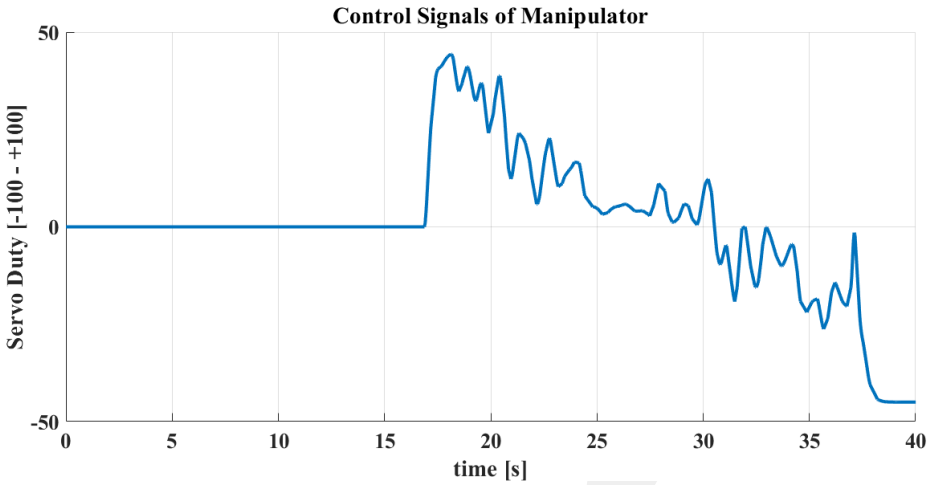
(c)


Attitude of Quadcopter
Roll
Pitch
Yaw
Euler Angles [deg]
time [s]
20
15
10
5
0
-5
-10
-15
0
5
10
15
20
25
30
35

(d)


Angular Velocities
Roll rate
Pitch Rate
Angular Rates [deg/s]
time [s]
150
100
50
0
-50
-100
0
5
10
15
20
25
30
35

(e)



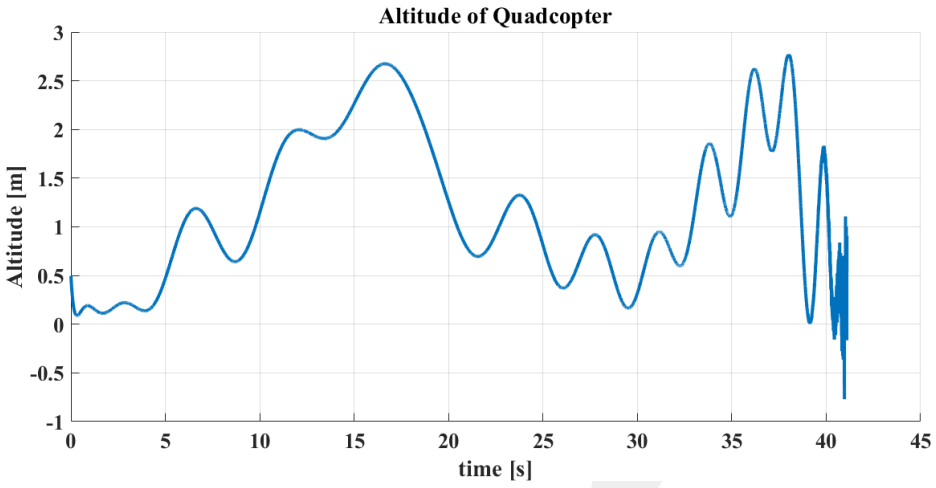
(f)

Figure 5.10 Manipulator Hover Control Test Results (a) Altitude (c)X Axis Position of Quadcopter (c) X-Y Trajectory (d) Attitude (e) Attitude Rates (f) Manipulator Servo PWM

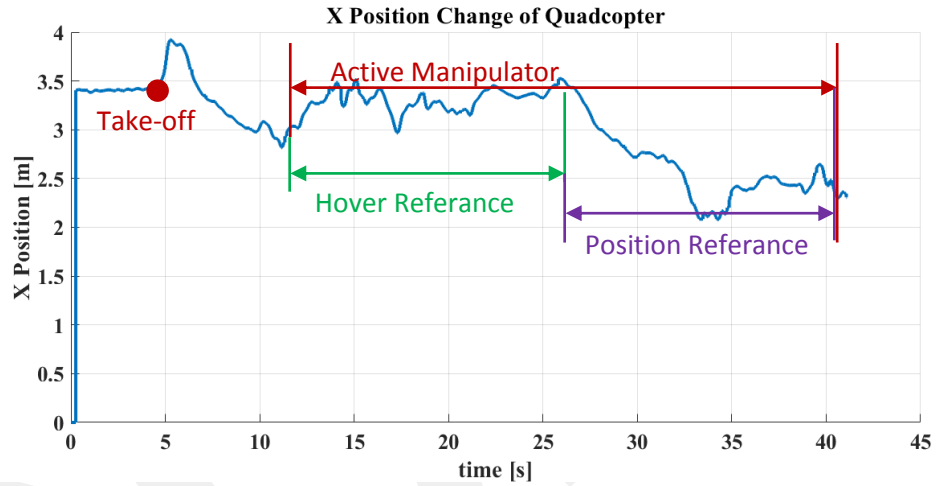
The pitch axis which the manipulator is connected is disrupted during take-off and made an error of about 1 meter on the X axis. However, after the manipulator was activated, the moment created by the manipulator helped the system to self-stabilize and returned to the desired hover position.

5.3.2 Positioning Test with the Active Manipulator

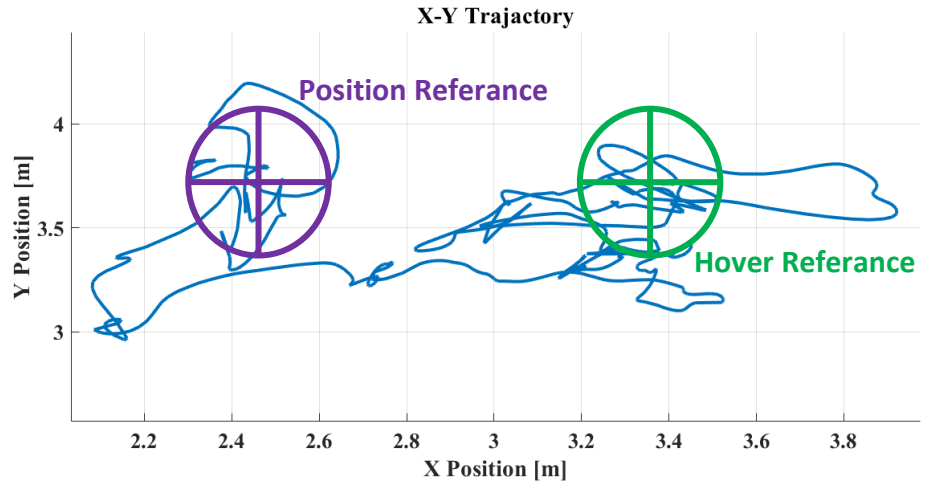
In this test, the manipulator was activated after the quadcopter took off and was referenced to remain in the hover at the desired area for a while. After a while, it was referenced to change its position on the X axis and hover at the desired position reference. The test results are given in Figure 5.11.


Altitude of Quadcopter
Altitude [m]
time [s]

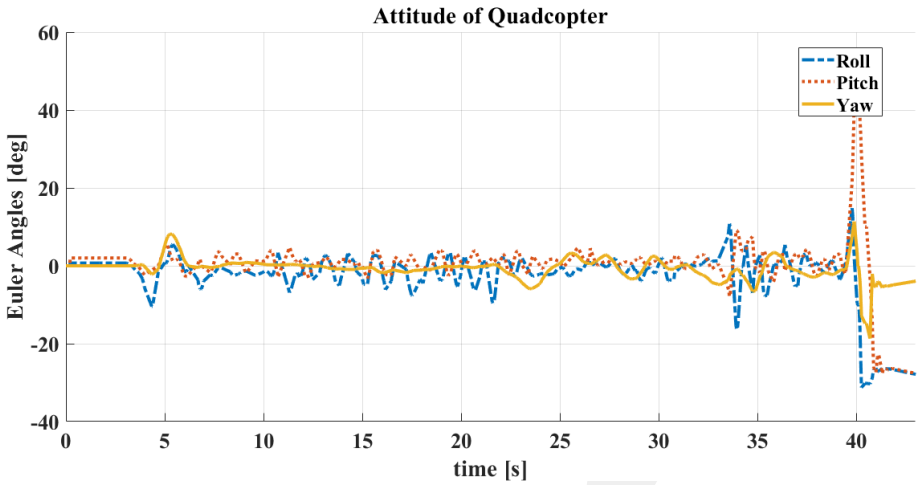
(a)


X Position Change of Quadcopter
Take-off
Active Manipulator
Hover Referance
Position Referance
X Position [m]
time [s]

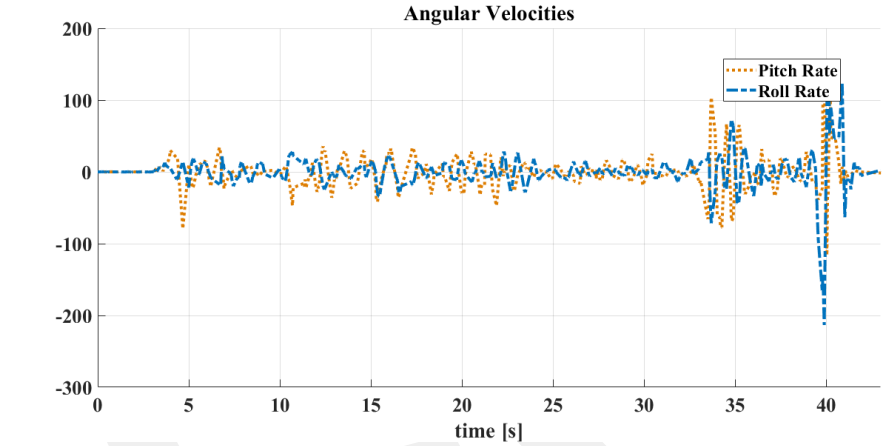
(b)


X-Y Trajactory
Position Referance
Hover Referance
Y Position [m]
X Position [m]

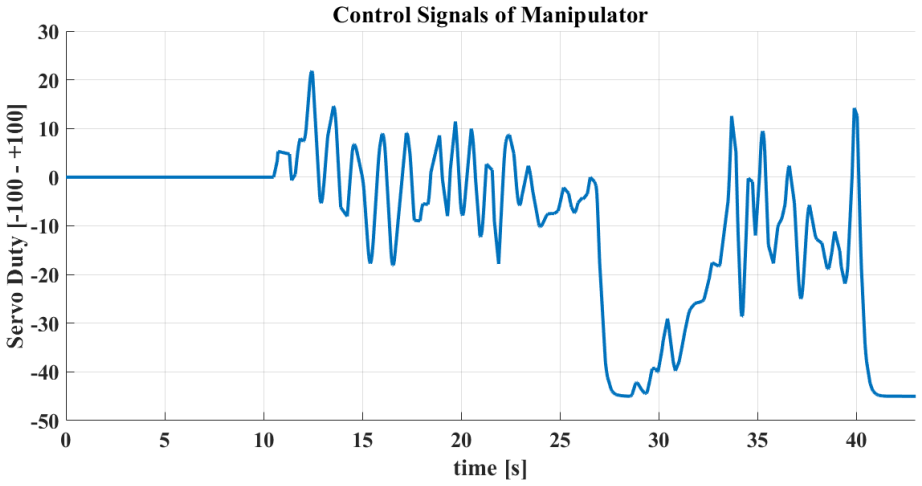
(c)


Attitude of Quadcopter
Roll
Pitch
Yaw
Euler Angles [deg]
time [s]

(d)


Angular Velocities
Pitch Rate
Roll Rate
time [s]

(e)


Control Signals of Manipulator
Servo Duty [-100 - +100]
time [s]

(f)

Figure 5.11 Manipulator Position Control Test Results (a) Altitude (c)X Axis Position of Quadcopter (c) X-Y Trajectory (d) Attitude (e) Attitude Rates (f) Manipulator Servo PWM

The test results show that the system is successful in both the hover task and the position change task.

CHAPTER 6

6. CONCLUSION AND DISCUSION

Aerial manipulation is achieved by a flying platform and a coupled robotic arm. This combination outcomes a lightly damped and a complex system which is prone to be disturbed easily. In this thesis, we classified the control systems in terms of the centralized and decentralized topologies. Several control algorithms are designed and implemented on a 2D nonlinear coupled dynamical model. Starting with the centralized control, the decentralized topology is discussed, as well. In the latter topology, the required torque for the regulation and control in the pitch plane is allocated by the robotic arm as an actuator sub system while the propulsion sub system is used to control the altitude. This is an unconventional way of multicopter control. In the field of aerial manipulation, such novel control allocations will be required to fully employ the dynamical potential of the system.

In the scope of the thesis, simulations are done with different control algorithms and the performance of the system under different disturbances are evaluated. In this context, different control structures such as PID, cascaded PID, LQR, T-LQR, and ADRC are designed. The best performance is obtained from the combination of ADRC and T-LQR.

In order to determine the moment of inertia, observe the response of the manipulator, and tune the attitude controllers, several tests are performed on the test bench. In order to perform the physical tests, Naze32 flight controller board is used to specify settings and limitations. The entire control algorithm of the system runs on the Raspberry Pi 3B + using MatLab/Simulink. At the same time, a Pyhton Script running in the background communicates with the MatLab interface to collect and package the sensor data on the system. This provides the necessary feedback to the controllers designed on MatLab/Simulink.

For performing system tests, the most reliable and necessary voltage/current distribution is integrated into the system. Each hardware is individually supplied with the required power. This allowed the system to perform an error-free flight test without being affected by possible low current or current peaks.

To best evaluate the effect of the manipulator on the system, flight tests are accomplished on a quadcopter performing autonomous flight. In the tests, the system is referenced to hover at a specified position and track to a given position reference. The test results are satisfactory. This shows that certain tasks can be performed using a manipulator on aerial vehicles, as well as stabilization, disturbance rejection, and attitude control.

As a future work, it is better to reduce the quadcopter chassis with high speed motors and small propellers to perform the indoor tests easily. Using control board with faster sampling time, the operating frequency of the system will be increased. In addition, motors with high torque and torque control are available for actuating the manipulator. This allows the system to be controlled dynamically by torque, without the need for any conversion.

REFERENCES

[1] H. B. Khamseh, F. Sharifi, A. Abdessameud “Aerial Manipulation - A Literature Survey”, September 2018

[2] F. Ruggiero, J. Cacace, H. Sadeghian, V. Lippiello, "Passivity-based control of VToL UAVs with a momentum-based estimator of external wrench and unmodeled dynamics.", *Robotics and Autonomous Systems*, 2015, pp.139-151.

[3] N. Bulut, “Modeling, simulation, and control of a quadrotor having a 2-dof robotic arm”,Master’s Thesis, METU, July 2019

[4] M. Aydemir, K. B. Arıkan,“Evaluation of the Disturbance Rejection Performance of an Aerial Manipulator”, *Journal of Intelligent & Robotic Systems*,2019

[5] A. Aksal, “Control of the Quadrotor Unmanned Aerial Vehicle Equipped with a Delta Robot Manipulator”, Master’s Thesis, Atılım University, 2016.

[6] S. Kannan, M. Alma, M. A. Olivares-Mendez and H. Voos, “Adaptive Control of Aerial Manipulation Vehicle”, November 2014

[7] A. Khalifa, M. Fanni, A. Ramadan and A. Abo-Ismail, “Adaptive Intelligent Controller Design for a New Quadrotor Manipulation System”, 2013

[8] S. Kim, S. Choi and H. J. Kim, “Aerial Manipulation Using a Quadrotor with a Two DOF Robotic Arm”, November 2013

[9] A.E.Jimenez-Cano, J.Martin, G.Heredia, A.Ollero, R.Cano, “Control of an aerial robot with multi-link arm for assembly tasks”, May 2013

[10] J.U. Alvarez-Mun˜oz, N. Marchand, F. Guerrero-Castellanos, S. Durand, A. E. Lopez-Luna, “Improving control of quadrotors carrying a manipulator arm”, October 2014

[11] J.L.J. Scholten, M. Fumagalli, S. Stramigioli and R. Carloni, “Interaction Control of an UAV Endowed with a Manipulator”, *IEEE International Conference on Robotics and Automation (ICRA)*, May 2013

[12] R. Zanella, “Decoupled Controllers for Mobile Manipulation with Aerial Robots”, Sweden 2016

[13] A. Khalifa, M. Fanni, A. Ramadan and A. Abo-Ismail, "New Quadrotor Manipulation System: Inverse Kinematics, Identification and Ric-Based Control", August 2015

[14] M. Orsag, C. Korpela, M. Pekala, and P. Oh, "Stability Control in Aerial Manipulation", 15 March 2013

[15] S. Kannan, S. Quintanar-Guzman, J. Dentler, M.A. Olivares-Mendez and H. Voos,"Control of Aerial Manipulation Vehicle in Operational Space", *ECAI*, 2016

[16] H. Lee, S. Kim and H.J. Kim "Control of an Aerial Manipulator using On-line Parameter Estimator", Jan 2016

[17] M. Tognon, B. Yüksel, G. Buondonno, A. Franchi, "Dynamic Decentralized Control for Protocentric Aerial", *IEEE Conf. on Robotics and Automation Singapore*, 2017

[18] P.E.I. Pounds, D.R. Bersak and A.M. Dollar, "Grasping from the air Hovering capture and load stability", *IEEE Conf. on Robotics and Automation Shanghai*, 2011

[19] L. Marconi, R. Naldi, L. Gentili, "Modeling and control of a flying robot interacting with the environment", *Automatica 47*, 2011

[20] V. Lippiello and F. Ruggiero, "Cartesian impudence control of a UAV with a robotic arm", *10th IFAC Symposium on Robot Control International Federation of Automatic Control ,* September 2012

[21] M. Orsag, C. Korpela, P. Oh "Modeling and control of MM-UAV Mobile manipulating unmanned aerial vehicle", *Journal of Intelligent Robotic Systems Springer Science,* 2013

[22] A. Khalifa, M. Fanni, A. Ramadan, A. Abo-Ismail "Modeling and control of a new quadrotor manipulation system",*IEEE* 2012

[23] M. Orsag, C. Korpela, S. Bogdan, and P. Oh, "Lyapunov based model reference adaptive control for aerial manipulation", *International Conference on Unmanned Aircraft Systems*, 2013

[24] V. Ghadiok, J. Goldin and W. Ren, "Autonomous indoor aerial gripping using a quadrotor", *Intelligent Robots and Systems IEEE*,2011

[25] G. Antonelli, F. Arrichiello, S. Chiaverini, P.R. Giordano, "Adaptive trajectory tracking for quadrotor mavs in presence of uncertainties and external disturbances", *Advanced Intelligent Mechatronics IEEE*,2013

[26] "Naze 32 Revision 6 Flight Controller illustration". Received from https://www.dronetrest.com/t/naze-32-revision-6-flight-controller-guide

[27] "XY-3606 Regulator Module illustration". Received from https://www.aliexpress.com/item/32775054308.html

[28] "Radiolink AT9S illustration". Received from https://www.roboshop.com.tr/radiolink-at9s-rc-9-kanal-kumanda

[29] "Pozyx Anchor and tag illustration". Received from https://www.pozyx.io

[30] “Raspberrry pi 3B+ illustration”. Received from https://www.raspberrypi.org/

[31] “Raspberrry pi 3B+ GPIO pins layout illustration”. Received from https://maker.robotistan.com/raspberry-pi-dersleri-4-gpio-ile-led-kontrolu/

APPENDIX

APPENDIX A

```
clear all
clc

% Equation of Motion
% Solver&Linerization----------------------------------------------------
syms phi theta psi phid thetad psid phidd thetadd psidd alpha alphad alphadd xdd ydd
zdd Fx Fy F1 F2 F3 F4  ...
   Kf m1 m2 g Ixx Iyy Izz Im Lb T Md1 Md2 Md3 Md4 Rm Kmot Km Jm omega1
omega2 omega3 omega4 pt Ct D R Cd A h L x xd y yd z zd ...
   a1 a2 a3 omega1d omega2d omega3d omega4d duty F duty1 duty2 duty3 duty4
vel Vs Kts Rs Mdis


eqn1=m1*xdd+Fx==(-F1-F3)*sin(phi);
eqn2=m1*zdd+Fy==(F1+F3)*cos(phi)-m1*g;
eqn3=Ixx*phidd+Mdis==(F3-F1)*L-T-Fx*h*cos(phi)-Fy*h*sin(phi);
eqn4=m2*xdd+phidd*h*cos(phi)*m2-alphadd*(Lb/2)*sin(alpha)*m2-
Fx==(h*(phid^2)*sin(phi)*m2)+((Lb/2)*(alphad^2)*cos(alpha)*m2);
eqn5=m2*zdd+m2*h*phidd*sin(phi)+m2*(Lb/2)*alphadd*cos(alpha)-
Fy==(Lb/2)*(alphad^2)*sin(alpha)*m2-h*(phid^2)*cos(phi)*m2-m2*g;
eqn6=Im*alphadd==T+Fx*(Lb/2)*sin(alpha)-Fy*(Lb/2)*cos(alpha);

eqn7=F1==((1250*duty1)/381 + (1250*(duty1^2 - (341407*duty1)/312500 +
268931/781250)^(1/2))/381 - 1675/762)*9.81;
eqn8=F3==((1250*duty3)/381 + (1250*(duty3^2 - (341407*duty3)/312500 +
268931/781250)^(1/2))/381 - 1675/762)*9.81;

sol=solve([eqn1 eqn2 eqn3 eqn4 eqn5 eqn6 eqn7 eqn8], ...% eqn9 eqn10
   [xdd, zdd, phidd, alphadd, Fx, Fy, F1, F3]); % T, Vser

%State-Space x_dot Selection--------------------------------------------
xddsol=sol.xdd;
zddsol=sol.zdd;
phiddsol=sol.phidd;
alphaddsol=sol.alphadd;

x2d=phiddsol;
x9d=xddsol;
x13d=zddsol;
x8d=alphaddsol;
```

```
% Linearize Active Manipulator----------------------------------------------------
u1=[duty1;duty3];
u2=[T];

x_al=[phi;phid;alpha;alphad;z;zd];
dx=[phid;x2d;alphad;x8d;zd;x13d];

x_al2=[alpha;alphad];
dx2=[alphad;x8d];

x_al3=[phi;phid;alpha;alphad];
dx3=[phid;x2d;alphad;x8d];

As=jacobian(dx,x_al);
Bs=jacobian(dx,u1);

As2=jacobian(dx2,x_al2);
Bs2=jacobian(dx2,u2);

As3=jacobian(dx3,x_al3);
Bs3=jacobian(dx3,u2);

%Parameters-----------------------------------------------------------

m1=2.7/2;
m2=.3;
L=.4;
Lb=.4;
Ixx=0.1124;%0.0457;
Im=(m2*(Lb^2))-(m2*((Lb/2)^2));
h=0.07;
g=9.81;

D=0.356;
R=D/2;
Kmot=0.0127;
Ke=Kmot;
d=7.5e-4;
Rm=0.0479;
Jm=3.68e-5;
A=(pi*((D/2)^2));
pt=1.3;
Ct=0.1027;
Cd=0.0118;
Iyy=0.0457;
Izz=0.0846;
a1=-((Kmot*Ke)/Rm);
a2=-(d/Jm);
a3=Kmot/(Jm*Rm);
```

```
Kf=pt*Ct*(D^4);
Km=Kf/58;

%Linearization Points----------------------------------------------------
phi=0;
phid=0;
alpha=-pi/2+phi;
alphad=0;

duty1=.6414;
duty2=.6414;
duty3=.6414;
duty4=.6414;
T=0;
Mdis=0;

%Numerical Matrixes-------------------------------------------------------
Bs=subs(Bs);
B=double(Bs);

As=subs(As);
Ad=double(As);

Bs2=subs(Bs2);
B2=double(Bs2);

As2=subs(As2);
Ad2=double(As2);

Bs3=subs(Bs3);
B3=double(Bs3);

As3=subs(As3);
Ad3=double(As3);

C=[1 0 0 0 0 0 0 0;
   0 1 0 0 0 0 0 0;
   0 0 1 0 0 0 0 0;
   0 0 0 1 0 0 0 0;
   0 0 0 0 0 0 1 0;
   0 0 0 0 0 0 0 1];

C1=[0 0 0 0 1 0];

C2=[0 0 1 0 0 0 0 0;
    0 0 0 1 0 0 0 0];

C2t=[1 0];

C3t=[1 0 0 0];
```

```
C3=[1 0 0 0 0 0 0 0;
    0 1 0 0 0 0 0 0;
    0 0 1 0 0 0 0 0;
    0 0 0 1 0 0 0 0];

Cct=[1 0 0 0;
     0 0 1 0];

Ab1=[Ad zeros(6,1);-C1 zeros(1)];
Bb1=[B;zeros(1,2)];

Ab2=[Ad2 zeros(2,1);-C2t zeros(1)];
Bb2=[B2;zeros(1,1)];

Ab3=[Ad3 zeros(4,1);-C3t zeros(1)];
Bb3=[B3;zeros(1,1)];

Abc=[Ad3 zeros(4,2);-Cct zeros(2)];
Bbc=[B3;zeros(2,1)];


Ic=[phi;0;alpha;0;0;0;1;0];

%LQR Design-----------------------------------------------------------

aa=1/sqrt(7);
aa2=1/sqrt(3);
aa3=1/sqrt(4);
aa3t=1/sqrt(5);
aac=1/sqrt(6);
%at=sqrt(4);

x1m=2*pi;
x2m=2*pi;
x7m=18*pi/9;
x13m=.005;
x14m=.05;
x1em=pi/180;
x13em=.005;
x7em=pi/180000000;

%Traking Maximum Values-----------------------------------------------

x1mt=1e100;
x2mt=1e100;
x7mt=pi/9;
x8mt=2*pi;
x1emt=2*pi;
x1cmt=3.2*pi;
```

```
x7cmt=3*pi;
%-------------------------------------------------------------------------

u1m=.95;
u2m=.95;
u3m=.95;
u4m=.95;
u5m=4;       %.25*g;

ro1=5;
ro=5;
bet=1/(sqrt(1));

Q=[((aa)^2)/((x1m)^2) zeros(1,6);
   zeros(1,1) ((aa)^2)/((x2m)^2) zeros(1,5);
   zeros(1,2) ((aa)^2)/((x7m)^2) zeros(1,4);
   zeros(1,3) ((aa)^2)/((x8m)^2) zeros(1,3);
   zeros(1,4) ((aa)^2)/((x13m)^2) zeros(1,2);
   zeros(1,5) ((aa)^2)/((x14m)^2) zeros(1,1);
   zeros(1,6) ((aa)^2)/((x13em)^2)];

Q2=[((aa2)^2)/((x7m)^2) zeros(1,2);
   zeros(1,1) ((aa2)^2)/((x8m)^2) zeros(1,1);
   zeros(1,2) ((aa2)^2)/((x7em)^2)];

Q3=[((aa3)^2)/((x1mt)^2) 0 0 0;
    0 ((aa3)^2)/((x2mt)^2)  0 0;
    0 0 ((aa3)^2)/((x7mt)^2) 0;
    0 0 0 ((aa3)^2)/((x8mt)^2)];

Q3t=[((aa3t)^2)/((x1mt)^2) zeros(1,4);
     zeros(1,1) ((aa3t)^2)/((x2mt)^2) zeros(1,3);
     zeros(1,2) ((aa3t)^2)/((x7mt)^2) zeros(1,2);
     zeros(1,3) ((aa3t)^2)/((x8mt)^2) zeros(1,1);
     zeros(1,4) ((aa3t)^2)/((x1emt)^2)];

Qc=diag([((aac)^2)/((x1m)^2) ((aac)^2)/((x2m)^2) ((aac)^2)/((x7m)^2)
((aac)^2)/((x8m)^2) ((aac)^2)/((x1cmt)^2) ((aac)^2)/((x7cmt)^2)]) ;
Rc= 2*((bet^2)/(u5m^2));


R1=ro1*[((bet^2)/(u1m^2)) 0;
        0 ((bet^2)/(u3m^2))];
R2=ro*((bet^2)/(u5m^2));

R3=10*((bet^2)/(u5m^2));

R3t=2*((bet^2)/(u5m^2));
%Controller Gain for Tracking LQR----------------------------------------
Kmix=lqr(Ab1,Bb1,Q,R1);
```

```
K=[zeros(2,4) Kmix(:,5:6)];
Ki=[Kmix(:,7)];


K2mix=lqr(Ab2,Bb2,Q2,R2);
K2=[K2mix(1,1) K2mix(1,2)];
K2i=[K2mix(1,3)];

K3=lqr(Ad3,B3,Q3,R3);

K3tmix=lqr(Ab3,Bb3,Q3t,R3t);
K3t=K3tmix(1,1:4);
K3ti=K3tmix(1,5);

Kcmix=lqr(Abc,Bbc,Qc,Rc);
Kc=Kcmix(1,1:4)
Kci=Kcmix(1,5:6)

%ADRC Design ----------------------------------------------------------------
Abart=[0 1 0;0 0 1;0 0 0];
Cbart=[1 0 0];
Dbart=[0];

Mpt=1e-2;
tst=1.2;
at=log(Mpt)/pi;
zett=sqrt((at^2)/((at^2)+1));
wnt=4/(zett*tst);
Lot=[3*wnt;3*wnt^2;wnt^3];
```

APPENDIX B

```
from pypozyx import PozyxSerial, get_first_pozyx_serial_port

from math import sqrt

from pypozyx import Coordinates, DeviceCoordinates

from pypozyx.core import PozyxCore

#from pypozyx.definitions.constants import PozyxConstants

from pypozyx.definitions import (PozyxBitmasks, PozyxRegisters, PozyxConstants,
POZYX_SUCCESS, POZYX_FAILURE, POZYX_TIMEOUT,
ERROR_MESSAGES)

from pypozyx.structures.byte_structure import ByteStructure

from pypozyx.structures.generic import XYZ, SingleSensorValue, SingleRegister,
Data, SingleRegister, dataCheck

from pypozyx import PozyxSerial, get_first_pozyx_serial_port, POZYX_SUCCESS,
SingleRegister, EulerAngles, Acceleration, AngularVelocity

from pypozyx.structures.device import NetworkID, UWBSettings, DeviceList,
Coordinates, RXInfo, DeviceCoordinates, FilterData, AlgorithmData

import time

import timeit

import serial

import struct

import sys              # for user input

import socket           # for UDP communication

import datetime         # for current time

import asyncore         # for asynchornous udp comm

import SocketServer     # for socketserver udp comm

import threading        # for using threads

import array as arr
```

```
global rollrc

global pitchrc

global yawrc

global throttlerc

##########################################################################
######

############################ Configuration
#################################

##########################################################################
######

# The following lines define the serial port

##########################################################################
######

class drone(object):

    FILE    =    0    # Save to a timestamped file, the data selected below

    TIME    =    1    # Save the difference of time between all the main functions
for perfomance logging

    FLYT    =    1    # Save the flight time in seconds

    ATT     =    1    # Ask and save the attitude of the multicopter

    ALT     =    0    # Ask and save the altitude of the multicopter

    RC      =    0    # Ask and save the pilot commands of the multicopter

    MOT     =    0    # Ask and save the PWM of the motors that the MW is
writing to the multicopter

    RAW     =    0    # Ask and save the raw imu data of the multicopter

    CMD     =    0    # Send commands to the MW to control it

    UDP     =    0    # Save or use UDP data (to be adjusted)

    ASY     =    0    # Use async communicacion

    SCK     =    0    # Use regular socket communication

    SCKSRV  =    0    # Use socketserver communication

    PRINT   =    1    # Print data to terminal, useful for debugging
```

```
ser=serial.Serial()
ser.port="/dev/ttyS0"        # This is the port that the MultiWii is attached to (for raspberry pie)
ser.baudrate=115200
ser.bytesize=serial.EIGHTBITS
ser.parity=serial.PARITY_NONE
ser.stopbits=serial.STOPBITS_ONE
ser.timeout=0
ser.xonxoff=False
ser.rtscts=False
ser.dsrdtr=False
ser.writeTimeout=2
ser1=serial.Serial()
ser1.port="/dev/ttyUSB0"        # This is the port that the MultiWii is attached to (for raspberry pie)
ser1.baudrate=115200
ser1.bytesize=serial.EIGHTBITS
ser1.parity=serial.PARITY_NONE
ser1.stopbits=serial.STOPBITS_ONE
ser1.timeout=0
ser1.xonxoff=False
ser1.rtscts=False
ser1.dsrdtr=False
ser1.writeTimeout=2
timeMSP_ATT = 0.005
timeMSP_ALT = 0.0065
timeMSP_RAW = 0.005
timeMSP = 0.005
################################
# Initialize Global Variables
```

################################

latitude = 0.0

longitude = 0.0

altitude = -0

heading = -0

timestamp = -0

gpsString = -0

numSats = -0

accuracy = -1

beginFlag = 0

roll = 0

pitch = 0

yaw = 0

throttle = 0

angx = 0.0

angy = 0.0

m1 = 0

m2 = 0

m3 = 0

m4 = 0

message = ""

ax = 0

ay = 0

az = 0

gx = 0

gy = 0

gz = 0

magx = 0

magy = 0

```
magz = 0
elapsed = 0
flytime = 0
udp_mess = ""
udp_mess2 = ""
numOfValues = 0
precision = 3
rcData = [1500, 1500, 1500, 1000] #order -> roll, pitch, yaw, throttle
#########################################################################
#
###################### MultiWii Serial Protocol#######################
#########################################################################
#
#  The following define the hex values required
#  to make various MSP requests
################################################
BASIC="\x24\x4d\x3c\x00"              #MSG Send Header (to MultiWii)
MSP_IDT=BASIC+"\x64\x64"              #MSG ID: 100
MSP_STATUS=BASIC+"\x65\x65"           #MSG ID: 101
MSP_RAW_IMU=BASIC+"\x66\x66"    #MSG ID: 102
MSP_SERVO=BASIC+"\x67\x67"            #MSG ID: 103
MSP_MOTOR=BASIC+"\x68\x68"            #MSG ID: 104
MSP_RC=BASIC+"\x69\x69"               #MSG ID: 105
MSP_RAW_GPS=BASIC+"\x6A\x6A"    #MSG ID: 106
MSP_ATTITUDE=BASIC+"\x6C\x6C"   #MSG ID: 108
MSP_ALTITUDE=BASIC+"\x6D\x6D"   #MSG ID: 109
MSP_BAT = BASIC+"\x6E\x6E"            #MSG ID: 110
MSP_COMP_GPS=BASIC+"\x71\x71"   #MSG ID: 111
MSP_SET_RC=BASIC+"\xC8\xC8"     #MSG ID: 200
CMD2CODE = {
```

'MSP_IDENT':100,

'MSP_STATUS':101,

'MSP_RAW_IMU':102,

'MSP_SERVO':103,

'MSP_MOTOR':104,

'MSP_RC':105,

'MSP_RAW_GPS':106,

'MSP_COMP_GPS':107,

'MSP_ATTITUDE':108,

'MSP_ALTITUDE':109,

'MSP_ANALOG':110,

'MSP_RC_TUNING':111,

'MSP_PID':112,

'MSP_BOX':113,

'MSP_MISC':114,

'MSP_MOTOR_PINS':115,

'MSP_BOXNAMES':116,

'MSP_PIDNAMES':117,

'MSP_WP':118,

'MSP_BOXIDS':119,

'MSP_SET_RAW_RC':200,

'MSP_SET_RAW_GPS':201,

'MSP_SET_PID':202,

'MSP_SET_BOX':203,

'MSP_SET_RC_TUNING':204,

'MSP_ACC_CALIBRATION':205,

'MSP_MAG_CALIBRATION':206,

'MSP_SET_MISC':207,

'MSP_RESET_CONF':208,

```
'MSP_SET_WP':209,
'MSP_SWITCH_RC_SERIAL':210,
'MSP_IS_SERIAL':211,
'MSP_DEBUG':254,
}
precision = 3
serial_port = get_first_pozyx_serial_port()
if serial_port is not None:
    pozyx = PozyxSerial(serial_port)
    print("Connection success!")
else:
    print("No Pozyx port was found")
################################################
# ATTITUDE(msp)
#       receives: msp altitude message
#       outputs:  prints data in nice format
#       returns:  Angle in X, Y and heading
################################################
def ATTITUDE(msp):
        global angx
        global angy
        global heading
        if str(msp) == "":
                #print(msp_hex)
                #print("Header: " + msp_hex[:6])
                #payload = int(msp_hex[6:8])
                #print("Payload: " + msp_hex[6:8])
                #print("Code: " + msp_hex[8:10])
                #print("RC data unavailable")
```

```
            return
        else:
            msp_hex = msp.encode("hex")
            if msp_hex[10:14] == "":
                #print("angx unavailable")
                return
            else:
                angx = littleEndian(msp_hex[10:14])/10.0

            if msp_hex[14:18] == "":
                #print("angy unavailable")
                return
            else:
                angy = littleEndian(msp_hex[14:18])/10.0

            if msp_hex[18:22] == "":
                #print("head unavailable")
                return
            else:
                heading = littleEndian(msp_hex[18:22])+5
################################################
# RC(msp)
#\t   receives: msp RC message
#\t   outputs:  prints data in nice format
#\t   returns:  Roll/Pitch/Yaw/Throttle
################################################
def RC(msp):
        global roll
        global pitch
```

```
global yaw
global throttle


if str(msp) == "":
        #print(msp_hex)
        #print("Header: " + msp_hex[:6])
        #payload = int(msp_hex[6:8])
        #print("Payload: " + msp_hex[6:8])
        #print("Code: " + msp_hex[8:10])
        #print("RC data unavailable")
        return
else:
        msp_hex = msp.encode("hex")
        if msp_hex[10:14] == "":
                #print("roll unavailable")
                return
        else:
                roll = float(littleEndian(msp_hex[10:14]))
        if msp_hex[14:18] == "":
                #print("pitch unavailable")
                return
        else:
                pitch = float(littleEndian(msp_hex[14:18]))
        if msp_hex[18:22] == "":
                #print("yaw unavailable")
                return
        else:
                yaw = float(littleEndian(msp_hex[18:22]))
```

```
        if msp_hex[22:26] == "":
            #print("throttle unavailable")
            return
        else:
            throttle = float(littleEndian(msp_hex[22:26]))
            return
        #print(str(roll) + " " + str(pitch) + " " + str(yaw) + " " + str(throttle))
################################################
# RAW(msp)
#\t   receives: msp raw message
#\t   outputs:  prints data in nice format
#\t   returns:  ax/ay/az/gx/gy/gz/magx/magy/magz
################################################
def RAW(msp):
    global ax
    global ay
    global az
    global gx
    global gy
    global gz
    global magx
    global magy
    global magz
    if str(msp) == "":
        #print(msp_hex)
        #print("Header: " + msp_hex[:6])
        #payload = int(msp_hex[6:8])
        #print("Payload: " + msp_hex[6:8])
        #print("Code: " + msp_hex[8:10])
```

```
        #print("RC data unavailable")
        return
    else:
        msp_hex = msp.encode("hex")
        if msp_hex[10:14] == "":
            #print("ax unavailable")
            return
        else:
            ax = float(littleEndian(msp_hex[10:14]))

        if msp_hex[14:18] == "":
            #print("ay unavailable")
            return
        else:
            ay = float(littleEndian(msp_hex[14:18]))
        if msp_hex[18:22] == "":
            #print("az unavailable")
            return
        #else:
            az = float(littleEndian(msp_hex[18:22]))

        if msp_hex[22:26] == "":
            #print("gx unavailable")
            return
        else:
            gx = float(littleEndian(msp_hex[22:26]))

        if msp_hex[26:30] == "":
            #print("gy unavailable")
```

```
            return
        else:
            gy = float(littleEndian(msp_hex[26:30]))
        if msp_hex[30:34] == "":
            #print("gz unavailable")
            return
        else:
            gz = float(littleEndian(msp_hex[30:34]))
            return
        #if msp_hex[34:38] == "":
            #print("magx unavailable")
        #       return
        #else:
        #       magx = float(littleEndian(msp_hex[34:38]))

        #if msp_hex[38:42] == "":
            #print("magy unavailable")
        #       return
        #else:
        #       magy = float(littleEndian(msp_hex[38:42]))

        #if msp_hex[42:46] == "":
            #print("magz unavailable")
        #       return
        #else:
        #       magz = float(littleEndian(msp_hex[42:46]))
        #       return
        #print(str(roll) + " " + str(pitch) + " " + str(yaw) + " " + str(throttle))
##############################################################
```

```
# littleEndian(value)
#       receives: a parsed, hex data piece
#       outputs:  the decimal value of that data
#       function: swaps byte by byte to convert little
#                   endian to big endian
#       function: calls 2's compliment to convert to decimal
#       returns:  The integer value
#################################################################
def littleEndian(value):
      length = len(value)     # gets the length of the data piece
      actual = ""
      for x in range(0, length/2):    #go till you've reach the halway point
            actual += value[length-2-(2*x):length-(2*x)]    #flips all of the bytes (the last shall be first)
            x += 1
      intVal = twosComp(actual)       # sends the data to be converted from 2's compliment to int
      return intVal                   # returns the integer value
#####################################################################
# twosComp(hexValue)
#       receives: the big endian hex value (correct format)
#       outputs:  the decimal value of that data
#       function: if the value is negative, swaps all bits
#                   up to but not including the rightmost 1.
#                   Else, just converts straight to decimal.
#                   (Flip all the bits left of the rightmost 1)
#       returns:  the integer value
#####################################################################
def twosComp(hexValue):
      firstVal = int(hexValue[:1], 16)
```

```
    if firstVal >= 8:       # if first bit is 1
        bValue = bin(int(hexValue, 16))
        bValue = bValue[2:]     # removes 0b header
        newBinary = []
        length = len(bValue)
        index = bValue.rfind('1')       # find the rightmost 1
        for x in range(0, index+1):     # swap bits up to rightmost 1
            if x == index:          #if at rightmost one, just append remaining bits
                newBinary.append(bValue[index:])
            elif bValue[x:x+1] == '1':
                newBinary.append('0')
            elif bValue[x:x+1] == '0':
                newBinary.append('1')
            x += 1
        newBinary = ''.join(newBinary)  # converts char array to string
        finalVal = -int(newBinary, 2)   # converts to decimal
        return finalVal

    else:          # if not a negative number, simply convert to decimal
        return int(hexValue, 16)
##############################################################
# askATT()
#       receives: nothing
#       outputs:  nothing
#       function: Do everything to ask the MW for data and save it on globals
#       returns:  nothing
##############################################################
def askATT():
    ser1.flushInput()             # cleans out the serial port
```

```
        ser1.flushOutput()
        ser1.write(MSP_ATTITUDE)        # sends MSP request
        time.sleep(timeMSP_ATT)         # gives adaquate time between MSP TX & RX
        response=ser1.readline()        # reads MSP response
        ATTITUDE(response)              # sends to ATTITUDE to parse and update
global variables
#################################################################
# askRAW()
#   receives: nothing
#   outputs:  nothing
#   function: Do everything to ask the MW for data and save it on globals
#   returns:  nothing
#################################################################
def askRAW():
        ser1.flushInput()               # cleans out the serial port
        ser1.flushOutput()
        ser1.write(MSP_RAW_IMU)         # gets raw data
        time.sleep(timeMSP_RAW)
        response=ser1.readline()
        RAW(response)
#################################################################
# askRC()
#       receives: nothing
#       outputs:  nothing
#       function: Do everything to ask the MW for data and save it on globals
#       returns:  nothing
#################################################################
def askRC():
        ser1.flushInput()               # cleans out the serial port
        ser1.flushOutput()
```

```
        ser1.write(MSP_RC)              # gets RC information
        time.sleep(timeMSP)
        response = ser1.readline()
        RC(response)
#########################################################################
######################## MAIN ###########################################
#########################################################################
# main()
#       receives: -
#       outputs:  -
#       function: opens serial port
#                 loops msp commands
#########################################################################
def main():
        global beginFlag
        global udp_mess2            #Need to be able to modify the value of this
variable
        global gyrod
        global comm
        if drone.ASY:
                print ("Beginning Asyncore UDP server on ")+str(udp_ip)
                AsyncoreServerUDP()
                loop_thread = threading.Thread(target=asyncore.loop, name="Asyncore
Loop")
                loop_thread.start()
        if drone.SCK:
                print ("Beginning regular UDP server on ")+str(udp_ip)
                sock.bind((udp_ip, udp_port))
        if drone.SCKSRV:
                print ("Beginning UDP socketserver on ")+str(udp_ip)
```

```
        server = SocketServer.UDPServer((udp_ip, udp_port),
SocketServerHandler)

        loop_thread = threading.Thread(target=server.serve_forever,
name="SocketServer Loop")

        loop_thread.start()

    print ("Beginning Multiwii - wait 14 seconds...")

    try:

        ser1.open()

    except Exception as e:          # catches any errors with opening serial ports

        print("Error open serial port: "+str(e))

        exit()

    if ser1.isOpen():

        time.sleep(5)          # Gives time for the MultiWii to calibrate and begin
sending live info

        if drone.FILE:

            st =
datetime.datetime.fromtimestamp(time.time()).strftime('%Y_%m_%d+%H-%M-
%S')+".csv"

            file = open("data/"+st, "w")

        if drone.FLYT:

            flytime = timeit.default_timer()


# initialize the data container

who_am_i = SingleRegister()

# get the data, passing along the container

status = pozyx.getWhoAmI(who_am_i)

i=0

Posx = str(0)

Posy = str(0)

Posz = str(0)

position = Coordinates()
```

```
# check the status to see if the read was successful. Handling failure is covered later.
if status == POZYX_SUCCESS:
    # print the container. Note how a SingleRegister will print as a hex string by default.
    print(who_am_i) # will print '0x43'
    ser.open()
    #ser1.open()
    while True:
        timestamp = timeit.default_timer()
        askATT()
        askRC()
        sign = struct.pack('<d', 500)
        AccxNaze = struct.pack('<d', ax)
        AccyNaze = struct.pack('<d', ay)
        AcczNaze = struct.pack('<d', az)
        VxNaze = struct.pack('<d', gx)
        VyNaze = struct.pack('<d', gy)
        VzNaze = struct.pack('<d', gz)
        RollNaze = struct.pack('<d', angx)
        PitchNaze = struct.pack('<d', angy)
        YawNaze = struct.pack('<d', heading)
        ser.flushInput()        # cleans out the serial port
        ser.flushOutput()
        ser.write(sign + angx + angy + angz)
##        time.sleep(0.0025)
```