Controlling UAV by an Embedded Controller

Submitted by:

Hamid Saeed Khan

Supervised by:

Dr. Muhammad Bilal Kadri

THESIS

Submitted to:

Department of Electronic and Power Engineering

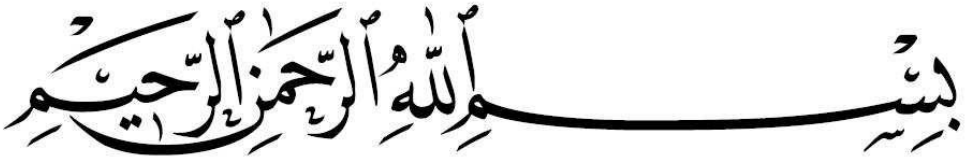
Pakistan Navy Engineering College, Karachi

National University of Science and Technology, Islamabad Pakistan

In fulfillment of requirement for the award of the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

With Specialization in Control Engineering



In the Name of Allāh, the Most Gracious, the Most Merciful

Abstract

The UAV is an acronym for Unmanned Aerial Vehicle, which is an aircraft with no pilot on board. UAVs can be remote controlled aircraft (e.g. flown by a pilot at a ground control station) or can fly autonomously based on pre-programmed flight plans or more complex dynamic automation systems.

Quadrotor UAV is selected for thesis work. Quadrotor is a rotorcraft that has four lift-generating propellers. Two of the propellers spin clockwise and the other two spin counter-clockwise. Control of the machine can be achieved by varying relative speed of the propellers. Quadrotor concept is not new, however, availability of high speed brushless motors and high power to weight ratio Li-polymer battery technology, quadrotors can be successfully designed and fabricated. A quadrotor offers a challenging control problem due to its highly unstable nature. An effective control methodology is therefore needed for such a unique airborne vehicle.

This thesis work presents the mathematical model and PID control of quadrotor. The PID controller is also implemented on embedded system (PSOC device). This embedded controller is tested on quadrotor model with the use of hardware in loop simulation technique. PID controller is a feedback controller, widely used in controls systems. It calculates the error (difference between set point and plant output) and attempts to minimize this error by adjusting the control signal.

This thesis work also presents the non-linear model predictive control of quadrotor. Model Predictive Control is an advanced control strategy, widely used in industries. It is an optimal control strategy that works on the principle of receding horizon. It predicts the plant output for the prediction horizon by using the dynamic model of the plant and attempts to find the optimal control signal with the help of optimization scheme.

Acknowledgement

First and foremost, I am very much grateful to ALLAH, the Almighty, who showers his blessings upon us and enable me to complete my research work.

I am deeply grateful to my research supervisor Dr. Muhammad Bilal Kadri, Assistant Professor, Department of Electronic and Power Engineering, Pakistan Navy Engineering College, NUST. I feel much honored to have him as my supervisor. His support and guidance helped me a lot throughout the research work. Without him, I would have never been able to complete my research work.

I am also very grateful to my GEC committee, comprises of the following faculty members, for providing guidance throughout my research period:

- Cdr Dr. Attaullah Memon PN
- Cdr Dr. Syed Sajjad Haider Zaidi PN
- Cdr Dr. Tariq Mairaj Rasool Khan PN

I also want to thank my family, especially my mother, for their prayers, love, and encouragement.

Contents

CHAPTER 1 INTRODUCTION

1 Introduction

1.1 Background

Unmanned air vehicle, commonly known as UAV, is a flying machine, which requires no pilot on board. It is controlled by using remote control from the ground station or it can be pre-programmed for the autonomous flight. The UAVs are mostly used for the military purposes, i.e. surveillance and also for attacking the enemy on ground. But UAVs have also been started for the use of civil purposes like in fire fighting operations, for surveillance of important installations.

There are two types of UAVs exist.

1. Fixed Wing UAVs
2. Rotary Wing UAVs

Fixed wing UAVs are like airplanes controlled from the ground station through radio link as shown in figure 1.1.

Figure 1.1: MQ-9 Reaper UAV

MQ-9 Reaper UAV is a fixed wing uav used by US airforce as discussed in [1].

Rotary wing UAVs are like helicopters without pilot on board, controlled by ground station using radio link as shown in figure 1.2. Rotary wing UAVs are capable of vertical take-off and landing (VTOL), requires no runway to fly.

Figure 1.2: MQ-8 Fire Scout

MQ-8 fire scout is a rotary wing UAV used by US Navy [2].

For this thesis, quadrotor is selected for research work. Quadrotor is a rotary wing uav, capable of vertical take-off and landing (VTOL) as shown in figure 1.3.

Figure 1.3: Parrot AR Drone

Parrot AR Drone is a commercially available quadrotor at [3].

As shown in figure 1.3, quadrotor has four lift generating propellers. Two of these propellers rotates clockwise and the other two rotates counter clockwise. By varying the speed of these propellers, the quadrotor attitude (roll angle, pitch angle and yaw angle), altitude and position

can be controlled. Due to the MIMO structure and very fast and complex dynamics, quadrotor controlling is a very challenging task and it requires a very sophisticated control scheme, which can also be embed on some embedded device.

1.2 Objectives of thesis

The main objectives of the thesis are to develop the controller of quadrotor, implement this controller on PSOC embedded device. The following are the sub-objectives for the thesis.

- In depth study of PSOC embedded device, hardware in loop simulation technique, system identification and model predictive controls.
- Mathematical modeling of quadrotor.
- PID control of quadrotor.
- Implement discrete PI controller on 8051 micro-controller and PSOC device.
- Test these embedded controllers on DC motor speed model with hardware-in-loop simulation technique.
- Implement PID controller of quadrotor on PSOC device and test this controller on quadrotor plant model with hardware-in-loop simulation technique.
- System identification and non-linear MPC control of quadrotor.

1.3 Organization of thesis

The thesis has been organized into seven chapters.

Chapter 1: presents the introductory background, objectives and organization of the thesis.

Chapter 2: presents the literature review related to control of quadrotor, hardware in loop simulation technique.

Chapter 3: presents the discrete PID control strategy for both inner loop and outer loop control of quadrotor.

Chapter4: presents the implementation of simple discrete PI controller on microcontroller and PSOC device and test on DC motor plant model with HIL. It also presents the outer loop control strategy of quadrotor implementation on PSOC and test on quadrotor plant model with HIL.

Chapter 5: presents the system identification and non-linear MPC control of quadorotor.

Chapter 6: presents the conclusion of the thesis.

CHAPTER 2 LITERATURE REVIEW

2 Literature Review

This chapter presents the research findings already carried out on quadrotor control and hardware in loop simulations.

2.1 Quadrotor Control

Quadrotor is an unstable system and many control techniques have been applied on it's controlling. Some of the techniques only stabilizes the quadrotor by controlling the attitude (roll angle, pitch angle and yaw angle). This type of control is called the "inner loop" control of quadrotor. Some techniques also control the position of the quadrotor in 3-dimensional space. This type of control is called the "outer loop" control of quadrotor. The majority of the quadrotor work is on simulations but some of the works also have been implemented on actual quadrotor hardware. These control techniques include linear, non-linear and also the artificial intelligence techniques like neural networks and fuzzy logic. Some of the research findings are discussed below.

2.1.1 Linear Control

In [4], linear control successfully stabilizes the prototype quadrotor X-4 Flyerhad, attached to a test platform, in the presence of step disturbances. Later a new Mark II prototype was tested by the same group without disturbances [5]. STARMAC-II prototype achieved free flight hovering using PID controls [6]. The control of this flight cause to fail in the presence of wind disturbances. Later the STARMAC-II team achieved outdoor path following [7]. Another prototype achieved autonomous flight with linear control, in the presence of small disturbances [8]. PID and LQ controllers were implemented and regulate the system in [9]. In [10], PD^2 feedback control is proposed with quaternion based feedback for the exponential attitude stabilization of quadrotor. In [11], switching model predictive attitude controller was implemented. It uses the piecewise affine models of the quadrotor and linear MPC controllers were computed for each piecewise affine model. The switching between these controllers was

governed by rate of rotation angles. The results were good on experimental test bed in the presence of wind disturbances.

2.1.2 Non-Linear Control

Linear control techniques are capable of stabilizing the quadrotor but non-linear control techniques can expand the region of angles that can be achieved for quadrotor. In [13], HMX-4 quadrotor used feedback linearization technique to achieve control. It uses state inputs from the camera. Integral sliding mode control with reinforcement learning is used to achieve multi agent control of quadrotor [14]. [15, 16] achieved the formation control by sliding mode controller and focused on obstacle avoidance by extracting state variables from Kalman filter. [17] developed backstepping controller with observer for quadrotor. [18] proposed a vision based control scheme which performs visual servo control by using a fixed target camera for hovering the quadrotor. In [19], Draganfly II quadrotor uses a pre-trained neural network for stabilizing the quadrotor in hover state without disturbances. Adaptive neural network controls have also successfully stabilized the quadrotor in simulations [20, 21].

2.2 Hardware in loop Simulation

Hardware in loop simulation is a technique used in control systems for analyzing the behavior of real hardware in close loop control. In HIL, there can be controller in simulation environment with real plant hardware in actual world. This method is used for the tuning of controller parameters on actual plant hardware. In other type of HIL, controller is on actual hardware (some embedded processor) with plant dynamic model in simulation environment. This method is used because sometimes the actual plant hardware is not available for testing or it is too costly. This method helps the controller designers to test the behavior of their designed controller before testing on actual plant hardware. Many researchers have used both type of HIL. Some of them are discussed below.

In [22], hardware in simulation technique is used for online identification of squirrel cage induction motor with using ARMA model and recursive least square algorithm. It also performed online controller parameters tuning. HIL simulation is used for testing engine control system hardware with dynamic model of diesel engine in simulation environment [23]. HIL simulation is used for testing the actual controller hardware on automatic gearbox model of a passenger car [24]. In [25], HIL simulation is used for controlling permanent magnet synchronous motor drive model by controller on actual hardware. HIL simulation is used for designing pareto-optimal controller for actual electric motor speed control with multi objective optimization algorithm [26]. HIL simulation is used for online PID controller tuning and fuzzy logic controller designing of DC motor motion control platform by using multi objective evolutionary methods [27].

2.3 Summary

In this chapter, a brief overview of the research work is presented related to the control of quadrotor and hardware-in-loop simulations. Linear and non-linear control techniques for quadrotor are discussed. It clearly explained the superiority of non-linear controls. Both type of hardware-in-loop simulation work is presented.

CHAPTER 3 PID Control of Quadrotor

3 PID Control of Quadrotor

Quadrotor is an un-manned aerial vehicle, capable of vertical take-off and landing (VTOL) and hover. It is an open loop unstable system and it requires some control strategy for its stable flight. This chapter presents discrete PID control of quadrotor. It presents both inner loop and outer loop control of quadrotor.

3.1 Quadrotor Dynamics

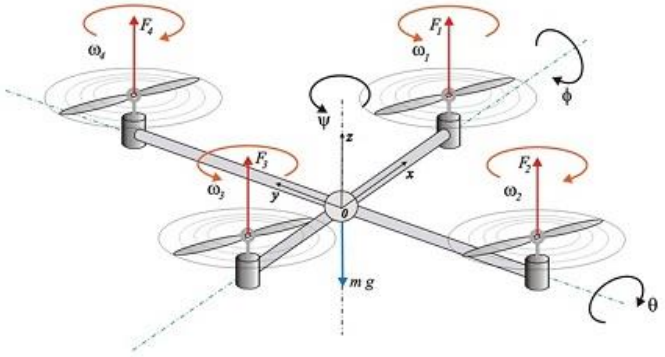


Figure 3.1: Quadrotor Diagram

As shown in figure 3.1 [34], Quadrotor has four lift generating propellers. Two propellers rotate clockwise and the other two rotates counter-clockwise. Quadrotor control is achieved by varying the propellers angular speed Ω_i (i = 1, 2, 3, 4).

Let (a) the rotation angles of quadrotor are roll angle (ø), pitch angle (θ) and yaw angle (ψ) and (b) the translational-vector movement of quadrotor centre of mass is [x, y, z].

The mathematical model of quadrotor [12] is:

$$\begin{aligned}
\ddot{\phi} &= \dot{\theta}\dot{\psi}\frac{I_{yy}-I_{zz}}{I_{xx}} + \dot{\theta}\frac{J_r}{I_{xx}}\Upsilon + \frac{L}{I_{xx}}U_2 \\
\ddot{\theta} &= \dot{\phi}\dot{\psi}\frac{I_{zz}-I_{xx}}{I_{yy}} - \dot{\phi}\frac{J_r}{I_{yy}}\Upsilon + \frac{L}{I_{yy}}U_3 \\
\ddot{\psi} &= \dot{\theta}\dot{\phi}\frac{I_{xx}-I_{yy}}{I_{zz}} + \frac{1}{I_{zz}}U_4 \\
\ddot{x} &= \frac{(\cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi)}{M}U_1 \\
\ddot{y} &= \frac{(\cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi)}{M}U_1 \\
\ddot{z} &= -g + \frac{(\cos\phi\cos\theta)}{M}U_1
\end{aligned} \tag{3.1}$$

Where M is the mass of the system, g is the acceleration due to gravity. U_i (i = 1, 2, 3, 4) and Υ are the control signals that are dependent on the propellers angular speed Ω_i (i = 1, 2, 3, 4).

The control signals are calculated as:

$$\begin{aligned}
U_1 &= b\Omega_1^2 + b\Omega_2^2 + b\Omega_3^2 + b\Omega_4^2 \\
U_2 &= b\Omega_4^2 - b\Omega_2^2 \\
U_3 &= b\Omega_3^2 - b\Omega_1^2 \\
U_4 &= d(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \\
\Upsilon &= \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4
\end{aligned} \tag{3.2}$$

The control signal U_1 is related to total thrust of the quadrotor. U_2, U_3 are related to roll angle (ø) and pitch angle (θ) respectively. U_4 is related to yaw angle (ψ). Υ is the residual propeller angular speed.

Table 3.1 defines the parameters used in model equations.

Parameter Symbols	Parameter Description
I_{xx}	x-axis inertia component
I_{yy}	y-axis inertia component
I_{zz}	z-axis inertia component
L	Length of the quadrotor arm
M	Mass of quadrotor
b	Thrust co-efficient
d	Drag co-efficient
J_r	Rotor inertia

Table 3.1: Quadrotor model parameters

The parameter values used here are given in table 3.2 [12]:

Parameter	Value	Units
M	0.8	Kg
L	0.3	m
J_r	6.01×10^{-5}	Kg m^2
I_{xx}	15.67×10^{-3}	Kg m^2
I_{yy}	15.67×10^{-3}	Kg m^2
I_{zz}	28.346×10^{-3}	Kg m^2
b	192.3208×10^{-7}	N s^2
d	4.003×10^{-7}	Nm s^2

Table 3.2: Quadrotor parameter values

3.2 PID Controller

PID controller is a linear feedback controller, which works on error (reference – plant output) and tries to minimize this error as shown in figure 3.2 [35]. It is most widely used controller in industry. PID controller exist both in continuous and discrete form. It can be implemented by using both analog and digital electronic devices.

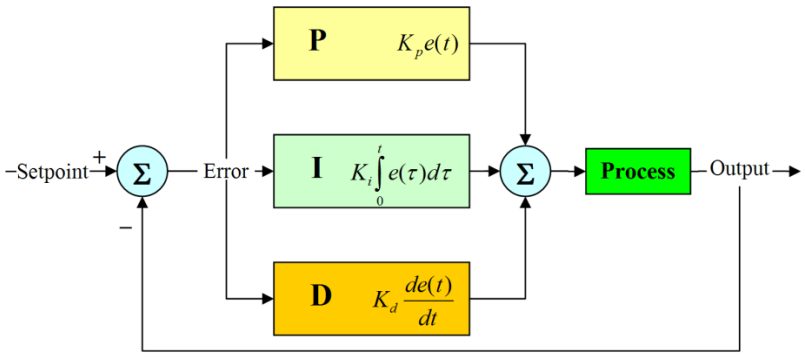


Figure 3.2: Continuous PID controller in close loop

3.2.1 Continuous PID Controller

As shown in figure 3.2, the control signal generated by PID controller is the sum of proportional, Integrator and derivative terms. The time domain equation of PID controller is:

$$u(\mathrm{t}) = K_p e(\mathrm{t}) + K_i \int_0^t e(\tau) d_\tau + K_d \frac{de(\mathrm{t})}{dt} \tag{3.3}$$

Where u(t) is the control signal, e(t) is the error signal, which is difference in reference signal and output. K_P, K_i and K_d are the proportional, integrator and differentiator gains respectively. These gains need to be tuned for required performance of close loop system. The laplace domain equation of PID controller is:

$$U(\mathrm{s}) = K_p E(\mathrm{s}) + \frac{K_i E(\mathrm{s})}{s} + K_d s E(\mathrm{s}) \tag{3.4}$$

The PI and PD controllers are also used. The PI controller equations are:

$$u(\mathrm{t}) = K_p e(\mathrm{t}) + K_i \int_0^t e(\tau) d_\tau \tag{3.5}$$

$$U(\mathrm{s}) = K_p E(\mathrm{s}) + \frac{K_i E(\mathrm{s})}{s} \tag{3.6}$$

The PD controller equations are:

$$u(\mathrm{t}) = K_p e(\mathrm{t}) + K_d \frac{de(\mathrm{t})}{dt} \tag{3.7}$$

$$U(\mathrm{s}) = K_p E(\mathrm{s}) + K_d s E(\mathrm{s}) \tag{3.8}$$

3.2.2 Discrete PID Controller

As discussed above, the PID controller can be implemented on digital device with its discrete time equations. These equations are obtained by applying Forward Euler's method [36] on continuous PID equations.

The discrete PID controller equations are:

$$\begin{aligned} u[\mathrm{k}] = {} & (K_p + K_d\,\mathrm{N})\,\mathrm{e}[\mathrm{k}] + (\mathrm{NT}_s K_p + K_i \mathrm{T}_s - 2K_d\,\mathrm{N} - 2K_p)\,\mathrm{e}[\mathrm{k}-1] + (K_p - K_p\,\mathrm{NT}_s \\ & - K_i \mathrm{T}_s + K_i\,\mathrm{NT}_s^2 + K_d\,\mathrm{N})\,\mathrm{e}[\mathrm{k}-2] - (\mathrm{NT}_s - 2)\,\mathrm{u}[\mathrm{k}-1] - (1 - \mathrm{NT}_s)\,\mathrm{u}[\mathrm{k}-2] \end{aligned} \tag{3.9}$$

$$U(\mathrm{z}) = K_p E(\mathrm{z}) + K_i \mathrm{T}_s \frac{1}{z-1} E(\mathrm{z}) + K_d \frac{\mathrm{N}}{1 + \mathrm{NT}_s \dfrac{1}{z-1}} E(\mathrm{z}) \tag{3.10}$$

The discrete PI controller equations are:

$$u[\mathrm{k}] = K_p\,\mathrm{e}[\mathrm{k}] + (K_i \mathrm{T}_s - K_p)\,\mathrm{e}[\mathrm{k}-1] + \mathrm{u}[\mathrm{k}-1] \tag{3.11}$$

$$U(\mathrm{z}) = K_p E(\mathrm{z}) + K_i \mathrm{T}_s \frac{1}{z-1} E(\mathrm{z}) \tag{3.12}$$

The discrete PD controller equations are:

$$u[\mathrm{k}] = (K_p + K_d\,\mathrm{N})\,\mathrm{e}[\mathrm{k}] + (\mathrm{NT}_s K_p - K_d\,\mathrm{N} - K_p)\,\mathrm{e}[\mathrm{k}-1] - (\mathrm{NT}_s - 1)\,\mathrm{u}[\mathrm{k}-1] \tag{3.13}$$

$$U(\mathrm{z}) = K_p E(\mathrm{z}) + K_d \frac{\mathrm{N}}{1 + \mathrm{NT}_s \dfrac{1}{z-1}} E(\mathrm{z}) \tag{3.14}$$

Equations 3.9, 3.11, 3.13 represents the controllers in time domain and equations 3.10, 3.12, 3.14 are in frequency domain (Z-domain). T_s is the sampling time and N is the filter coefficient.

3.3 Inner Loop Control of Quadrotor

This section presents the inner loop control of quadrotor i.e. roll angle, pitch angle and yaw angle. Figure 3.3 shows the simulation layout.

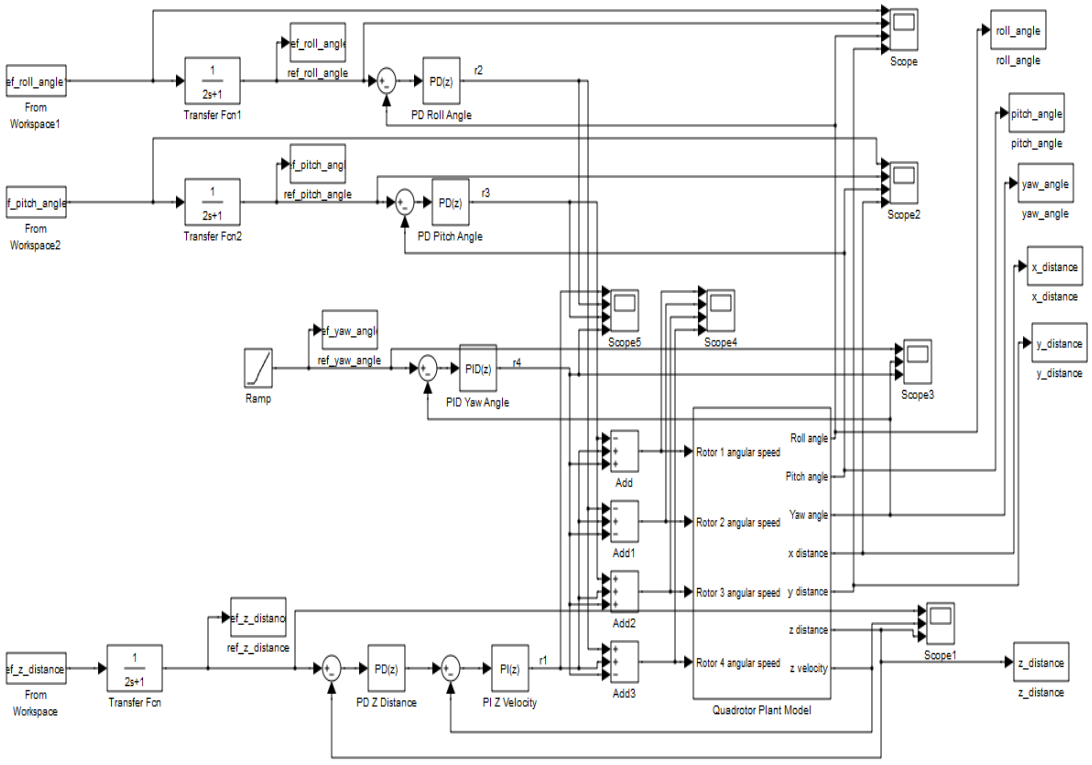


Figure 3.3: Inner loop control of quadrotor

Z distance and Z velocity control is part of outer loop control but it needs to be included here. Achieving inner loop control is baseless without hover the quadrotor at some height.

As shown in figure 3.3, PD control is used for roll angle and pitch angle controlling and PID is used for yaw angle controlling. r_i (i=1, 2, 3, 4) are the control signals generated by controllers

and these signals form the final propeller angular speed signals for quadrotor input with following relations.

$$\begin{aligned}\Omega_1 &= r_1 - r_3 + r_4 \\ \Omega_2 &= r_1 - r_2 - r_4 \\ \Omega_3 &= r_1 + r_3 + r_4 \\ \Omega_4 &= r_1 + r_2 - r_4\end{aligned} \tag{3.15}$$

r1 is related to vertical distance control of quadrotor from ground. r2, r3 and r4 are related to roll angle, pitch angle and yaw angle control respectively. The gains and other parameters used in controllers are given in table 3.3.

Controller	Parameters
Roll angle PD Controller	K_p=5, K_d=5, N=100, T_s=0.01sec
Pitch angle PD Controller	K_p=5, K_d=5, N=100, T_s=0.01sec
Yaw angle PID Controller	K_p= 0.5, K_d=2, K_i=0.01, N=100, T_s=0.01sec
Z velocity PI Controller	K_p= 5, K_i=5, T_s=0.01sec
Z distance PD Controller	K_p=20, K_d=20, N=100, T_s=0.01sec

Table 3.3: Inner loop controller's parameters

3.3.1 Results

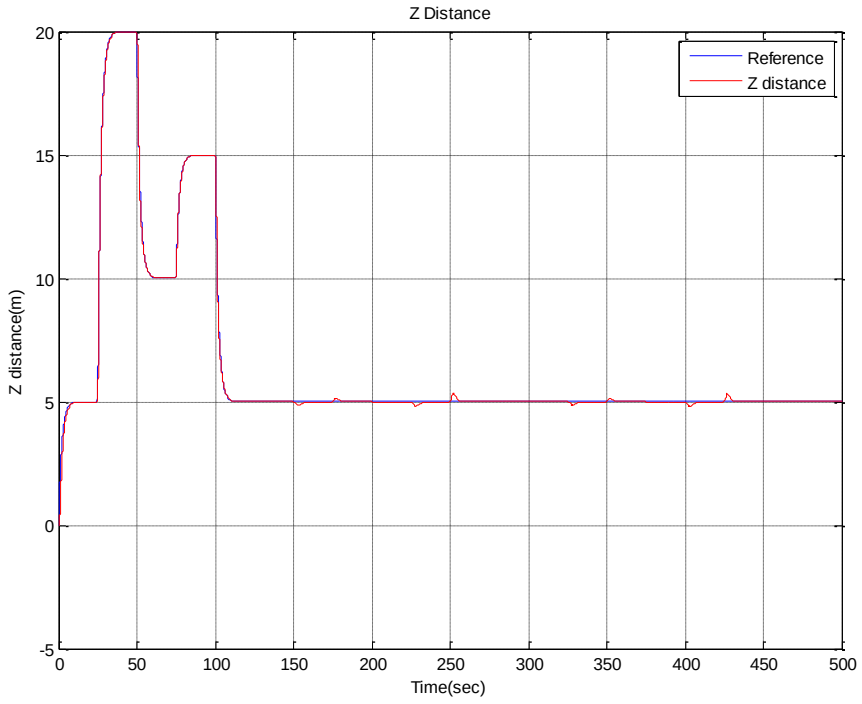


Figure 3.4: Height control of quadrotor

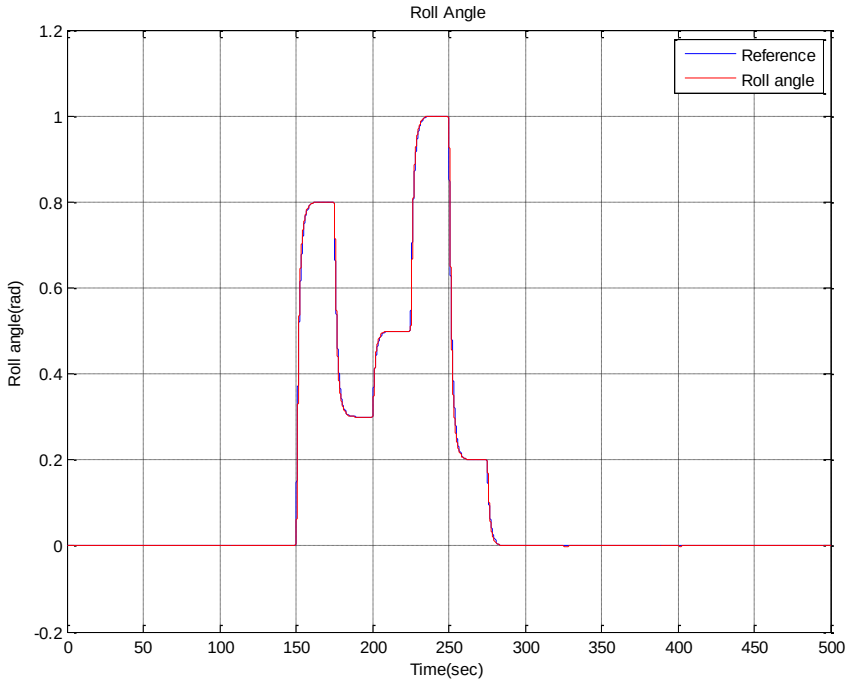


Figure 3.5: Roll angle control of quadrotor

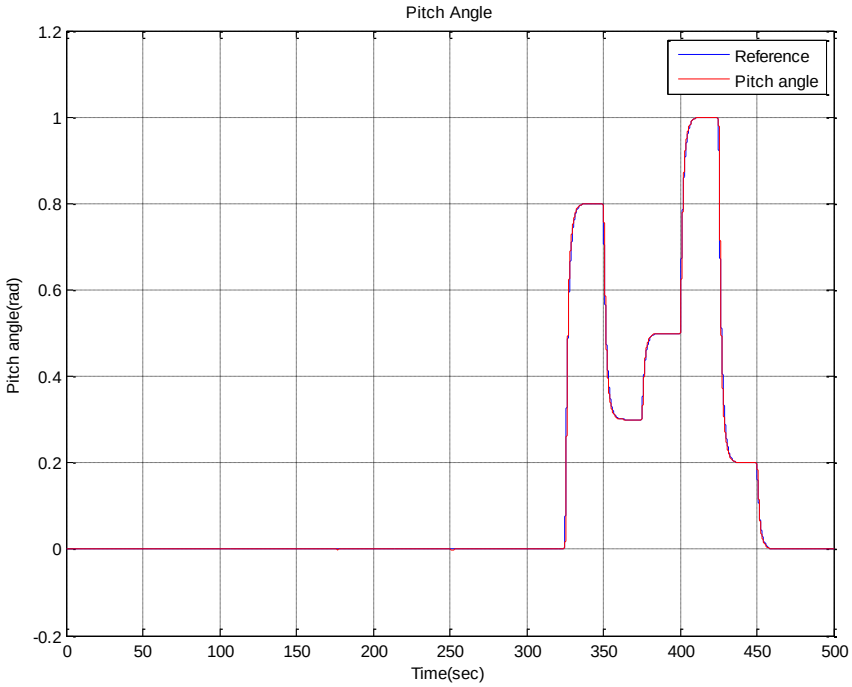


Figure 3.6: Pitch angle control of quadrotor

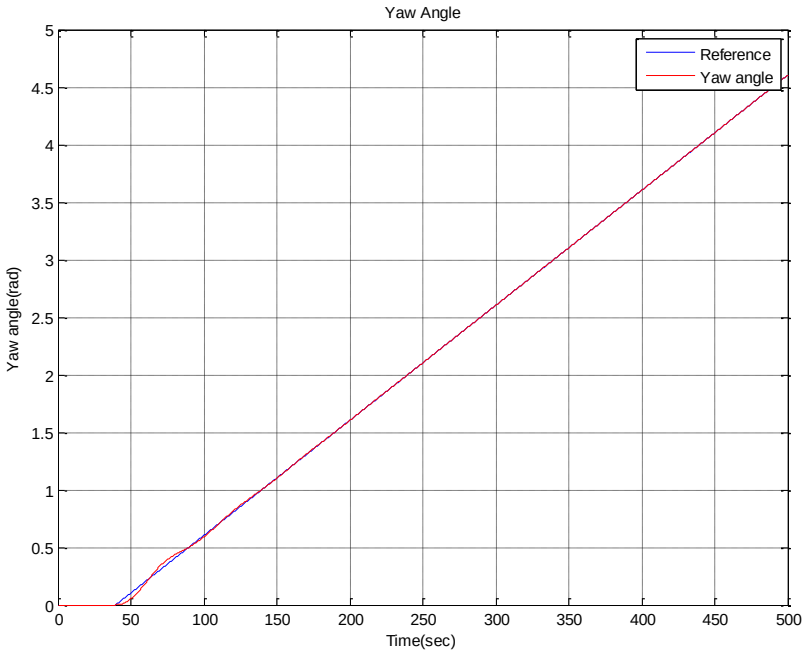


Figure 3.7: Yaw angle control of quadrotor

Figure 3.4 - 3.7 clearly shows the effectiveness of controller. It can be seen that the controller is able to track different level of reference signals for roll angle, pitch angle, yaw angle and height of quadrotor. The control signals (propellers angular speed) generated by control strategy are given below:

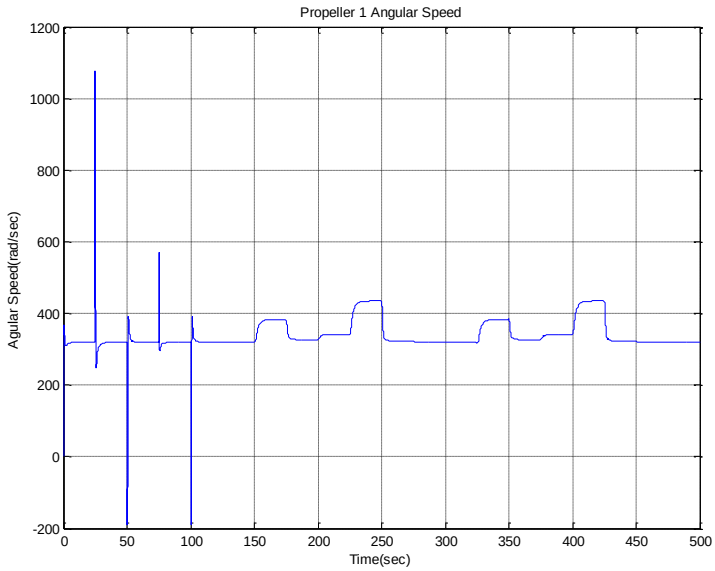


Figure 3.8: Propeller 1 Angular Speed

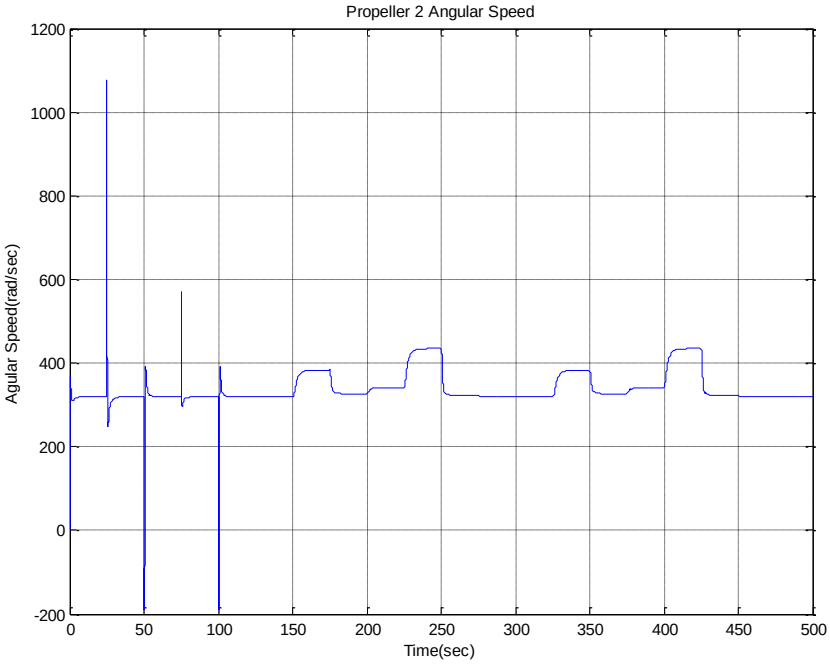


Figure 3.9: Propeller 2 Angular Speed

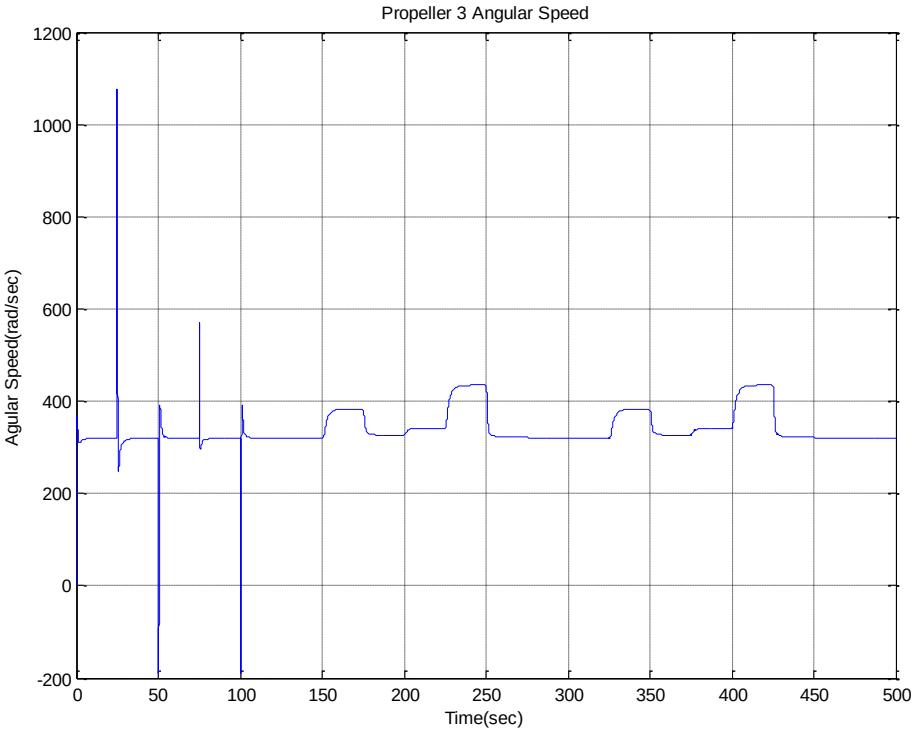


Figure 3.10: Propeller 3 Angular Speed

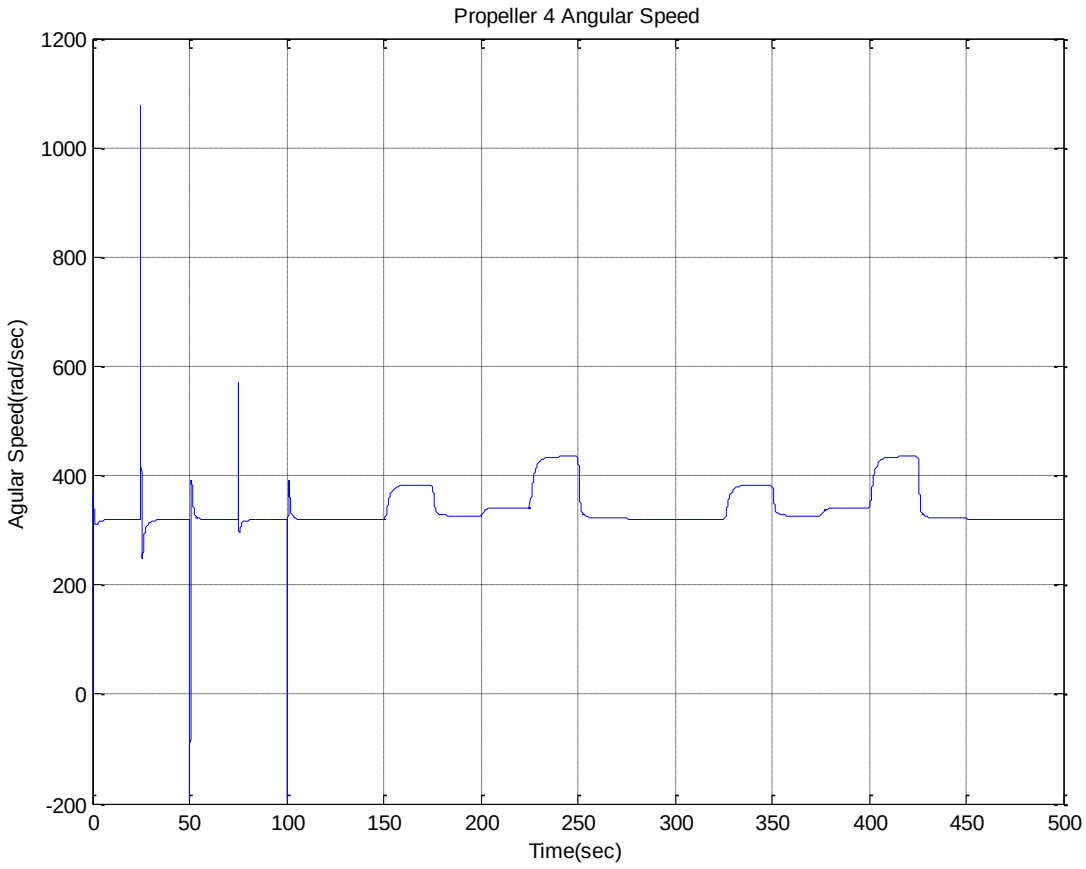


Figure 3.11: Propeller 4 Angular Speed

3.4 Outer Loop Control of Quadrotor

This section presents the outer loop control of quadrotor i.e. X-distance, Y-distance and Z-distance (Height). Figure 3.12 shows the simulation layout.

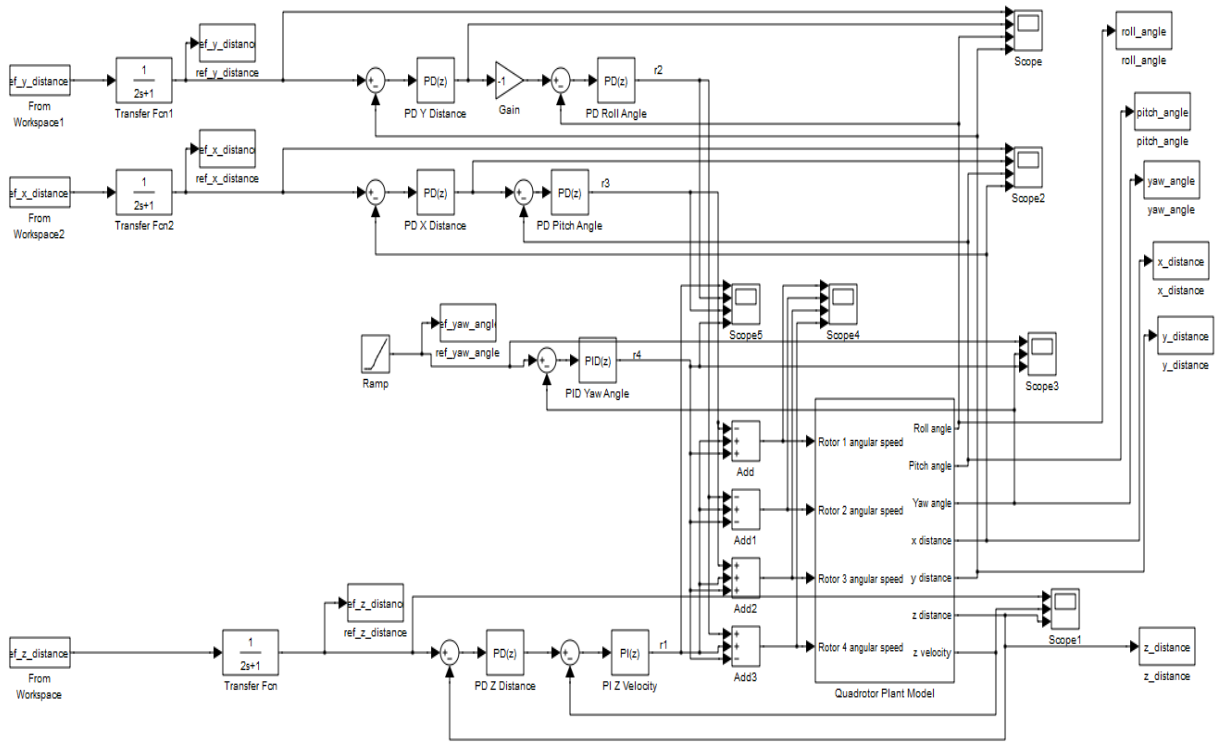


Figure 3.12: Outer loop control of quadrotor

As shown in figure 3.12, the whole control strategy is similar to inner loop control, except the two PD control loops for X-distance and Y-distance control.

The control signal of PD Y distance controller becomes the reference signal for PD roll angle controller. There is a gain block of -1 value, due to opposite relation between Y distance and roll angle i.e. if roll angle is positive, Y distance increase in negative and vice versa. The control signal of PD X distance controller becomes the reference signal for PD pitch controller.

The gains and other parameters used in controllers are given in table 3.4.

Controller	Parameters
Roll angle PD Controller	K_p=5, K_d=5, N=100, T_s=0.01sec
Pitch angle PD Controller	K_p=5, K_d=5, N=100, T_s=0.01sec
Yaw angle PID Controller	K_p= 0.5, K_d=2, K_i=0.01, N=100, T_s=0.01sec
Z velocity PI Controller	K_p= 5, K_i=5, T_s=0.01sec
Z distance PD Controller	K_p=20, K_d=20, N=100, T_s=0.01sec
Y distance PD Controller	K_p=0.01, K_d=0.05, N=100, T_s=0.01sec
X distance PD Controller	K_p=0.01, K_d=0.05, N=100, T_s=0.01sec

Table 3.4: Inner loop and Outer loop controller's parameters

3.4.1 Results

The Y-distance, X-distance and Z-distance results are given below. As shown in figure 3.12, the Y distance loop is link with roll angle loop and X distance loop is link with pitch angle loop. This relation can also be seen in figures 3.14 and 3.16, where roll angle and pitch angle becomes zero when Y distance and X distance becomes steady. Figures 3.13 – 3.17 clearly shows the effectiveness of controller. The Y-distance, X-distance and Z-distance are achieved for their given references.

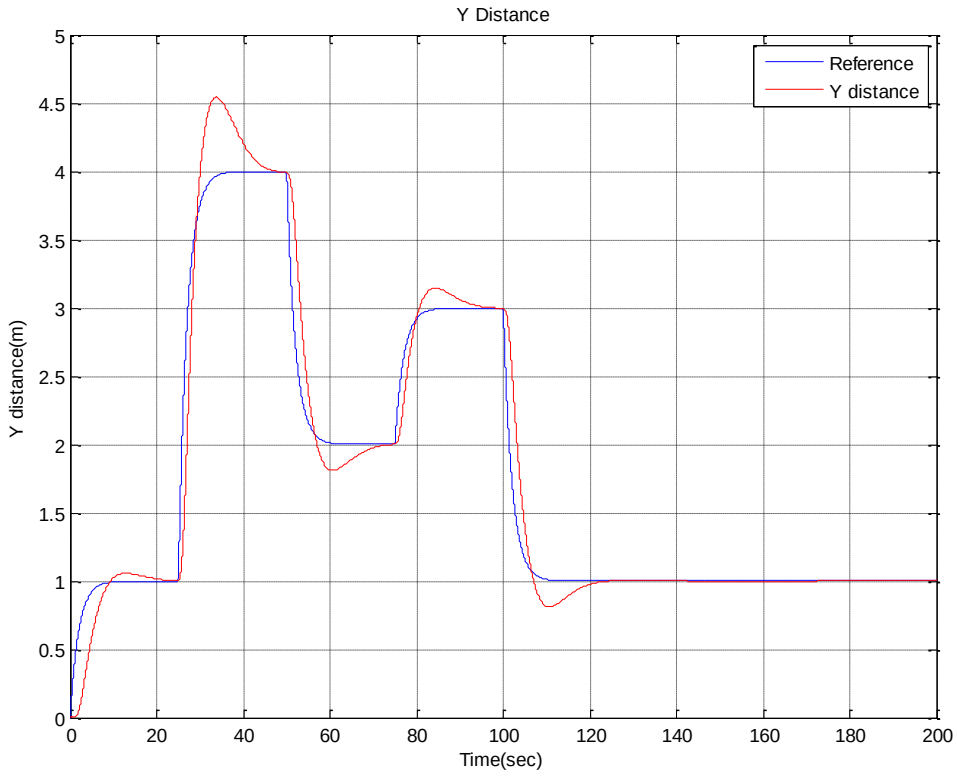


Figure 3.13: Y distance control of quadrotor

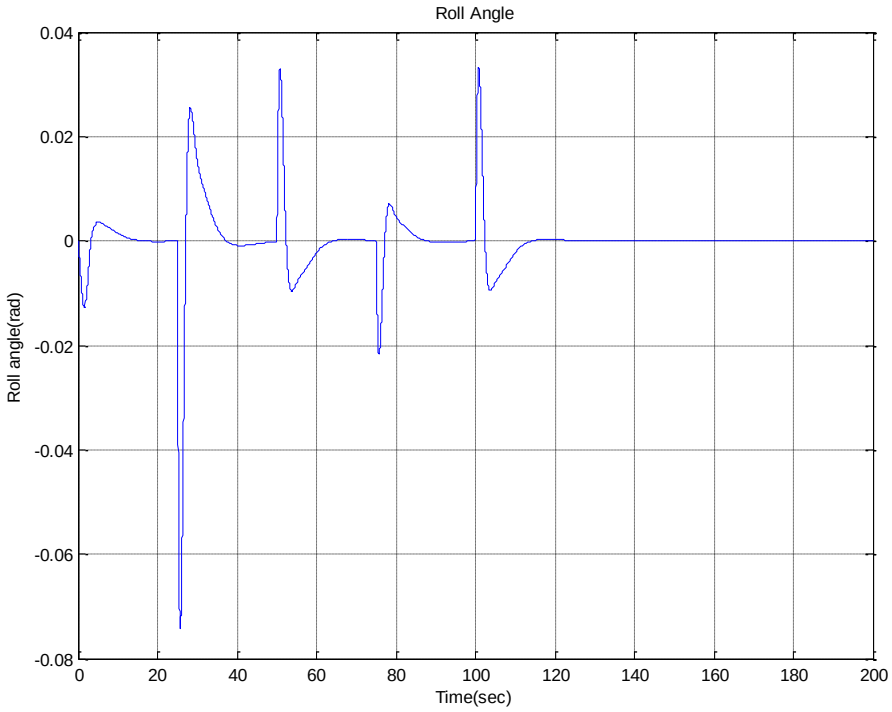


Figure 3.14: Roll angle

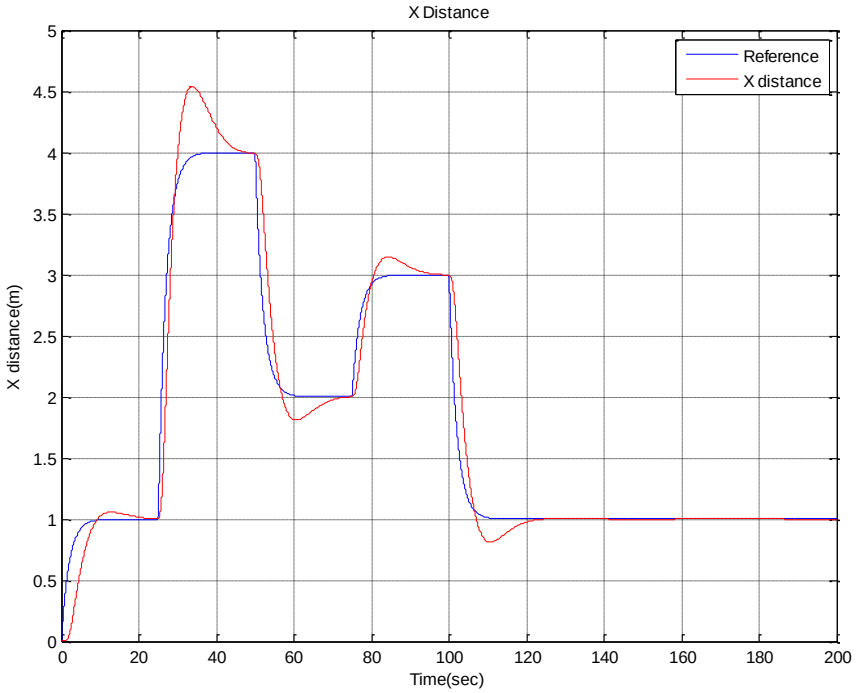


Figure 3.15: X distance control of quadrotor

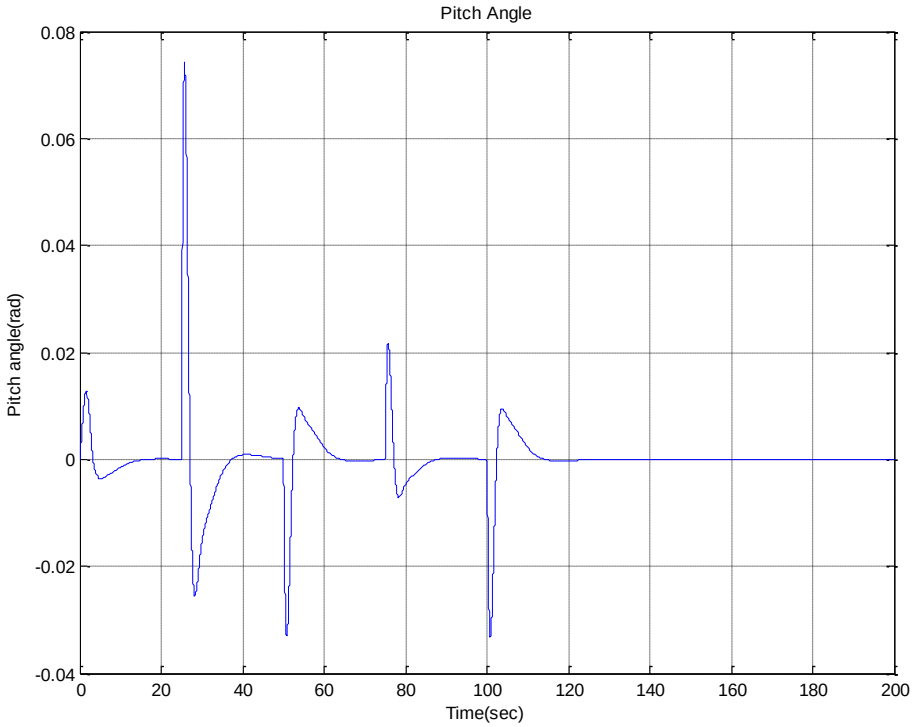


Figure 3.16: Pitch angle

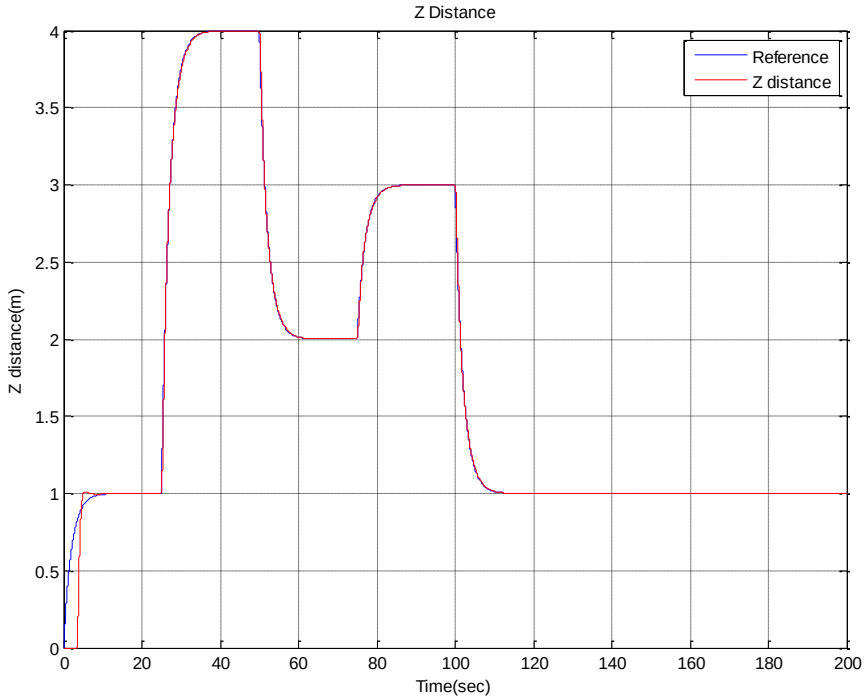


Figure 3.17: Z distance control of quadrotor

The control signals (propellers angular speed) are given below.

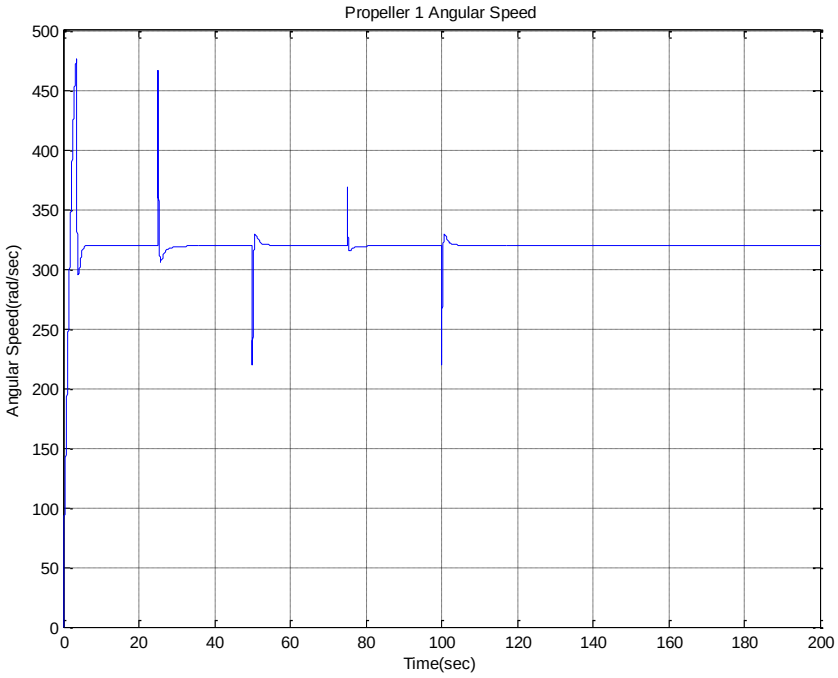


Figure 3.18: Propeller 1 Angular Speed

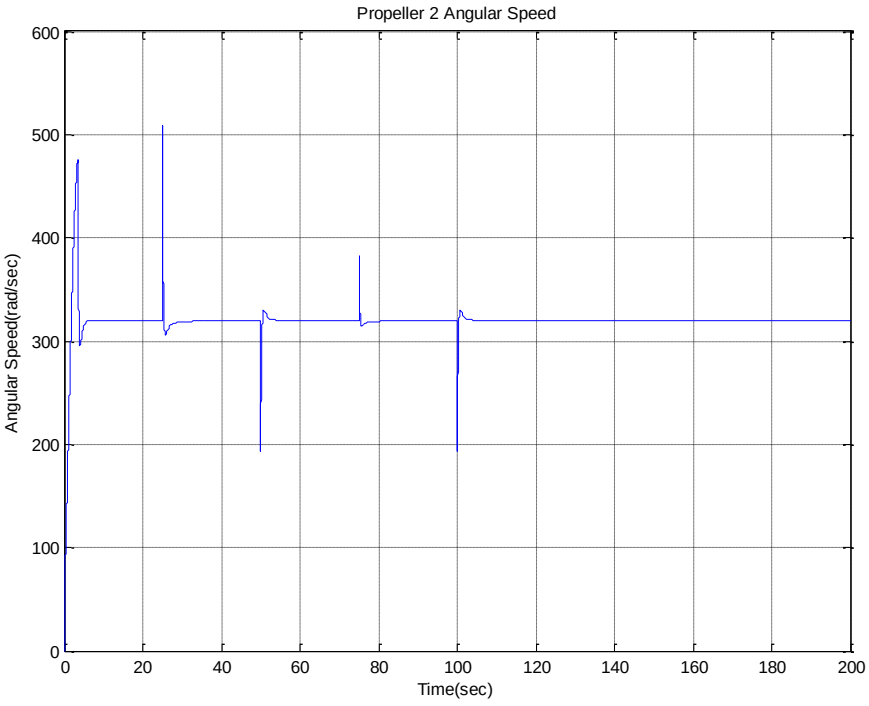


Figure 3.19: Propeller 2 Angular Speed

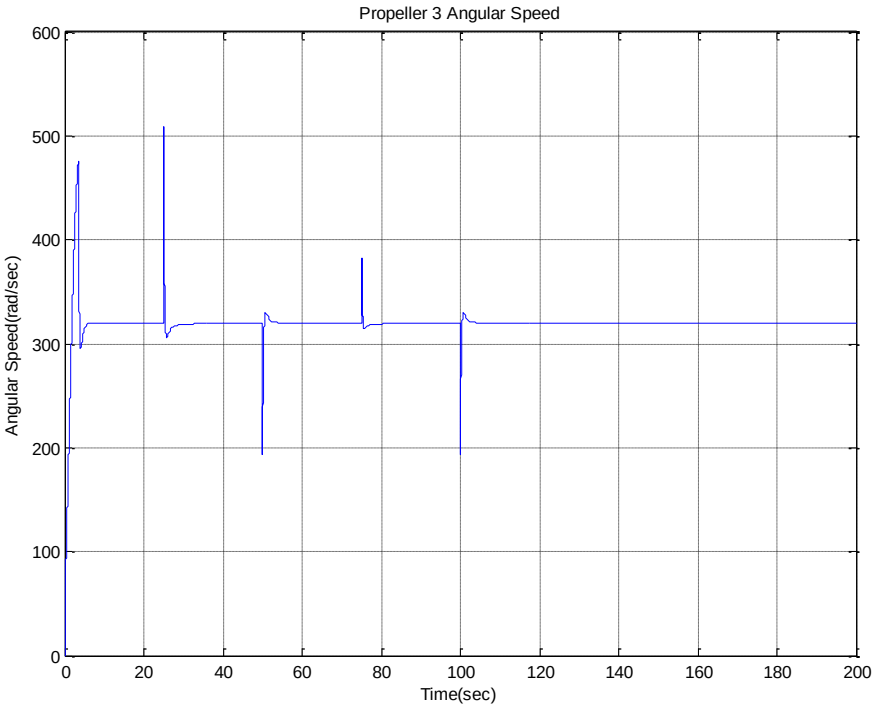


Figure 3.20: Propeller 3 Angular Speed

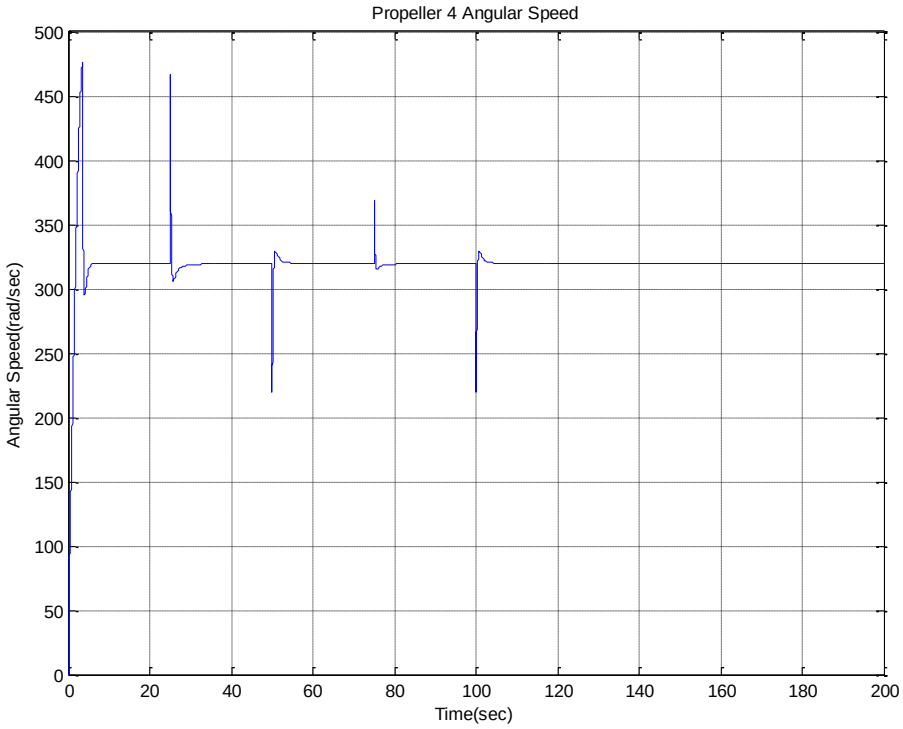


Figure 3.21: Propeller 4 Angular Speed

This control strategy is able to move quadrotor to a given reference trajectory in 3-dimensional space as shown in figure 3.22 and 3.23.

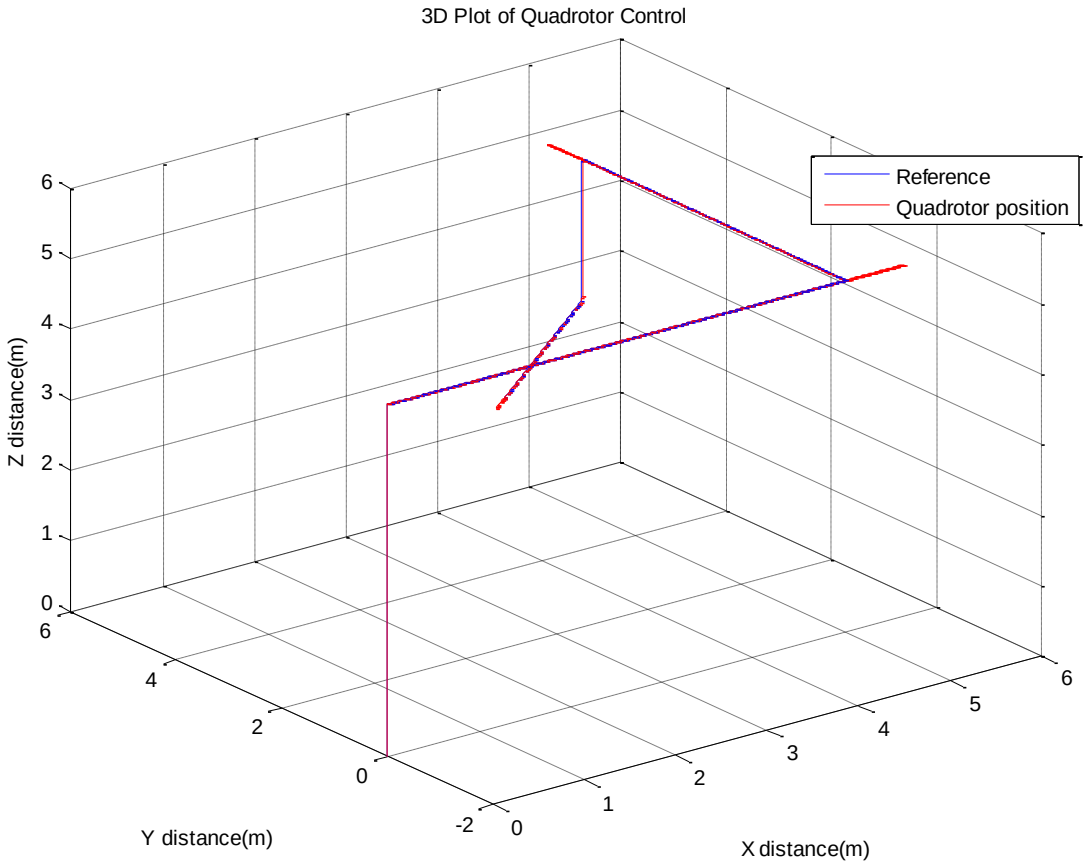


Figure 3.22: 3D Plot of Quadrotor Control

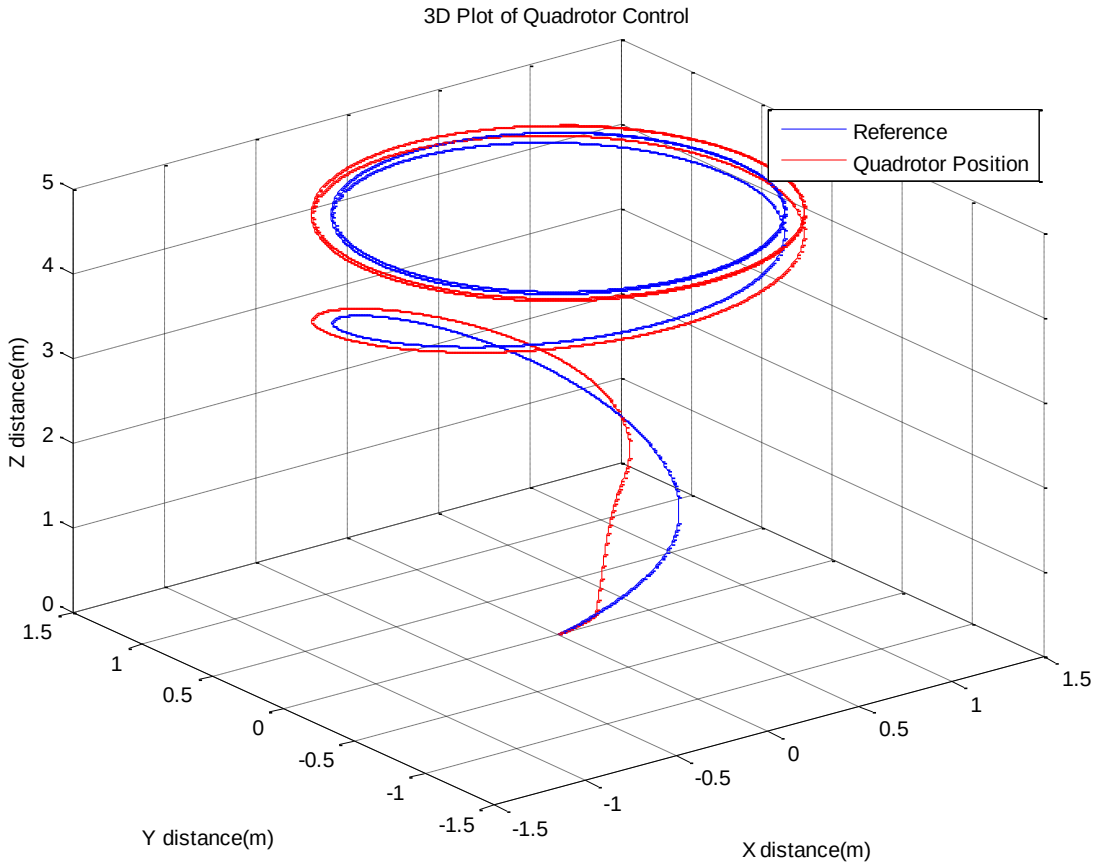


Figure 3.23: 3D Plot of Quadrotor Control

3.5 Summary

This chapter has presented the quadrotor mathematical model with the parameters used in simulations. The continuous and discrete PD, PI and PID controllers are also discussed. The inner loop and outer loop control is presented with the use of discrete PD, PI and PID controllers. The simulation results have clearly shown the effectiveness of the controllers.

CHAPTER 4 Embedded Control of Quadrotor

4 Embedded Control of Quadrotor

Once the controller is designed, it needs to be implemented on some embedded device like microprocessor, microcontroller or FPGA. Then this embedded controller runs in close loop with plant. This chapter presents the implementation of simple discrete PI controller on 8051 microcontroller and PSOC device, and these embedded controllers runs on DC motor speed plant model by using hardware-in-loop simulation technique. The outer loop control strategy based on discrete PI, PD and PID controllers is also implemented on PSOC device, and runs on quadrotor simulink model by using hardware-in-loop simulation technique.

4.1 DC Motor Speed Control

The conventional controllers such as PD, PI and PID have been widely used in industry and have demonstrated good control performance for different industrial plants. These conventional controllers have always been the very first choice for controlling any industrial plant due to its simple structure. The conventional controllers have been implemented in industry by analog circuits. These analog circuits impose some limitations, when there is a need for retuning the controller. Sometimes, the complete hardware of the controller needs to be modified. Today, a wide variety of embedded processors are available and these processors can be used to implement the conventional controllers. This section presents an implementation of digital PI controller on an 8-bit microcontroller and PSOC device. These PI controllers are tested for the plant model of DC motor by hardware-in-loop simulation technique. In these HIL simulations, the PI controller is on implemented on a microcontroller or PSOC and the plant model is in simulation environment.

4.1.1 Control Strategy

The control strategy for hardware-in-loop simulation is shown in figure 4.1.

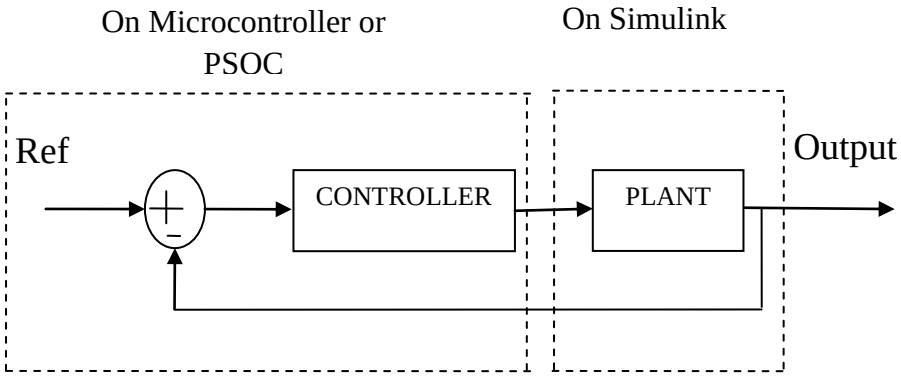


Figure 4.1: Block Diagram of close loop system

The left dotted part of figure 4.1 is implemented on 8-bit microcontroller hardware and PSOC device, the right dotted part is in simulation environment i.e. Simulink. The plant used for simulation is a linear model of DC motor speed plant [37][38]. The plant transfer function is

$$G_p(s) = \frac{2029.826}{(s+26.29)(s+2.296)} \tag{4.1}$$

The PI controller is used, due to its property of eliminating the steady state error and its simple structure. To implement the controller on an embedded system, the digital version of PI controller is used. The difference equation of digital PI controller [39] is

$$U(nT) = U(nT-T) + G_1 \times e(nT) - G_2 \times e(nT-T) \tag{4.2}$$

$$G_1 = K_p + \frac{K_I T}{2} \tag{4.3}$$

$$G_2 = K_p - \frac{K_I T}{2} \tag{4.4}$$

Equation (4.2) is implemented on microcontroller and PSOC with calculated values of G_1 and G_2 from equations (4.3) and (4.4), respectively. Here K_P is proportional gain, K_I is Integral gain and T is the sampling time. T should be kept as smaller as possible because more T will cause more overshoots in close loop system response [36]. K_P and K_I should be chosen appropriately by any

of the PID tuning method [40] for the required close loop performance. Here, the both K_P and K_I values are taken "1".

4.1.2 Microcontroller Implementation

The 8-bit microcontroller (ATMEL 8051) [41] is used for controller implementation. It has 128 bytes of RAM, 4K bytes of ROM, 2 timers, 6 interrupt sources and 1 serial port. There are more advanced versions of 8051 are available but this basic one is enough for PI controller implementation due to the usage of only 153 bytes of ROM, 5 bytes of RAM and 1 serial port. The plant is not present in real, so its mathematical model is used in simulation with controller in real world. This HIL simulation is performed with serial communication between microcontroller and Simulink. In this simulation, 9600 baud rate is used. The other baud rates can also be used. The microcontroller receives the plant output value from Simulink, calculates the error signal, then calculates the control signal by using equation (4.2) and sends it back to Simulink. This process is implemented by the flowcharts shown in figure 4.2(a), 4.2(b).

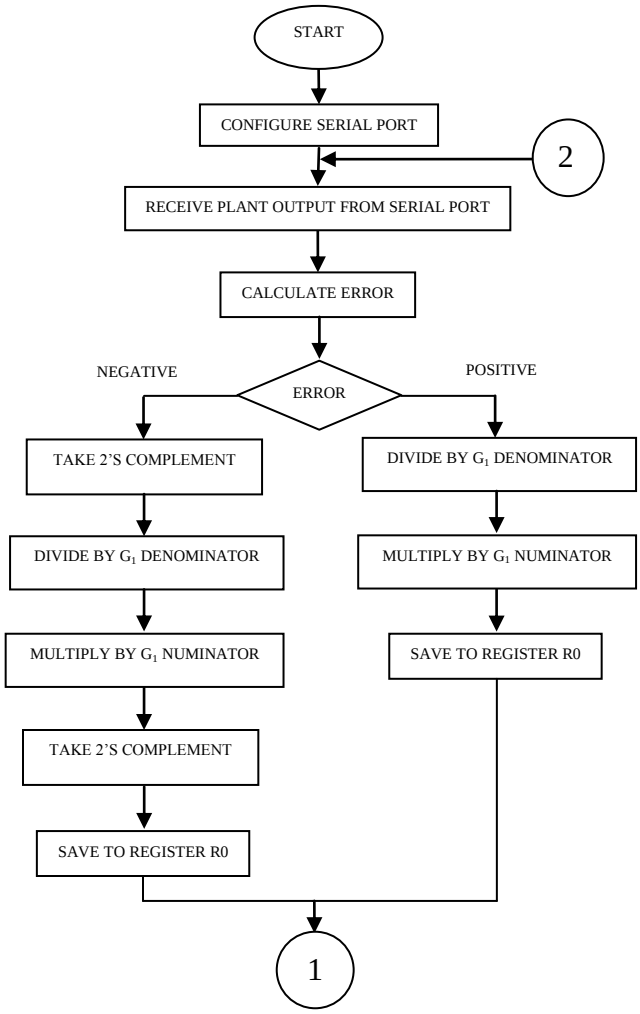


Figure 4.2(a): Flowchart of PI controller implementation (connected to figure 4.2(b))

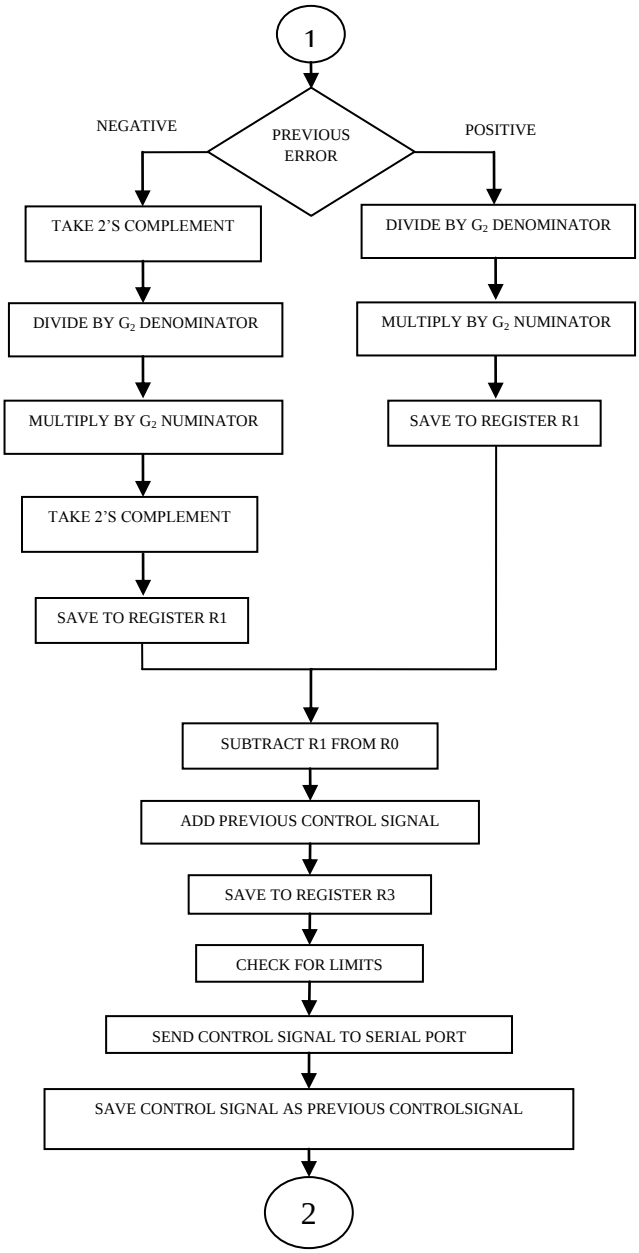


Figure 4.2(b): Flowchart of PI controller implementation (connected to figure 4.2(a))

The "Check for limits" block receives the control signal save in register R3 and limits this control signal within +1 and -1 value. This limitation may affect the system response to some degree. This thing is called the saturation of control signal. The PI controller is implemented using assembly language. The code is given in appendix A, section 8.1:

4.1.3 PSOC Implementation

PSOC (Programmable system on chip) [42] is an embedded device. PSOC 5 is used in this work. PSOC 5 contains Cortex M3 ARM processor with analog and digital configurable blocks. These configurable blocks make this device different from traditional microcontrollers. PSOC can used to implement a complete system on a single chip. It is programmed by using C language. Here, PSOC 5 is used to implement discrete PI controller. The C language code is given in appendix A, section 8.2.

4.1.4 Experimental Setup

The experimental setup consist of hardware in loop simulation with DC motor plant in Simulink and digital PI controller on microcontroller or PSOC device as shown in figure 4.1. The Simulink block diagram is shown in figure 4.3.

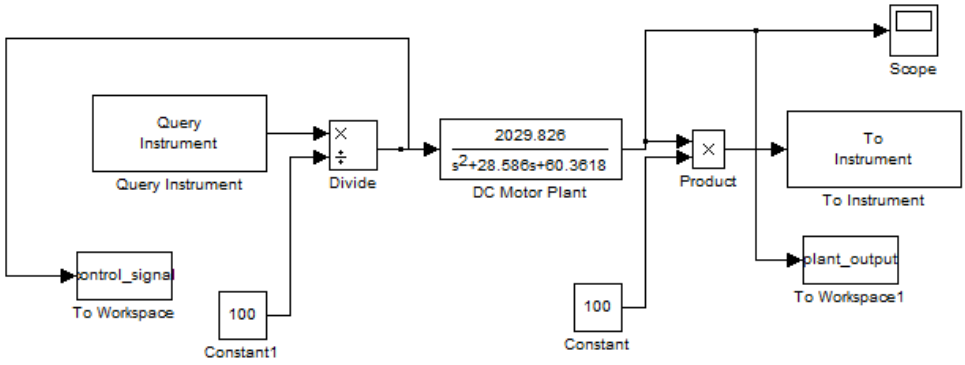


Figure 4.3: Simulink Blocks of HIL simulation for 8051

The “query instrument” and “to instrument” blocks are from the “Instrument control toolbox” of Matlab. The blocks are used for serial communication. It is assumed that plant step response will remain between “0” to “2”. A constant “100” is used for scaling the signals. The scaling is necessary in order to amplify the signals. The scaled signals enable them to be processed as floating point numbers on 8-bit microcontroller till two spaces after the decimal point. In case of PSOC HIL simulation, multiplying and dividing constants are kept “10,000” as shown in figure 4.4, to enabled signals to be processed till 4 decimal spaces. The signals are of 16-bit resolution and PSOC is able to handle it due to its 32-bit ARM processor.

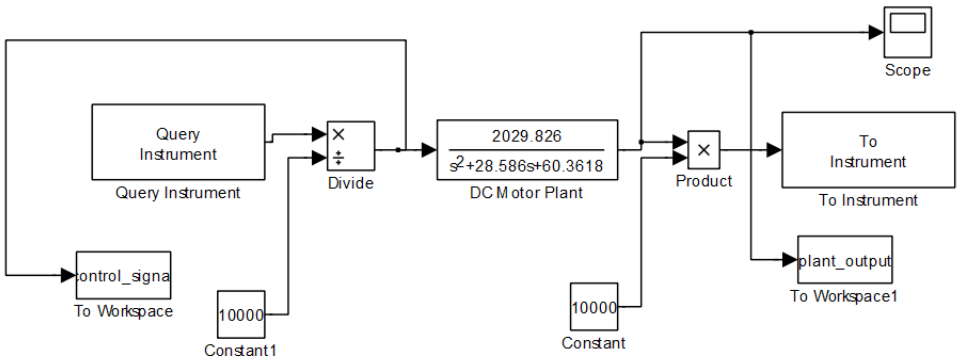


Figure 4.4: Simulink Blocks of HIL simulation for PSOC

4.1.5 Simulation Results

The simulation is performed with the values of K_P, K_I and T as 1, 1 and 0.01 sec, respectively. These values are chosen for achieving the best possible control performance (least steady state error). The calculated values of G_1 and G_2 are 1.005 and 0.995, respectively. In simulation, G_1 and G_2 are approximated to 1, for making the calculation possible on 8-bit microcontroller. This approximation may cause a very small error in the close loop system performance. The plant output, control signal and error plot for microcontroller implementation are shown in figure 4.5, 4.6 and 4.7 respectively.

The digital PI controller on microcontroller has good control performance with plant in HIL simulation with some small steady state error and oscillations at steady state level. The main reason of these oscillations and steady state error is the 8-bit wordlength of the microcontroller registers. The finite wordlength contributes towards the poor control performance during the steady state. The performance can be improved using registers with more number of bits like 16-bit or 32-bit microcontrollers.

The PSOC implemented PI controller gave better performance than 8-bit microcontroller because of 32-bit Cortex M3 ARM processor. The plant output, control signal and error plot for PSOC implementation are also shown in figure 4.5, 4.6 and 4.7 respectively.

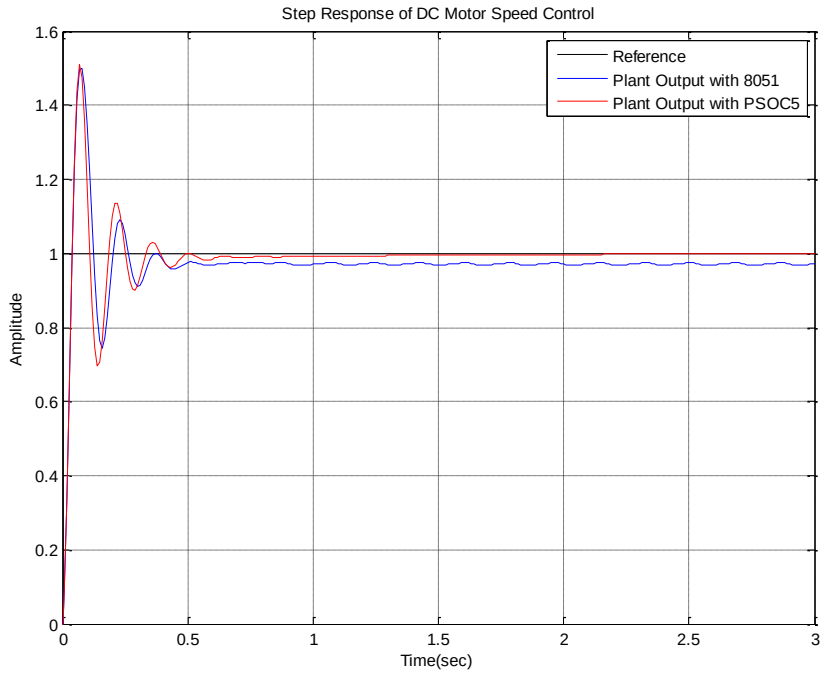


Figure 4.5: Step Response of close loop HIL simulation

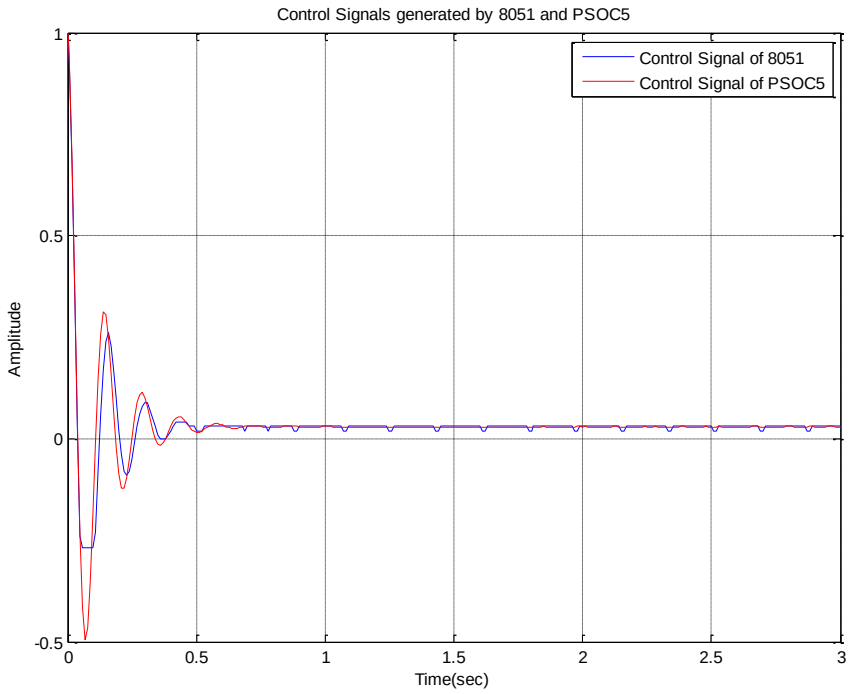


Figure 4.6: Control Signal

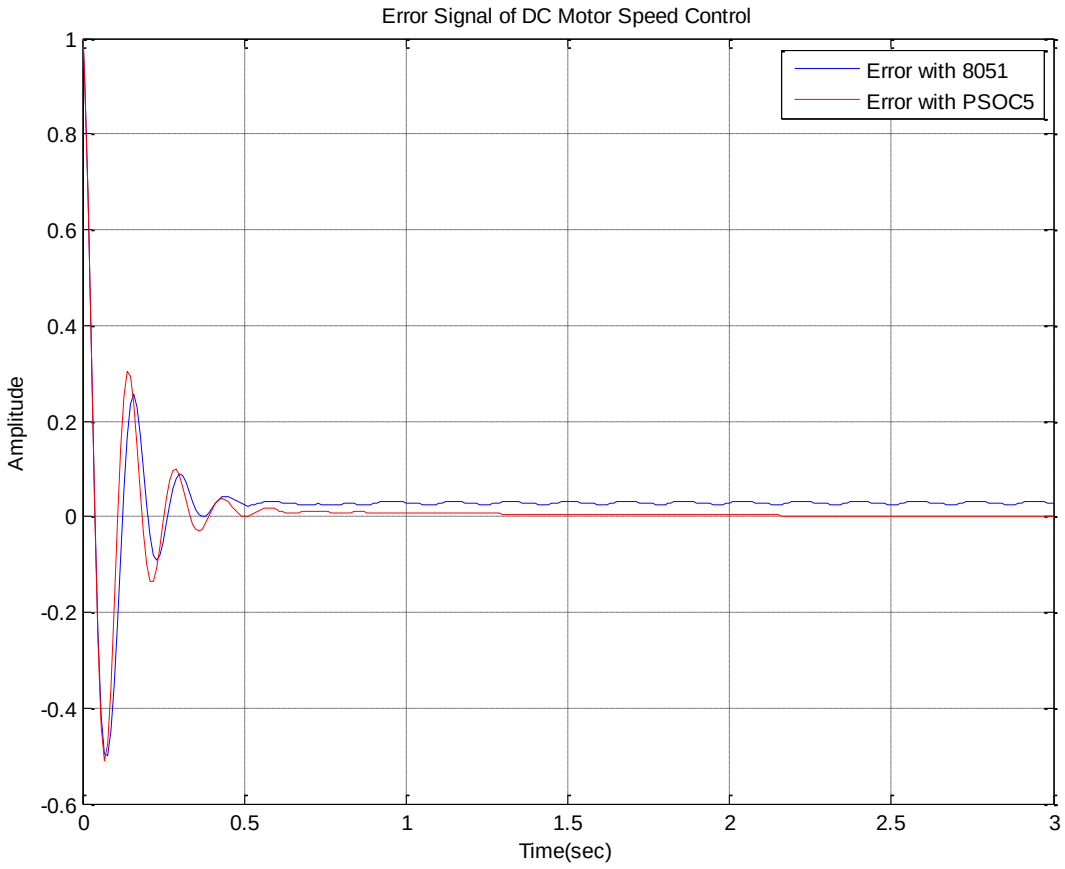


Figure 4.7: Error plot for HIL simulation

The error plot completely shows the better performance of PSOC than microcontroller. The root mean square of error (RMSE) values of both simulations also confirms this. The RMSE value of microcontroller simulation is “0.1157” and RMSE value of PSOC simulation is “0.1119”. The RMSE value of PSOC simulation is less than microcontroller simulation.

4.1.6 Conclusion

This section has presented the discrete PI controller implementation on 8-bit microcontroller and 32-bit ARM processor device PSOC and tested both controllers on DC motor plant model by using hardware in loop simulation technique. It is clearly visible from the results that PSOC performance is better than microcontroller. It clearly eliminated the steady state error, whereas the microcontroller was unable to eliminate.

4.2 Quadrotor Control

This section presents the outer loop control strategy (discussed in section 3.4) implementation on PSOC and runs this embedded controller on quadrotor plant model with hardware in loop simulation. Quadrotor is a MIMO system and its control strategy is also a MIMO system. In HIL simulation, PSOC receives the roll angle, pitch angle, yaw angle, x distance, y distance, z distance and z velocity and gives the calculated all four rotors speed. The strategy is implemented using C language. The code is given in appendix A, section 8.3. The experimental setup is shown in figure 4.8.

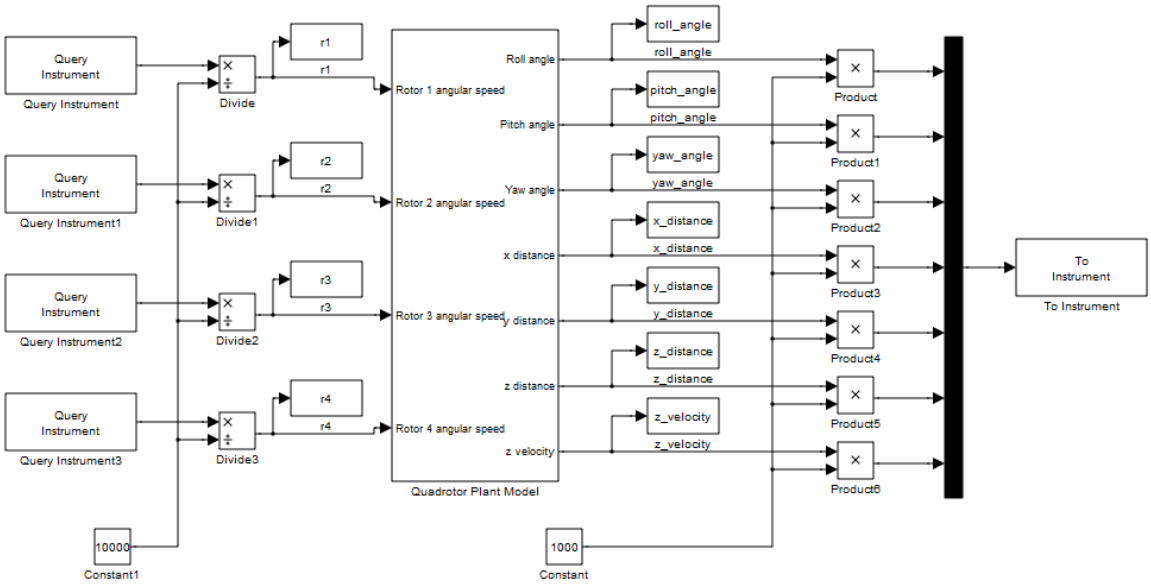


Figure 4.8: Simulink Blocks of Quadrotor HIL Simulation

As discussed in previous section, the "To Instrument" and "Query Instrument" block are used for serial communication between simulink and PSOC device. The constants are used to make points calculation possible on PSOC board.

4.2.1 Simualtion Results

As discussed, this simulation is outer loop control of quadrotor. The X distance, Y distance and Z distance are achieved for their respective references and Yaw angle is also achieved. The X distance, Y distance, Z distance and Yaw angle results are given in figures 4.9, 4.11, 4.13 and 4.14 respectively. The results show the good performance of our embedded controller and gives proof of successful implementation of control strategy on PSOC device.

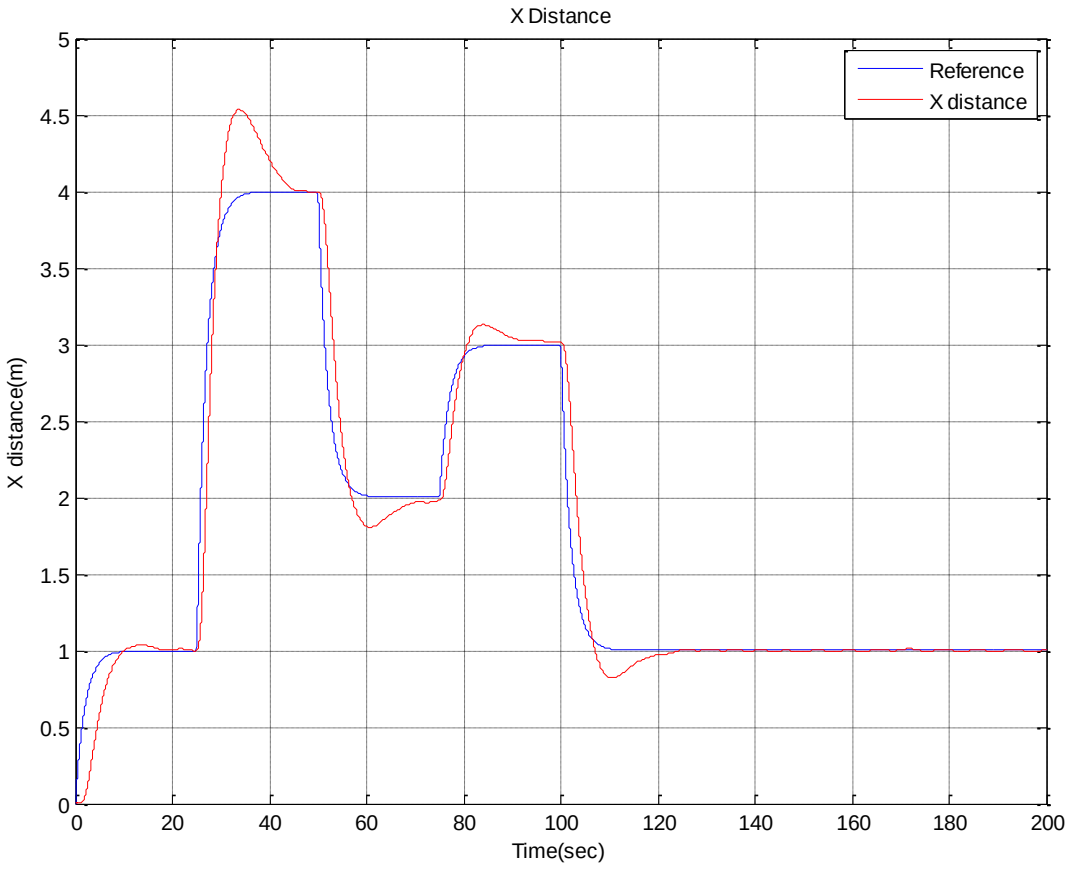


Figure 4.9: X distance control of quadrotor

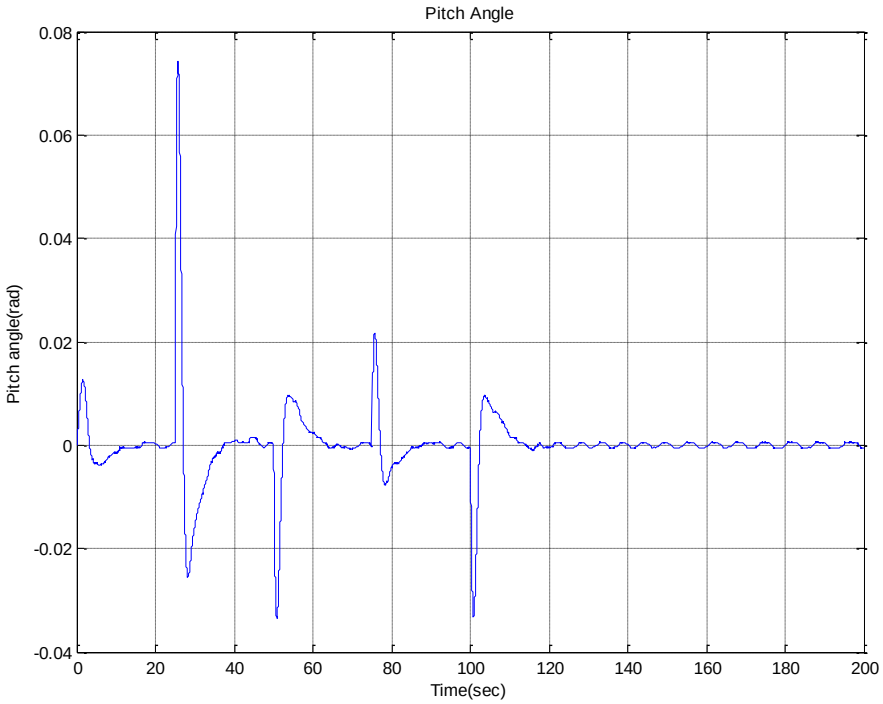


Figure 4.10: Pitch Angle

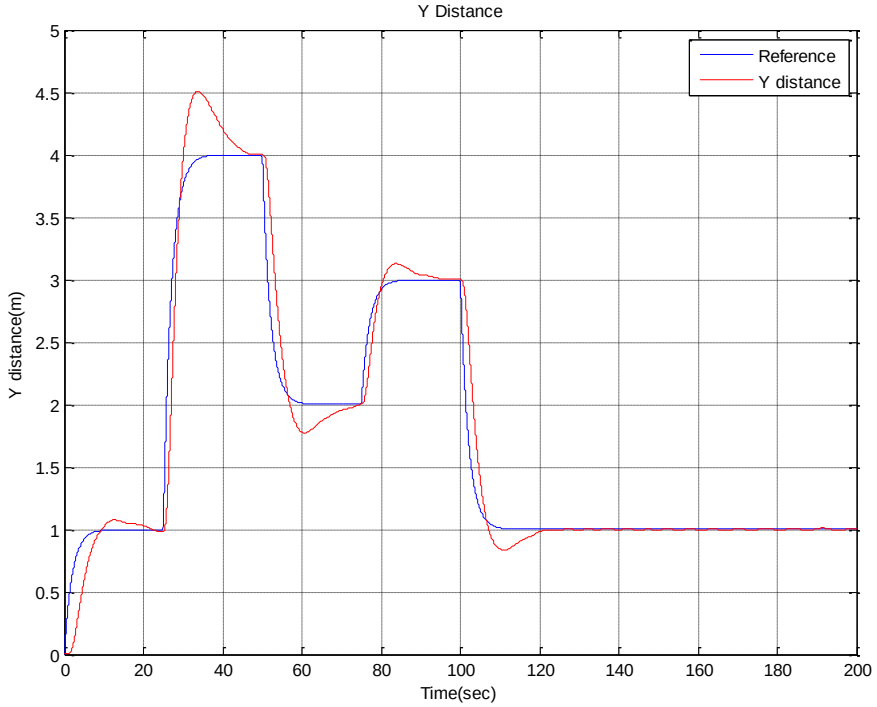


Figure 4.11: Y distance control of quadrotor

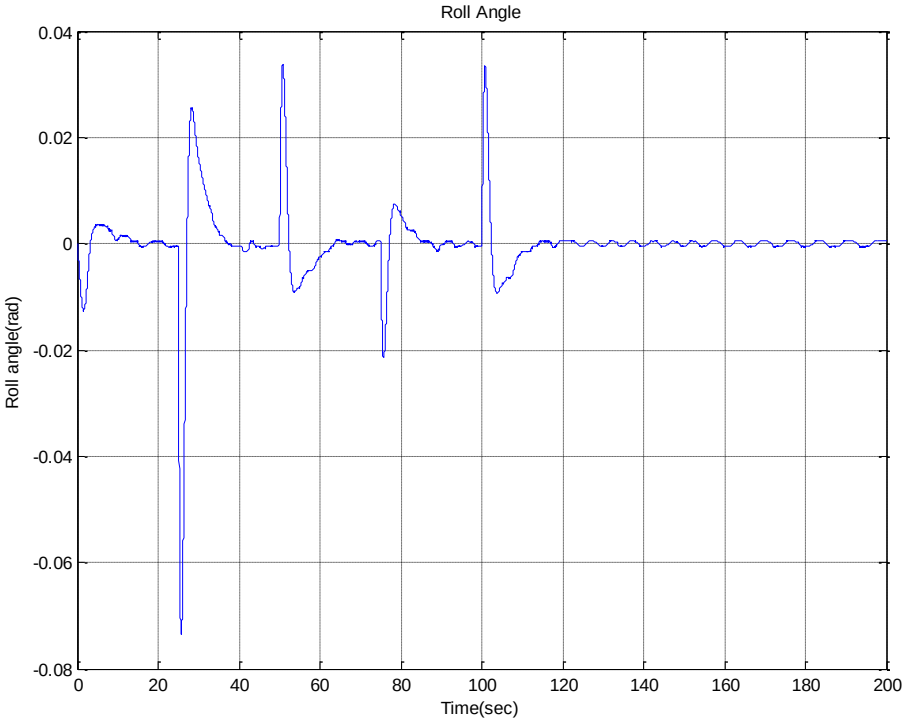


Figure 4.12: Roll Angle

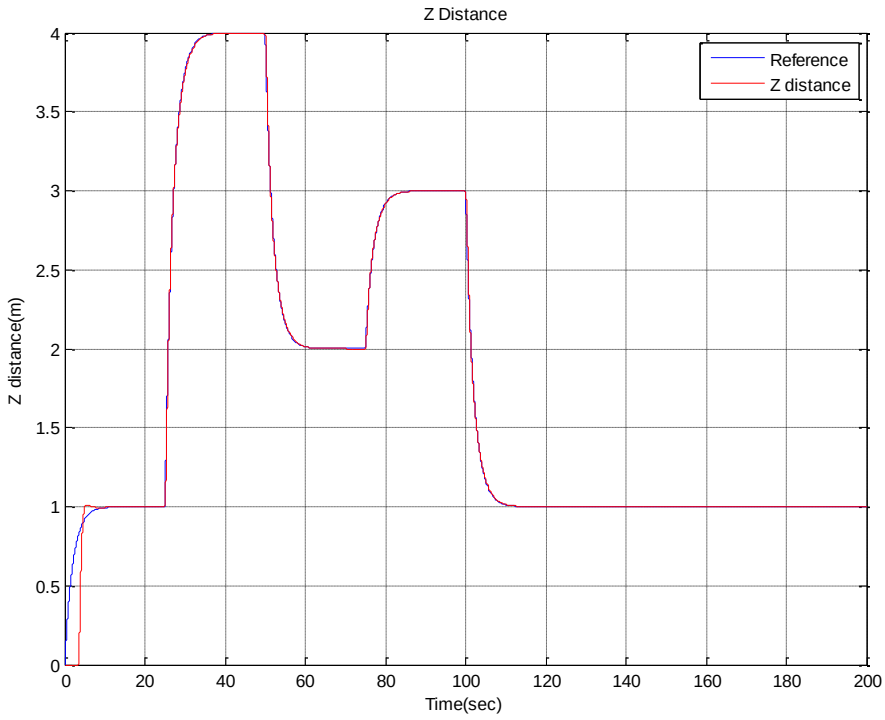


Figure 4.13: Z distance control of quadrotor

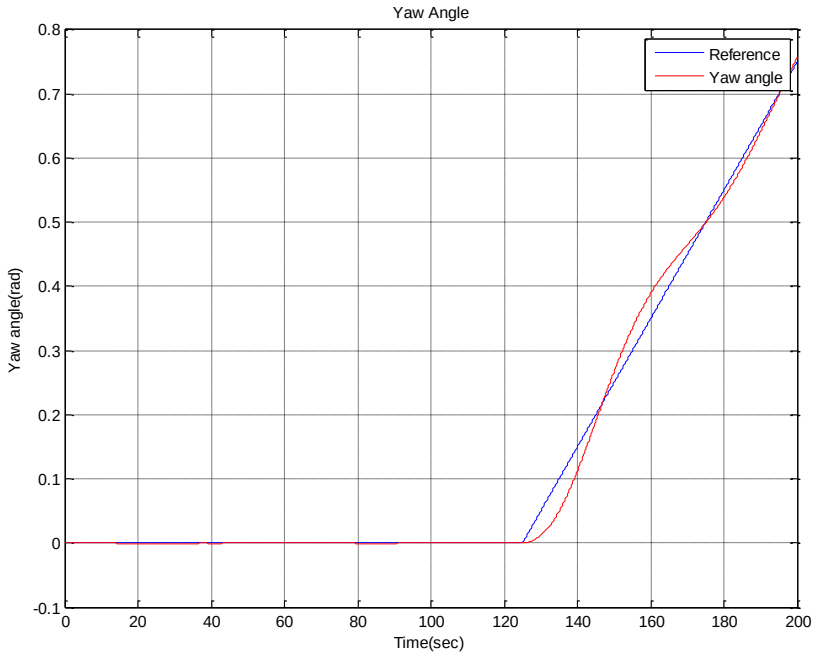


Figure 4.14: Yaw angle control of quadrotor

The PSOC board receives the quadrotor output signals and gives back the control signals, which are propellers angular speed. The control signals are given below:

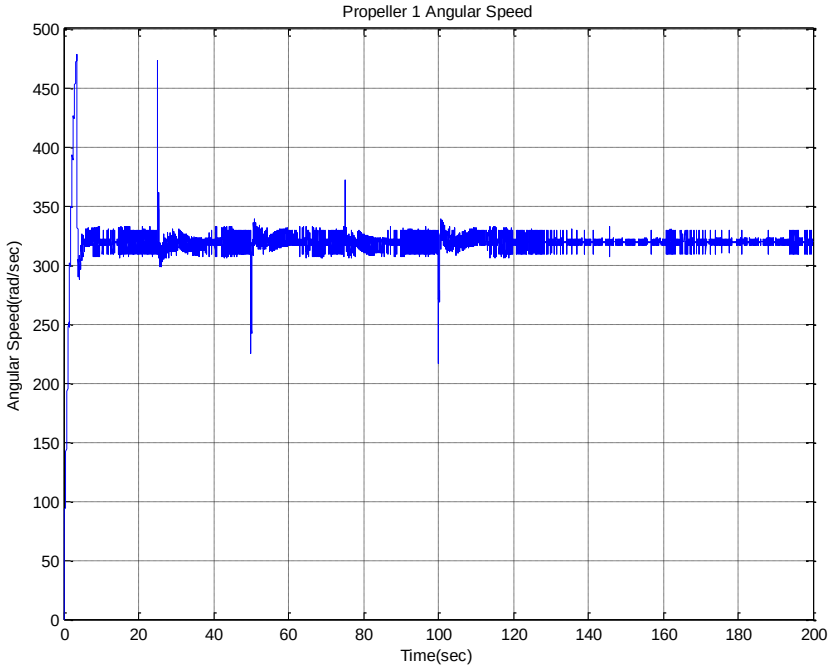


Figure 4.15: Propeller 1 angular speed of quadrotor

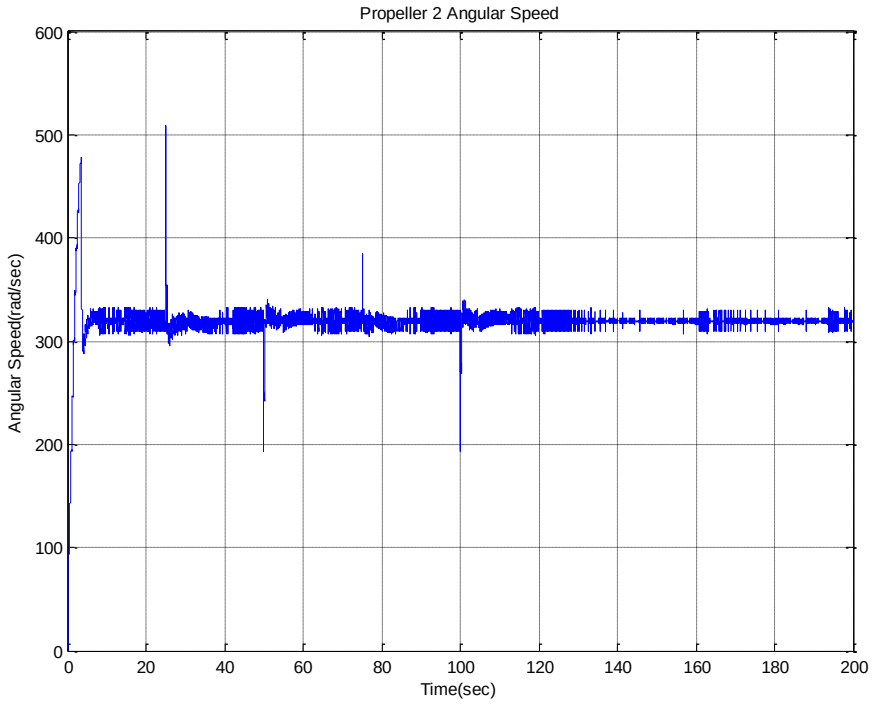


Figure 4.16: Propeller 2 angular speed of quadrotor

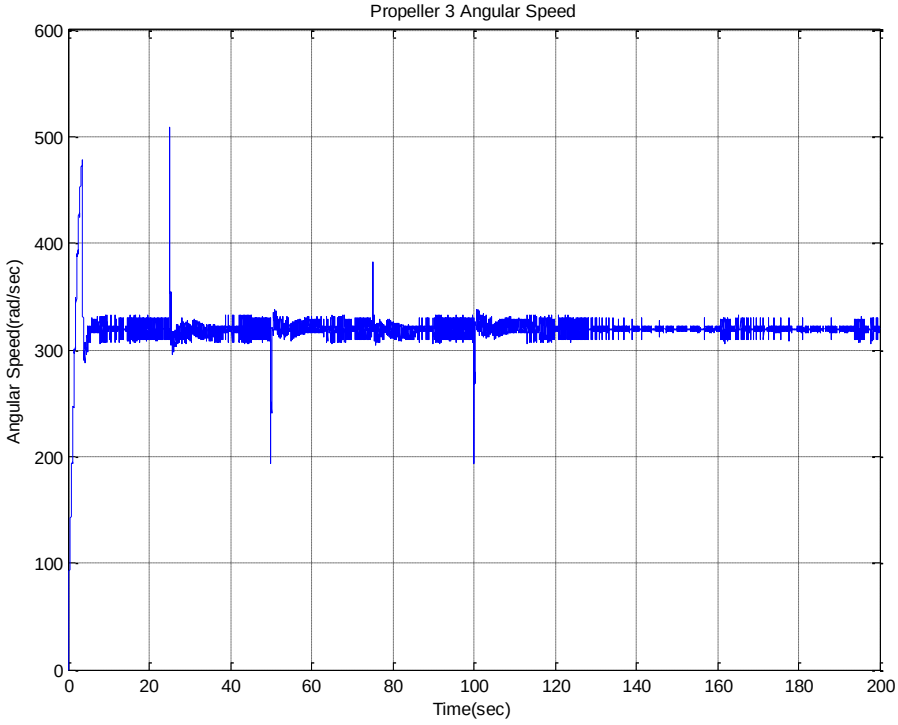


Figure 4.17: Propeller 3 angular speed of quadrotor

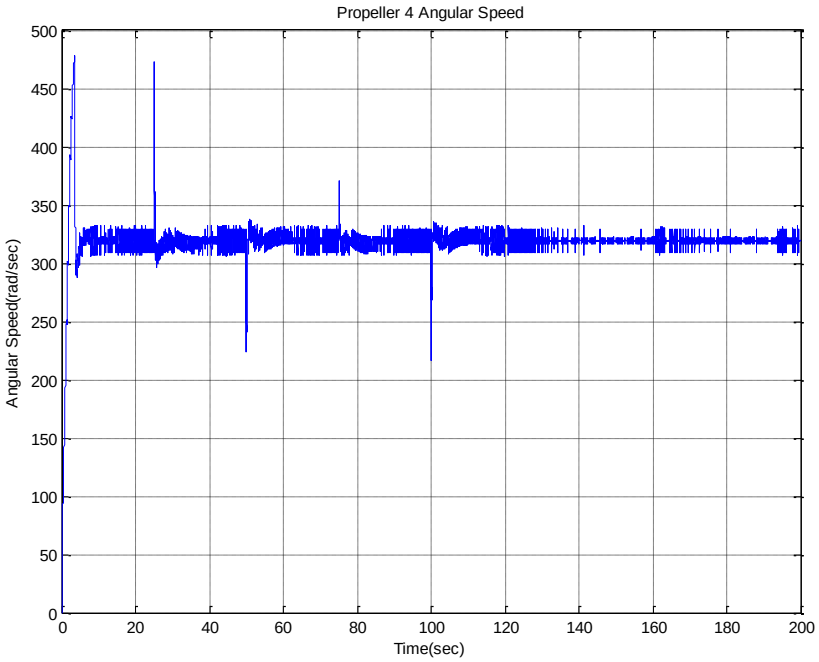


Figure 4.18: Propeller 4 angular speed of quadrotor

As shown in figure 3.8, the outer loop control is link with inner loop control. The roll angle effects Y distance and pitch angle effects X distance. The roll angle and pitch angle are shown in figures 4.19 and 4.20 respectively.

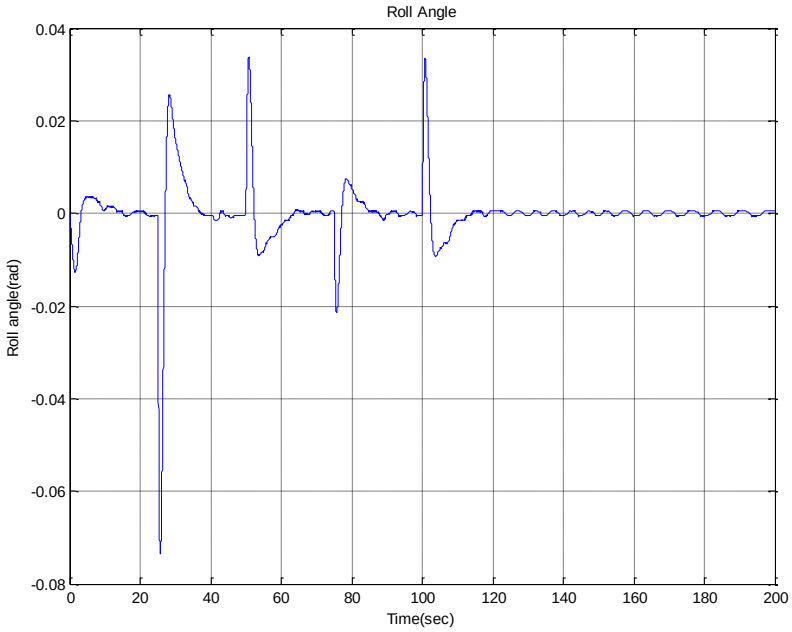


Figure 4.19: Roll Angle in HIL Simulation

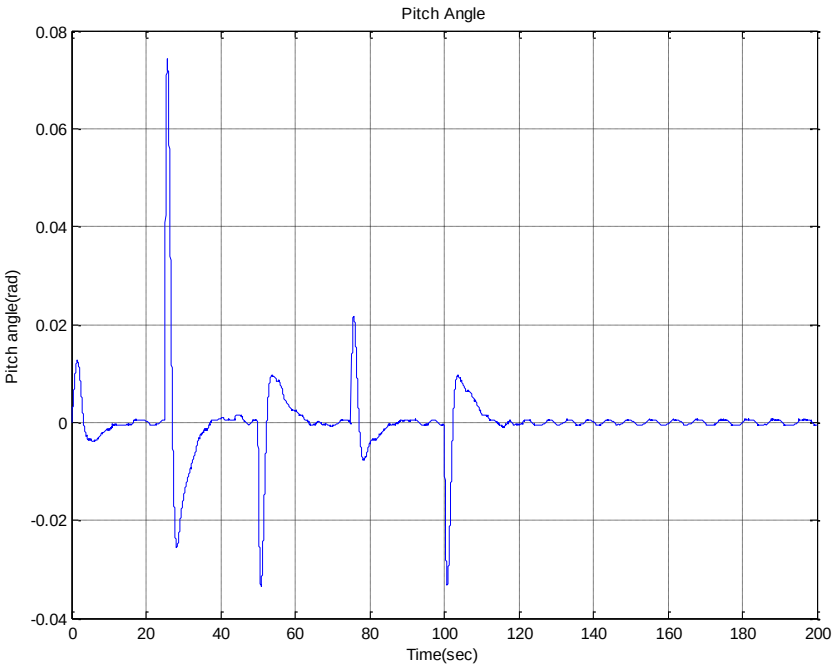


Figure 4.20: Pitch Angle in HIL Simulation

This outer loop control strategy implemented on PSOC is able to move quadrotor to a given reference trajectory in 3-dimensional space as shown in figure 4.21 and 4.22.

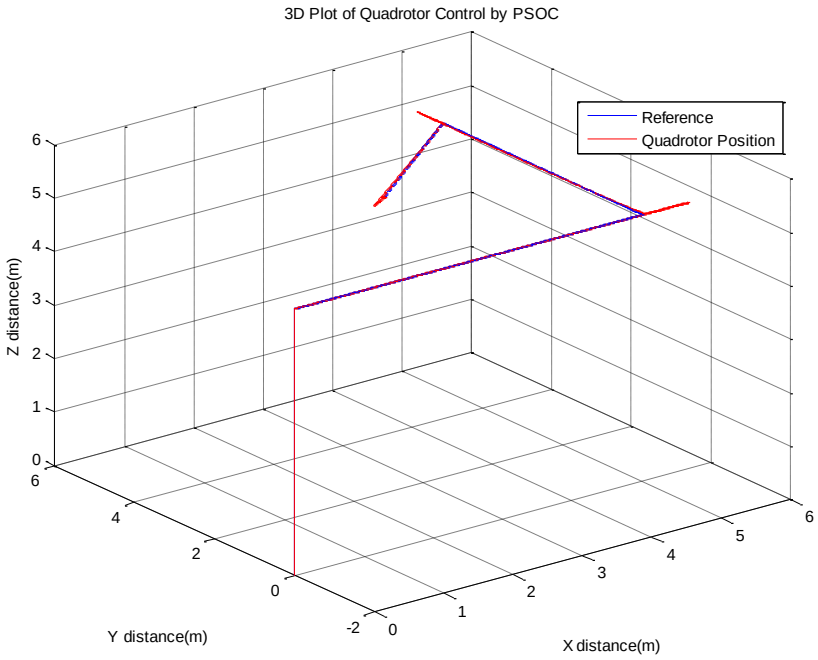


Figure 4.21: 3D Plot of Quadrotor Control by PSOC

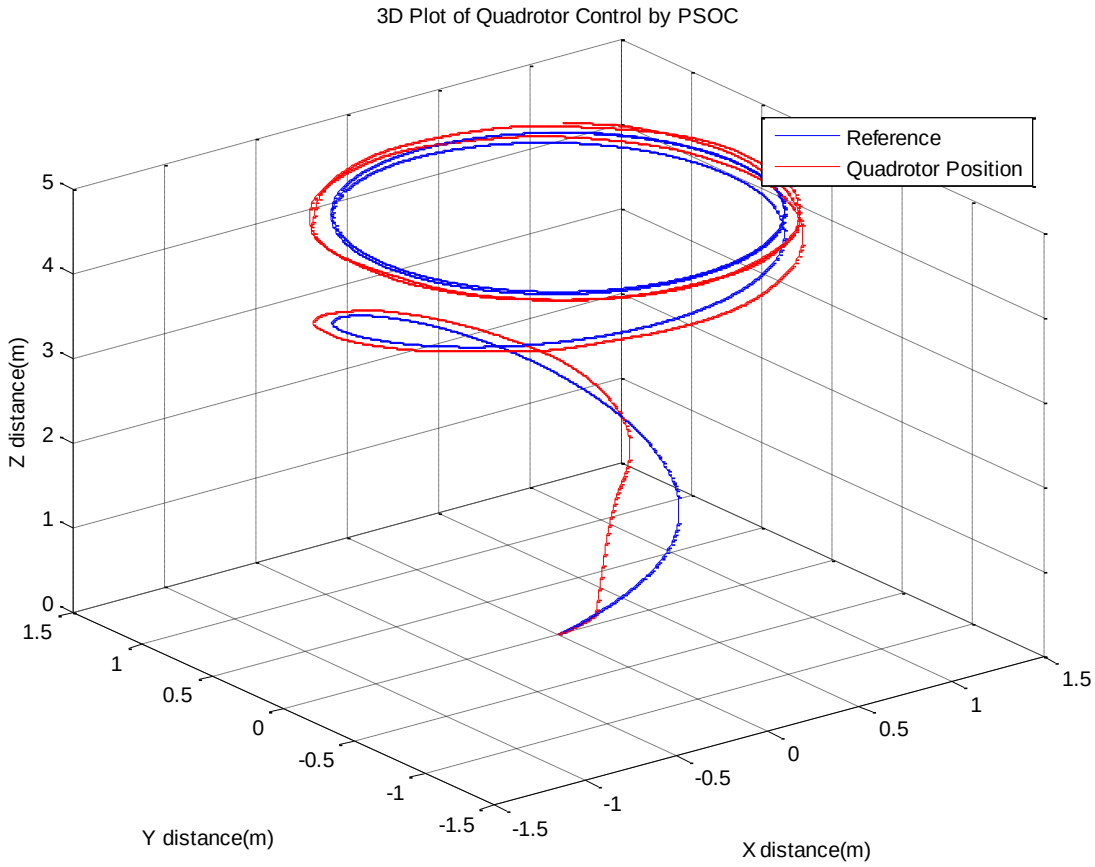


Figure 4.22: 3D Plot of Quadrotor Control by PSOC

4.3 Summary

This chapter has presented the simple discrete PI controller implementation on microcontroller and PSOC device and tested on DC motor speed model with hardware in loop simulation. Results clearly showed the better performance of PSOC device. This chapter also presented the quadrotor MIMO control strategy implementation on PSOC device. Results clearly showed the successful implementation and good control performance. This makes the PSOC device, a good choice for the implementation of control algorithms.

CHAPTER 5 MPC Control of Quadrotor

5 MPC Control of Quadrotor

Model predictive control (MPC) strategy is an advanced optimal control strategy widely used in process industry. MPC is a digital control technique. The MPC requires the plant model in order to find the control signals. Sometimes the accurate model parameters are not available, so it needs to be found. This chapter presents the system identification applied on quadrotor plant model to find the parameters and use these parameters in MPC to find control signals. This is non-linear MPC because of non-linear model usage.

5.1 MPC Control Technique

The model predictive control technique includes the prediction of plant output up to some prediction horizon N_p and tries to find the control signals up to some control horizon N_c by using some constrained optimization method as shown in figure 5.1. Thus gives the optimal control signals and optimal control results.

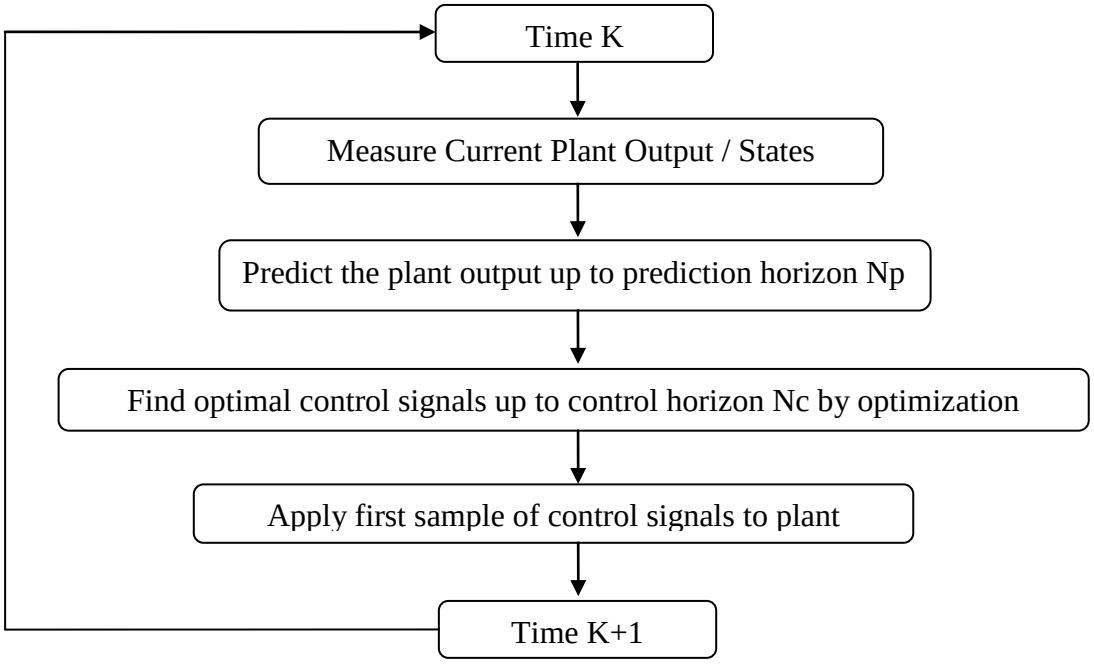


Figure 5.1: MPC Algorithm

The prediction of plant output over prediction horizon is shown in figure 5.2[43].

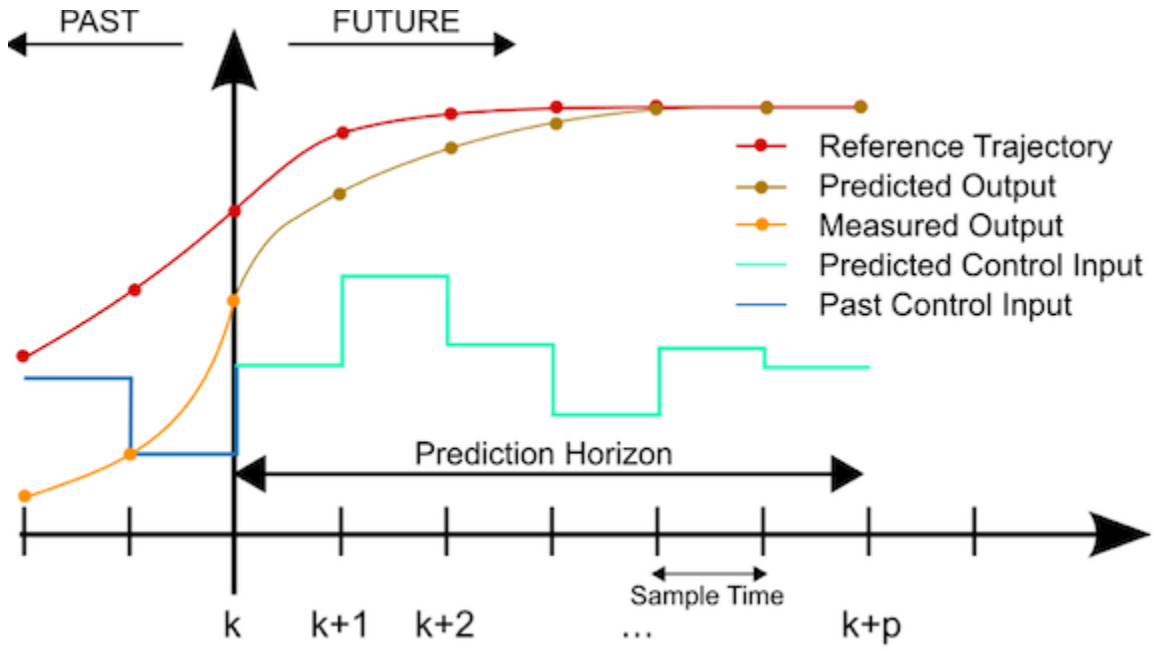


Figure 5.2: Plant output Prediction

The plant output is predicted over prediction horizon and control signal is found but only first sample is applied to plant and the whole process is repeated again on next sample instant.

5.2 Discrete Model of Quadrotor

In model predictive control, a plant model is required for prediction. Here, the plant model given by equations 3.1 and 3.2 is used but it is discretized by using Newton Euler method [36]. The symbols used in discrete model are:

$\emptyset_1$ = Roll angle

$\emptyset_2$ = Roll angle derivative

θ_1 = Pitch angle

θ_2 = Pitch angle derivative

ψ_1 = Yaw angle

ψ_2 = Yaw angle derivative

$x1$ = X distance

$x2$ = X distance derivative

$y1$ = Y distance

$y2$ = Y distance derivative

$z1$ = z distance

$z2$ = z distance derivative

The discrete plant model is given below:

$$
\begin{aligned}
&\phi_1[\mathrm{k}+1] = \phi_1[\mathrm{k}] + \mathrm{T_s}\,\phi_2[\mathrm{k}] \\
&\phi_2[\mathrm{k}+1] = \phi_2[\mathrm{k}] + \mathrm{T_s}\left(\dot{\theta}\dot{\psi}\,\frac{I_{yy}-I_{zz}}{I_{xx}} + \dot{\theta}\,\frac{J_r}{I_{xx}}\,\Upsilon + \frac{L}{I_{xx}}U_2\right) \\
&\theta_1[\mathrm{k}+1] = \theta_1[\mathrm{k}] + \mathrm{T_s}\,\theta_2[\mathrm{k}] \\
&\theta_2[\mathrm{k}+1] = \theta_2[\mathrm{k}] + \mathrm{T_s}\left(\dot{\phi}\dot{\psi}\,\frac{I_{zz}-I_{xx}}{I_{yy}} - \dot{\phi}\,\frac{J_r}{I_{yy}}\,\Upsilon + \frac{L}{I_{yy}}U_3\right) \\
&\psi_1[\mathrm{k}+1] = \psi_1[\mathrm{k}] + \mathrm{T_s}\,\psi_2[\mathrm{k}] \\
&\psi_2[\mathrm{k}+1] = \psi_2[\mathrm{k}] + \mathrm{T_s}\left(\dot{\theta}\dot{\phi}\,\frac{I_{xx}-I_{yy}}{I_{zz}} + \frac{1}{I_{zz}}U_4\right) \\
&x_1[\mathrm{k}+1] = x_1[\mathrm{k}] + \mathrm{T_s}\,x_2[\mathrm{k}] \\
&x_2[\mathrm{k}+1] = x_2[\mathrm{k}] + \mathrm{T_s}\left(\frac{(\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)}{M}U_1\right) \\
&y_1[\mathrm{k}+1] = y_1[\mathrm{k}] + \mathrm{T_s}\,y_2[\mathrm{k}] \\
&y_2[\mathrm{k}+1] = y_2[\mathrm{k}] + \mathrm{T_s}\left(\frac{(\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)}{M}U_1\right) \\
&z_1[\mathrm{k}+1] = z_1[\mathrm{k}] + \mathrm{T_s}\,z_2[\mathrm{k}] \\
&z_2[\mathrm{k}+1] = z_2[\mathrm{k}] + \mathrm{T_s}\left(-g + \frac{(\cos\phi\cos\theta)}{M}U_1\right)
\end{aligned}
\tag{5.1}
$$

The U1, U2, U3 and U4 are:

$$
\begin{aligned}
U_1 &= b\Omega_1^2 + b\Omega_2^2 + b\Omega_3^2 + b\Omega_4^2 \\
U_2 &= b\Omega_4^2 - b\Omega_2^2 \\
U_3 &= b\Omega_3^2 - b\Omega_1^2 \\
U_4 &= d(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \\
\Upsilon &= \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4
\end{aligned}
\tag{5.2}
$$

5.3 System Identification of Quadrotor

This discrete model given by equation 5.1 and 5.2 can be used in MPC for output prediction and parameters given in table 3.2 can also be used but most of the time these parameters are not accurately known. So the equation 5.1 and 5.2 are modified as given below:

$$
\begin{aligned}
\phi_1[\mathrm{k+1}] &= \phi_1[\mathrm{k}] + \mathrm{T}_s\,\phi_2[\mathrm{k}] \\
\phi_2[\mathrm{k+1}] &= \phi_2[\mathrm{k}] + \mathrm{T}_s\left(\dot{\theta}\dot{\psi}P_1 + \dot{\theta}P_2\Upsilon + P_3U_2\right) \\
\theta_1[\mathrm{k+1}] &= \theta_1[\mathrm{k}] + \mathrm{T}_s\,\theta_2[\mathrm{k}] \\
\theta_2[\mathrm{k+1}] &= \theta_2[\mathrm{k}] + \mathrm{T}_s\left(\dot{\phi}\dot{\psi}P_4 - \dot{\phi}P_5\Upsilon + P_6U_3\right) \\
\psi_1[\mathrm{k+1}] &= \psi_1[\mathrm{k}] + \mathrm{T}_s\,\psi_2[\mathrm{k}] \\
\psi_2[\mathrm{k+1}] &= \psi_2[\mathrm{k}] + \mathrm{T}_s\left(P_7U_4\right) \\
z_1[\mathrm{k+1}] &= z_1[\mathrm{k}] + \mathrm{T}_s\,z_2[\mathrm{k}] \\
z_2[\mathrm{k+1}] &= z_2[\mathrm{k}] + \mathrm{T}_s\left(-g + P_8(\cos\phi\cos\theta)U_1\right)
\end{aligned}
\tag{5.3}
$$

The U1, U2, U3 and U4 are:

$$
\begin{aligned}
U_1 &= \Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2 \\
U_2 &= \Omega_4^2 - \Omega_2^2 \\
U_3 &= \Omega_3^2 - \Omega_1^2 \\
U_4 &= \Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2 \\
\Upsilon &= \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4
\end{aligned}
\tag{5.4}
$$

The model given in equations 5.3 and 5.4 is used in MPC with calculated values of P_1-P_8. These parameters P_1-P_8 are calculated by recursive least square algorithm (RLS) [44] and data obtain from simulation of section 3.3. The data used for system identification is given below.

The Quadrotor input signals, which are propellers angular speed, are:

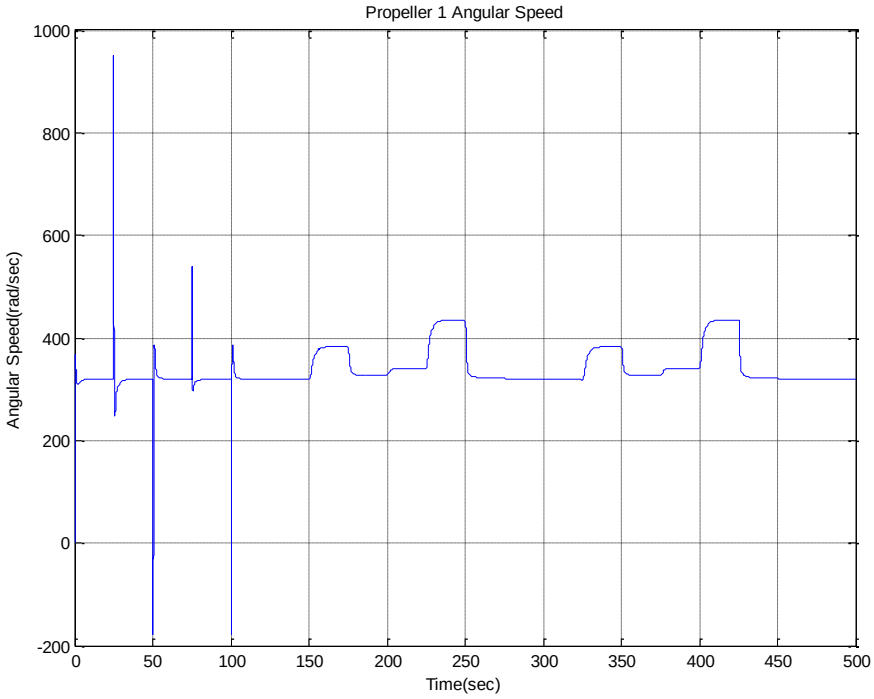


Figure 5.3: Propeller 1 Angular Speed

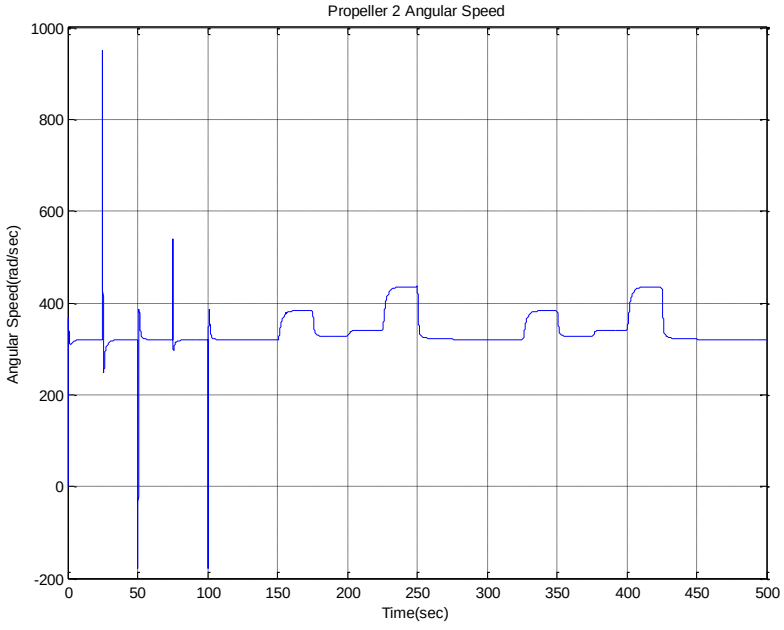


Figure 5.4: Propeller 2 Angular Speed

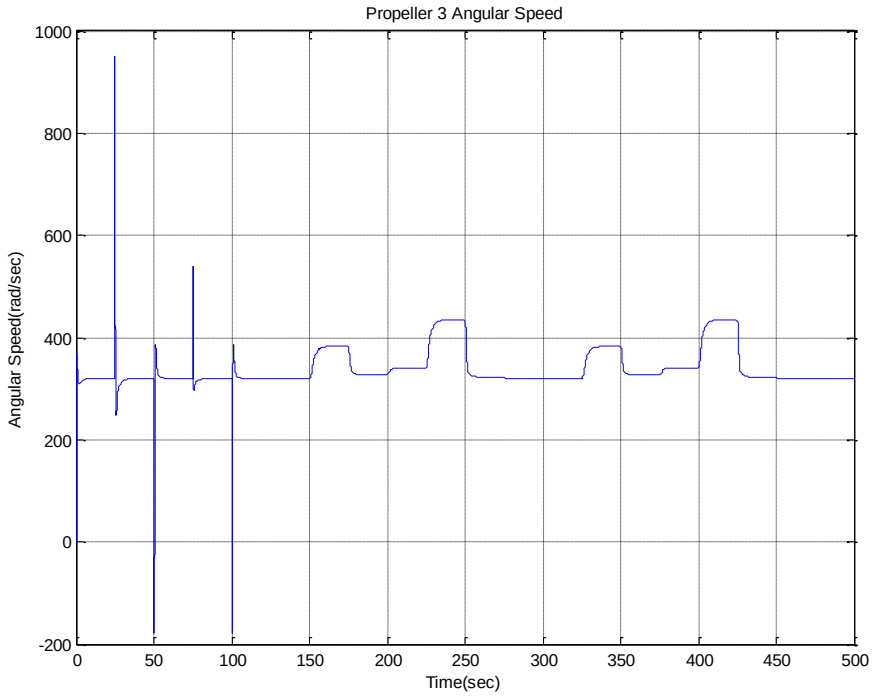


Figure 5.5: Propeller 3 Angular Speed

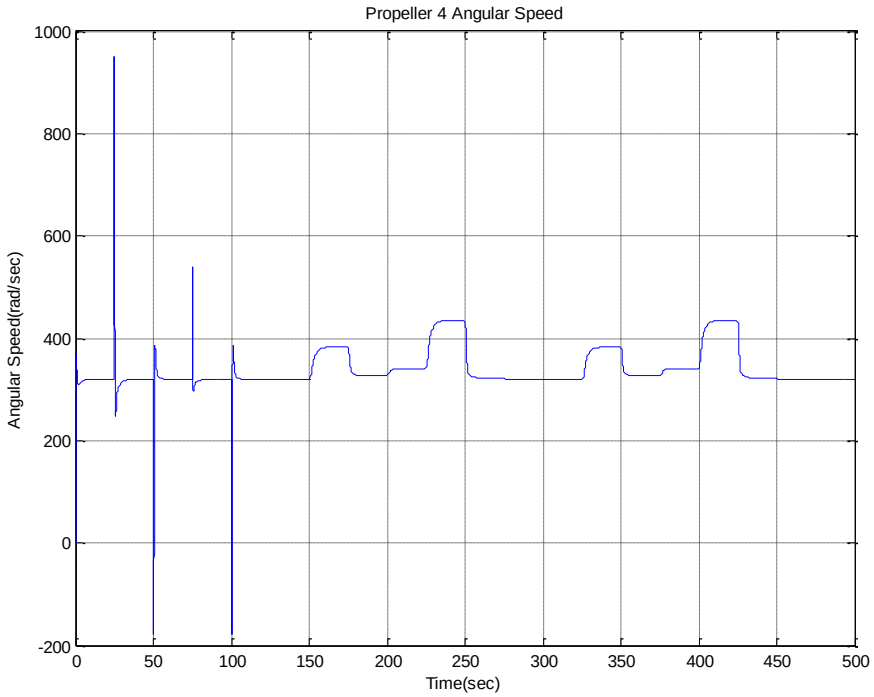


Figure 5.6: Propeller 4 Angular Speed

The Quadrotor output signals, which are roll angle, roll angle derivative, pitch angle, pitch angle derivative, yaw angle derivative and z distance derivative, are:

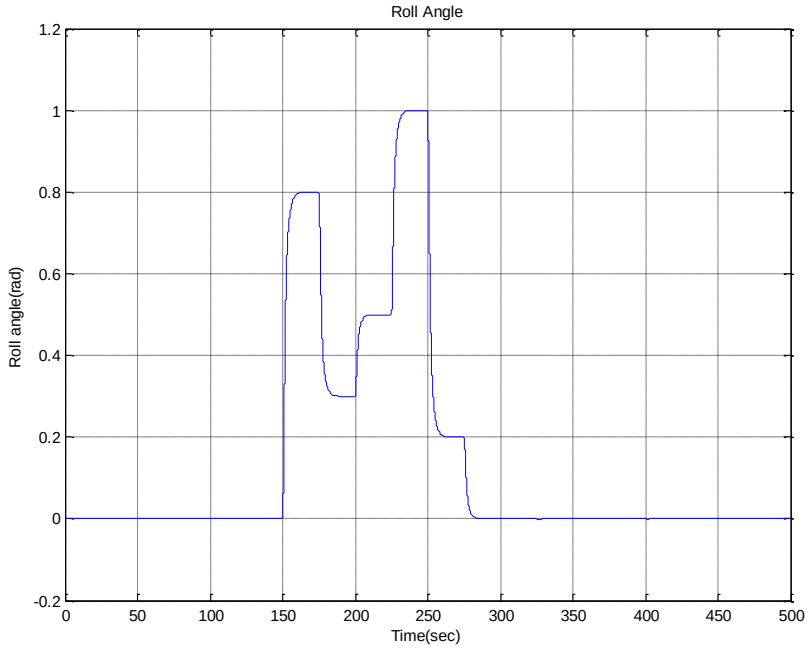


Figure 5.7: Roll Angle

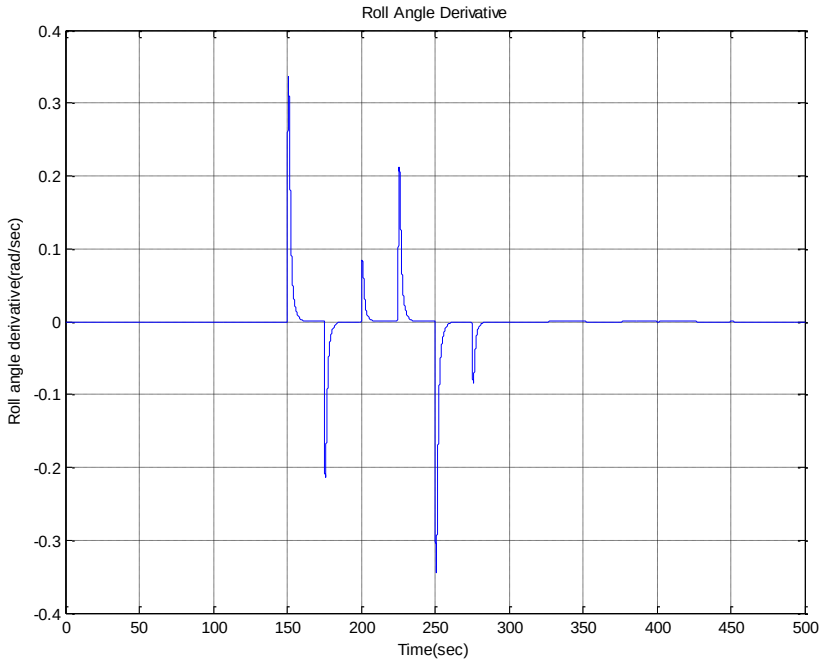


Figure 5.8: Roll Angle Derivative

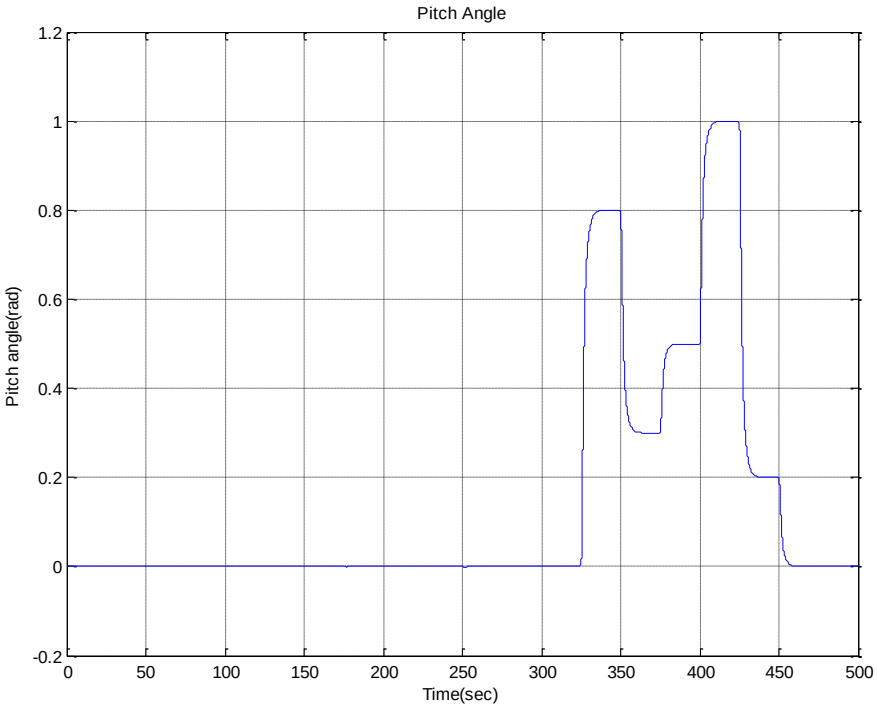


Figure 5.9: Pitch Angle

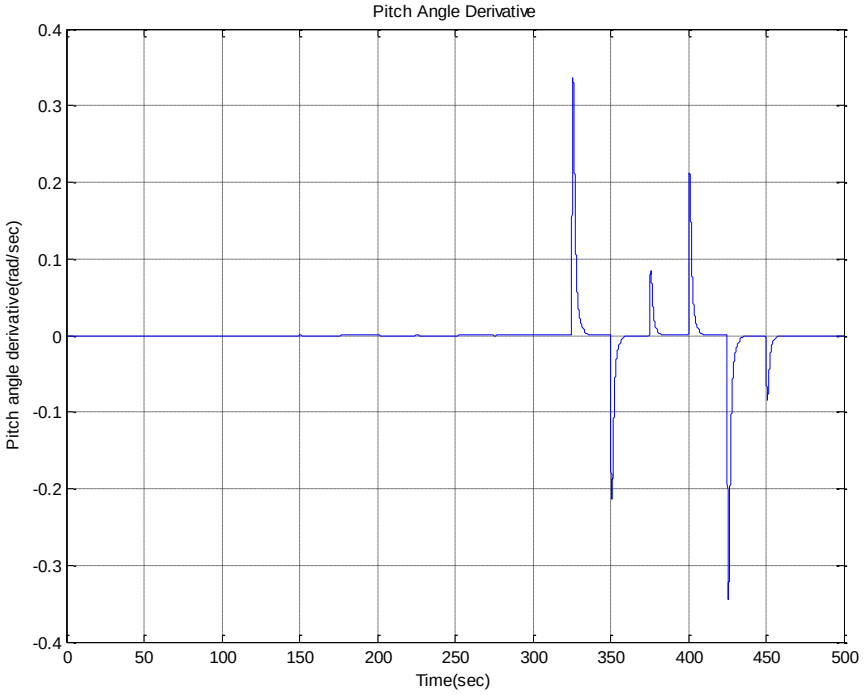


Figure 5.10: Pitch Angle Derivative

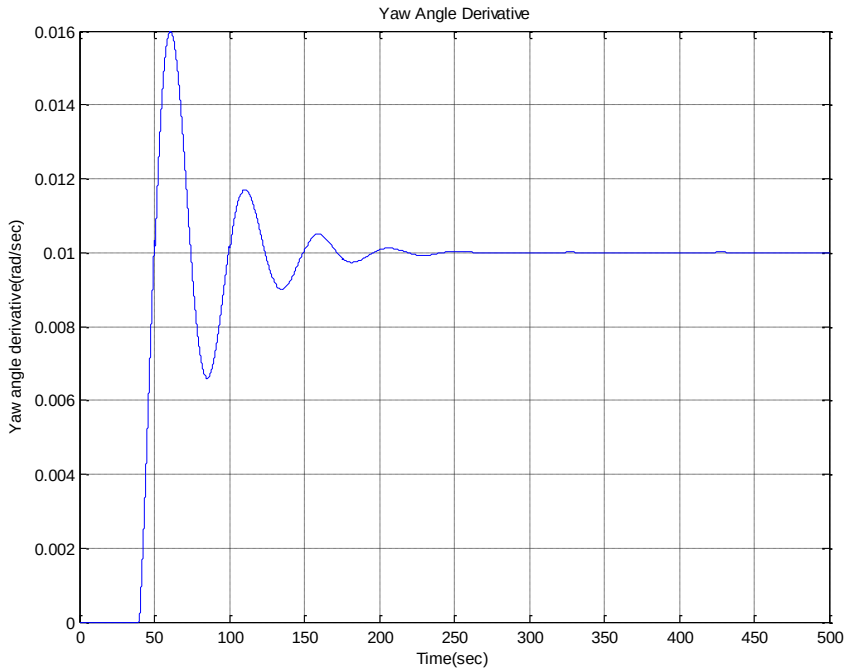


Figure 5.11: Yaw Angle Derivative

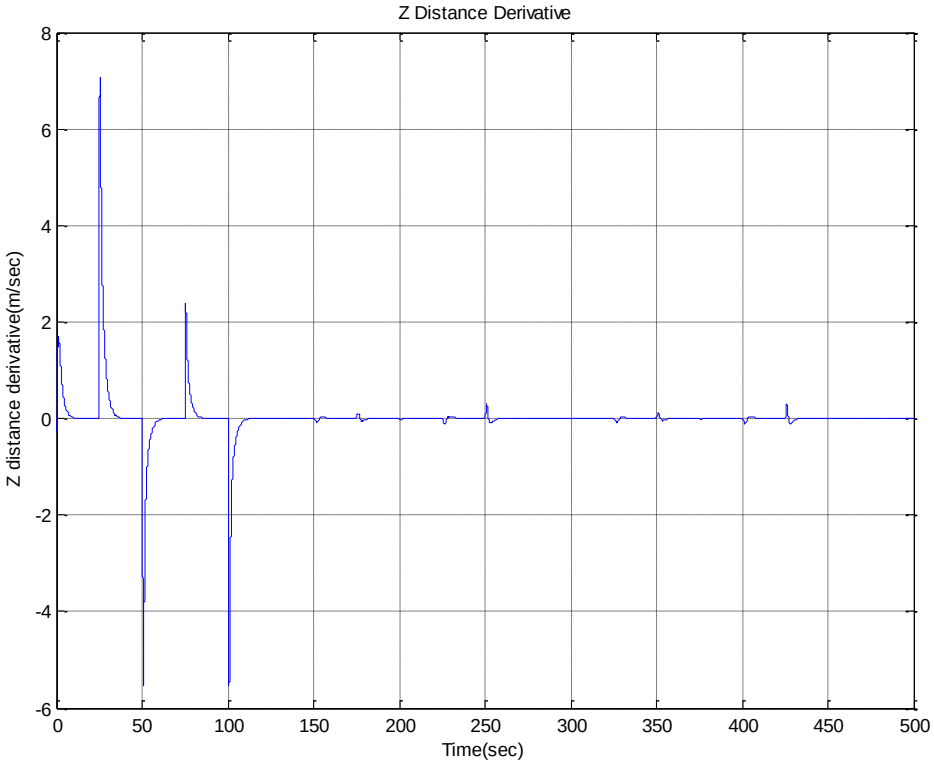


Figure 5.12: Z Distance Derivative

The RLS algorithm equations are:

$$\begin{aligned}
&\Theta(\mathrm{k})_{n\times 1} = \Theta(\mathrm{k}-1)_{n\times 1} + K(\mathrm{k})(\mathrm{y}(\mathrm{k}) - \varphi^T(\mathrm{k})_{1\times n}\Theta(\mathrm{k}-1)_{n\times 1}) \\
&\mathrm{K}(\mathrm{k})_{n\times 1} = \mathrm{P}(\mathrm{k})_{n\times n}\varphi(\mathrm{k})_{n\times 1} \\
&P(\mathrm{k})_{n\times n} = P(\mathrm{k}-1)_{n\times n} - P(\mathrm{k}-1)_{n\times n}\varphi(\mathrm{k})_{n\times 1}(1+\varphi^T(\mathrm{k})_{1\times n}\,\mathrm{P}(\mathrm{k}-1)_{n\times n}\varphi(\mathrm{k})_{n\times 1})^{-1}\varphi^T(\mathrm{k})_{1\times n}P(\mathrm{k}-1)_{n\times n}
\end{aligned} \tag{5.5}$$

Here $\Theta(\mathrm{k})_{n\times 1}$ is the required parameters vector. "n" is the number of parameters to be found. $\mathrm{P}(\mathrm{k})_{n\times n}$ is the covariance matrix, usually initialize with identity matrix of size "n" multiplied with a large number like 100. $\varphi(\mathrm{k})_{n\times 1}$ is a vector dependent on model and signals used for identification. y(k) is model output signal also dependent on model and signals used for identification. The sampling time $\mathrm{T_s}$ is 0.01sec.

To find P_1, P_2 and P_3:

n=3.

$$\begin{aligned}
&\Theta(\mathrm{k})_{3\times 1} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} \\
&\varphi(\mathrm{k})_{3\times 1} = \begin{bmatrix} \theta_2[\mathrm{k}-1]\psi_2[\mathrm{k}-1] \\ \theta_2[\mathrm{k}-1]\Upsilon[\mathrm{k}-1] \\ U_2[\mathrm{k}-1] \end{bmatrix} \\
&y(\mathrm{k}) = \frac{\phi_2[\mathrm{k}]-\phi_2[\mathrm{k}-1]}{T_s}
\end{aligned} \tag{5.6}$$

At k=0,

$$\begin{aligned}
&\Theta(\mathrm{k}-1)_{3\times 1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\
&P(\mathrm{k}-1)_{3\times 3} = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix}
\end{aligned} \tag{5.7}$$

To find P_4, P_5 and P_6:

n=3.

$$\Theta(\mathrm{k})_{3\times1}=\begin{bmatrix}P_4\\P_5\\P_6\end{bmatrix}$$

$$\varphi(\mathrm{k})_{3\times1}=\begin{bmatrix}\phi_2[\mathrm{k}-1]\psi_2[\mathrm{k}-1]\\-\phi_2[\mathrm{k}-1]\Upsilon[\mathrm{k}-1]\\U_3[\mathrm{k}-1]\end{bmatrix} \tag{5.8}$$

$$y(\mathrm{k})=\frac{\theta_2[\mathrm{k}]-\theta_2[\mathrm{k}-1]}{T_s}$$

At k=0,

$$\Theta(\mathrm{k}-1)_{3\times1}=\begin{bmatrix}0\\0\\0\end{bmatrix}$$

$$P(\mathrm{k}-1)_{3\times3}=\begin{bmatrix}100&0&0\\0&100&0\\0&0&100\end{bmatrix} \tag{5.9}$$

To find P_7:

n=1.

$$\Theta(\mathrm{k})_{1\times1}=P_7$$

$$\varphi(\mathrm{k})_{1\times1}=U_4[\mathrm{k}-1] \tag{5.10}$$

$$y(\mathrm{k})=\frac{\psi_2[\mathrm{k}]-\psi_2[\mathrm{k}-1]}{T_s}$$

At k=0,

$$\Theta(\mathrm{k}-1)_{1\times1}=0$$

$$P(\mathrm{k}-1)_{1\times1}=100 \tag{5.11}$$

To find P_8:

n=1.

$$\Theta(\mathrm{k})_{1\times1} = P_8$$
$$\varphi(\mathrm{k})_{1\times1} = \cos(\phi_1[\mathrm{k}-1])\cos(\theta_1[\mathrm{k}-1])U_1[\mathrm{k}-1] \tag{5.12}$$
$$y(\mathrm{k}) = \frac{z_2[\mathrm{k}] - z_2[\mathrm{k}-1] + \mathrm{g}\,T_s}{T_s}$$

At k=0,

$$\Theta(\mathrm{k}-1)_{1\times1} = 0$$
$$P(\mathrm{k}-1)_{1\times1} = 100 \tag{5.13}$$

The P_1-P_8 values are found by RLS with the help of given data. The values are:

Parameters found by RLS	Parameters Values
P1	-0.2676
P2	0.0032
P3	0.0004
P4	0.2626
P5	-0.0601
P6	0.0004
P7	$1.4119*10^{-5}$
P8	$2.4036*10^{-5}$

Table 5.1: Parameters found by RLS

5.4 MPC Control of Quadrotor

As discussed in section 5.1, the MPC algorithm requires a plant model for future output prediction, so the model given in equations 5.3 and 5.4 are used with parameters given in table 5.1. The MPC algorithm also requires some constrained optimization method, which can minimize the following cost function and find control signals, which gives optimal output. The lower limit of control signal is "0rad/sec" and upper limit is "400rad/sec". The cost function is:

$$j = \sum_{i=k}^{k+N_p} (\phi_{1\mathrm{Ref}}[\mathrm{i}] - \phi_1[\mathrm{i}])^2 + (\theta_{1\mathrm{Ref}}[\mathrm{i}] - \theta_1[\mathrm{i}])^2 + (\psi_{1\mathrm{Ref}}[\mathrm{i}] - \psi_1[\mathrm{i}])^2 + (\mathrm{z}_{1\mathrm{Ref}}[\mathrm{i}] - \mathrm{z}_1[\mathrm{i}])^2 \tag{5.14}$$

The "sequential programming" optimization method is used with the help of Matlab Optimization toolbox command "fmincon". The prediction horizon is taken "N_p= 20" and control horizon is taken "N_c= 2". The sampling time for MPC is "T_s= 0.1sec".

5.4.1 Results

The roll angle, pitch angle, yaw angle and Z distance are achieved as given below. Multiple simulations are performed.

Simulation 1:

Quadrotor is initially at ground and it has to move at z distance (height) of "1m" with achieving yaw angle of "1rad". Roll angle and Pitch angle has to remain at "0rad".

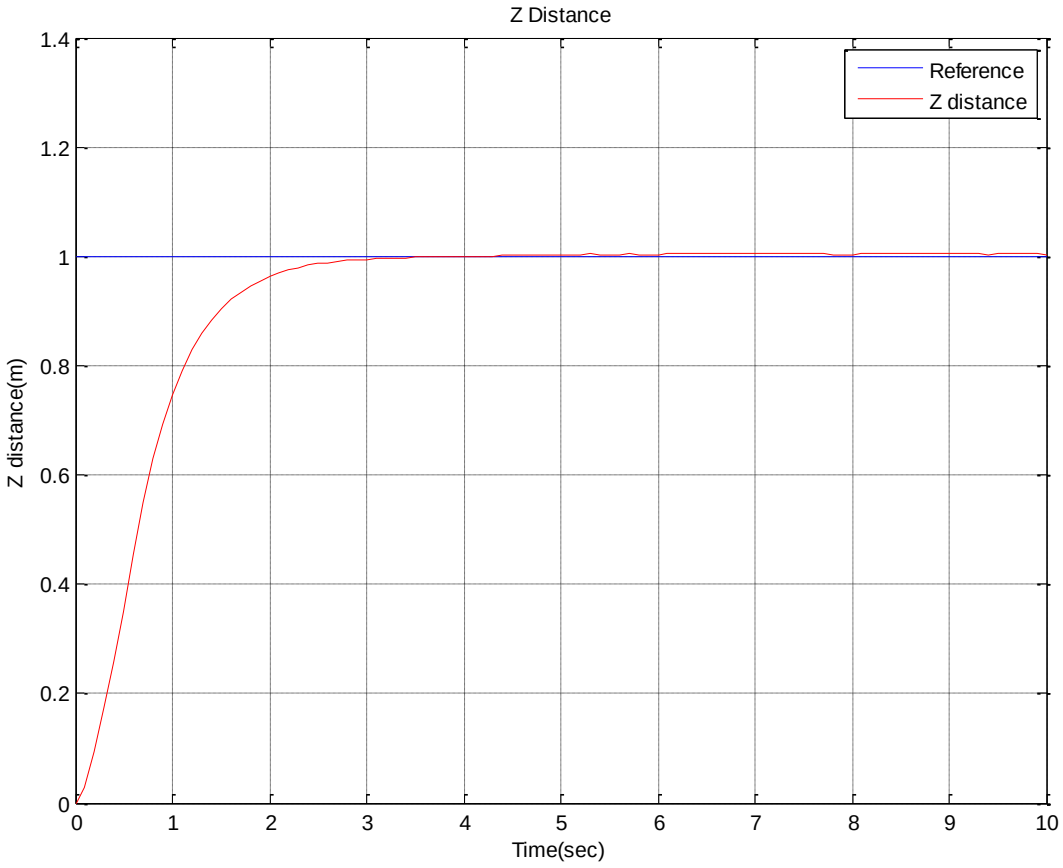


Figure 5.13: Z Distance

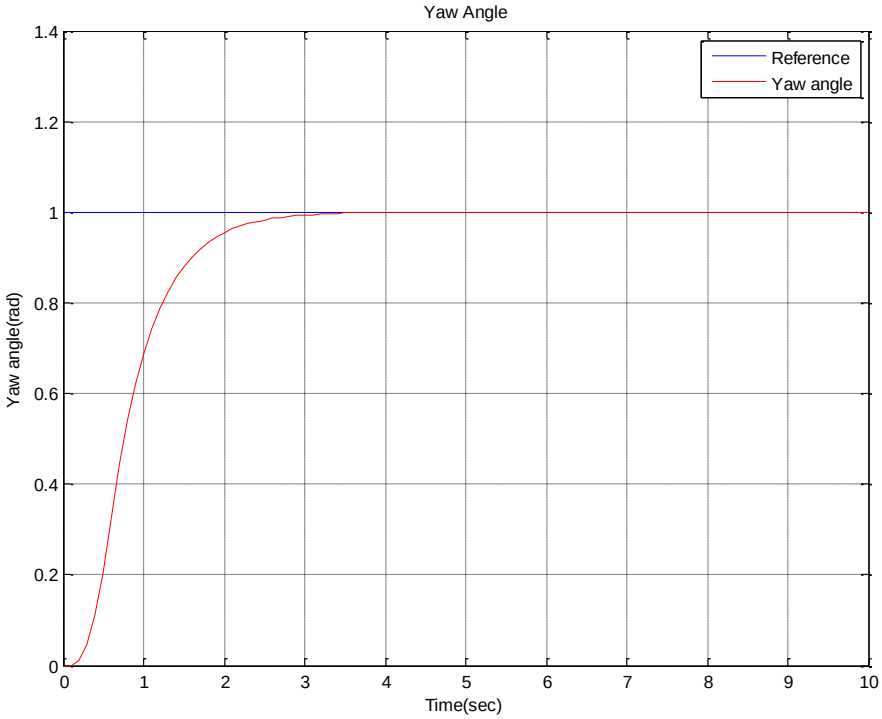


Figure 5.14: Yaw Angle

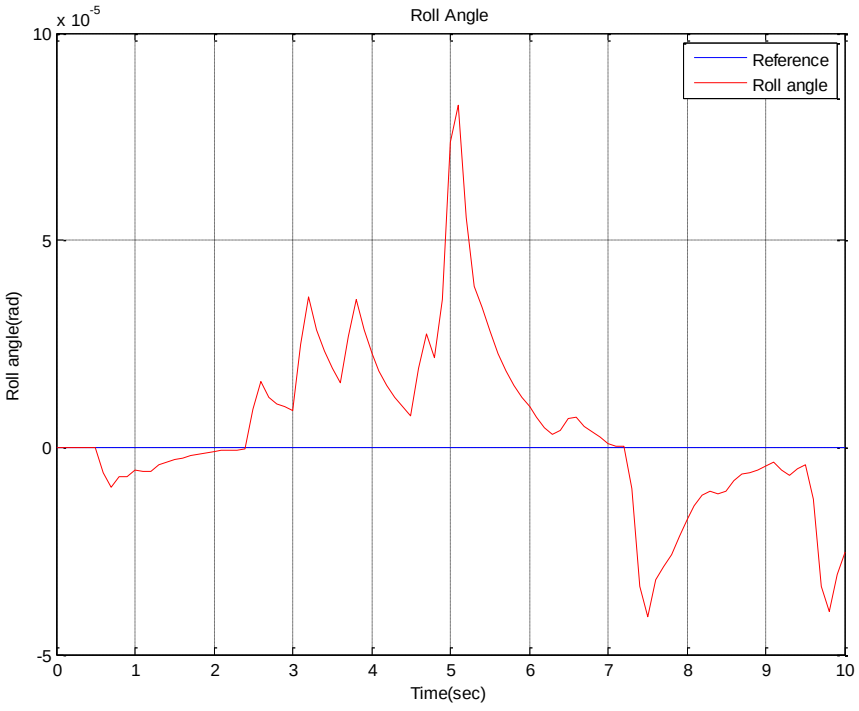


Figure 5.15: Roll Angle

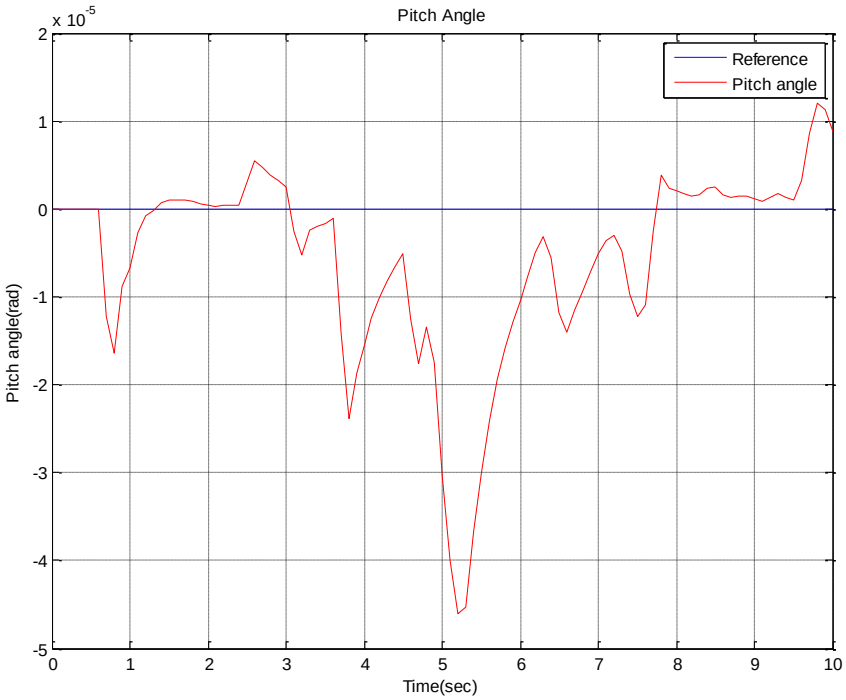


Figure 5.16: Pitch Angle

The control signals generated by MPC are:

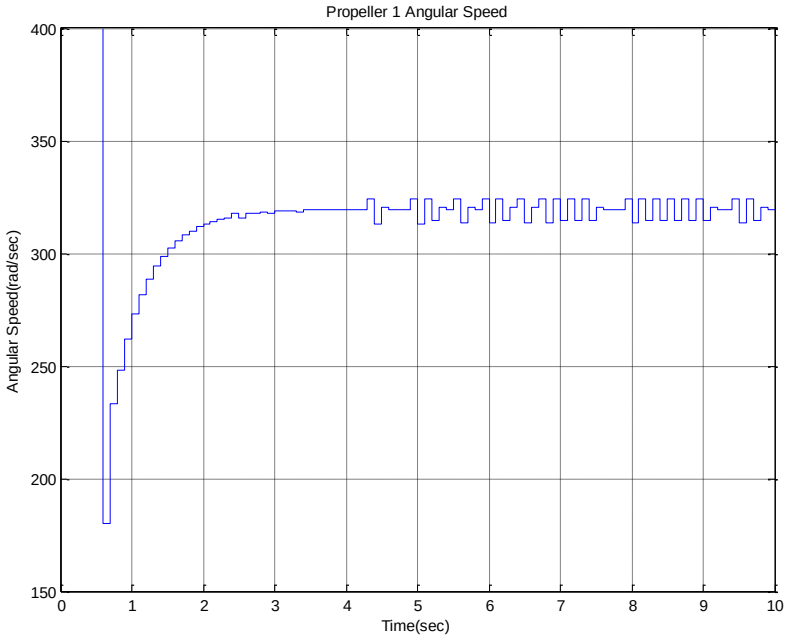


Figure 5.17: Propeller 1 Angular Speed

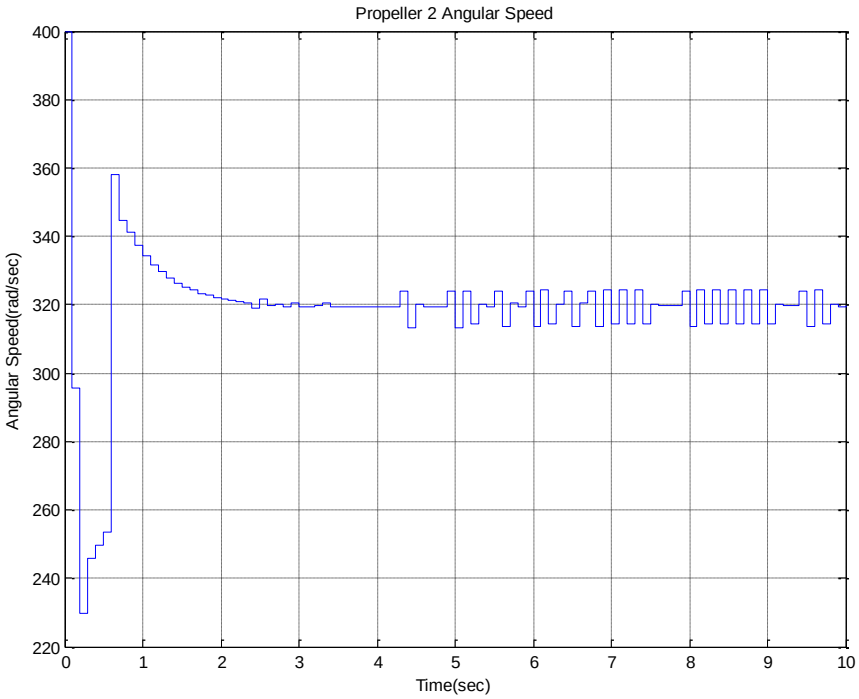


Figure 5.18: Propeller 2 Angular Speed

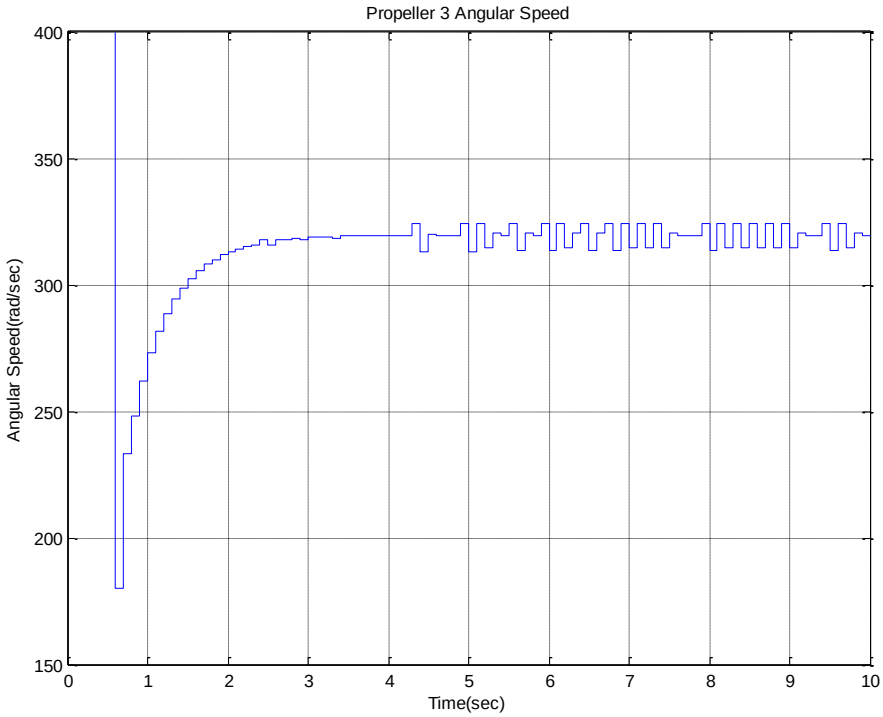


Figure 5.19: Propeller 3 Angular Speed

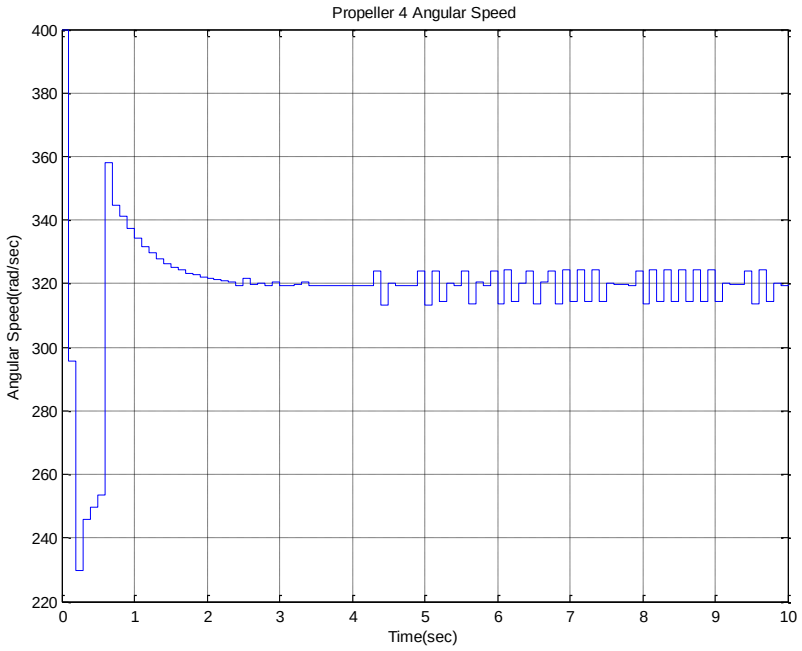


Figure 5.20: Propeller 4 Angular Speed

Simulation 2:

Quadrotor is initially at ground and it has to move at z distance (height) of “2m” with achieving yaw angle of “2rad”. Roll angle and Pitch angle has to remain at “0rad”.

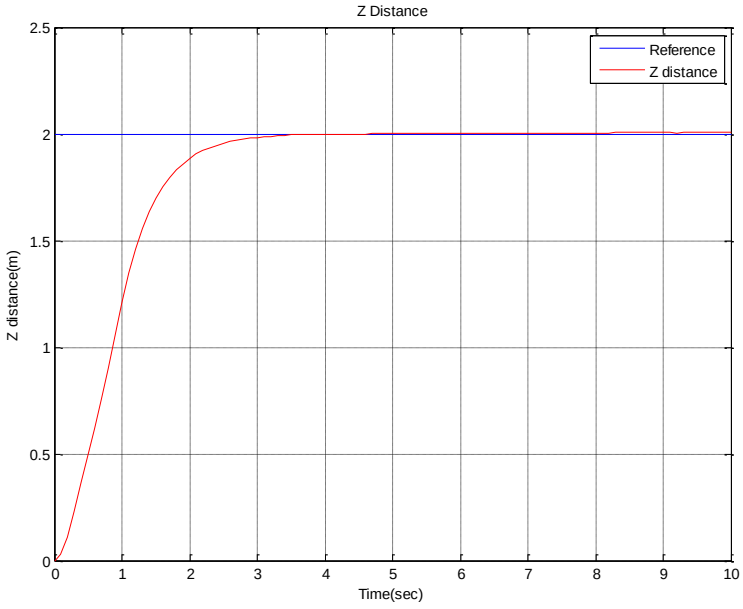


Figure 5.21: Z Distance

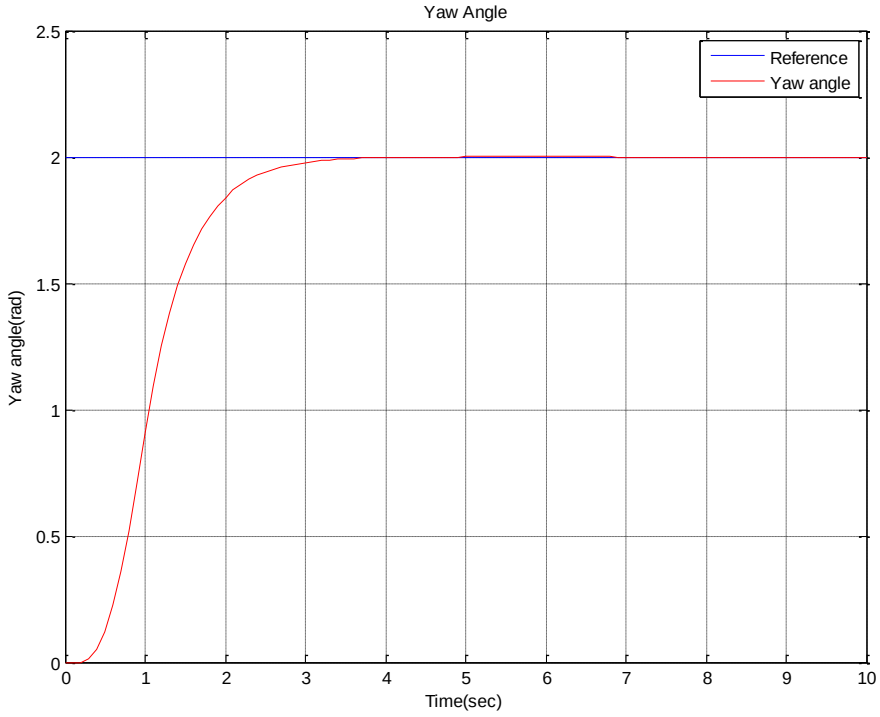


Figure 5.22: Yaw Angle

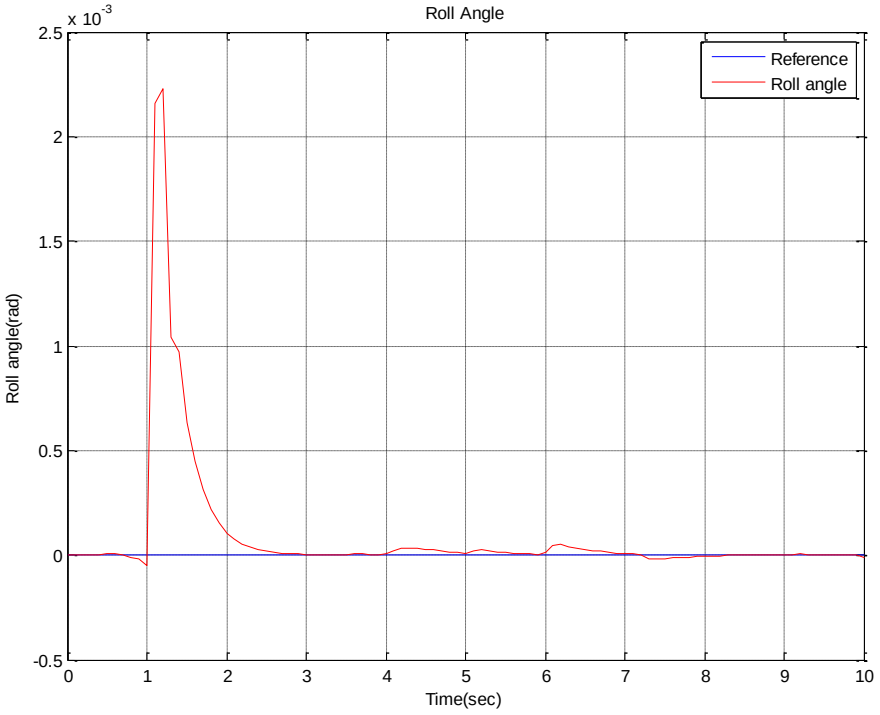


Figure 5.23: Roll Angle

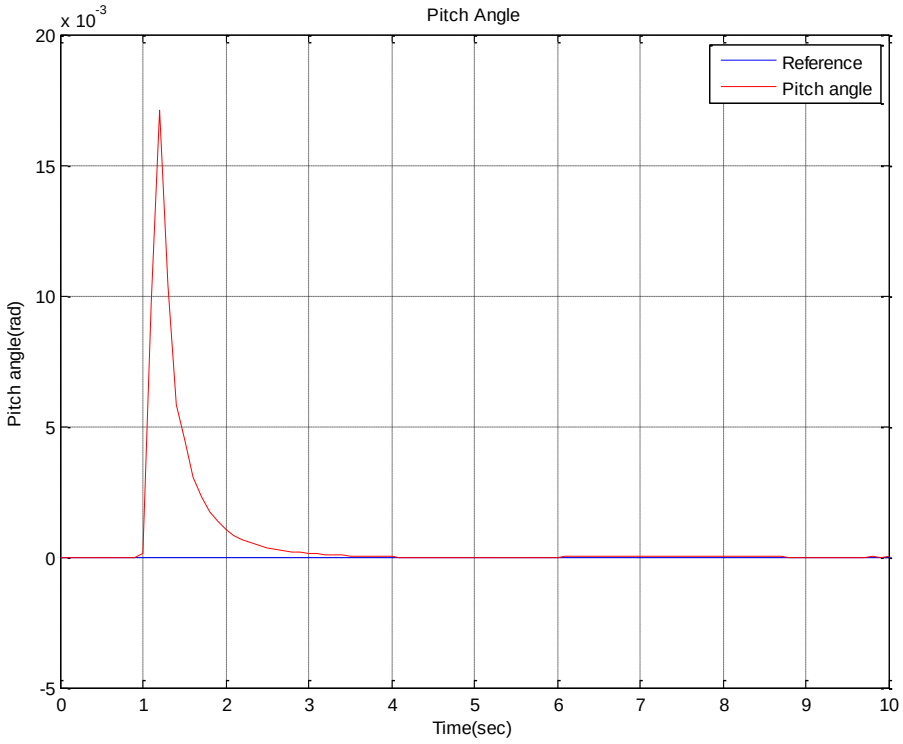


Figure 5.24: Pitch Angle

The control signals generated by MPC are:

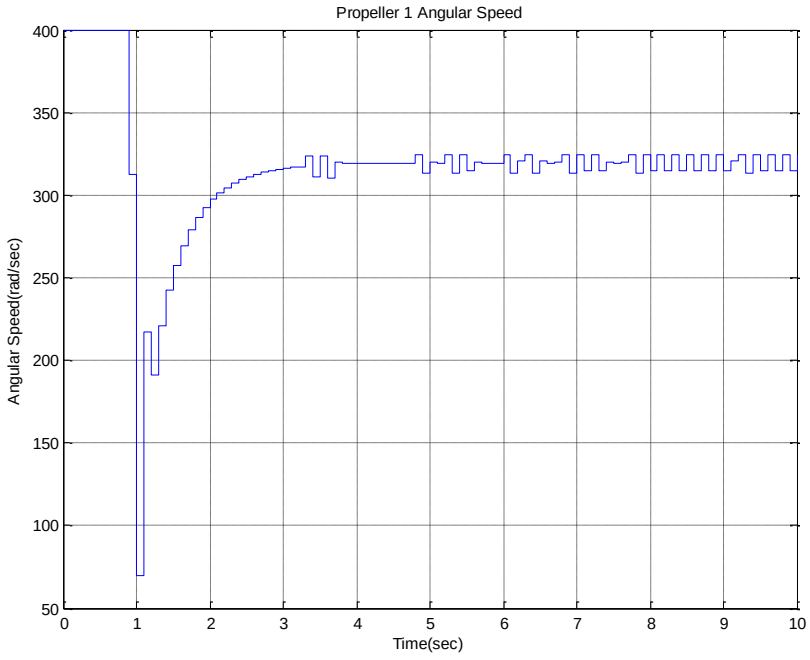


Figure 5.25: Propeller 1 Angular Speed

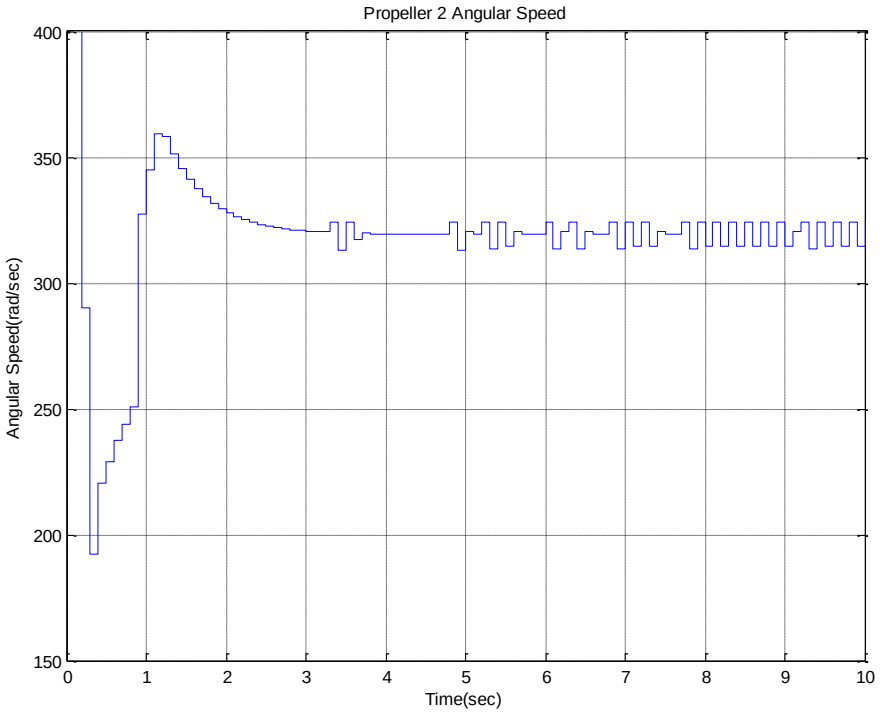


Figure 5.26: Propeller 2 Angular Speed

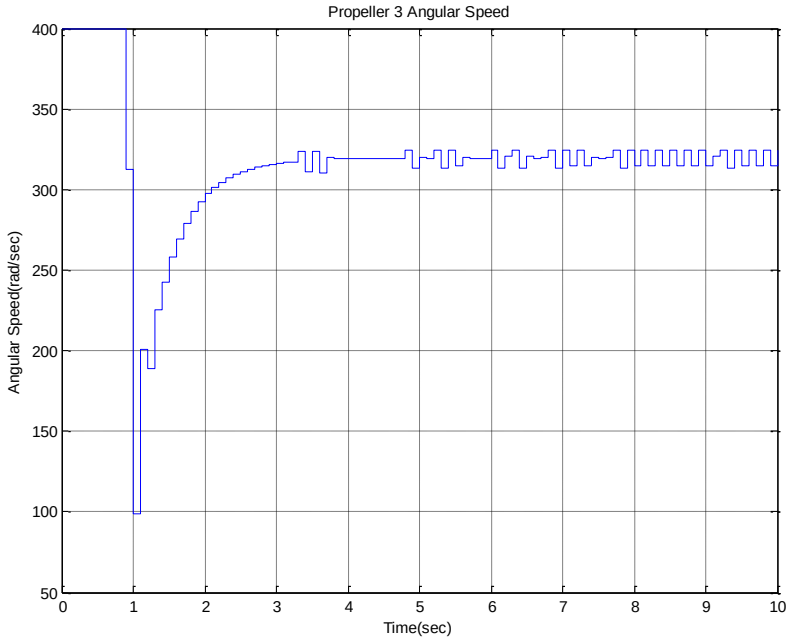


Figure 5.27: Propeller 3 Angular Speed

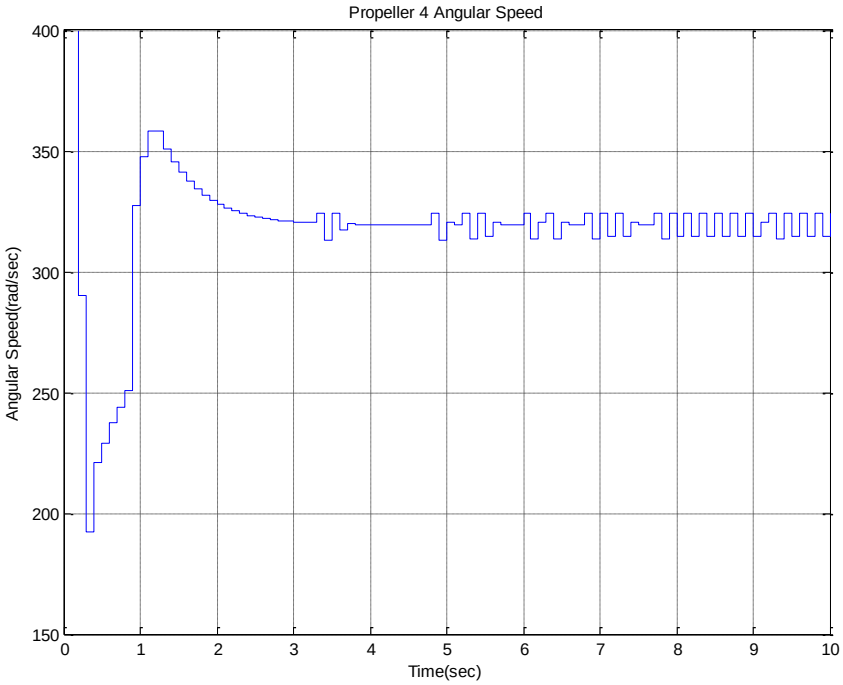


Figure 5.28: Propeller 4 Angular Speed

Simulation 3:

Quadrotor is initially at height of “1m” and it has to maintain “1m” height with yaw angle at “0rad”. Roll angle and Pitch angle are to be achieved at “0.5rad”.

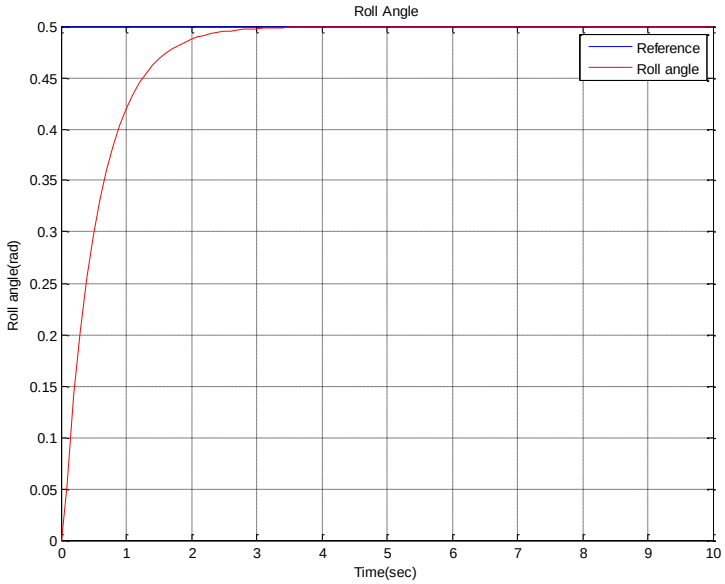


Figure 5.29: Roll Angle

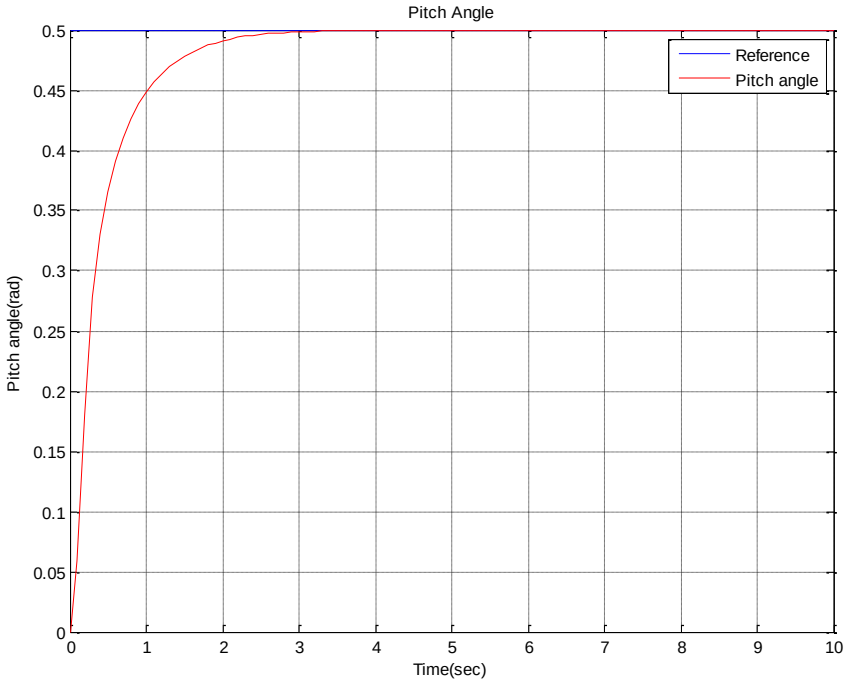


Figure 5.30: Pitch Angle

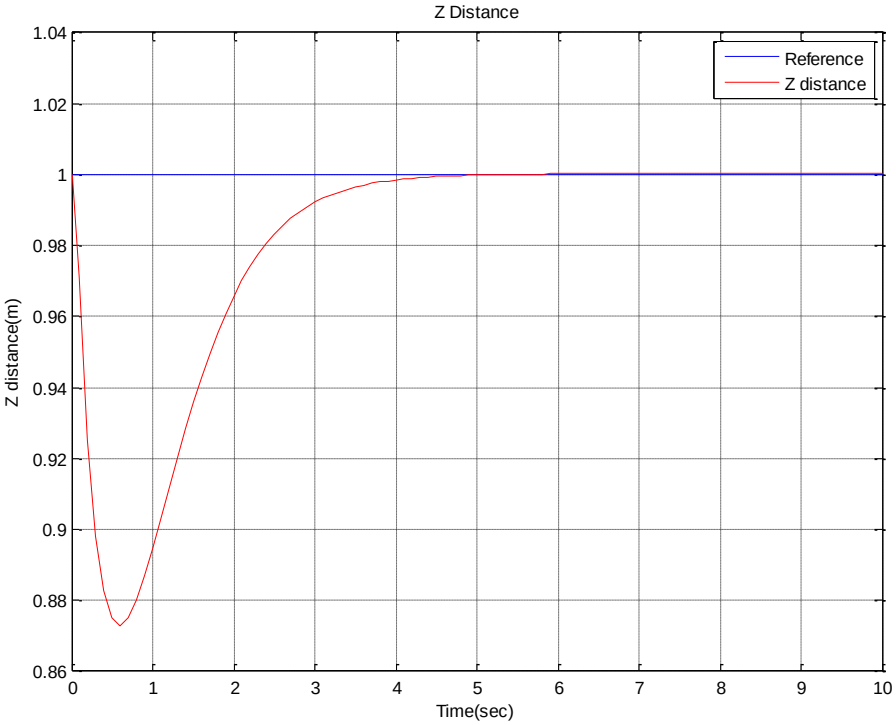


Figure 5.31: Z Distance

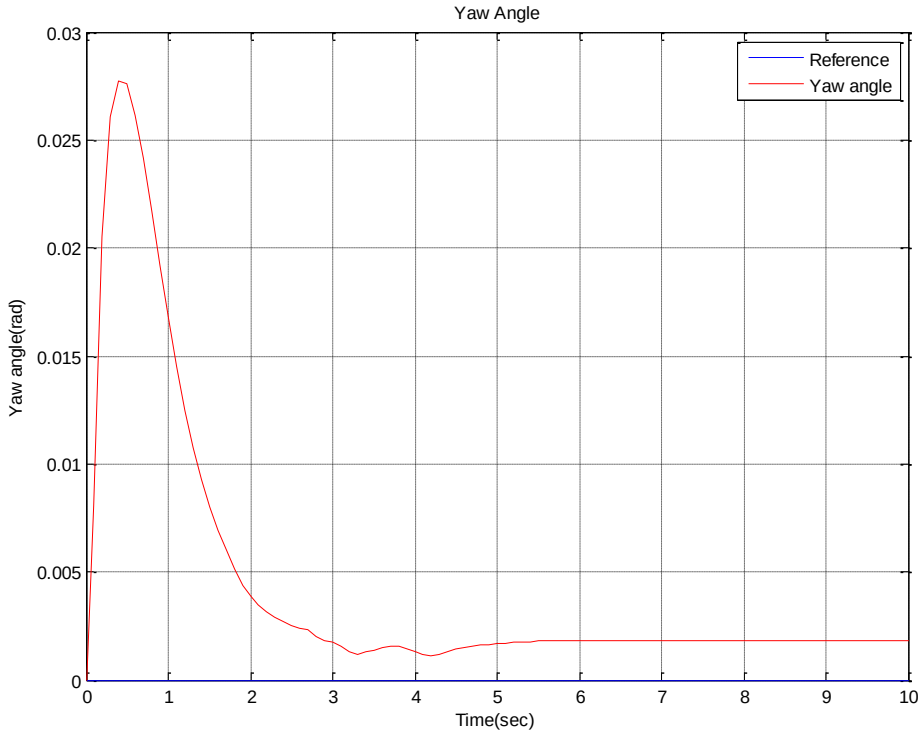


Figure 5.32: Yaw Angle

The control signals generated by MPC are:

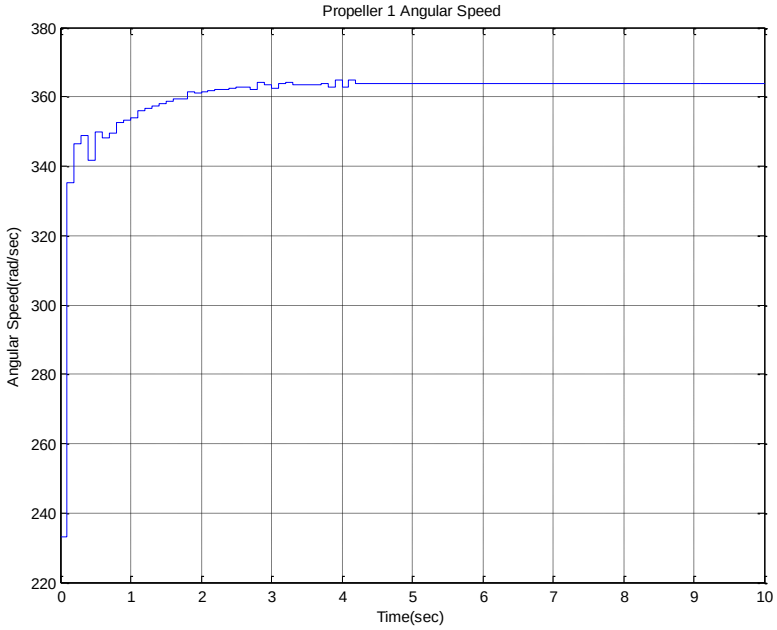


Figure 5.33: Propeller 1 Angular Speed

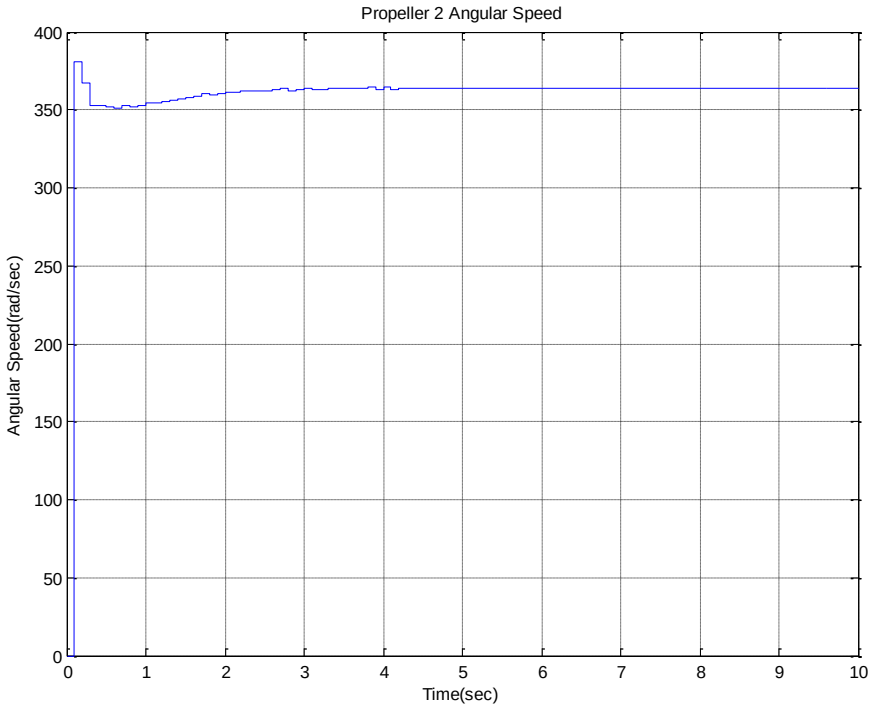


Figure 5.34: Propeller 2 Angular Speed

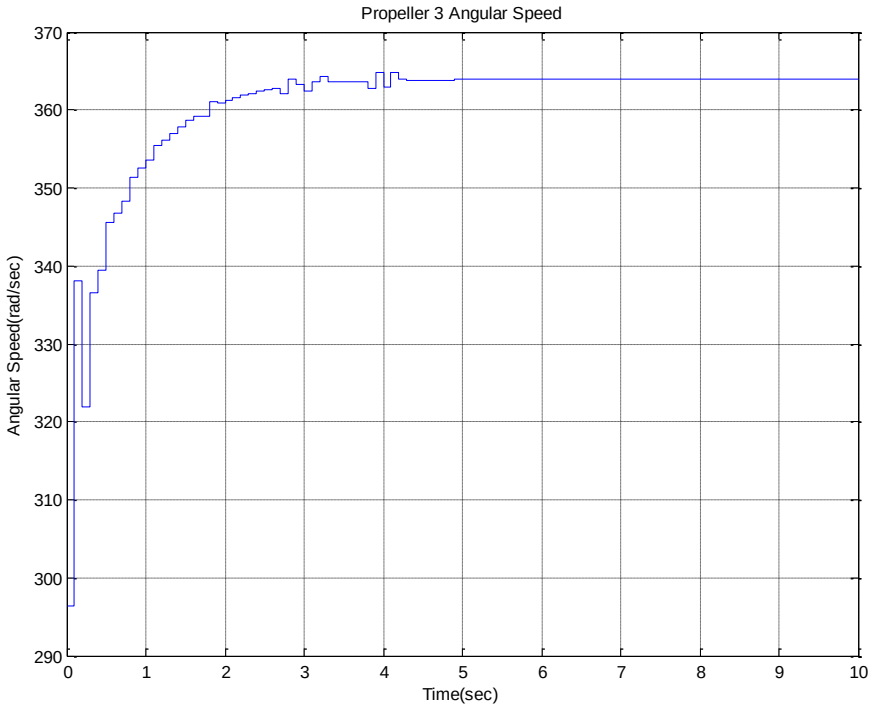


Figure 5.35: Propeller 3 Angular Speed

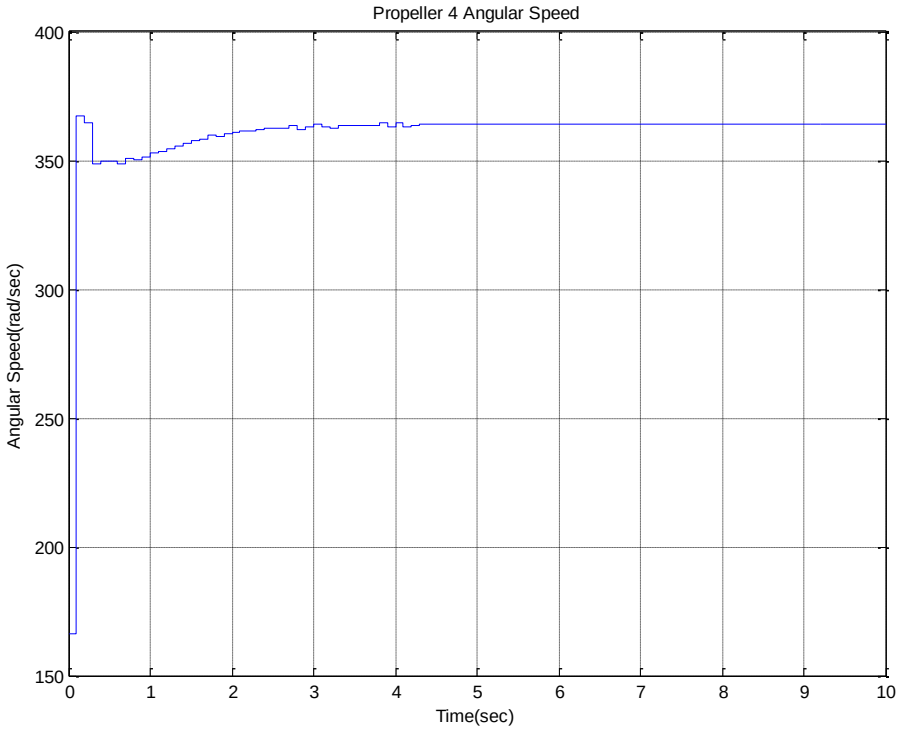


Figure 5.36: Propeller 4 Angular Speed

3 different simulations are conducted to show the effectiveness of MPC controller in different scenarios and different references. In all simulations, results have shown good control performance and reference tracking.

5.5 Summary

This chapter has presented discrete non-linear model of quadrotor and system identification of the model by using recursive least squares algorithm. This discrete non-linear model with identified parameters, found by RLS, is used in model predictive control strategy. The main advantage of the proposed scheme lies in system identification. Most of the time, the accurate parameters are not known. This makes the practical implementation very difficult. Proposed scheme finds the parameters directly from quadrotor data. The MPC is an optimal control strategy, so output will always be at optimal performance. The proposed controller has shown good performance of quadrotor.

CHAPTER 6 Conclusion of Thesis

6 Conclusion of Thesis

The main focus of the thesis is on quadrotor controlling. Quadrotor is unstable system and it can not fly in open loop. In order to operate quadrotor, a close loop control strategy is required. This thesis has presented two control strategies for quadrotor controlling. One strategy is based on discrete PID control. This strategy uses discrete PD, PI and PID controllers. This PID based scheme is able to control both inner loop control (roll angle, pitch angle and yaw angle) of quadrotor and also outer loop control of quadrotor (a specific point in 3-dimensional space). This PID scheme is chosen to be discrete, so it can be implemented on any digital embedded system, like processor.

The outer loop control strategy is implemented on PSOC device, which contain 32-bit ARM processor. The implemented control strategy is tested on quadrotor dynamic model by using hardware in loop simulation. The results have shown the satisfactory performance of this embedded controller. In thesis, a simple discrete PI controller is also implemented on 8051 microcontroller by using assembly language and on PSOC by using C language. This controller is tested on DC motor speed model with hardware in loop simulation. This also has shown satisfactory performance.

The PID control scheme is able to control quadrotor but its results are not optimal. This thesis has also presented model predictive control (MPC) of quadrotor, which is optimal in nature. MPC scheme requires a plant model, so quadrotor non-linear model is discretized and used. Quadrotor model has some parameters, which needs to be accurate for good performance but these parameters are not always known. This thesis has presented the system identification of these parameters from data obtained by PID control of quadrotor. MPC uses discrete non-linear model with identified parameters. MPC find the optimal control signals with the help of constrained optimization method like sequential programming and gives the optimal control performance. The results have shown good performance.

6.1 Future Recommendations

- The designed embedded PID controller needs to be tested on real quadrotor hardware.
- In MPC, branch and bound optimization method has to be included. It makes the control scheme, parallel in nature and its sampling time will be reduced. So for practical implementation, it will not require a very high speed processor but it will require device capable of doing parallel processing like FPGA.

References

7 References

[1] "Fact Sheet Display: MQ-9 Reaper". United States Air Force. 18 August 2010. Retrieved 27 September 2013.

[2] http://www.bga-aeroweb.com/Defense/MQ-8-Fire-Scout.html.

[3] http://ardrone2.parrot.com.

[4] Pounds P, Mahony R, Hynes P, Roberts J. Design of a four-rotor aerial robot. In: Australasian conference on robotics & automation, Auckland, New Zealand; 2002. p. 145–50.

[5] Pounds P, Mahony R, Corke P. Modelling and control of a quad-rotor robot. In: Proceedings of the Australasian conference on robotics & automation, Auckland, New Zealand; 2006.

[6] Hoffmann G, Huang H, Waslander S, Tomlin C. Quadrotor helicopter flight dynamics and control: theory and experiment. In: Proceedings of the AIAA guidance, navigation & control conference, Hilton Head, SC, USA, 2007 [aIAA Paper Number 2007-6461].

[7] Hoffmann G, Waslander S, Tomlin C. Quadrotor helicopter trajectory tracking control. In: Proceedings of the AIAA guidance, navigation & control conference, Honolulu, HI, USA; 2008 [aIAA Paper Number 2008-7410].

[8] Bouabdallah S, Siegwart R. Field and service robotics. Springer tracts in advanced robotics. Berlin/Heidelberg: Springer; 2006. p. 429–40 [chapter: Towards intelligent miniature flying robots].

[9] S. Bouabdallah, A. Noth, R. Siegwart, PID vs LQ control techniques applied to an indoor micro quadrotor, in: Proceedings of the IEEE International Conference on Robotics and Automation, Barcelona, Spain, April, (2005), pp. 2259–2264.

[10] A. Tayebi, S. McGilvray, Attitude stabilization of a VTOL quadrotor aircraft, IEEE Transactions on Control Systems Technology 14 (May (3)) (2006).

[11] Kostas Alexis, George Nikolakopoulos, Anthony Tzes, Switching model predictive attitude control for a quadrotor helicopter subject to atmospheric disturbances, Control Engineering Practice 19 (2011) 1195-1207.

[12] Mehmet Onder Efe, Neural network assisted computationally simple $PI^{\lambda}D^{\mu}$ control of a quadrotor UAV, IEEE Transactions on Industrial Informatics, Vol. 7, No. 2, May 2011.

[13] Altug E, Taylor C. Vision-based pose estimation and control of a model helicopter. In: Proceedings of the 2004 IEEE international conference on mechatronics, Istanbul, Turkey; 2004.

[14] S.L. Waslander, G. Hoffmann, J.S. Jang, C.J. Tomlin, Multi-agent X4-flyer testbed control design: integral sliding mode vs. reinforcement learning, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005, 468–473.

[15] G. Hoffmann, D.G. Rajnarayan, S.L. Waslander, D. Dostal, J.C. Jang, C.J. Tomlin, The stanford testbed of autonomous rotorcraft for multi-agent control (STARMAC), in: Proceedings of the 23rd Digital Avionics System Conference, Salt Lake City, UT, November, 2004.

[16] G. Hoffmann, S.L. Waslander, Distributed cooperative search using information— theoretic costs for particle filters, with quadrotor applications, in: AIAA Guidance, Navigation, and Control Conference and Exhibit, Keystone, CO, August, 2006.

[17] L. Mederreg, F. Diaz, N.K. M'sirdi, Nonlinear backstepping control with observer design for a 4 rotors helicopter, in: AVCS'04, International Conference on Advances in Vehicle Control and Safety, Genova, Italy, October, (2004), pp. 28–31.

[18] T. Hamel, R. Mahony, Pure 2D Visual Servo control for a class of under-actuated dynamic systems, in: Proceedings of the 2002 IEEE International Conference on Robotics and Automation, New Orleans, LA, (2004), pp. 2229–2235.

[19] Dunfied J, Tarbouchi M, Labonte G. Neural network based control of a four rotor helicopter. In: Proceedings of the 2004 IEEE international conference on industrial technology, Hammamet, Tunisia; 2004. p. 1543–8.

[20] Cesareo Raimundez J, Fernandez Villaverde A. Adaptive tracking control for a quad-rotor. In: Proceedings of ENOC-2008, Saint Petersburg, Russia; 2008.

[21] Dierks T, Jagannathan S. Neural network output feedback control of a quadrotor UAV. In: Proceedings of the 2008 IEEE conference on decision and control, Cancun, Mexico; 2008. p. 3633–9.

[22] Ashraf Saleem, Rateb Issa, Tarek Tutuni, "Hardware-in-the-loop for on-line identification and control of three-phase squirrel cage induction motors," Simulation Modelling Practice and Theory 18 (2010) 277-290.

[23] R. Isermann, J. Schaffnit, S. Sinsel, "Hardware-in-the-loop simulation for the design and testing of engine-control systems," Control Engineering Practice 7 (1999) 643}653.

[24] Schlegel C., Bross M., Beater P., "HIL Simulation of the hydraulics and mechanics of an automatic gearbox," 2nd International Modelica conference, proceedings, PP. 67-75.

[25] Masaya Harakawa, Hisanori Yamasaki, Tetsuaki Nagano, Simon Abourida, Christian Dufour, Jean Bélanger, "Real-Time Simulation of a Complete PMSM Drive at 10 μs Time Step," Paper presented at the 2005 International Power Electronics Conference, Niigata, Japan (IPEC-Niigata 2005).

[26] Piotr Wozniak, "Preferences in multi-objective evolutionary optimization of electric motor speed control with hardware in the loop", Applied Soft Computing 11 (2011) 49-55.

[27] P. Stewart, D.A.Stone, P.J.Fleming, "Design of robust fuzzy logic control systems by multi-objective evolutionary methods with hardware in the loop", Engineering Applications of Artificial Intelligence 17 (2004) 275-284.

[28] Dukelow, S. G., The Control of Boilers, 2nd ed. Instrument Society of America, 1991.

[29] Shinskey, F. G., Averaging level control. Chem. Eng. Process. 60 ~9!, 58 ~1997!.

[30] Eborn, J., Panagopoulos, H., and Åstro¨m, K. J., Robust PID control of steam generator water level. IFAC'99 14th World Congress of IFAC. Beijing, P. R. China, 1999.

[31] Panagopoulos, H., Åstro¨m, K. J., and Ha¨gglund, T., Design of PID controllers based on constrained optimization. 1999 American Control Conference. San Diego, California, 1999.

[32] David Cartes, Lei Wu, "Experimental evaluation of adaptive three-tank level control", ISA Transaction 44 (2005) 283-293.

[33] A. Boubakir, F.B., C. Boubakir and S.Laboid, A fuzzy sliding mode controller using nonlinear sliding surface applied to coupled tanks system International Journal of Fuzzy Systems, 2008. 10(2).

[34] http://lucasamor.im/wp-content/uploads/2014/01/quadrotor_diagram.jpg

[35] http://radhesh.files.wordpress.com/2008/05/pid.jpg

[36] Franklin, G.F., Powell, D.J., and Workman, M.L., "Digital Control of Dynamic Systems (3rd Edition)" Prentice Hall, 1997.

[37] Kuo, B. C. (1987). Automatic control systems. Englewood Cliffs, NJ: Prentice Hall. Ch4.

[38] Tipsuwan, Y., & Chow, M.-Y., 1999. Fuzzy logic microcontroller implementation for DC motor speed control. The 25th annual conference of the IEEE industrial electronics society (IECON 99), Vol. 3 (pp. 1271–1276). San Jose, CA.

[39] Guanrong chen, Trung Tat Pham, "Introduction to fuzzy sets, fuzzy logic, and fuzzy control systems," CRC Press LLC, 2000 N.W. Corporate Blvd., Boca Raton, Florida 33431.

[40] Aidan O'Dwyer (*Dublin Institute of Technology, Ireland*), "Handbook Of PI And PID Controller Tuning Rules," 3rd Edition, ISBN: 978-1-84816-242-6.

[41] ATMEL 8051 Microcontroller hardware manual.

[42] http://www.cypress.com/psoc5lp/?source=CY-ENG-HEADER

[43] http://upload.wikimedia.org/wikipedia/commons/thumb/1/11/MPC_scheme_basic.svg/434px MPC_scheme_basic.svg.png

[44] Karl J. Astrom, Bjorn Wittenmark, Adaptive control, 2nd Ed, Addison-Wesley Publishing Company, Inc.

Appendix A

8 Appendix A

8.1 Discrete PI Controller in Assembly Language

The assembly language code for the implementation of discrete PI controller discussed in section 4.1.2 is given below:

```
ERRORC    EQU   30H
ERRORP    EQU   31H
CSIGC     EQU   32H
CSIGP     EQU   33H
OUTPUT    EQU   34H

          ORG   0H

          MOV   ERRORC,#0
          MOV   ERRORP,#0
          MOV   CSIGC,#0
          MOV   CSIGP,#0
          MOV   OUTPUT,#0

          MOV   TMOD,#20H
          MOV   TH1,#-3
          MOV   SCON,#50H
          SETB  TR1

HERE:     JNB   RI,HERE
          MOV   A,SBUF
          CLR   RI

          MOV   OUTPUT,A
          MOV   A,#100
          CLR   C
          SUBB  A,OUTPUT
          MOV   ERRORC,A

          JB    ACC.7,L1
```

```
        MOV  B,#1
        DIV  AB
        MOV  B,#1
        MUL  AB
        MOV  R0,A
        JMP  L2

L1:     CPL  A
        INC  A
        MOV  B,#1
        DIV  AB
        MOV  B,#1
        MUL  AB
        CPL  A
        INC  A
        MOV  R0,A

L2:     MOV  A,ERRORP
        JB   ACC.7,L3
        MOV  B,#1
        DIV  AB
        MOV  B,#1
        MUL  AB
        MOV  R1,A
        JMP  L4

L3:     CPL  A
        INC  A
        MOV  B,#1
        DIV  AB
        MOV  B,#1
        MUL  AB
        CPL  A
        INC  A
        MOV  R1,A

L4:     MOV  A,R0
        CLR  C
        SUBB A,R1
        ADD  A,CSIGP
        MOV  R3,A

        CJNE A,#100,NEXT
NEXT:   JC   H1
        MOV  A,#100
        JMP  H2
```

```
H1:         CJNE  A,#-100,NEXT1
NEXT1:      JNC   H2
            MOV   A,#-100

H2:         MOV   CSIGC,A

            CLR   TI
            MOV   SBUF,CSIGC
HERE1:      JNB   TI,HERE1
            CLR   TI

            MOV   ERRORP,ERRORC
            MOV   CSIGP,CSIGC

            JMP   HERE

            END
```

8.2 Discrete PI Controller in C Language

The C language code for the implementation of discrete PI controller discussed in section 4.1.3 is given below:

```
#include <device.h>

void main()
{
    uint8 a2,a1,num=0;/*Error Current*/   /*Little Endian a2,a1*/
    uint8 c2,c1;/*Control Signal Current*/
    uint16 aa2,aa1,aa,l;
    int16 cc,cc2,cc1;
    float errorc=0,errorp=0,csigc=0,csigp=0,output=0;


    UART_Start();
    isr_1_Start();
    isr_2_Start();
    LCD_Start();

    for(;;)
    {   if (UART_GetRxBufferSize()!=0 && num==0)
        {a1=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==1)
        {a2=UART_ReadRxData();num++;}
        if (num==2)
        {
        aa1=(uint16)a1;
        aa2=(uint16)a2;
        aa2=(aa2<<8)&0xff00;
```

```
        aa=aa2|aa1;
        LCD_ClearDisplay();
        LCD_Position(0,0);
        LCD_PrintNumber(aa);

        output=(float)aa;
        errorc=10000-output;
        csigc=csigp+1.005*errorc-0.995*errorp;
        if (csigc>=10000)
        {csigc=10000;}
        else if(csigc<=-10000)
        {csigc=-10000;}
        errorp=errorc;
        csigp=csigc;

        cc=(int16)csigc;
        l=(uint16)cc;
        LCD_Position(1,0);
        LCD_PrintNumber(l);

        cc1=cc;cc2=cc;
        cc2=(cc2>>8)&0x00ff;
        c2=(uint8)cc2;
        cc1=cc1&0x00ff;
        c1=(uint8)cc1;

        UART_WriteTxData(c1);
        UART_WriteTxData(c2);
        num=0;
        }

    }
}

/* [] END OF FILE */
```

8.3 Quadrotor Control Strategy on C Language

The C language code for the implementation of quadrotor control strategy discussed in section 4.2 is given below:

```
#include <device.h>
#include <math.h>

void main()
{
    uint8 a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,num=0;
    uint16
aa2,aa1,aaa12,aa4,aa3,aaa34,aa6,aa5,aaa56,aa8,aa7,aaa78,aa10,aa9,aaa910,aa12,
aa11,aaa1112,aa14,aa13,aaa1314,sample=0;
    int16 ac12,aa34,aa56,aa78,aa910,aa1112,aa1314;
    int32 vvv1,vv1,vv2,vv1a,vv2a;
    uint8 v1,v2,v1a,v2a,v3,v4,v3a,v4a,v5,v6,v5a,v6a,v7,v8,v7a,v8a;
    double T=0.01,N=100,r1,r2,r3,r4,time;
```

```
    double
roll_angle,ref_roll_angle,e_roll_angle,ep_roll_angle=0,u_roll_angle,up_roll_a
ngle=0,P_roll_angle=5,D_roll_angle=5;
    double
pitch_angle,ref_pitch_angle,e_pitch_angle,ep_pitch_angle=0,u_pitch_angle,up_p
itch_angle=0,P_pitch_angle=5,D_pitch_angle=5;
    double
yaw_angle,ref_yaw_angle,e_yaw_angle,ep_yaw_angle=0,epp_yaw_angle=0,u_yaw_angl
e,up_yaw_angle=0,upp_yaw_angle=0,P_yaw_angle=0.5,I_yaw_angle=0.01,D_yaw_angle
=2;
    double
x_distance,ref_x_distance,e_x_distance,ep_x_distance=0,u_x_distance,up_x_dist
ance=0,P_x_distance=0.01,D_x_distance=0.05;
    double
y_distance,ref_y_distance,e_y_distance,ep_y_distance=0,u_y_distance,up_y_dist
ance=0,P_y_distance=0.01,D_y_distance=0.05;
    double
z_distance,ref_z_distance,e_z_distance,ep_z_distance=0,u_z_distance,up_z_dist
ance=0,P_z_distance=20,D_z_distance=20;
    double
z_velocity,ref_z_velocity,e_z_velocity,ep_z_velocity=0,u_z_velocity,up_z_velo
city=0,P_z_velocity=5,I_z_velocity=5;
    UART_Start();
    isr_1_Start();
    isr_2_Start();


    for(;;)
    {
    if (UART_GetRxBufferSize()!=0 && num==0)
        {a1=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==1)
        {a2=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==2)
        {a3=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==3)
        {a4=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==4)
        {a5=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==5)
        {a6=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==6)
        {a7=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==7)
        {a8=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==8)
        {a9=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==9)
        {a10=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==10)
        {a11=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==11)
        {a12=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==12)
        {a13=UART_ReadRxData();num++;}
        if (UART_GetRxBufferSize()!=0 && num==13)
        {a14=UART_ReadRxData();num++;}
```

```
if(num==14)
{
aa1=(uint16)a1;
aa2=(uint16)a2;
aa2=(aa2<<8)&0xff00;
aaa12=aa2|aa1;
ac12=(int16)aaa12;
roll_angle=(double)ac12;
roll_angle=roll_angle/1000;

aa3=(uint16)a3;
aa4=(uint16)a4;
aa4=(aa4<<8)&0xff00;
aaa34=aa4|aa3;
aa34=(int16)aaa34;
pitch_angle=(double)aa34;
pitch_angle=pitch_angle/1000;

aa5=(uint16)a5;
aa6=(uint16)a6;
aa6=(aa6<<8)&0xff00;
aaa56=aa6|aa5;
aa56=(int16)aaa56;
yaw_angle=(double)aa56;
yaw_angle=yaw_angle/1000;

aa7=(uint16)a7;
aa8=(uint16)a8;
aa8=(aa8<<8)&0xff00;
aaa78=aa8|aa7;
aa78=(int16)aaa78;
x_distance=(double)aa78;
x_distance=x_distance/1000;

aa9=(uint16)a9;
aa10=(uint16)a10;
aa10=(aa10<<8)&0xff00;
aaa910=aa10|aa9;
aa910=(int16)aaa910;
y_distance=(double)aa910;
y_distance=y_distance/1000;

aa11=(uint16)a11;
aa12=(uint16)a12;
aa12=(aa12<<8)&0xff00;
aaa1112=aa12|aa11;
aa1112=(int16)aaa1112;
z_distance=(double)aa1112;
z_distance=z_distance/1000;

aa13=(uint16)a13;
aa14=(uint16)a14;
aa14=(aa14<<8)&0xff00;
aaa1314=aa14|aa13;
aa1314=(int16)aaa1314;
z_velocity=(double)aa1314;
z_velocity=z_velocity/1000;
```

```
        time=sample*T;
        if(time>=0 && time<25)
        {ref_x_distance=1-exp(-time/2);
        ref_y_distance=1-exp(-time/2);
        ref_z_distance=1-exp(-time/2);}
        else if(time>=25 && time<50)
        {ref_x_distance=3-3*exp(-(time-25)/2)+1;
        ref_y_distance=3-3*exp(-(time-25)/2)+1;
        ref_z_distance=3-3*exp(-(time-25)/2)+1;}
        else if(time>=50 && time<75)
        {ref_x_distance=2*exp(-(time-50)/2)+2;
        ref_y_distance=2*exp(-(time-50)/2)+2;
        ref_z_distance=2*exp(-(time-50)/2)+2;}
        else if(time>=75 && time<100)
        {ref_x_distance=1-exp(-(time-75)/2)+2;
        ref_y_distance=1-exp(-(time-75)/2)+2;
        ref_z_distance=1-exp(-(time-75)/2)+2;}
        else if(time>=100 && time<125)
        {ref_x_distance=2*exp(-(time-100)/2)+1;
        ref_y_distance=2*exp(-(time-100)/2)+1;
        ref_z_distance=2*exp(-(time-100)/2)+1;}
        else
        {ref_x_distance=1;
        ref_y_distance=1;
        ref_z_distance=1;}


        if (time>=125)
        {ref_yaw_angle=0.01*(time-125);}
        else {ref_yaw_angle=0;}

        //y_distance(PD)
        e_y_distance=ref_y_distance-y_distance;

u_y_distance=(P_y_distance+D_y_distance*N)*e_y_distance+(P_y_distance*N*T-
P_y_distance-D_y_distance*N)*ep_y_distance-(N*T-1)*up_y_distance;

        //roll_angle(PD)
        ref_roll_angle=-u_y_distance;
        e_roll_angle=ref_roll_angle-roll_angle;

u_roll_angle=(P_roll_angle+D_roll_angle*N)*e_roll_angle+(P_roll_angle*N*T-
P_roll_angle-D_roll_angle*N)*ep_roll_angle-(N*T-1)*up_roll_angle;

        //x_distance(PD)
        e_x_distance=ref_x_distance-x_distance;

u_x_distance=(P_x_distance+D_x_distance*N)*e_x_distance+(P_x_distance*N*T-
P_x_distance-D_x_distance*N)*ep_x_distance-(N*T-1)*up_x_distance;

        //pitch_angle(PD)
        ref_pitch_angle=u_x_distance;
        e_pitch_angle=ref_pitch_angle-pitch_angle;

u_pitch_angle=(P_pitch_angle+D_pitch_angle*N)*e_pitch_angle+(P_pitch_angle*N*
T-P_pitch_angle-D_pitch_angle*N)*ep_pitch_angle-(N*T-1)*up_pitch_angle;
```

```
        //yaw_angle(PID)
        e_yaw_angle=ref_yaw_angle-yaw_angle;

u_yaw_angle=(P_yaw_angle+D_yaw_angle*N)*e_yaw_angle+(N*T*P_yaw_angle+I_yaw_an
gle*T-2*D_yaw_angle*N-2*P_yaw_angle)*ep_yaw_angle+(P_yaw_angle-
P_yaw_angle*N*T-I_yaw_angle*T+I_yaw_angle*N*T*T+D_yaw_angle*N)*epp_yaw_angle-
(N*T-2)*up_yaw_angle-(1-N*T)*upp_yaw_angle;

        //z_distance(PD)
        e_z_distance=ref_z_distance-z_distance;

u_z_distance=(P_z_distance+D_z_distance*N)*e_z_distance+(P_z_distance*N*T-
P_z_distance-D_z_distance*N)*ep_z_distance-(N*T-1)*up_z_distance;

        //z_velocity(PI)
        ref_z_velocity=u_z_distance;
        e_z_velocity=ref_z_velocity-z_velocity;
        u_z_velocity=P_z_velocity*e_z_velocity+(I_z_velocity*T-
P_z_velocity)*ep_z_velocity+up_z_velocity;

        r1=-u_pitch_angle+u_z_velocity+u_yaw_angle;
        r2=-u_roll_angle+u_z_velocity-u_yaw_angle;
        r3=u_pitch_angle+u_z_velocity+u_yaw_angle;
        r4=u_roll_angle+u_z_velocity-u_yaw_angle;

        ep_roll_angle=e_roll_angle;
        up_roll_angle=u_roll_angle;

        ep_pitch_angle=e_pitch_angle;
        up_pitch_angle=u_pitch_angle;

        epp_yaw_angle=ep_yaw_angle;
        ep_yaw_angle=e_yaw_angle;
        upp_yaw_angle=up_yaw_angle;
        up_yaw_angle=u_yaw_angle;

        ep_x_distance=e_x_distance;
        up_x_distance=u_x_distance;

        ep_y_distance=e_y_distance;
        up_y_distance=u_y_distance;

        ep_z_distance=e_z_distance;
        up_z_distance=u_z_distance;

        ep_z_velocity=e_z_velocity;
        up_z_velocity=u_z_velocity;

        r1=r1*10000;
        r2=r2*10000;
        r3=r3*10000;
        r4=r4*10000;

        vvv1=(int32)r1;
        vv1=vvv1;vv2=vvv1,vv1a=vvv1;vv2a=vvv1;
        vv2a=(vv2a>>24)&0x000000ff;
```

```
v2a=(uint8)vv2a;
vv1a=(vv1a>>16)&0x000000ff;
v1a=(uint8)vv1a;
vv2=(vv2>>8)&0x000000ff;
v2=(uint8)vv2;
vv1=vv1&0x000000ff;
v1=(uint8)vv1;

vvv1=(int32)r2;
vv1=vvv1;vv2=vvv1,vv1a=vvv1;vv2a=vvv1;
vv2a=(vv2a>>24)&0x000000ff;
v4a=(uint8)vv2a;
vv1a=(vv1a>>16)&0x000000ff;
v3a=(uint8)vv1a;
vv2=(vv2>>8)&0x000000ff;
v4=(uint8)vv2;
vv1=vv1&0x000000ff;
v3=(uint8)vv1;

vvv1=(int32)r3;
vv1=vvv1;vv2=vvv1,vv1a=vvv1;vv2a=vvv1;
vv2a=(vv2a>>24)&0x000000ff;
v6a=(uint8)vv2a;
vv1a=(vv1a>>16)&0x000000ff;
v5a=(uint8)vv1a;
vv2=(vv2>>8)&0x000000ff;
v6=(uint8)vv2;
vv1=vv1&0x000000ff;
v5=(uint8)vv1;

vvv1=(int32)r4;
vv1=vvv1;vv2=vvv1,vv1a=vvv1;vv2a=vvv1;
vv2a=(vv2a>>24)&0x000000ff;
v8a=(uint8)vv2a;
vv1a=(vv1a>>16)&0x000000ff;
v7a=(uint8)vv1a;
vv2=(vv2>>8)&0x000000ff;
v8=(uint8)vv2;
vv1=vv1&0x000000ff;
v7=(uint8)vv1;

UART_WriteTxData(v1);
UART_WriteTxData(v2);
UART_WriteTxData(v1a);
UART_WriteTxData(v2a);
num++;
}
if (UART_GetTxBufferSize()==0 && num==15)
{
UART_WriteTxData(v3);
UART_WriteTxData(v4);
UART_WriteTxData(v3a);
UART_WriteTxData(v4a);
num++;
}
if (UART_GetTxBufferSize()==0 && num==16)
{
```

```
        UART_WriteTxData(v5);
        UART_WriteTxData(v6);
        UART_WriteTxData(v5a);
        UART_WriteTxData(v6a);
        num++;
        }
        if (UART_GetTxBufferSize()==0 && num==17)
        {
        UART_WriteTxData(v7);
        UART_WriteTxData(v8);
        UART_WriteTxData(v7a);
        UART_WriteTxData(v8a);
        num=0;
        sample++;
        }

    }
}

/* [] END OF FILE */
```