DESIGN AND IMPLEMENTATION OF A MODEL PREDICTIVE TRAJECTORY TRACKING CONTROLLER FOR A QUADCOPTER

by

Kübra Hilal Salman

B.S., Mechanical Engineering, Boğaziçi University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Mechanical Engineering
Boğaziçi University

2022

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to my advisor Assist. Prof. Sinan Öncü for his valuable guidance, support, and advice throughout my research and education. I would also like to thank Prof. H. Işıl Bozma and Assoc. Prof. Volkan Sezer, for participating in my jury.

I would like to thank my laboratory colleagues and friends from Smart and Autonomous Mobility Laboratory, İlhan Umur Ayberk, Göksel Berker Gönül, Alperen Gezer, Atacan Aslan, and Berkin Aral, for their support, understanding, and friendship during my thesis period. My sincere thanks must also go to Erol Kayataş and Çağrı Çiftçi for helping me gain the necessary qualifications to progress in my thesis. I would also like to thank the entire Department of Mechanical Engineering staff in BOUN, especially Selamet Çevik from the Mechanical Engineering Laboratory, for his willing help in solving problems we encountered in the experimental setup.

I would like to thank The Scientific and Technology Research Council of Turkey (TUBITAK) for financial support throughout my graduate education under the 2210-a program. This project was co-funded by TUBITAK 1071-120N785 and the European Union's Horizon 2020 research and innovation programme under grant agreement No 862665 ERA-NET ICT-AGRI-FOOD.

Mom, Dad, Bilgenur, and Alperen Kemal, thank you very much for being there for me whenever I need you and making me feel that you will always be. Nagihan, I am grateful to you for always believing in me and supporting me in every aspect of my life. My dear friends, Esra, Aslı, and Serenay thank you for being with me in my journey to make sense of my life. Finally, I would like to thank my friend Tarık, who is no longer with us but will always be in our memories. We will never forget you as a good person.

ABSTRACT

DESIGN AND IMPLEMENTATION OF A MODEL PREDICTIVE TRAJECTORY TRACKING CONTROLLER FOR A QUADCOPTER

In this study, a trajectory tracking controller was designed for a quadcopter unmanned aerial vehicle (UAV), and its performance was evaluated by performing the necessary flight tests. As the controller approach, the model predictive controller (MPC), a newly popular method in aviation, has been chosen. Firstly, nonlinear dynamic equations of quadcopter were derived by Newton-Euler approximation. Then, a control algorithm was created, and a quadcopter system was produced, together with the sensors and flight computers necessary for the algorithm to work. While creating the control algorithm, it was decided to control the angle of the quadcopter UAV with the ArduPilot flight control algorithm, which is open-source software. ArduPilot software is an algorithm consisting of cascaded PID controllers and runs on the Pixhawk flight computer. The MPC algorithm was designed for trajectory tracking by considering the derived linear equations of motion. With the simulation created in the MATLAB environment, trials were made before the real flight tests and the design parameters of the MPC were decided. The tested controller software was written in Python and run on a Raspberry Pi computer. The communication between Raspberry Pi and Pixhawk flight computers was ensured, the system was made ready for flight, and trajectory tracking tests were carried out.

ÖZET

DÖRT PERVANELİ İNSANSIZ HAVA ARACI İÇİN YÖRÜNGE TAKİP EDEN MODEL ÖNGÖRÜLÜ KONTROLCÜ TASARIMI VE UYGULAMASI

Bu çalışmada, dört pervaneli bir insansız hava aracı (İHA) için yörünge takip kontrolcüsü tasarlanmış ve gerekli uçuş testleri yapılarak performansı değerlendirilmiştir. Kontrolcü yaklaşımı olarak, havacılıkta yeni popüler olan bir yöntem olan model önsezili kontrolcü (MPC) seçilmiştir. İlk olarak, Newton-Euler yaklaşımı ile dört pervanelinin doğrusal olmayan dinamik denklemleri türetilmiştir. Daha sonra bir kontrol algoritması oluşturulmuş ve algoritmanın çalışması için gerekli sensörler ve uçuş bilgisayarları ile birlikte bir dört pervaneli İHA sistemi üretilmiştir. Kontrol algoritması oluşturulurken, açık kaynaklı yazılım olan ArduPilot uçuş kontrol algoritması ile dört pervaneli İHA'nın duruş açılarının kontrol edilmesine karar verilmiştir. ArduPilot yazılımı, katmanlı PID kontrolcülerinden oluşan ve Pixhawk uçuş bilgisayarı üzerinde çalışan bir algoritmadır. MPC algoritması, türetilen doğrusal hareket denklemleri dikkate alınarak yörünge takibi için tasarlanmıştır. MATLAB ortamında oluşturulan simülasyon ile gerçek uçuş testleri öncesi denemeler yapılmış ve MPC'nin tasarım parametrelerine karar verilmiştir. Test edilen denetleyici yazılımı Python'da yazılmış ve bir Raspberry Pi bilgisayarında çalıştırılmıştır. Raspberry Pi ve Pixhawk uçuş bilgisayarları arasındaki iletişim sağlanmış, sistem uçuşa hazır hale getirilmiş ve yörünge takip testleri yapılmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

b	Aerodynamic thrust coefficient of propeller
d	Aerodynamic moment coefficient of propeller
F	Force vector
F_i	i_{th} Propeller's force around z axis of BFF
J	Inertia matrix of the quadcopter
m	Mass of the quadcopter
M	Moment vector
M_i	i_{th} Propeller's moment around z axis of BFF
p	Angular velocity around x axis of BFF
q	Angular velocity around y axis of BFF
r	Angular velocity around z axis of BFF
R_{EB}	Rotation matrix from BFF to EFF
R_{BE}	Rotation matrix from EFF to BFF
R_r	Transformation matrix
u	Translational velocity on x axis of BFF
U	Input vector of the plant model
v	Translational velocity on y axis of BFF
V_B	Translational velocity vector in BFF
V_E	Translational velocity vector in EFF
w	Translational velocity on z axis of BFF
x	Position on x axis of EFF
X	State vector of the plant model
y	Position on y axis of EFF
z	Position on z axis of EFF
$(.)_m$	Belong to the control-oriented model
$(.)_d$	Belong to the discretized control-oriented model
$(.)_a$	Belong to the augmented control-oriented model

η	Orientation vector about EFF
θ	Pitch angle
$\dot{\theta}$	Angular velocity about y axis of EFF
$\ddot{\theta}$	Angular acceleration about y axis of EFF
ξ	Position vector in EFF
ϕ	Roll angle
$\dot{\phi}$	Angular velocity about x axis of EFF
$\ddot{\phi}$	Angular acceleration about x axis of EFF
ψ	Yaw angle
$\dot{\psi}$	Angular velocity about z axis of EFF
$\ddot{\psi}$	Angular acceleration about z axis of EFF
ω	Angular velocity vector in BFF
Ω_i	Angular velocity of the i_{th} propeller

LIST OF ACRONYMS/ABBREVIATIONS

BFF	Body-Fixed Frame
EFF	Earth-Fixed Frame
EOM	Equation of Motion
ESC	Electronic Speed Controller
QP	Quadratic Programming
LQR	Linear Quadratic Regulator
MATLAB	Matrix Labrotary
MIMO	Multi-Input Multi-Output
MPC	Model Predictive Controller
NED	North East Down
PID	Proportional-Integral-Derivative
RC	Remote Control
RTK	Real-Time Kinematics
SISO	Single-Input Single-Output
UAV	Unmaned Aerial Vehicle
VTOL	Vertical Take Off and Landing

1. INTRODUCTION

Unmanned aerial vehicles (UAVs) are aerial vehicles that are remotely controlled. Autonomously flying them has been the subject of many studies to date. The modern UAVs are usually portable-sized small, and highly maneuverable electromechanical systems. Therefore, controlling them requires high piloting skills. Studies on controllers that will enable autonomous flight are critical as they eliminate piloting. In Figure 1.1, the quadcopter system that is developed for this thesis is shown.

Figure 1.1. Quadcopter System Developed for This Thesis.

1.1. Motivation and Background

UAVs are generally divided into two categories depending on their wing type. These are fixed-wing and rotary-wing types. Fixed-wing UAVs are more efficient in energy consumption, but they have limited mobility. Rotary-wing UAVs can have three, four, six, or eight propellers and are named tricopter, quadcopter, octocopter, and hexacopter, respectively. In this thesis, a rotary-wing type quadcopter UAV with high maneuverability is considered. Quadcopters are under-actuated systems because

they have four inputs but six degrees of freedom to control. The inputs of a quadcopter system are its actuators. A quadcopter has four electric motors counted as actuators and four connected propellers. Generally, a quadcopter's control architecture is treated under two subsystems, a high-level controller and a low-level controller. The high-level controller uses position errors to generate attitude reference set points for the low-level controller. The low-level controller acts on attitude errors to generate system inputs as motor RPMs. In Figure 1.2 the control architecture of a quadrotor is shown.

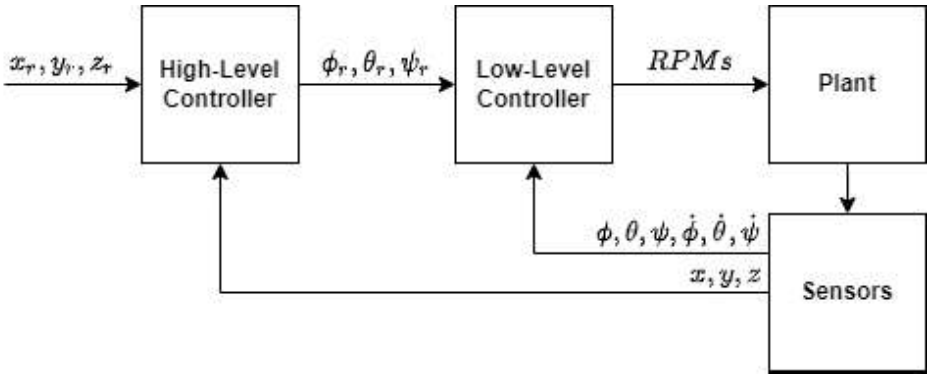

Figure 1.2. High-Level and Low-Level Control Architecture.

The main goal of the thesis is to design a high-level controller for the trajectory tracking stage. The controller is planned to work with the mathematically generated reference trajectories. For this purpose, model predictive controller (MPC) is chosen as the high-level controller while commercial flight controller ArduPilot is chosen as the low-level controller. An MPC is a controller that has objectives to compute future control sequences by considering the plant model. It solves an optimization problem to select the best control action before sending each input to the system. It is preferred due to its high preview capability in addition to its ability to handle multi-input-multi-output (MIMO) systems and systems' constraints.

1.2. Problem Statement and Objectives

This thesis aims to observe performance of the linear MPC as a high-level controller named trajectory tracking controller. The trajectory tracking controller uses

trajectory errors to produce attitude references. Dynamic relations behind trajectory tracking controller of quadcopter has highly nonlinear and coupled. Therefore, it aims to use MPC's advantage to solve trajectory tracking problems by considering the system constraints and future predictions in optimization. On the other hand, the ArduPilot's attitude and angular rate controllers were selected as low-level controllers. It is aimed to run MPC on the Raspberry Pi board, while the ArduPilot's controllers will run on the Pixhawk board. The objectives of the thesis are listed below.

- To design a linear model predictive controller in MATLAB as a trajectory tracking controller for a quadcopter.
- To simulate the nonlinear plant model and linear MPC together in MATLAB environment.
- To program the model predictive trajectory tracking controller in Python language.
- To build a working quadcopter frame with required hardware components.
- To implement MPC on Raspberry Pi board as a high-level controller, while ArduPilot on the Pixhawk board runs as the low-level controller.
- To do flight tests to assess the system's feasibility and the controller's performance.

1.3. Outline of the Thesis

Outline of the thesis is summarized below.

- Chapter 2 presents a literature survey of usage areas of UAVs, other works on the modeling and control of a quadcopter, and precisely MPC approach in any other systems.
- Chapter 3 presents modeling the dynamics of quadcopter including translational dynamics and rotational dynamics.
- Chapter 4 presents the algorithms of the Ardupilot used as the low-level controller and designed MPC used as the high-level controller.

- Chapter 5 presents MATLAB simulation results with the nonlinear model of the quadcopter and designed MPC.
- Chapter 6 presents the test results and discussion.
- Chapter 7 presents the hardware design including hardware components and performance calculations required component selection.
- Chapter 8 presents the conclusion and planned future works.

2. LITERATURE SURVEY

The areas where unmanned aerial vehicles (UAV) have been used overgrew with the increasing abilities of UAVs. Today, they are used in many applications such as photography, mapping, search and rescue operations, wildfire surveillance, military, agricultural and marine operations, etc. Moreover, developments in micro UAVs have accelerated with significant progress in sensing technology, high-density power storage, and data processing. A project initiated in the Autonomous Systems Laboratory in EPFL, named OS4, has done significant research on the fully autonomous control of the micro UAVs [1]. Due to the increasing demand for UAV applications, more effort has been focused on their control. The controller design researches have focused on linear control methods, including proportional-integral-derivative controller and linear quadratic requlators, and nonlinear control methods, including backstepping approach, feedback linearization, and model predictive control.

Proportional-integral-derivative (PID) controller is one of the widely employed controllers because it is straightforward and effective. It can be used only for single-input single-output (SISO) systems. A classical PID controller calculates the error between the desired set point and the measured variable, then uses this error to produce a corrective action depending on the proportional, integral, and derivative gains. With the developments in the aviation industry, the PID controllers were starting to be used in the position, attitude, and altitude control problems in the quadcopter's control applications [2–5]. Moreover, today's most popular open-source flight controllers for UAVs, ArduPilot, and PX4, are also based on variations of PID control strategies. In the quadcopter control terminology, the position and velocity control processes are named as the high-level task, while the attitude and angular rate control processes are named as low-level tasks. In commercial flight controllers, high-level and low-level controllers consist of cascaded PID algorithms because the cascaded PID algorithm allows the pilot to fly the UAV in the desired mode. The researches validated with the simulation and flight tests have shown that the cascaded PID controller is more stable

and has higher performance in attitude control tasks than the single loop PIDs [6–8]. Another way to implement variations of classical PID controller is to directly use the desired or measured process value with some gains instead of using the error between them to produce the controller's input [9]. The feedforward term is an actual example of this implementation. Because in the feedforward control, the desired set-point value is directly used to produce the controller's input, and it helps to reduce the error. There are many methods to tune PID gains, such as Ziegler-Nicholes and Tyreus-Luyben methods by considering the system response, but another popular way is to tune the gains during the flight. For online tuning, fuzzy logic is the most popular method. The simulation and actual flight tests results on the quadcopter system show that fuzzy PID performs well than classical PID [10, 11].

Linear quadratic regulator (LQR) is a popular linear control algorithm that provides a dynamic operating system at minimum cost. The cost of the system is defined by the linear-quadratic functional derived from the linear differential equations of the system. If the system is nonlinear, the nonlinear equations must be linearized around an operating point to design an LQR controller. It is used for simulation and actual flight tests for the quadcopter systems' position, attitude, and altitude control [12–14]. Gain scheduling is sometimes used in LQR applications. Because the controller gains are calculated before a flight, and their values depend on the linearized model around an operating point, modifying gains during operation might be helpful. When the gains are designed concerning the error between the desired set-point and measured variable, LQR gains change depending on the magnitude of the error [15]. Because PID and LQR are linear control strategies, many researchers compare their control performance on quadrotors. PID controller gives better stability than the LQR controller in the simulation environment prepared for a quadcopter system, but the system is stabilized either using LQR or PID [16]. In the trajectory tracking application [17, 18], the LQR controller gives more smooth and less disturbance control actions than the PID controllers. The performance of the PID and LQR controllers is validated with experimental setup and actual flight tests. Sometimes, one can implement the PID controller whose gain is obtained by an LQR loop instead of choosing one of them.

The method is used to control the quadcopter's altitude and compared with the classical PID controller. While the PID controller gives the fastest results, PID tuned by the LQR controller shows robust performance [19].

Feedback linearization is a standard method used to control nonlinear systems in which changing variable is performed to convert the nonlinear system into a linear one. The feedback linearization has been performed to control the highly nonlinear attitude dynamics of the quadcopter. The presented method's successful trajectory tracking performance is validated with simulation and experimental results [20]. Also, it can be used in a cascade control algorithm that uses model predictive controller in the high-level controller, and robust feedback linearization in the low-level. The results show the feasibility of the algorithm [21].

The backstepping control approach is a recursive controller design method that uses the Lyapunov stability theorems and feedback controller to guarantees the global asympotic stability of the system [22]. For an attitude tracking of a quadcopter, backstepping control approach gives better control performance comparing with the classical PID method in simulation environmenet [23]. For a trajectory tracking task of a quadcopter, an integral backstepping controller was tested in simulation environment and compared with simplified command filtered integral backstepping approach. The results show that proposed method provided robustness against the external disturbances while reducing the control effort [24].

Model predictive controller (MPC) is a controller that has objectives to compute future control sequence by considering the plant model and solving an optimization problem to select the best control action. MPC can be linear and nonlinear depending on the plant model used in optimization. The nonlinear MPC is not widely used as linear MPC because it requires considering nonlinear constraints and systems, making the controller appropriate for systems with slow dynamics. However, with the progress in the optimal control algorithm, nonlinear MPC is getting popular in fast dynamical systems [25]. For example, linear and nonlinear MPC controller in a trajectory tracking

task of a micro octocopter was compared, and results show that their performance is comparable, while nonlinear MPC showed better disturbance rejection capacity [26]. The trajectory tracking MPC controller of the quadcopter system can be designed with one loop or two loops, in other terms cascaded, including the low-level and the high-level controllers as mentioned in the PID examples. Generally, controller designers prefer to consider all the system dynamics together and prepare an MPC controller which produces the electric motor inputs directly for the trajectory tracking task [27, 28]. The simulation results show that MPC gives a smooth response due to its advantage in considering constraints and system dynamics. Compared with the cascaded PID controller, MPC shows better settling times and with no peak overshoot [29]. Also, as done in the PID applications, the cascaded MPC and centralized MPC can be compared. It is seen that cascaded MPC can apply more constraints on the whole system [30]. Finally, as with the usage of this thesis, there is another study which can combine designed MPC algorithm for the attitude control and open-source flight controller for the angular rate control. The PX4 algorithm is preferred as an open-source flight controller and the simulation and flight tests results show that MPC angular rate controller is able to success close reference tracking performance [31].

3. MODELING THE DYNAMICS OF THE QUADCOPTER

The dynamic model represents the behaviors of an object over time by describing the relation between the forces and moments acting on the quadcopter's body frame and the resulting translational and rotational motions. It is essential to derive a realistic dynamic model as it is used in the simulation of the system and also for the design of the model predictive controller (MPC). In this chapter, the dynamic model of the quadcopter is derived using the Newton-Euler formalism. First, reference frames and the transformation matrix between those frames are introduced. Then, control inputs of the system will be defined by considering the simplification of the dynamic equations. Next, translational and rotational dynamics will be investigated by considering the forces and moments acting on the quadcopter's body. Finally, a state-space representation of the system will be derived. Before starting derivation of dynamic equations with Newton-Euler formalism, some assumptions are considered to simplify modeling the dynamics. These assumptions are listed below.

- The mechanical structure of the quadcopter is rigid and symmetrical.
- The center of gravity of the quadcopter coincides with the origin of the body-fixed frame.
- The propellers are rigid.
- Connections between propellers, motors, frame, and other hardware devices are rigid.
- Thrust and moments generated by the propellers are proportional to the square of the propeller's speed.
- The gyroscopic effects due to the relative angular velocity of propellers to the quadcopter's body are neglected.
- The aerodynamic effects due to air friction except for the thrust generation are neglected.

3.1. Reference Frames and Transformation Matrices

There are two reference frames needed to define the motions of a quadcopter. The first is the earth-fixed reference frame (EFF) represented with the $[X_E, Y_E, Z_E]$ axis system. EFF is taken as the North-East-Down (NED) reference frame in this thesis, where the X_E represents the north, Y_E represents the east, and Z_E represents the down. The second is the body-fixed reference frame (BFF) represented with the $[X_B, Y_B, Z_B]$ axis system attached to the quadcopter's CoG. In Figure 3.1, these two reference coordinate frames are seen.

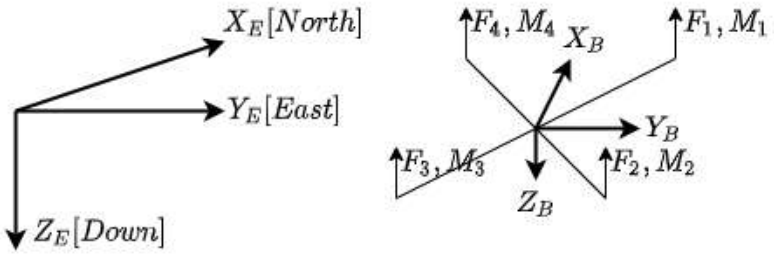

Figure 3.1. Reference Axis Systems.

The quadcopter moves in the EFF. The position and orientation of the quadcopter can be defined with the following notations as

$$\begin{aligned} \xi &= [x\ y\ z]^T \\ \eta &= [\phi\ \theta\ \psi]^T \end{aligned} \tag{3.1}$$

where $\xi \in \mathbb{R}^3$ represents the position and $\eta \in \mathbb{R}^3$ represents the orientation in the EFF. The quadcopter's orientation is defined using the Euler angles (ϕ,θ,ψ). Because the BFF is attached to the quadcopter's CoG, Euler angles are also used to define BFF's orientation with respect to the EFF.

By taking the derivative of the positions defined in (3.1), one can find the translational and rotational velocity in EFF. On the other hand, it is necessary to define

the translational and rotational velocities of the quadcopter in the BFF as

$$\begin{aligned} V_B &= [u\ v\ w]^T \\ \omega &= [p\ q\ r]^T \end{aligned} \tag{3.2}$$

where $V_B \in \mathbb{R}^3$ represents the translational velocity and $\omega \in \mathbb{R}^3$ represents the rotational velocity. The definition made in the BFF makes the results of the acting forces and moments more understandable.

A rotation matrix is required to express vectors in another axis system. In this chapter, two axes systems, EFF and BFF, are used to model the dynamics of the quadcopter. Although the dynamic equations will be set in the BFF, the quadcopter's motion in the EFF is considered for trajectory tracking. Therefore, a rotation matrix (R_{BE}) that can change the definition of vectors from EFF to BFF is needed. The rotation matrix should be formed in the order of ψ, θ, and ϕ angle rotations as

$$\begin{aligned} R_{BE} &= R_\phi R_\theta R_\psi \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & s(\phi) \\ 0 & -s(\phi) & c(\phi) \end{bmatrix} \begin{bmatrix} c(\theta) & 0 & -s(\theta) \\ 0 & 1 & 0 \\ s(\theta) & 0 & c(\theta) \end{bmatrix} \begin{bmatrix} c(\psi) & s(\psi) & 0 \\ -s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{3.3}$$

where the matrices R_ϕ, R_θ, and R_ψ provide rotations in x, y, and z axis, respectively. For notational simplicity, cosine and sine functions are denoted with $c(.)$ and $s(.)$ symbols. After multiplication of rotation matrices R_ϕ, R_θ and R_ψ, the rotation matrix R_{BE}, from EFF to BFF, is derived [1] as

$$R_{BE} = \begin{bmatrix} c(\theta)c(\psi) & c(\theta)s(\psi) & -s(\theta) \\ s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & s(\phi)c(\theta) \\ c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) & c(\phi)c(\theta) \end{bmatrix}. \tag{3.4}$$

In addition to that, the translational velocity of the quadcopter in the BFF (V_B) can be expressed by using the translational velocity in the EFF ($V_E \in \mathbb{R}^3$) and rotation matrix

(R_{BE}). Because the V_E is equal to the derivative of the position vector ($\dot{\xi} = [\dot{x}, \dot{y}, \dot{z}]^T$), the expression of V_B can be written as

$$V_B = R_{BE} V_E = R_{BE} \dot{\xi}. \tag{3.5}$$

In order to find V_E, the inverse of the rotation matrix (R_{EB}) should be taken. Because the rotation matrix is orthogonal, one can apply the transpose operation instead of taking its inverse as

$$R_{EB} = (R_{BE})^{-1} = (R_{BE})^T \tag{3.6}$$

where $(.)^{-1}$ represents the inverse operation and $(.)^T$ represents the transpose operation. That is one of the useful properties of orthogonal matrices. Therefore, R_{EB} can be written as

$$R_{EB} = \begin{bmatrix} c(\theta)c(\psi) & s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) & c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) \\ c(\theta)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix}. \tag{3.7}$$

After defining the rotation matrix R_{EB}, translational velocity in the EFF (V_E) can also be represented, in terms of V_B and R_{EB}, as

$$V_E = R_{EB} V_B = \dot{\xi}. \tag{3.8}$$

The angular velocity in the BFF ($\omega = [p\ q\ r]^T$) can be related with Euler rates vector ($[\dot{\phi}\ \dot{\theta}\ \dot{\psi}]^T$) as

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = R_\phi R_\theta R_\psi \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + R_\phi R_\theta \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_\phi \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} = R_r \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.9}$$

where R_r represents the transformation matrix [1]. Therefore, the angular velocity in

the BFF ($\omega = [p\ q\ r]^T$) can be simplified and rewritten as

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -s(\theta) \\ 0 & c(\phi) & s(\phi)c(\theta) \\ 0 & -s(\phi) & c(\phi)c(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.10}$$

in terms of the Euler rates and the transformation matrix. The close form of the definition is written as

$$\omega = R_r \dot{\eta}. \tag{3.11}$$

The R_r can also be simplified further by using a small angle approximation. The small-angle approximation assumes the angles are small and the cosine of the angle approximates to 1, while the sine of the angle approximates to angle itself or nearly 0. Therefore, the approximation simplifies R_r matrix into a I_{3x3} eye matrix, and the angular velocity vector in the BFF and EFF can be assumed as equal.

3.2. Definition of Control Inputs

The quadcopter has four electric motors that are controlled to realize desired moves. The system is under-actuated because it has six degrees of freedom to be controlled, while it has four inputs. There are two possible control configurations for quadcopters that differ regarding the orientation of propellers and rotors. They are known as plus-orientation and cross-orientation. In Figure 3.2, plus and cross orientations are presented.

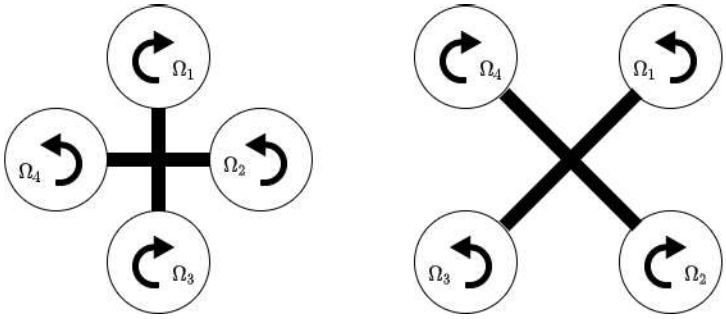

Figure 3.2. Plus (on the left) and Cross (on the right) Quadcopter Orientations.

Typically, electric motors of the quadcopter are selected as four inputs of the system. On the other hand, desired moves to be controlled consist of roll, pitch and yaw motions, and altitude change. Therefore, instead of choosing the control inputs as four propeller speeds, choosing them from the mathematical combination of the four propeller speeds is preferred to simplify the complexity of controller design. The four control inputs ($U_i \in \mathbb{R}^1$) for the quadcopter system are selected as

$$\begin{aligned} U_1 &= F_1 + F_2 + F_3 + F_4 \\ U_2 &= l(-F_1 - F_2 + F_3 + F_4) \\ U_3 &= l(F_1 - F_2 - F_3 + F_4) \\ U_4 &= -M_1 + M_2 - M_3 + M_4 \end{aligned} \tag{3.12}$$

where F_i represents the thrust, and M_i represents the moment generated by each propeller. Also, l represents the moment arm of each rotor that is assumed as equal for each rotor. Considering the control inputs' equations, they are expressed as the following definitions below.

- U_1: Throttle input represents the summation of forces generated by propellers.
- U_2: Roll input represents the moment generated by propellers around x-axis.
- U_3: Pitch input represents the moment generated by propellers around y-axis.
- U_4: Yaw input represents the moment generated by propellers around z-axis.

3.3. Nonlinear Dynamic Model

In order to derive the nonlinear dynamic model of the quadcopter, Newton-Euler formalism is used. Newton's second law of motion describes a simple relation between the acceleration, mass, and total force acting on the body ($F = ma$). Generally, the equations of motions are written in the EFF. In this case, Newton's law of motion will be implemented on the BFF, so the additional terms coming from the rotation of the BFF to the EFF have to be considered. The general form of the translational and rotational equations of motions (EOMs) are expressed as

$$\begin{aligned} \sum F &= m\dot{V_B} + \omega \times mV_B \\ \sum M &= J\dot{\omega} + \omega \times J\omega \end{aligned} \tag{3.13}$$

where F represents the total force acting on the body, M represents the total moment acting on the body, m represents the mass, and J represents the inertia of the quadcopter.

3.3.1. Forces and Moments Acting on the Quadcopter

The forces and moments acting on the quadcopter generate its motion. These forces and moments are represented in the BFF. Because the gyroscopic effects of propellers and air friction due to air velocity are neglected, the total forces acting on the quadcopter's body consist of the aerodynamic force generated by the propellers and the gravitational force. Also, the total moment acting on the quadcopter's body consists of aerodynamic moments generated by propellers. The propellers' aerodynamic forces and moments are proportional to the square of propellers' speed. By specifying each rotor with an index of "i", following force and moment equations can be written as

$$\begin{aligned} F_i &= b{\Omega_i}^2 \\ M_i &= d{\Omega_i}^2 \end{aligned} \tag{3.14}$$

where coefficients b and d represent the aerodynamic thrust and moment contributions. These factors depend on the geometry of the propeller, air density, rotational speed of the rotor, and experimentally found aerodynamic coefficients of propellers by considering the blade element theory. The blade element theory uses the lift and drag forces on the propeller's cross-sectional area, which has an airfoil shape. The theory explains the generation of thrust and moment through lift and drag. The force coming from propellers can be written, by substituting (3.14) in to (3.12), as

$$F_{prop,B} = \begin{bmatrix} 0 \\ 0 \\ -b({\Omega_1}^2 + {\Omega_2}^2 + {\Omega_3}^2 + {\Omega_4}^2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -F_1 - F_2 - F_3 - F_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -U_1 \end{bmatrix} \tag{3.15}$$

where U_1 represents the summation of forces generated by propellers. The moment acting on the quadcopter due to its propellers can be written, by substituting (3.14) into (3.12), as

$$M_{prop,B} = \begin{bmatrix} bl(-{\Omega_1}^2 - {\Omega_2}^2 + {\Omega_3}^2 + {\Omega_4}^2) \\ bl({\Omega_1}^2 - {\Omega_2}^2 - {\Omega_3}^2 + {\Omega_4}^2) \\ d(-{\Omega_1}^2 + {\Omega_2}^2 - {\Omega_3}^2 + {\Omega_4}^2) \end{bmatrix} = \begin{bmatrix} -F_1 l - F_2 l + F_3 l + F_4 l \\ F_1 l - F_2 l - F_3 l + F_4 l \\ -M_1 + M_2 - M_3 + M_4 \end{bmatrix} = \begin{bmatrix} U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{3.16}$$

where U_2, U_3, and U_4 represents the summation of moments generated by propellers on each axis.

Gravitational force can also be expressed in BFF frame ($F_{grav,B}$) by using the rotation matrix R_{BE} in (3.4). First of all, the gravitational force in EFF frame ($F_{grav.E}$) is defined as

$$F_{grav,E} = m \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{3.17}$$

where m represents the mass of the quadcopter and g represents the gravitational

acceleration. Then, the definition of gravitational force is transforming into BFF, by using the (R_{BE}), as

$$F_{grav,B} = R_{BE}F_{grav,E} = mR_{BE}\begin{bmatrix}0\\0\\g\end{bmatrix}. \tag{3.18}$$

Finally, total forces acting on the quadcopter's body can be obtained by using the forces defined in (3.15) and (3.18) as

$$\sum F = R_{BE}F_{grav,E} + F_{prop,B} = mR_{BE}\begin{bmatrix}0\\0\\g\end{bmatrix} + \begin{bmatrix}0\\0\\-U_1\end{bmatrix} \tag{3.19}$$

and the total moment acting on the quadcopter's body can be obtained by using the moment defined in (3.16) as

$$\sum M = M_{prop,B} = \begin{bmatrix}U_2\\U_3\\U_4\end{bmatrix}. \tag{3.20}$$

3.3.2. Translational Equations of Motion

In this section translational EOMs will be reconstructed by substituting the total force equation acting on the quadrotor's body which is defined in (3.19) into the general form of EOMs described in (3.13) as

$$\begin{aligned}\sum F &= m\dot{V}_B + \omega \times mV_B\\ R_{BE}F_{grav,E} + F_{prop,B} &= m\dot{V}_B + \omega \times mV_B.\end{aligned} \tag{3.21}$$

Because the general form of the EOMs is constructed on the BFF, the term V_B requires further simplification. Then, the translational position in EFF (ξ) derived in (3.5), and the rotation matrix R_{EB} is used to define V_E as

$$V_E = R_{EB}V_B = \dot{\xi}. \tag{3.22}$$

Then, the derivative of V_E is taken as

$$\begin{aligned} \dot{V}_E &= R_{EB}(\dot{V}_B + \omega \times V_B) \\ \ddot{\xi} &= R_{EB}(\dot{V}_B + \omega \times V_B) \end{aligned} \tag{3.23}$$

to express the $\dot{V}_B$ as

$$\dot{V}_B = R_{BE}\ddot{\xi} - \omega \times V_B. \tag{3.24}$$

If the expression of $\dot{V}_B$ is substituted in to the general form derived in (3.13), the translational EOMs is rewritten as

$$\begin{aligned} R_{BE}F_{grav,E} + F_{prop,B} &= m(R_{BE}\ddot{\xi} - \omega \times V_B) + \omega \times mV_B \\ R_{BE}F_{grav,E} + F_{prop,B} &= mR_{BE}\ddot{\xi}. \end{aligned} \tag{3.25}$$

Multiplying both sides with R_{EB} and dividing them to m, the simplified form of the translational EOMs is derived as

$$\ddot{\xi} = (1/m)(F_{grav,E} + R_{EB}F_{prop,B}). \tag{3.26}$$

Then, the final version of translational EOMs can be obtained by putting $[\ddot{x}, \ddot{y}, \ddot{z}]$ into $[\ddot{\xi}]$ as

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + R_{EB} \begin{bmatrix} 0 \\ 0 \\ -U1/m \end{bmatrix}. \tag{3.27}$$

The translational equation of motion in (3.27) are going to be used to design MPC in the following chapter.

3.3.3. Rotational Equation of Motion

After the derivation of translational EOMs, the rotational EOMs will be reconstructed by substituting the total moment equation acting on the quadcopter's body defined in (3.20) into the general form of the EOMs described in (3.13) as

$$\begin{aligned} \sum M &= J\dot{\omega} + \omega \times J\omega \\ M_{prop,B} &= J\dot{\omega} + \omega \times J\omega. \end{aligned} \tag{3.28}$$

If the derivative of angular velocity is taken, $\dot{\omega}$ is written as

$$\dot{\omega} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}. \tag{3.29}$$

Because the quadcopter's frame is assumed as symmetric, an inertia matrix is obtained diagonal whose off-diagonal elements are zero as

$$J = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{3.30}$$

where I_{xx}, I_{yy} and I_{zz} symbols represent moments of inertia around the principle axes. Next, the rotational EOMs can be rewritten by substituting (3.29) and (3.30) into (3.13) as

$$\begin{bmatrix} U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{3.31}$$

Then, the expression of $[\dot{p}\dot{q}\dot{r}]^T$ can be derived as

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} (I_{yy} - I_{zz})qr/I_{xx} \\ (I_{zz} - I_{xx})qr/I_{yy} \\ (I_{xx} - I_{yy})qr/I_{zz} \end{bmatrix} + \begin{bmatrix} U_2/I_{xx} \\ U_3/I_{yy} \\ U_4/I_{zz} \end{bmatrix}. \tag{3.32}$$

In order to represent rotational EOMs, $[p, q, r]$ should have to be replaced with $[\dot{\phi}, \dot{\theta}, \dot{\psi}]$ as shown in (3.11). By doing that final version of the rotational EOMs can be derived successfully as

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} (I_{yy} - I_{zz})\dot{\theta}\dot{\psi}/I_{xx} \\ (I_{zz} - I_{xx})\dot{\phi}\dot{\psi}r/I_{yy} \\ (I_{xx} - I_{yy})\dot{\phi}\dot{\theta}/I_{zz} \end{bmatrix} + \begin{bmatrix} U_2/I_{xx} \\ U_3/I_{yy} \\ U_4/I_{zz} \end{bmatrix}. \tag{3.33}$$

Although the rotational equations of motions in (3.33) are not going to be used to design MPC directly, their derivation is important to understand the low-level behavior of the quadcopter.

3.4. State Variables and Equations

The state-space form represents derived translational and rotational EOMs. The EOMs derived in this chapter will be used in the model predictive controller design process and as a plant in the simulation of the system. In the quadcopter system, there are twelve states and four inputs. States are decided as the quadcopter's positions and

their derivatives because of the translational EOMs, and the Euler angles and their derivatives because of the rotational EOMs. Thus, the states can be written as

$$X = \begin{bmatrix} x & \dot{x} & y & \dot{y} & z & \dot{z} & \phi & \dot{\phi} & \theta & \dot{\theta} & \psi & \dot{\psi} \end{bmatrix}^T. \tag{3.34}$$

Next, by using the control inputs of the system ($U \in \mathbb{R}^4$) as defined in (3.12), the input matrix can be written as

$$U = \begin{bmatrix} U_1 & U_2 & U_3 & U_4 \end{bmatrix}^T. \tag{3.35}$$

If the translational EOMs are separately written by using (3.7) and (3.27) as

$$\begin{aligned} \ddot{x} &= -(U_1/m)(cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi)) \\ \ddot{y} &= -(U_1/m)(cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi)) \\ \ddot{z} &= -(U_1/m)(cos(\phi)cos(\psi)) + g \end{aligned} \tag{3.36}$$

an explicit equations are created for the translational dynamics. If the rotational equations are separately written by using (3.33) as

$$\begin{aligned} \ddot{\phi} &= (I_{yy} - I_{zz})\dot{\theta}\dot{\psi}/I_{xx} + U_2/I_{xx} \\ \ddot{\theta} &= (I_{zz} - I_{xx})\dot{\phi}\dot{\psi}/I_{yy} + U_3/I_{yy} \\ \ddot{\psi} &= (I_{xx} - I_{yy})\dot{\phi}\dot{\theta}/I_{zz} + U_4/I_{zz} \end{aligned} \tag{3.37}$$

an explicit equations are created for the rotational dynamics.

4. CONTROLLER DESIGN

In this section, the control architecture of the quadrotor will be explained. The control architecture contains two control loops. They are low-level and high-level controllers. The low-level controller consists of the attitude and angular rate controllers from the ArduPilot open-source flight controller which are based on the variations of the PID controllers. The high-level controller contains linear MPC, designed specifically for the thesis application. In addition to the low-level and the high level controllers, there is an altitude controller in the control architecture of the quadcopter. The altitude control of the quadcopter is also a PID controller supplied by the Ardupilot. In Figure 4.1 the entire control algorithm structure including MPC and PID controllers are shown together.

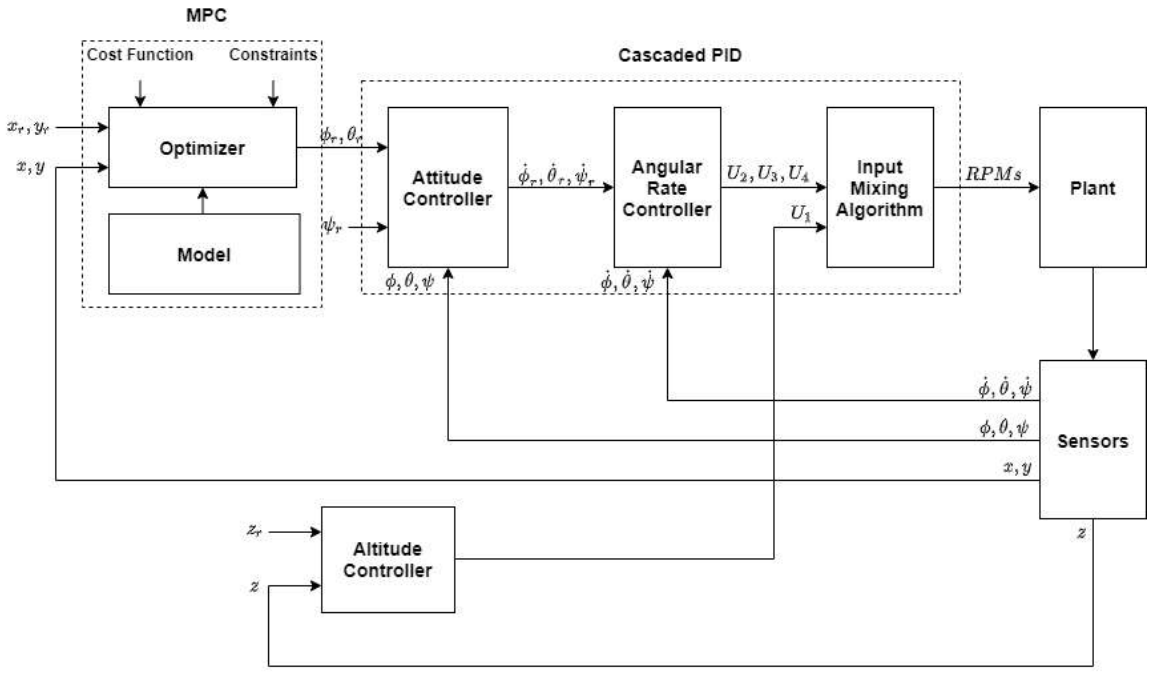

Figure 4.1. Control Architecture of Quadcopter.

In the scope of this thesis the low-level PID controllers and altitude PID controller will be run on the Pixhawk board with an open-source flight controller ArduPilot and

the high-level MPC will be run on the Raspberry Pi board. PID controllers and MPC will be further detailed in the following sections.

4.1. PID Controllers

PID controller is known also proportional-integral-derivative controller as it contains three gains. The control input of the system is produced by using the error and proportional, integrator, and derivative gains. PID controllers handle only single-input single-output (SISO) systems. In this thesis, PID controllers are used as the low-level controller of the quadcopter consisting of attitude and angular rate controllers, as seen in Figure 4.1. Because the objective of the thesis is to design an MPC controller for trajectory tracking processes, PID controllers from the ArduPilot open-source flight controller are utilized. The following section briefly explains the open-source flight controller's attitude and angular rate controllers.

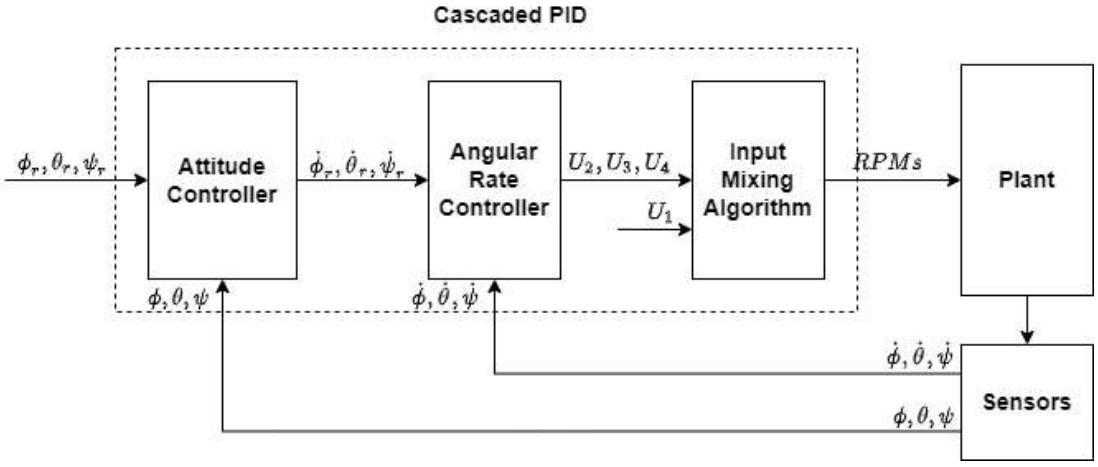

Figure 4.2. Cascaded PID Diagram.

4.1.1. Attitude Controller

The attitude controller uses the orientation errors in EFF to produce references for the following controller. The orientation errors consists of the difference between

the target angles $(\phi_r, \theta_r, \psi_r)$ and the actual angles (ϕ, θ, ψ). The actual angles are estimated by the filters in ArduPilot because they are not measured directly. On the other, hand the target angles are produced by the trajectory tracking MPC. The attitude controller has only proportional (P) gain, and an additional feed forward (FF) term as it is shown in Figure 4.3.

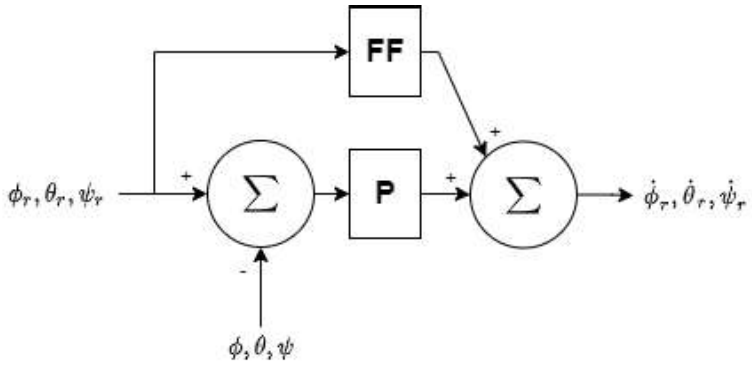

Figure 4.3. ArduPilot Attitude Controller Diagram.

The feed forward term reduces the errors faster or keeps the errors smaller than relying on the PID algorithm alone. The inputs of the attitude controller are consists of the angular rate references that are going to be used in the following angular rate controller.

4.1.2. Angular Rate Controller

Angular rate controller uses angular rate errors in EFF to produce the low-level control inputs. The angular rate errors consists of the difference between target angular rates $(\dot{\phi}_r, \dot{\theta}_r, \dot{\psi}_r)$ and actual rates $(\dot{\phi}, \dot{\theta}, \dot{\psi})$. The actual angular rates are measured by the gyroscope in the Pixhawk board, while the desired angular rates are produced by the attitude controller. The angular rate controller has proportional (P), integral (I) and derivative (D) gains. In Figure 4.4, angular rate controller diagram is represented.

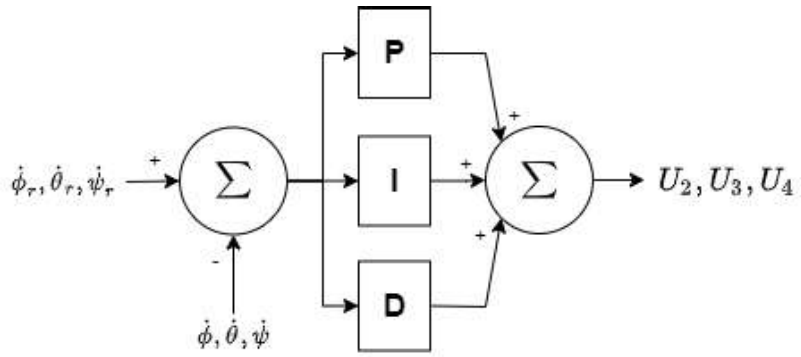

Figure 4.4. Ardupilot Angular Rate Controller Diagram.

The angular rate controller produces some of the system inputs defined in (3.12). Then the input mixing algorithm shown in Figure 4.1 convert them in to the angular speed of electric motors (RPMs) and send to the quadcopter system.

4.2. Linear Model Predictive Controller

The model predictive controller is a multivariable control algorithm that uses the internal dynamic model called the control-oriented model. MPC is based on the iterative optimization of the control-oriented model's output while also considering the constraints. The model predictive controller can be linear and nonlinear depending on the linearity of the control-oriented model. Both have some advantages and disadvantages regarding the controller's speed and performance. Due to nonlinear MPC considering the full system model requires more time to solve optimization problems but find proper control inputs. In this case, linear MPC is preferred because the quadrotor system requires to produce a fast dynamical response against the disturbances.

In the following figure, the diagram of MPC is presented. It is designed to reach reference x_r and y_r positions, and to produce ϕ_r and θ_r attitude references for the low-level controller by considering the cost function and constraints as shown in Figure 4.5.

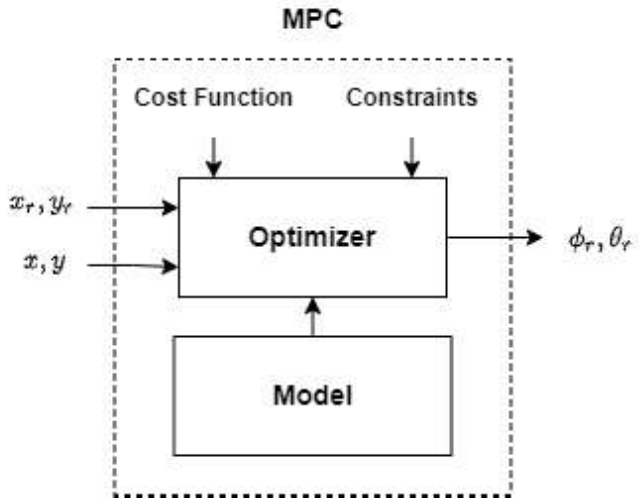

Figure 4.5. MPC Diagram.

The model predictive controller has some design parameters that affect the controller performance and also the optimization complexity. The design parameters consist of sample time (T_s), prediction horizon (N_p), control horizon (N_c), system's constraints ($u_{max}, u_{min}, \Delta u_{max}$) and the input weight matrix (W). At each iteration, the current plant state is sampled with the selected sample time, then until the prediction horizon, a cost function of the optimization problem is created. The cost function (J) is a combination of the output errors and the input. It can be rearranged by changing the input weight matrix. By doing that, one can determine the importance of tracking error and controller performance. In each optimization problem, the optimum inputs are produced as much as the number of the control horizon. However, only the first input is sent to the system.

MPC solves an online optimization problem at the current time by considering the actions over the prediction horizon. The prediction horizon keeps being shifted forward, and for this reason, MPC is also called receding horizon control. In Figure 4.6, calculated optimum control inputs and the outputs due to application of first control input at each instant time is shown. The dotted line presents the reference trajectory.

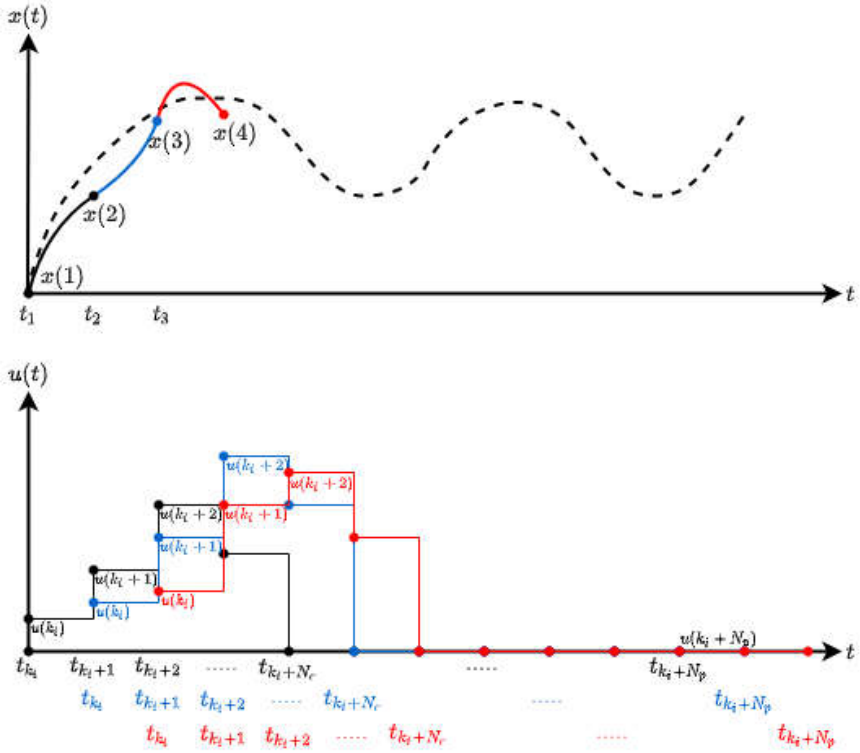

Figure 4.6. MPC Shifted Prediction Horizon Representation.

In this chapter, firstly, the control-oriented model is going to be derived and linearized by using the translational equations of motion derived in Chapter 3. They are going to be represented in the state-space form. Then, the linear control-oriented model is going to be discretized. After discretization, the state-space representation of the discrete control-oriented model will be represented in the augmented state-space form. The new state variable will be introduced for the augmented representation. Then, the visualization of the output predictions within one optimization window will be done at the current time (k_i). After all, the system's constraints will be defined. Finally, the quadratic programming (QP) solution for optimization process will be explained [32].

4.2.1. Control-Oriented Model and State-Space Representation

The control-oriented model is the model on which the model predictive controller is based. In this case, because the MPC is used to control position to complete the

trajectory tracking task, the control-oriented model consists of the translational EOMs. The translational EOMs were derived in (3.36) in Chapter 3 as

$$\begin{aligned} \ddot{x} &= -(U_1/m)(cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi)) \\ \ddot{y} &= -(U_1/m)(cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi)) \\ \ddot{z} &= -(U_1/m)(cos(\phi)cos(\psi)) + g. \end{aligned} \tag{4.1}$$

They have highly nonlinear and coupled dynamics. Therefore, the translational equations of motion have to be linearized and simplified in order to design a linear MPC with the assumptions listed below.

- Small-angle approximation can be used for the pitch and roll angles which is valid when $\phi, \theta \approx 0$. Therefore, trigonometric functions sine and cosine can be simplified as $\cos(\phi) \approx 1, \sin(\phi) \approx \phi$.
- The yaw angle is assumed to be zero ($\psi = 0$).
- The altitude change is assumed to be zero ($\dot{z} = 0$). Therefore, total thrust forces acting on the quadcopter are assumed to equal its weight ($U_1 = mg$).

After the implementation of the assumptions, and substituting mg into U_1, the translational EOMs can be rewritten as

$$\begin{aligned} \ddot{x} &= -g\theta \\ \ddot{y} &= g\phi. \end{aligned} \tag{4.2}$$

Then, a control-oriented model is created to design a linear MPC as

$$\begin{aligned} \dot{x}_m &= A_m x_m + B_m u_m \\ y_m &= C_m x_m \end{aligned} \tag{4.3}$$

by choosing the states as $x_m = [x\ \dot{x}\ y\ \dot{y}]^T$, inputs as $u_m = [\phi\ \theta]^T$ and outputs as $y_m = [x\ y]^T$. Also the state matrices are defined as

$$A_m = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad B_m = \begin{bmatrix} 0 & 0 \\ 0 & -g \\ 0 & 0 \\ g & 0 \end{bmatrix}, \quad C_m = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{4.4}$$

4.2.2. Discretization

Discretization is transferring continuous functions, models, or equations into discrete form. Because while the continuous problems have infinite degrees of freedom and discrete problems have finite, discretization is necessary due to the finite nature of the subsequent calculation process. In this section, the state-space representation derived from the control-oriented model will be discretized using Euler's Method as

$$\begin{aligned} \dot{x}_m(k) &= A_m x_m(k) + B_m u_m(k) \\ \dot{x}_m(k) &= \frac{x_m(k+1) - x_m(k)}{T_s} \end{aligned} \tag{4.5}$$

where T_s represents the sampling time, and k represents the instant time iteration. If the first equation in (4.5) is substituted into the second as

$$\begin{aligned} \frac{x_m(k+1) - x_m(k)}{T_s} &= A_m x_m(k) + B_m u_m(k) \\ x_m(k+1) - x_m(k) &= T_s(A_m x_m(k) + B_m u_m(k)) \end{aligned} \tag{4.6}$$

the discrete equations can be derived. The equation of $x_m(k+1)$ is written as

$$x_m(k+1) = (T_s A_m + I)x_m(k) + T_s B_m u_m(k) \tag{4.7}$$

where I represents the identity matrix. In order to simplify discrete representation, new state matrices are defined. For the output equation, there is no need for a derivative discretization. Therefore, the discrete state-space model is derived as

$$\begin{aligned} x_m(k+1) &= A_d x_m(k) + B_d u_m(k) \\ y_m(k) &= C_d x_m(k) \end{aligned} \tag{4.8}$$

where the new state matrices are defined as $A_d = T_s A_m + I$, $B_d = T_s B_m$ and $C_d = C_m$.

4.2.3. Augmented State-Space Matrices

The linear and discrete state-space model is defined in the previous section. In this section, augmented state-space model is derived to solve optimization problem with matrix calculations including the constraints. For the augmented representation, a new state variable vector which is related to $\Delta x_m(k)$ and $y_m(k)$ is chosen as

$$x_a(k) = [\Delta x_m(k)^T y_m(k)^T]^T. \tag{4.9}$$

By considering the definition of $x_m(k+1)$ in (4.8), the equation of $x_m(k)$ can be written as

$$\begin{aligned} x_m(k+1) &= A_d x_m(k) + B_d u_m(k) \\ x_m(k) &= A_d x_m(k-1) + B_d u_m(k-1). \end{aligned} \tag{4.10}$$

By defining $\Delta x_m(k) = x_m(k) - x_m(k-1)$ and $\Delta u_m(k) = u_m(k) - u_m(k-1)$, equation of $\Delta x_m(k+1)$ can be written as

$$\Delta x_m(k+1) = A_d \Delta x_m(k) + B_d \Delta u_m(k). \tag{4.11}$$

In order to relate the output $y_m(k)$ to the state variable $\Delta x_m(k)$, the equation of $\Delta y_m(k+1)$ is defined as

$$\begin{aligned} \Delta y_m(k+1) &= y_m(k+1) - y_m(k) \\ \Delta y_m(k+1) &= C_d x_m(k+1) - C_d x_m(k) \\ \Delta y_m(k+1) &= C_d \Delta x_m(k+1) \end{aligned} \tag{4.12}$$

by using the $y_m(k)$ definition in (4.8). Then, equation of $\Delta x_m(k+1)$ is substituted into the expression of $\Delta y_m(k+1)$ as

$$\begin{aligned} \Delta y_m(k+1) &= C_d \Delta x_m(k+1) \\ &= C_d A_d \Delta x_m(k) + C_d B_d \Delta u_m(k). \end{aligned} \tag{4.13}$$

By defining $\Delta y_m(k+1) = y_m(k+1) - y_m(k)$, definition of $y(k+1)$ can be written as

$$y_m(k+1) = C_d A_d \Delta x_m(k) + C_d B_d \Delta u_m(k) + y_m(k) \tag{4.14}$$

in terms of $\Delta x_m(k)$ and $y_m(k)$. Finally using the new state variable vector $x_a(k) = [\Delta x_m(k)^T \; y_m(k)^T]^T$, the augmented stat-space representation for a MIMO system is written as

$$\begin{aligned} x_a(k+1) &= \begin{bmatrix} \Delta x_m(k+1) \\ y_m(k+1) \end{bmatrix} = \begin{bmatrix} A_d & {o_{N_s \times N_o}}^T \\ C_d A_d & I_{N_o \times N_o} \end{bmatrix} \begin{bmatrix} \Delta x_m(k) \\ y_m(k) \end{bmatrix} + \begin{bmatrix} B_d \\ C_d B_d \end{bmatrix} \Delta u_m(k) \\ y_a(k) &= \begin{bmatrix} y_m(k) \end{bmatrix} = \begin{bmatrix} o_{N_s \times N_o} & I_{N_o \times N_o} \end{bmatrix} \begin{bmatrix} \Delta x_m(k) \\ y_m(k) \end{bmatrix}. \end{aligned} \tag{4.15}$$

Also the input vector can be written as $\Delta u_a(k)$ instead of $\Delta u_m(k)$, due to make last state-space representation clear. Finally, by defining the new state matrices A, B and

C, the close form of the augmented state-space representation is written as

$$\begin{aligned} x_a(k+1) &= Ax_a(k) + B\Delta u_a(k) \\ y_a(k) &= Cx_a(k) \end{aligned} \tag{4.16}$$

where $I_{N_o \times N_o}$ represents the identity matrix with the dimension of number of outputs times number of outputs ($N_o \times N_o$), and $o_{N_s \times N_o}$ represents a zero matrix with the dimension of number of states times number of outputs ($N_s \times N_o$).

4.2.4. Output Predictions within One Optimization Windows

In this section, an optimization problem over a prediction horizon will be solved, step by step. In order to find a solution for model predictive control of MIMO systems, $x_a(k_i)$ which represents predicted state variables, and $\Delta u_a(k_i)$ which represents future control movements, are defined as follows. Assuming that at the sampling instant k_i the state variable vector $x_a(k_i)$ is available through measurement. The future control movements are defined until the number of control horizons (N_c), while the predicted states are defined until the number of prediction horizons (N_p) as

$$\begin{gathered} \Delta u_a(k_i),\ \Delta u_a(k_i+1),\ ...,\ \Delta u_a(k_i+N_c-1) \\ x_a(k_i+1 \mid k_i),\ x_a(k_i+2 \mid k_i),\ x_a(k_i+3 \mid k_i),\ ...,\ x_a(k_i+N_p \mid k_i). \end{gathered} \tag{4.17}$$

Based on the state-space model derived in (4.16), the future state variables are calculated sequentially using the set of future control inputs in (4.17). Because the number of control horizon (N_c) can be equal or less than number of prediction horizon (N_p), the missing terms of the Δu_a are taken as zero. The predicted state variables in one

optimization window are defined as

$$\begin{aligned}
x_a(k_i+1 \mid k_i) =& Ax_a(k_i) + B\Delta u_a(k_i) \\
x_a(k_i+2 \mid k_i) =& Ax_a(k_i+1 \mid k_i) + B\Delta u_a(k_i+1) \\
=& A^2x_a(k_i) + AB\Delta u_a(k_i) + B\Delta u_a(k_i+1) \\
x_a(k_i+3 \mid k_i) =& Ax_a(k_i+2 \mid k_i) + B\Delta u_a(k_i+2) \\
=& A^3x_a(k_i) + A^2B\Delta u_a(k_i) + AB\Delta u_a(k_i+1) + B\Delta u_a(k_i+2) \\
& \vdots \\
x_a(k_i+N_p \mid k_i) =& A^{N_p}x_a(k_i) + A^{N_p-1}B\Delta u_a(k_i) + A^{N_p-2}B\Delta u_a(k_i+1) + \ ... \ + \\
& A^{N_p-N_c}B\Delta u_a(k_i+N_c-1).
\end{aligned} \tag{4.18}$$

By using the predicted state variables, predicted output variables are derived as

$$\begin{aligned}
y_a(k_i+1 \mid k_i) =& CAx_a(k_i) + CB\Delta u_a(k_i) \\
y_a(k_i+2 \mid k_i) =& CA^2x_a(k_i) + CAB\Delta u_a(k_i) + CB\Delta u_a(k_i+1) \\
y_a(k_i+3 \mid k_i) =& CA^3x_a(k_i) + CA^2B\Delta u_a(k_i) + CAB\Delta u_a(k_i+1) + CB\Delta u_a(k_i+2) \\
& \vdots \\
y_a(k_i+N_p \mid k_i) =& CA^{N_p}x_a(k_i) + CA^{N_p-1}B\Delta u_a(k_i) + CA^{N_p-2}B\Delta u_a(k_i+1) + \ ... \ + \\
& CA^{N_p-N_c}B\Delta u_a(k_i+N_c-1).
\end{aligned} \tag{4.19}$$

After defining the predicted output variables for a given prediction horizon, the vectors Y_a and ΔU_a are defined as

$$\begin{aligned}
Y_a =& \Big[y_a(k_i+1 \mid k_i)^T \ y_a(k_i+2 \mid k_i)^T \ y_a(k_i+3 \mid k_i)^T \ ... \ y_a(k_i+N_p \mid k_i)^T\Big]^T \\
\Delta U_a =& \Big[\Delta u_a(k_i)^T \ \Delta u_a(k_i+1)^T \ \Delta u_a(k_i+2)^T \ .... \ \Delta u_a(k_i+N_c-1)^T\Big]^T
\end{aligned} \tag{4.20}$$

in order to represent one optimization window in the model predictive controller. Where the dimension of Y is represented by N_p times the number of outputs (N_o) and the dimension of ΔU is represented by N_c times the number of outputs (N_o).

After all, the compact matrix form is written as

$$Y_a = Px_a(k_i) + H\Delta U_a \tag{4.21}$$

where,

$$P = \begin{bmatrix} CA \\ CA^2 \\ CA^3 \\ \vdots \\ CA^{N_p} \end{bmatrix}, \quad H = \begin{bmatrix} CB & 0 & 0 & ... & 0 \\ CAB & CB & 0 & ... & 0 \\ CA^2B & CAB & CB & ... & 0 \\ \vdots & & & & \\ CA^{N_p-1}B & CA^{N_p-2}B & CA^{N_p-3}B & ... & CA^{N_p-N_C}B \end{bmatrix}. \tag{4.22}$$

The compact matrix form derived in (4.21) will be used to set cost function (J) of the optimization.

4.2.5. Constraints

Before setting up the cost function used in the optimization problem, the final step is to define the system constraints. All systems have operational constraints due to their limited physical capacities. Constraints are separated as input constraints, output constraints, or rate constraints. In the quadcopter systems, the thrust generated by the propulsion system is one of the essential limited physical properties counted as input constraints. This limitation also causes other restrictions in maneuverability. In this case, constraints will be defined over these angles because the MPC is used as a trajectory tracking controller, which produces pitch and roll angles as inputs. The magnitude of the constraints were decided during simulation tests. Then, they were tested in the real flight tests. The constraints on the control variable of the trajectory tracking controller are expressed as

$$\begin{aligned} u_{a,min} \leqslant u_a(k) \leqslant u_{a,max} \\ \Delta u_{a,min} \leqslant \Delta u_a(k) \leqslant \Delta u_{a,min} \end{aligned} \tag{4.23}$$

where $u_{a,min}$ and $u_{a,max}$ represents the input constraints, while $\Delta u_{a,min}$ and $\Delta u_{a,max}$ represents the input rate constraints. Because there are two control variables in the MPC, each of them is subjected to distinct constraints as

$$\begin{aligned} u_{a1,min} \leqslant u_{a1}(k) \leqslant u_{a1,max} \\ u_{a2,min} \leqslant u_{a2}(k) \leqslant u_{a2,max} \end{aligned} \tag{4.24}$$

where $u_{a1,min}$ and $u_{a1,max}$ represents the constraints on ϕ angle, while $u_{a2,min}$ and $u_{a2,max}$ represents the constraints on θ angle. Then, the constraints on the inputs can be grouped as

$$U_{a,min} = \begin{bmatrix} u_{a1,min} \\ u_{a2,min} \end{bmatrix}, \; U_a(k) = \begin{bmatrix} u_{a1}(k) \\ u_{a2}(k) \end{bmatrix}, \; U_{a,max} = \begin{bmatrix} u_{a1,max} \\ u_{a2,max} \end{bmatrix} \tag{4.25}$$

and represented with the inequality as

$$U_{a,min} \leqslant U_a(k) \leqslant U_{a,max}. \tag{4.26}$$

The same procedure is also applicable for the rates of input change. The rate of change constraints are expressed as

$$\begin{aligned} \Delta u_{a1,min} \leqslant \Delta u_{1a}(k) \leqslant \Delta u_{a1,max} \\ \Delta u_{a2,min} \leqslant \Delta u_{2a}(k) \leqslant \Delta u_{a2,max} \end{aligned} \tag{4.27}$$

where where $\Delta u_{a1,min}$ and $\Delta u_{a1,max}$ represents the constraints on ϕ angle, while $\Delta u_{a2,min}$ and $\Delta u_{a2,max}$ represents the constraints on θ angle. Then, the constraints on the input rate of change can be grouped as

$$\Delta U_{a,min} = \begin{bmatrix} \Delta u_{a1,min} \\ \Delta u_{a2,min} \end{bmatrix}, \; \Delta U_a(k) = \begin{bmatrix} \Delta u_{a1}(k) \\ \Delta u_{a2}(k) \end{bmatrix}, \; \Delta U_{a,max} = \begin{bmatrix} \Delta u_{a1,max} \\ \Delta u_{a2,max} \end{bmatrix} \tag{4.28}$$

and represented with the inequality as

$$\Delta U_{a,min} \leqslant \Delta U_a(k) \leqslant \Delta U_{a,max}. \tag{4.29}$$

These constraints can be represented into separate inequalities, and as it is shown in (4.20), they can be defined through the one optimization window as

$$\Delta U_a(k) \leqslant \Delta U_{a,max}, \begin{bmatrix} \Delta U_a(k) \leqslant \Delta U_{a,max} \\ \Delta U_a(k+1) \leqslant \Delta U_{a,max} \\ \vdots \\ \Delta U_a(k+N_c-1) \leqslant \Delta U_{a,max} \end{bmatrix} \tag{4.30}$$

and

$$-\Delta U_a(k) \leqslant -\Delta U_{a,min}, \begin{bmatrix} -\Delta U_a(k) \leqslant -\Delta U_{a,min} \\ -\Delta U_a(k+1) \leqslant -\Delta U_{a,min} \\ \vdots \\ -\Delta U_a(k+N_c-1) \leqslant -\Delta U_{a,min} \end{bmatrix}. \tag{4.31}$$

Suppose the inequalities are written into a vector representation. In that case, the following equation is derived as

$$\begin{bmatrix} -\Delta U_a \leqslant -\Delta U_{a,min} \\ \Delta U_a \leqslant \Delta U_{a,max} \end{bmatrix} \tag{4.32}$$

where the ΔU_a, $\Delta U_{a,min}$, and $\Delta U_{a,max}$ are vectors with their length equal to the size of the control horizon (N_c) multiplied by the size of the number of inputs (N_i). In order to construct a relation between input constraints and the rate of input change

constraints, the equation of $u_a(k)$ can be written as

$$\begin{aligned} u_a(k) &= u_a(k-1) + \Delta u_a(k) \\ u_a(k+1) &= u_a(k) + \Delta u_a(k+1) \\ u_a(k+1) &= u_a(k-1) + \Delta u_a(k) + \Delta u_a(k+1) \\ &\vdots \\ u_a(k+N_c-1) &= u_a(k-1) + \Delta u_a(k) + \Delta u_a(k+1) + ... + \Delta u_a(k+N_c-1) \end{aligned} \tag{4.33}$$

by using the definition of $\Delta u_a(k)$. Then, $u_a(k)$ is substituted in to the equation of $\Delta u_a(k+1)$. This relation can be represented in the matrix form as

$$\begin{bmatrix} u_a(k) \\ u_a(k+1) \\ \vdots \\ u_a(k+N_c-1) \end{bmatrix} = \begin{bmatrix} I \\ I \\ \vdots \\ I \end{bmatrix} u_a(k-1) + \begin{bmatrix} I & 0 & 0 & ... & 0 \\ I & I & 0 & ... & 0 \\ \vdots & & & & \\ I & I & I & ... & I \end{bmatrix} \begin{bmatrix} \Delta u_a(k) \\ \Delta u_a(k+1) \\ \vdots \\ \Delta u_a(k+N_c-1) \end{bmatrix}. \tag{4.34}$$

Because the left side of the equation in (4.34) is the expression of U_a from k_{th} variable to $(k+Nu-1)_{th}$ variable, same equation can be expressed compactly as

$$\begin{aligned} -(C_1 u_a(k-1) + C_2 \Delta U_a) &\leqslant -U_{a,min} \\ (C_1 u_a(k-1) + C_2 \Delta U_a) &\leqslant U_{a,max} \end{aligned} \tag{4.35}$$

where

$$C_1 = \begin{bmatrix} I \\ I \\ \vdots \\ I \end{bmatrix}, \; C_2 = \begin{bmatrix} I & 0 & 0 & ... & 0 \\ I & I & 0 & ... & 0 \\ \vdots & & & & \\ I & I & I & ... & I \end{bmatrix}. \tag{4.36}$$

Then the control input and rates of input change can be grouped as

$$\begin{bmatrix} C_{u_a} \\ C_{\Delta u_a} \end{bmatrix} \Delta U_a \leqslant \begin{bmatrix} d_{u_a} \\ d_{\Delta u_a} \end{bmatrix} \tag{4.37}$$

where

$$C_{u_a} = \begin{bmatrix} -C_2 \\ C_2 \end{bmatrix}, d_{u_a} = \begin{bmatrix} -U_{a,min} + C_1 u_a(k-1) \\ U_{a,max} - C_1 u_a(k-1) \end{bmatrix}, C_{\Delta u_a} = \begin{bmatrix} -I \\ I \end{bmatrix}, d_{\Delta u_a} = \begin{bmatrix} -\Delta U_{a,min} \\ \Delta U_{a,max} \end{bmatrix}. \tag{4.38}$$

Then the equation in (4.37) can be rewritten as

$$C\Delta U_a \leqslant d \tag{4.39}$$

where

$$CC = \begin{bmatrix} C_{\Delta u_a} \\ C_{u_a} \end{bmatrix}, d = \begin{bmatrix} d_{\Delta u_a} \\ d_{u_a} \end{bmatrix}. \tag{4.40}$$

The CC and d parameters will be used in the quadratic programming solution.

4.2.6. Optimization

Assuming that $r(k)$ is the set-point signal at the sample time k, in the given prediction horizon, the objective of the predictive control system is to find predicted output as close as possible to the given set-point signal. It might be assumed that the set-points are constant during the prediction horizon. However, this assumption causes a waypoint tracking MPC. In order to design a trajectory tracking MPC the set points are taken same as the reference trajectory. Because there are two number of outputs in the control-oriented model, there should be two references for each output. Therefore,

$r(k)$ can be defined as

$$r(k) = [x_r(k)\ y_r(k)]^T. \tag{4.41}$$

To design a model predictive trajectory tracking controller, the references over one optimization window should be shifted in each time. Therefore, in each optimization window, the vector includes references should be updated. The reference vector (R_s) is defined as

$$R_s = \begin{bmatrix} r(k_i+1|k_i) \\ r(k_i+2|k_i) \\ \vdots \\ r(k_i+N_p|k_i) \end{bmatrix} \tag{4.42}$$

by considering the updated references in each optimization window. After the expression of R_s is defined, the cost function that reflects the control objective is written as

$$J = (R_s - Y_a)^T(R_s - Y_a) + \Delta U_a{}^T W \Delta U_a. \tag{4.43}$$

The W represents a diagonal matrix that includes tuning parameters in diagonal terms for the desired closed-loop performance and it is named as weight matrix. If the expression of Y_a is substituted into the equation, the cost function can be rewritten as

$$J = (R_s - Px_a(k))^T(R_s - Px_a(k)) - 2\Delta U^T H^T(R_s - Px_a(k)) + \Delta U_a{}^T(H^T H + W)\Delta U_a. \tag{4.44}$$

By taking the derivative of J as

$$\frac{\partial J}{\partial \Delta U_a} = -2H^T(R_s - Px_a(k)) + 2(H^T H + W)\Delta U_a = 0 \tag{4.45}$$

the control inputs that minimize J are obtained. Finally, the optimal solution for the control inputs is found as

$$\Delta U_a = (H^T H + W)^{-1} H^T (R_s - P x_a(k)). \tag{4.46}$$

In order to simplify the solution, one can define the equations of $E = 2(H^T H + W)$ and $F = -2H^T(R_s - P x_a(k))$. Inside the QP solution, calculations are done through the variables E and F, and the optimum solution is found as $\Delta U_a = E^{-1}F$.

5. SIMULATION RESULTS

In order to foresee the responses of the model predictive trajectory tracking controller, a MATLAB simulation was prepared. In the simulation, the translational equations of motion in (3.36) were used to represent the nonlinear plant model. Although the MPC is linear, using the nonlinear equations of motion as a plant model helps to make simulation more realistic. Then, MPC algorithm was written in MATLAB script as it is designed in Chapter 4. The controller's design parameters were selected depending on the simulation tests or physical limitations.

The sample time (T_s) is the time interval depending on which the control-oriented model of the MPC is simulated. If the sample time is selected very small, the optimization complexity will increase, and the solution will take more time. If it is selected very large, the controller might not respond to instant disturbances in real-life applications. During the flight tests, it was determined that the GPS data came in 0.05 second intervals. Therefore, T_s is assumed as 0.05. Moreover, due to the small-angle approximation is often made when creating the control-oriented model, the minimum and maximum input constraints are selected as $\pm 10^\circ$ to provide a safe flight and to restrict the altitude loss. The input rate constraint is selected as physically proper value 100, but there is no information about the quadcopter's capacity to change thrust in a distinct time interval.

The prediction horizon (N_p) is the number of iterations MPC uses in one optimization window. If the prediction horizon is small, then the controller might not produce an input against the disturbances, which are required more time than the prediction horizon considers. A large prediction horizon also causes an increase in optimization complexity. The control horizon (N_c) is the number of optimal solutions that the MPC can solve in one optimization window. Like the prediction horizon, large control horizon also increases the complexity of the optimization problem. The input weight matrix (W) is another design parameter of MPC. It is multiplied by the input

terms and added to the error terms to produce the cost function. If the multiplication parameter of the input is increased, then the optimization results give smaller inputs. Since the quadcopter is assumed as symmetric in this application, the roll and pitch inputs should have the same weight in the cost function. Regarding the simulation results, N_p is selected as 40, N_c is selected as 4, and W is selected as 1000 for the best controller performance. However, other tests with different design parameters are also tried in actual flight tests and will be represented in Chapter 6. In Table 5.1, selected design parameters for MPC are listed.

Table 5.1. Design Parameters for MPC.

Parameter	**Value**
Sample Time	0.05
Prediction Horizon	40
Control Horizon	4
Input Constraints	+-10°
Input Rate Constraints	100
Input Weights	1000

The low-level controller will run on the quadcopter in real-life applications, but it is assumed that the low-level controller works without fault in the MATLAB simulation. It is assumed that it can follow the roll and pitch references without delay and overshoot. In the following sections, four different reference trajectories and MPC responses will be represented. The trajectory tracking performance will be investigated by changing the prediction horizon (N_p), and the control horizon (N_c) of the MPC, firstly. Then the input weight (W) will be increased while keeping the N_p and N_c constant, and results will be compared.

5.1. Reference Waypoint

First of all, simulation tests were carried out using the waypoint reference. The waypoint reference was generated as

$$r(t) = \begin{bmatrix} x_r(t) \\ y_r(t) \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \end{bmatrix}. \tag{5.1}$$

In Figure 5.1, the start point and the waypoint reference are shown on the EFF.

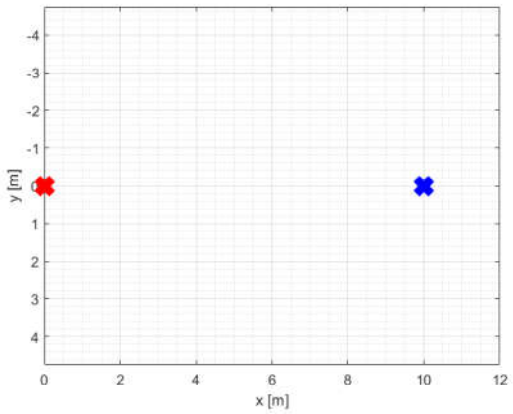

Figure 5.1. Waypoint Reference.

The output results in Figure 5.2 show that higher N_p values reduce the overshoot around the reference. The input results in Figure 5.3 show that increasing N_p and N_c result in smooth input generations. Moreover, additional Figures 5.4, and 5.5 show that increasing W decreases the control effort and reduces the overshoot around the reference.

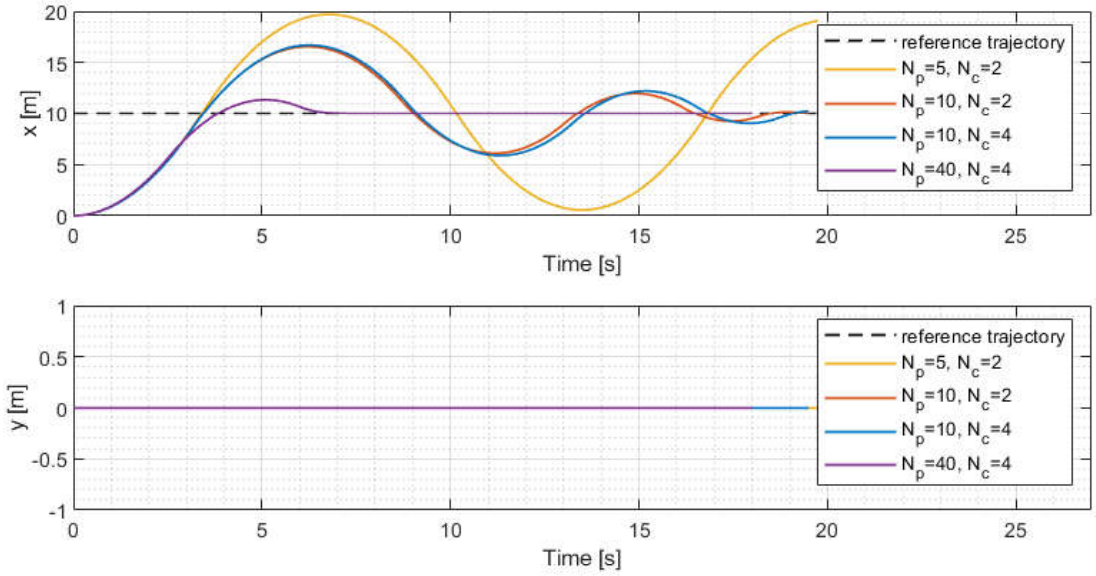

Figure 5.2. Simulation Output Results for the Waypoint Reference ($W = 10$).

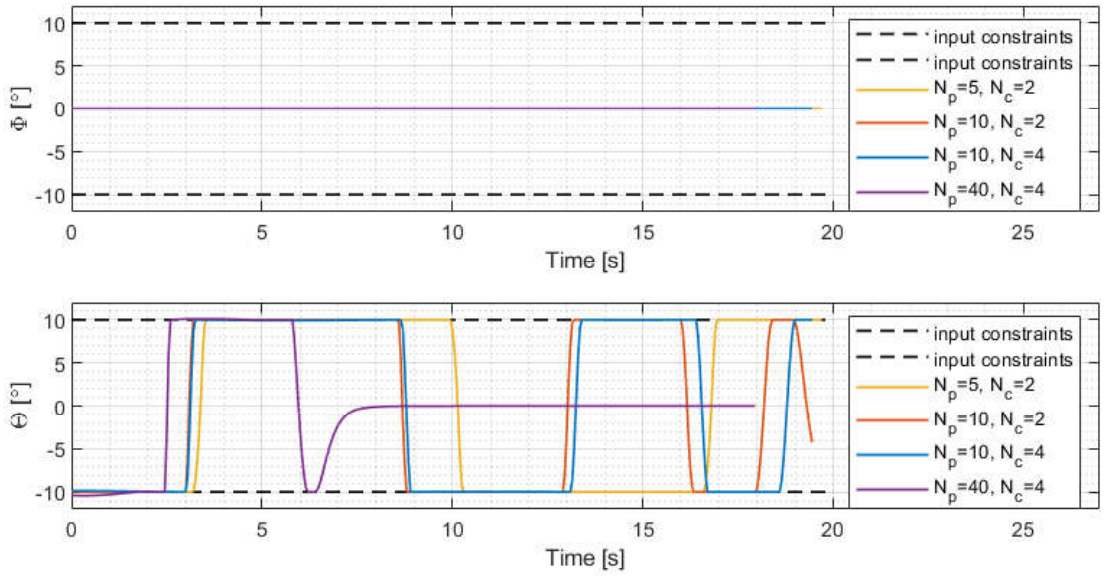

Figure 5.3. Simulation Input Results for the Waypoint Reference ($W = 10$).

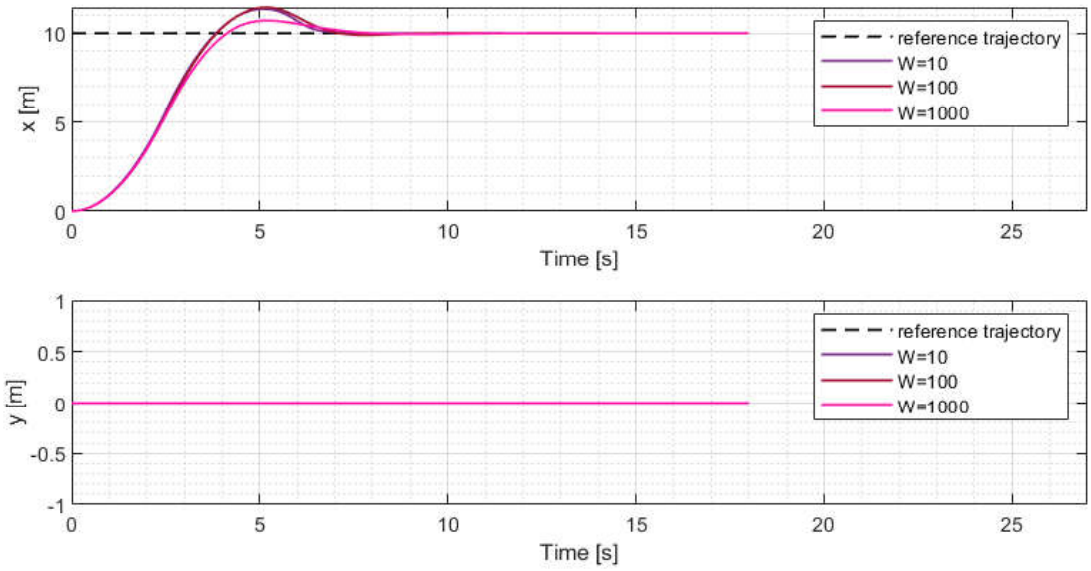

Figure 5.4. Simulation Output Results for the Waypoint Reference ($N_p = 40, N_c = 4$).

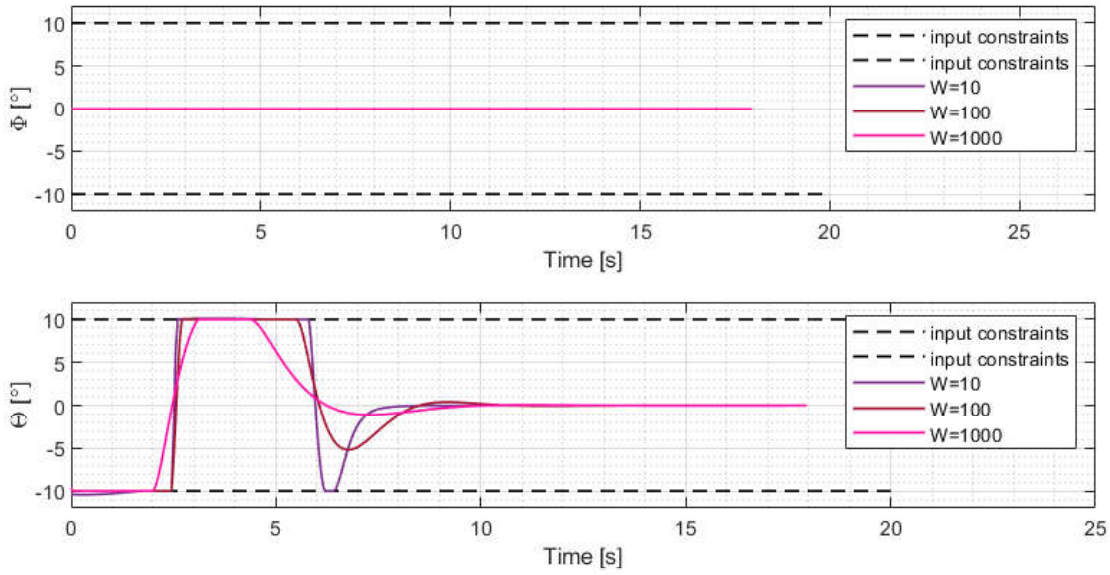

Figure 5.5. Simulation Input Results for the Waypoint Reference ($N_p = 40, N_c = 4$).

5.2. Longitudinal Reference Trajectory

Simulation tests were continued using a longitudinal reference trajectory. The longitudinal reference trajectory was generated as

$$r(t) = \begin{bmatrix} x_r(t) \\ y_r(t) \end{bmatrix} = \begin{bmatrix} t \\ 0 \end{bmatrix}. \tag{5.2}$$

It has a constant velocity of 1 m/s in the x-direction. In Figure 5.6, the reference trajectory, including the start and end points, is shown on the EFF.

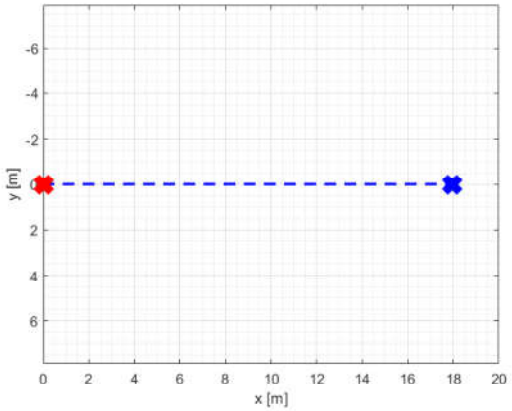

Figure 5.6. Longitudinal Trajectory Reference.

Looking at Figures 5.7, and 5.8, it is observed that accepting N_p as 10 and N_c as 4 is sufficient for a good controller performance. However, disturbances such as windy weather conditions were not included when preparing the simulation. Therefore, N_p and N_c parameters can be chosen as high as possible to achieve good performance with less control effort, despite disturbances. Also Figure 5.10 shows how the control effort decreases with increasing W value.

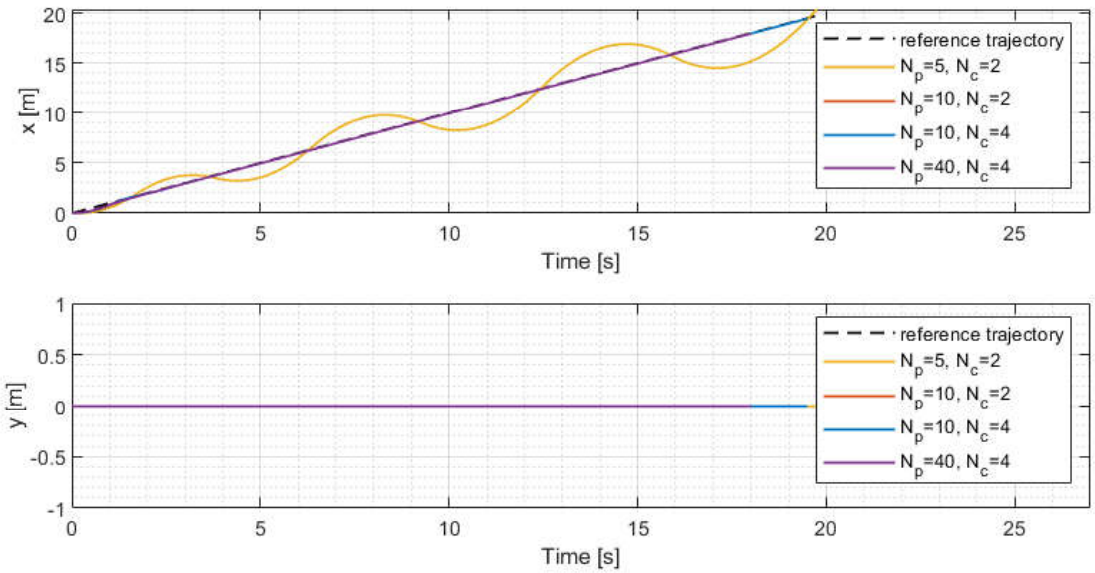

Figure 5.7. Simulation Output Results for the Longitudinal Trajectory Reference ($W = 10$).

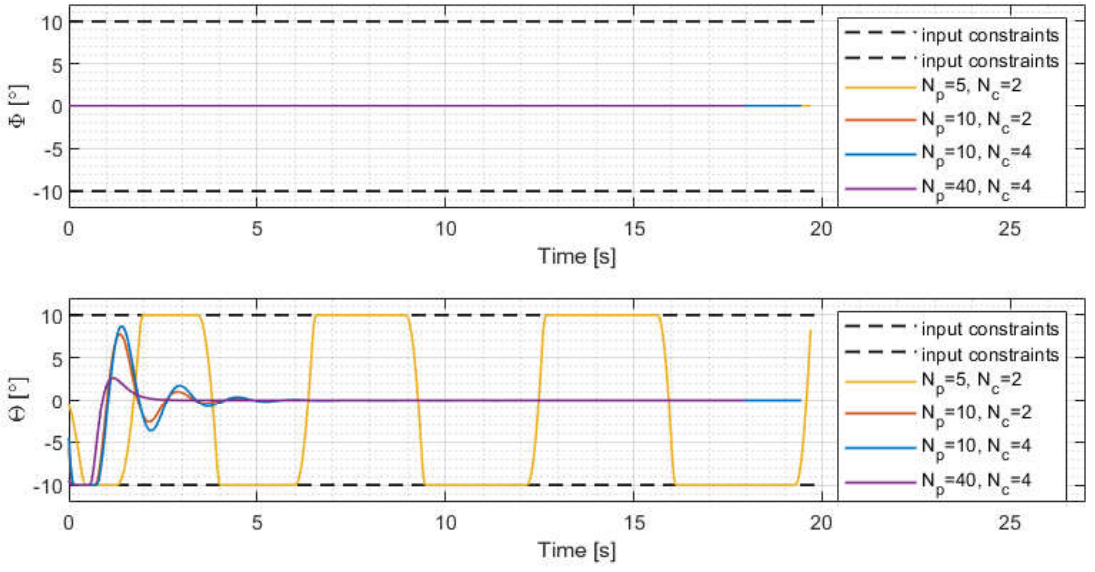

Figure 5.8. Simulation Input Results for the Longitudinal Trajectory Reference ($W = 10$).

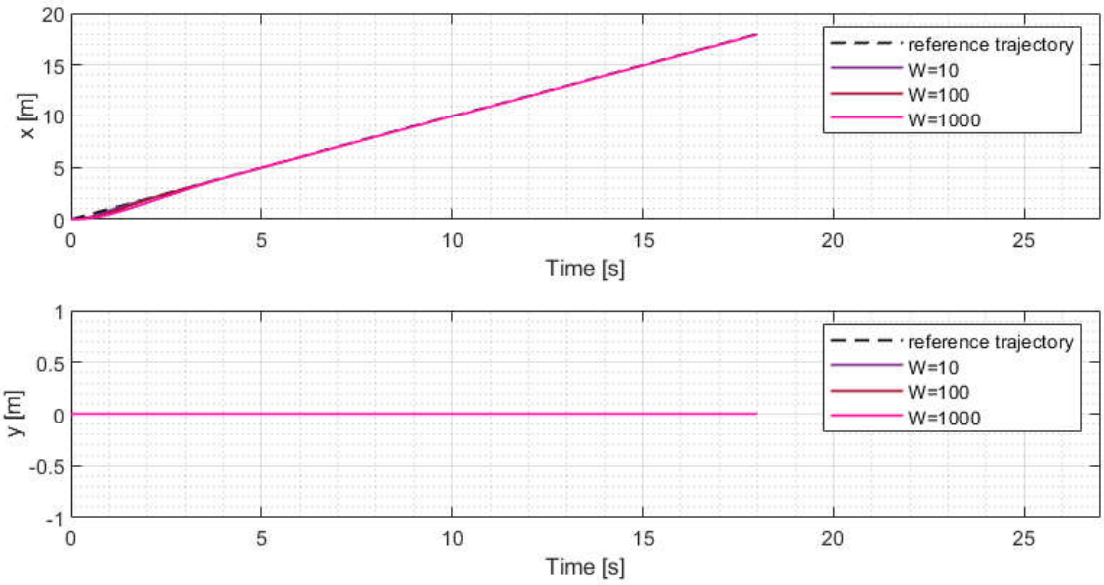

Figure 5.9. Simulation Output Results for the Longitudinal Trajectory Reference ($N_p = 40, N_c = 4$).

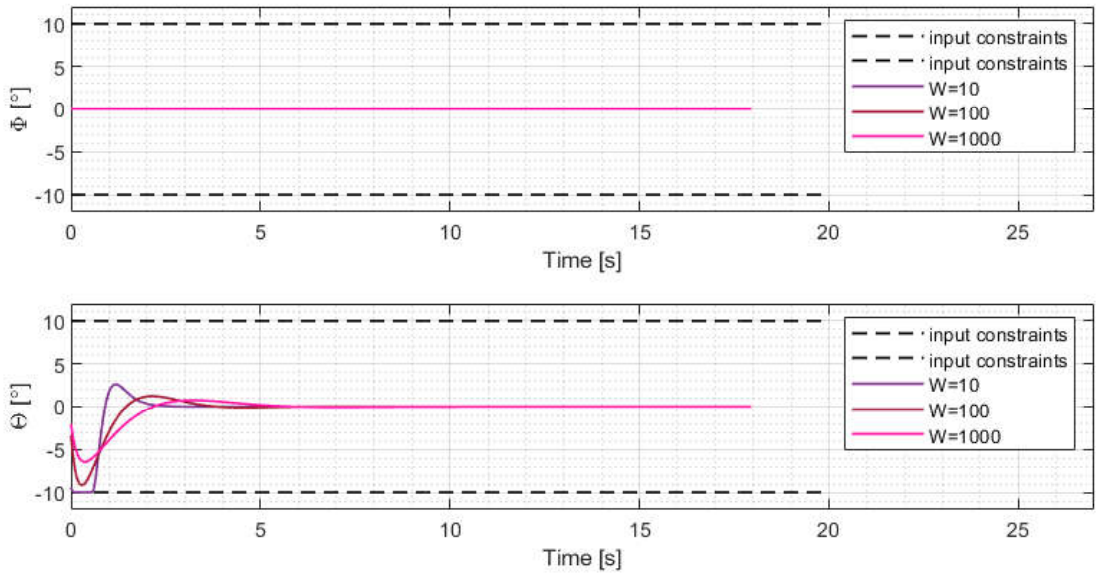

Figure 5.10. Simulation Input Results for the Longitudinal Trajectory Reference ($N_p = 40, N_c = 4$).

5.3. Sinusoidal Reference Trajectory

The simulation tests were continued using a sinusoidal reference trajectory. The sinusoidal reference trajectory was generated as

$$r(t) = \begin{bmatrix} x_r(t) \\ y_r(t) \end{bmatrix} = \begin{bmatrix} t \\ 3sin(0.4t) \end{bmatrix}. \tag{5.3}$$

It has a constant velocity of 1 m/s in the x-direction and a variable velocity in the y-direction. In Figure 5.11, the reference trajectory, including the start and end points, is shown on the EFF.

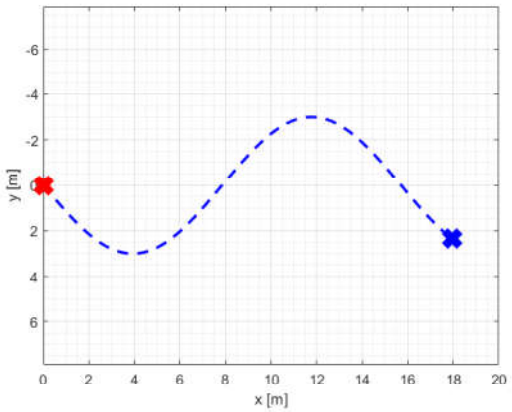

Figure 5.11. Sinusoidal Trajectory Reference.

Looking at Figures 5.12, and 5.13, it is observed that accepting N_p as 10 and N_c as 4 is sufficient for a good controller performance. However, as stated in the test using longitudinal reference, disturbances were neglected in the simulation. Therefore, N_p and N_c parameters can be chosen as high as possible to achieve good performance with less control effort, despite disturbances. Also Figure 5.15 shows how the control effort decreases with increasing W value.

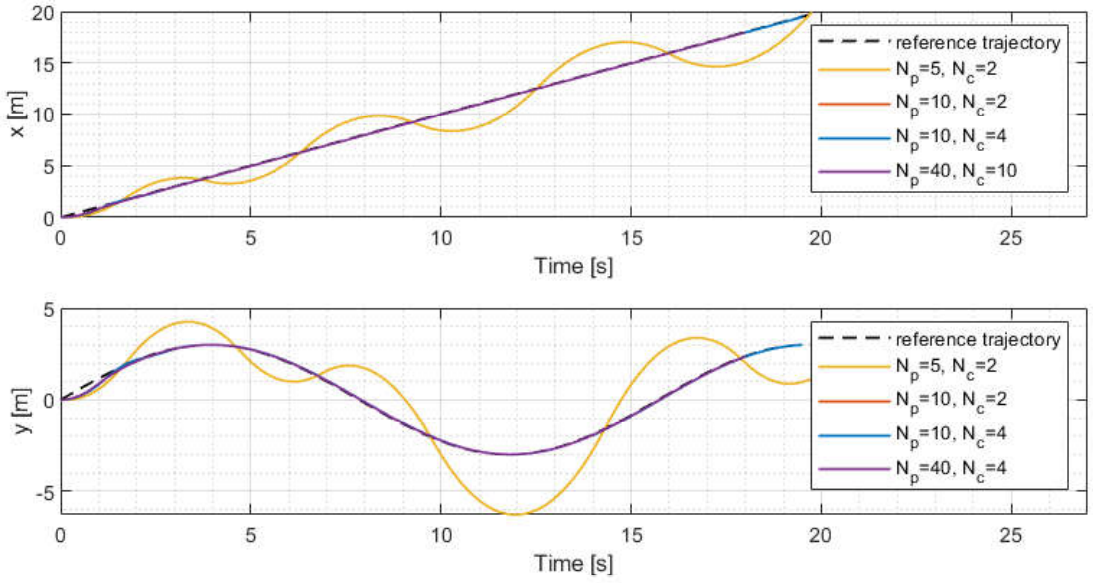

Figure 5.12. Simulation Output Results for the Sinusoidal Trajectory Reference ($W = 10$).

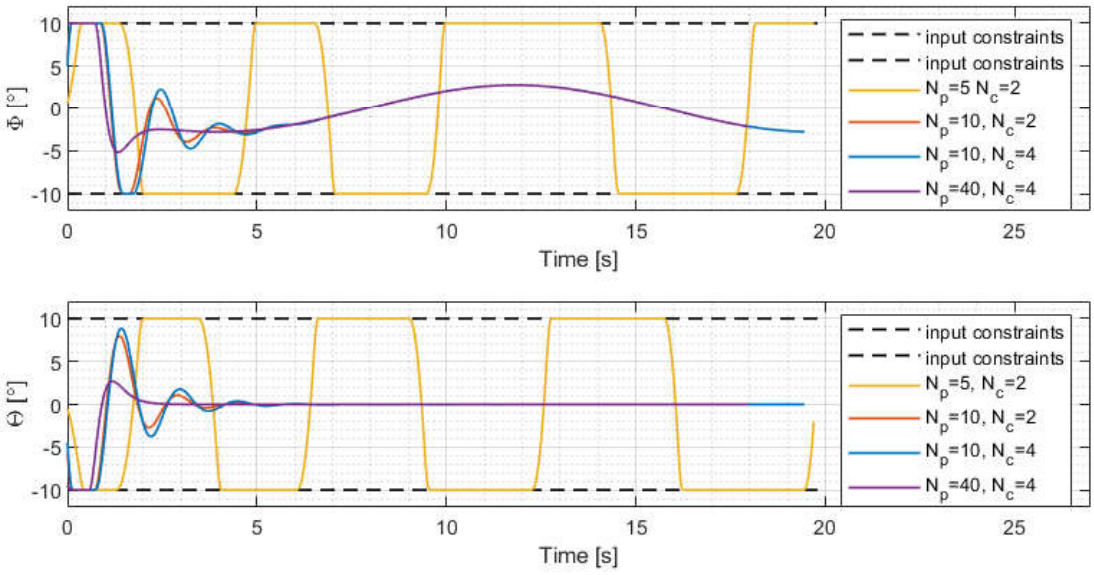

Figure 5.13. Simulation Input Results for the Sinusoidal Trajectory Reference ($W = 10$).

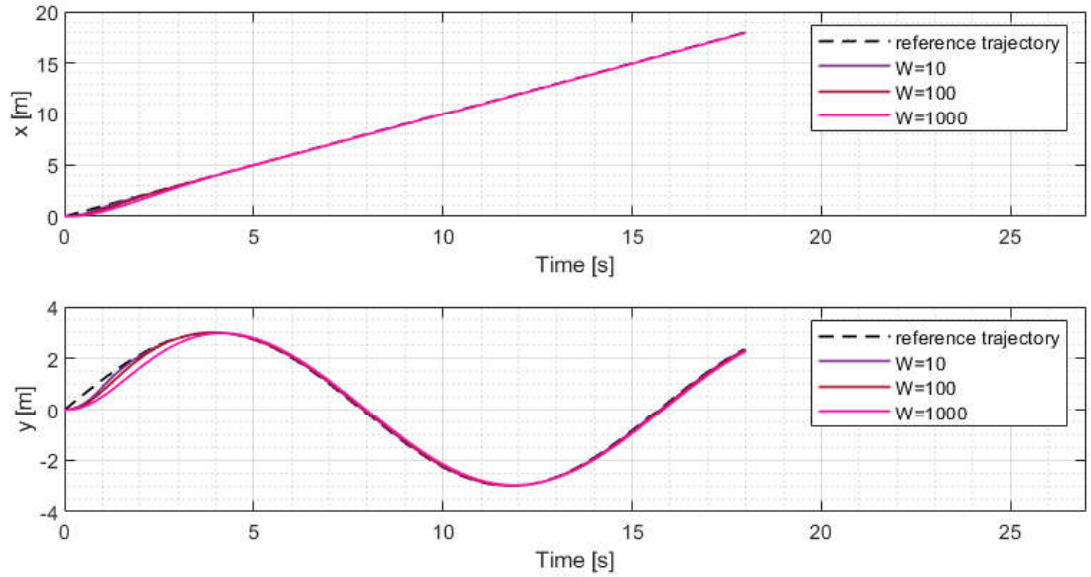

Figure 5.14. Simulation Output Results for the Sinusoidal Trajectory Reference ($N_p = 40, N_c = 4$).

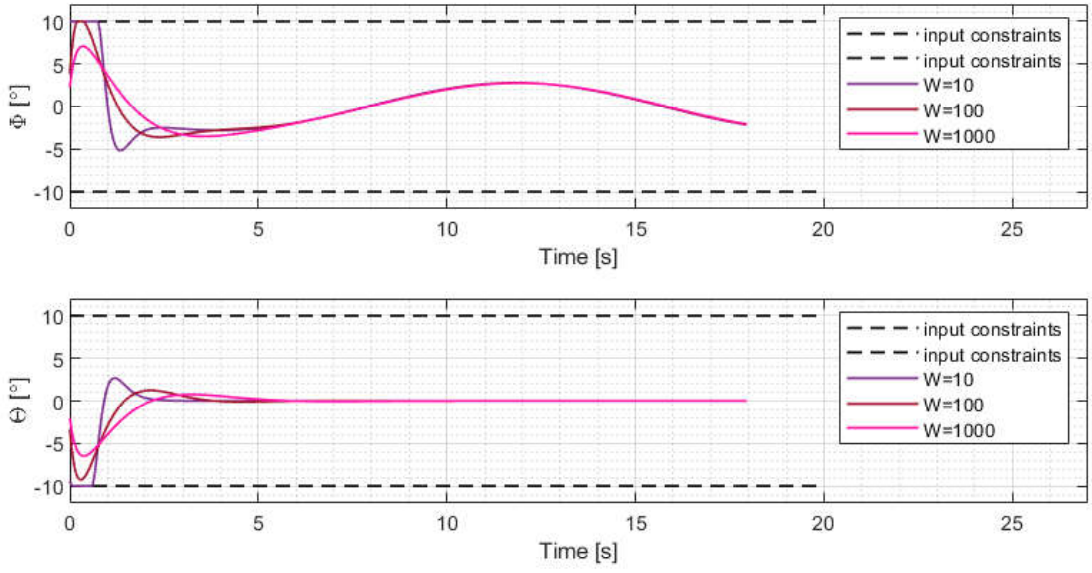

Figure 5.15. Simulation Input Results for the Sinusoidal Trajectory Reference ($N_p = 40, N_c = 4$).

5.4. Circular Reference Trajectory

The simulation tests were continued using a circular reference trajectory. The circular reference trajectory was generated by using the sine and cosine functions as

$$r(t) = \begin{bmatrix} x_r(t) \\ y_r(t) \end{bmatrix} = \begin{bmatrix} 3 - 3cos(0.4t) \\ 3sin(0.4t) \end{bmatrix}. \tag{5.4}$$

In Figure 5.16, the reference trajectory, including start and end points, is shown on the EFF.

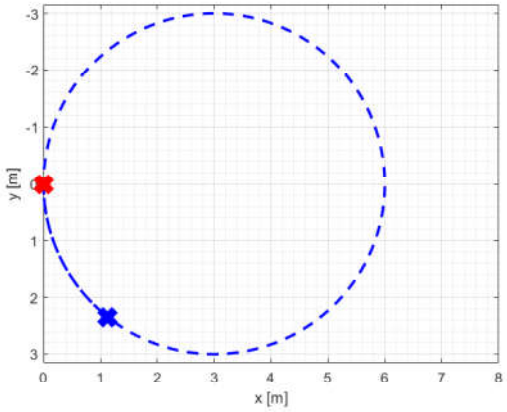

Figure 5.16. Circular Trajectory Reference.

As can be seen from the results in Figures 5.17 and 5.18, the system is not stable in the test where N_p is 5 and N_c is 2. Desired control performance was achieved only by increasing the N_c value. However, as stated in the test using longitudinal reference, disturbances were neglected in the simulation. For this reason, choosing high N_p and N_c parameters may be preferable in real flight tests as it will reduce control effort. Furthermore, the results in which the W parameter is changed and the N_p and N_c parameters are kept constant are shown in Figures 5.19 and 5.20. It is observed that there is a delay in trajectory tracking in the y-direction with increasing W parameter.

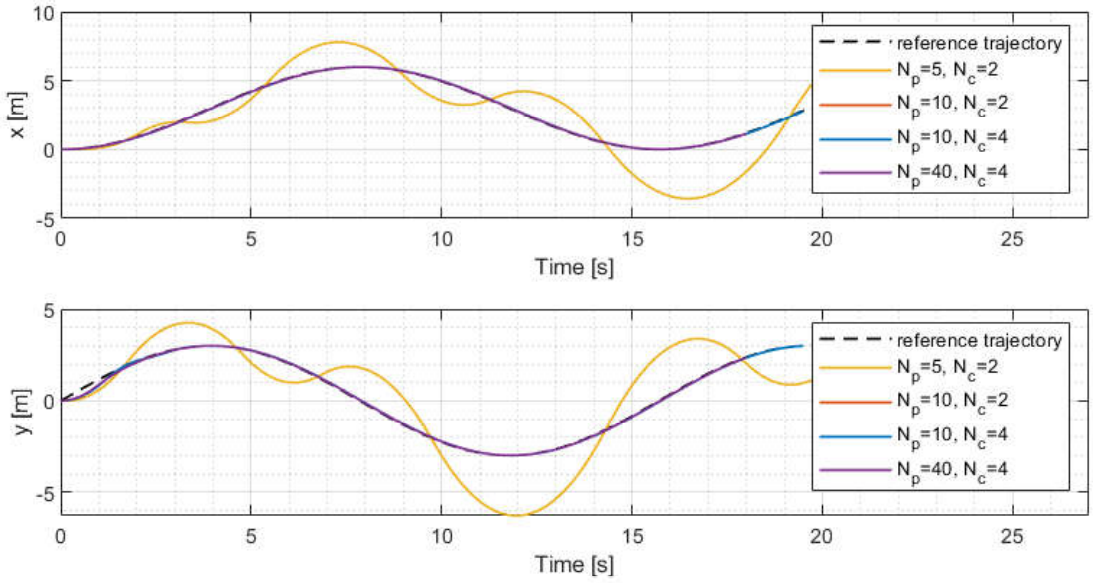

Figure 5.17. Simulation Output Results for the Circular Trajectory Reference ($W = 10$).

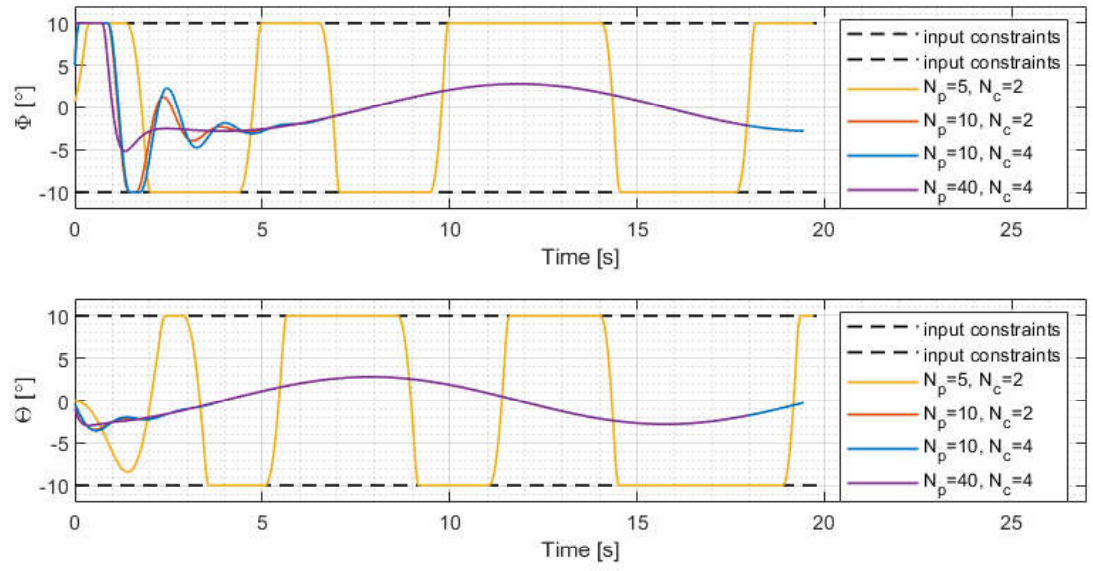

Figure 5.18. Simulation Input Results for the Circular Trajectory Reference ($W = 10$).

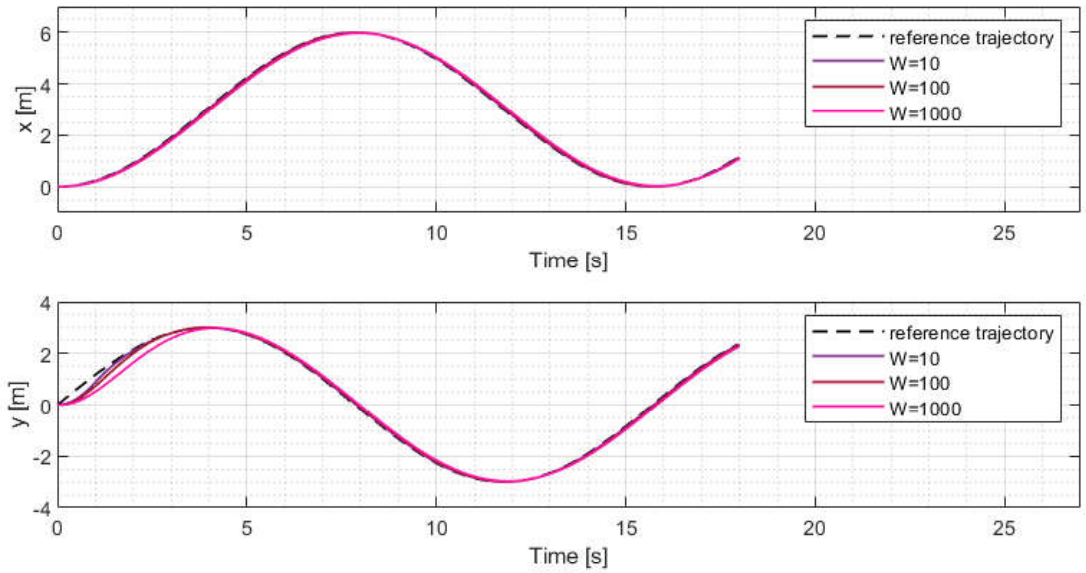

Figure 5.19. Simulation Output Results for the Circular Trajectory Reference ($N_p = 40, N_c = 4$).

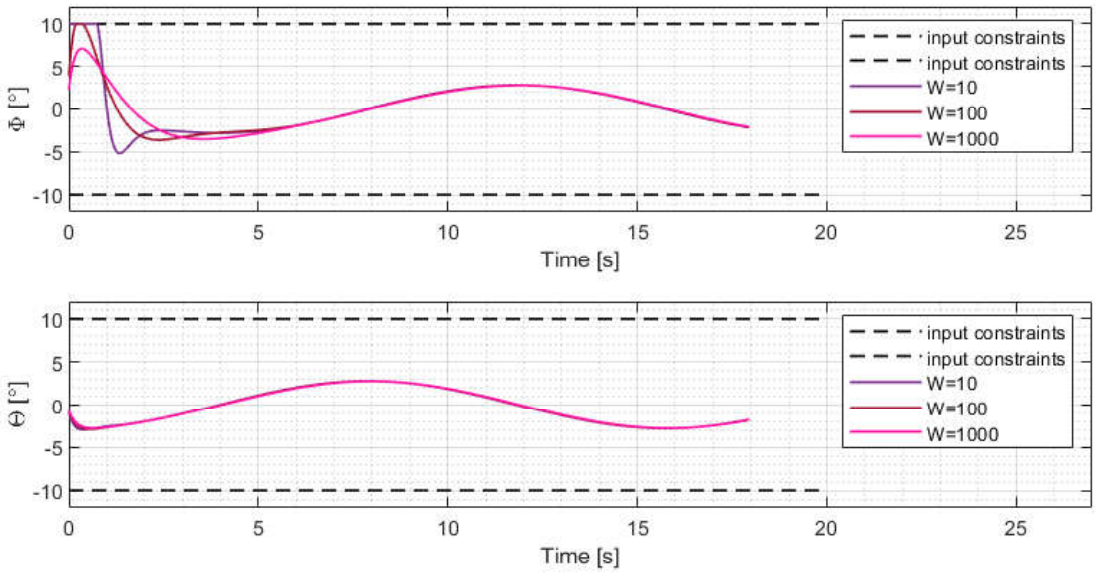

Figure 5.20. Simulation Input Results for the Circular Trajectory Reference ($N_p = 40, N_c = 4$).

6. FLIGHT TEST RESULTS AND DISCUSSION

The flight test results consist of the data logged by the Raspberry Pi. The flight data includes the reference trajectories, current position, and the system inputs generated by the MPC. The reference trajectories are generated from equations in the Raspberry Pi board, while the current position is taken from the estimator of the ArduPilot in the Pixhawk board. The position estimator highly depends on the GPS data connected to the Pixhawk board. The serial connection provides the communication between the Raspberry Pi and Pixhawk boards. Finally, the inputs are created by considering each iteration's reference trajectories and actual outputs. They are sent to the Pixhawk's low-level controller with the help of the DroneKit library. DroneKit library consists of functions that contain commands that Pixhawk can understand. The takeoff, attitude send, and landing functions are used from the DroneKit library. The takeoff and landing functions provide the quadcopter's desired altitude while maintaining its x and y positions constant. On the other hand, the attitude send function does not deal with the position of the quadcopter. It sends the roll and pitch commands produced in the high-level controller to the low-level controller in each optimization cycle.

Finally, the reference, output, and input data were recorded in each flight test. The flight tests were started by using the design parameters tested in the simulation environment, in Chapter 5. The reference trajectories used in the flight tests were also the same as those used in the simulation tests. Therefore, flight data for four different trajectories will be presented in this chapter. First of all, the case named Test-1 with the design parameters $N_p = 40$, $N_c = 4$, and $W = 10$ will be investigated for all trajectories. Then, the case named Test-2 with the design parameters $N_p = 40$, $N_c = 4$, and $W = 1000$ will be investigated to see the effect of input weights on control performance. After all, the case named Test-3 with the design parameters $N_p = 40$, $N_c = 6$, and $W = 1000$ will be investigated to see the control horizon's effect on the controller performance.

6.1. Reference Waypoint

The reference waypoint, which was used in simulation tests, was also used in flight tests. The equation is rewritten as

$$r(t) = \begin{bmatrix} x_r(t) \\ y_r(t) \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \end{bmatrix}. \tag{6.1}$$

6.1.1. Test-1 with the Design Parameters $N_p = 40$, $N_c = 4$, and $W = 10$

Test-1 was conducted to examine the controller performance when the design parameter W is low. In Figure 6.1, the resultant trajectory of the quadcopter on the EFF is shown.

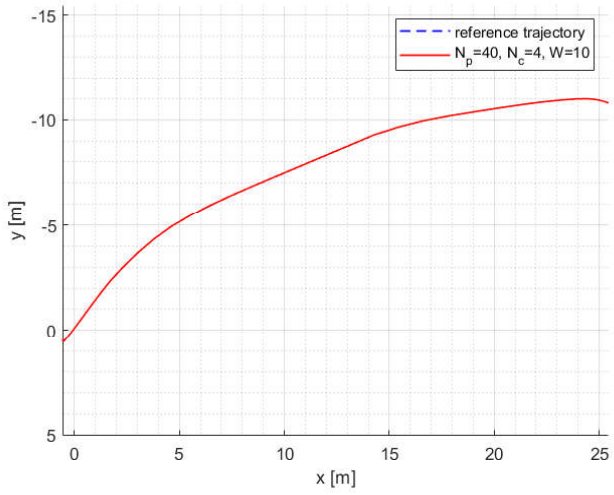

Figure 6.1. Test-1 Results for the Reference Waypoint.

In the simulation tests, it was observed that low W caused an overshoot. In Figure 6.2, the quadcopter approaches the reference waypoint in the x-direction. In Figure 6.3, it is observed that as the trajectory error in the x-direction decreases, the input θ changes to slow down the movement. However, the input cannot prevent the overshoot. In the y-direction, a negative input ϕ is produced because the starting point

is at positive y. After the quadcopter came to 0, the positive input ϕ started to be produced, but the quadcopter could stop.

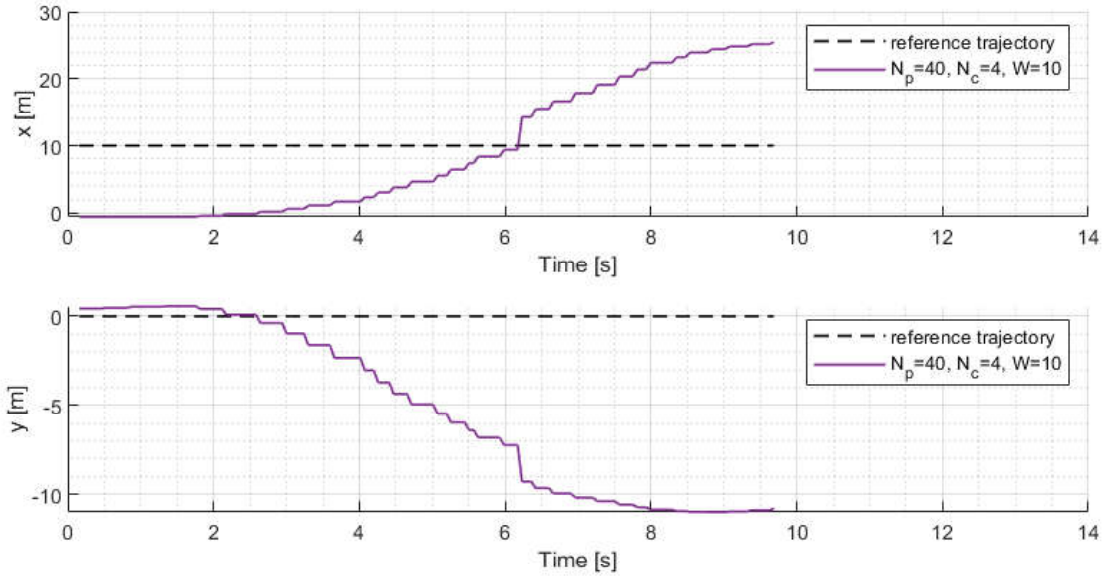

Figure 6.2. Test-1 Output Results for the Reference Waypoint.

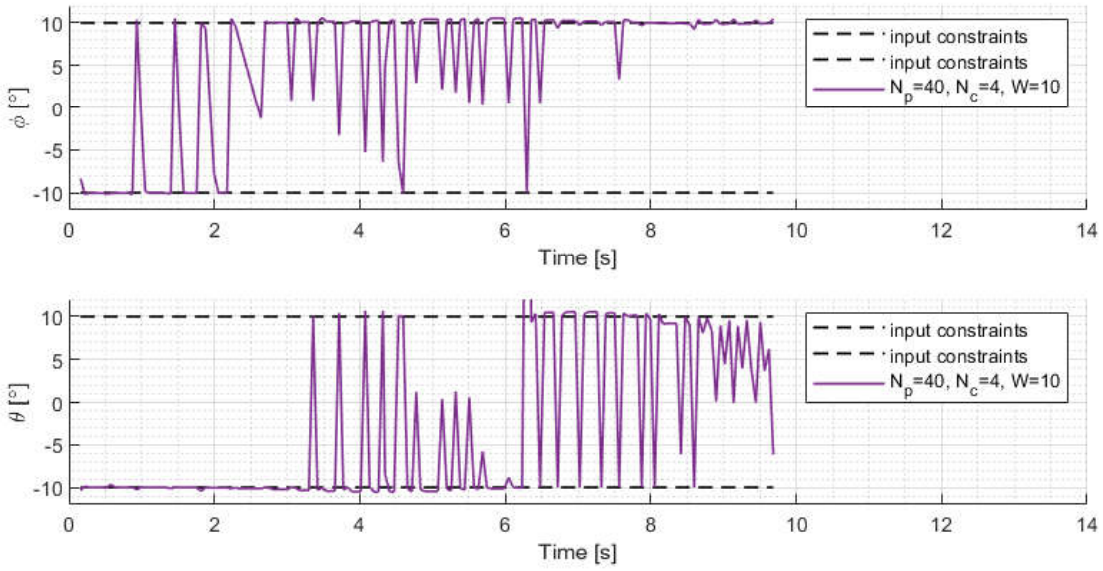

Figure 6.3. Test-1 Input Results for the Reference Waypoint.

6.1.2. Test-2 with the Design Parameters $N_p = 40$, $N_c = 4$, and $W = 1000$

After Test-1 was performed, Test-2 was performed to see the effect of high W on the controller performance, while N_p and N_c kept constant. In Figure 6.4, the resultant trajectory of the quadcopter on the EFF is shown.

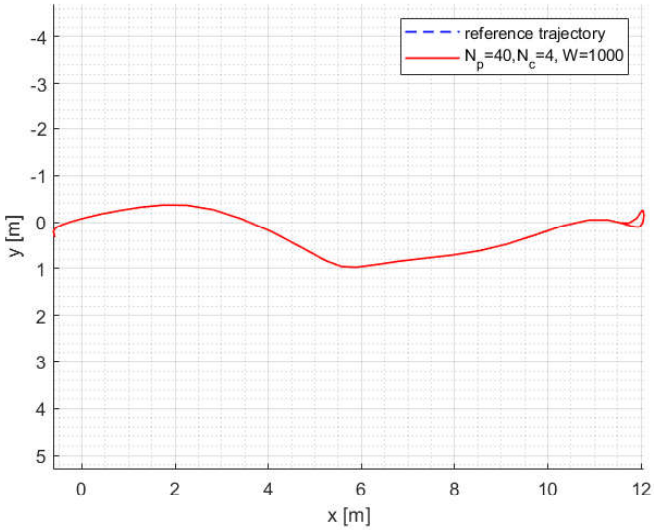

Figure 6.4. Test-2 Results for the Reference Waypoint.

In simulation results, it was observed that high W causes reduced control effort and prevents overshoot. In Figure 6.5, it is observed that the quadcopter maintains its position without overshooting after reaching the reference waypoint in the x-direction. Also, in the y-direction, it holds its position. In Figure 6.6, it is observed that the generated inputs are softer. Therefore, the control effort is reduced as is expected. It has been concluded that the trajectory tracking controller gives satisfactory results with the determined coefficients. However, the chattering problem in the produced inputs still persists.

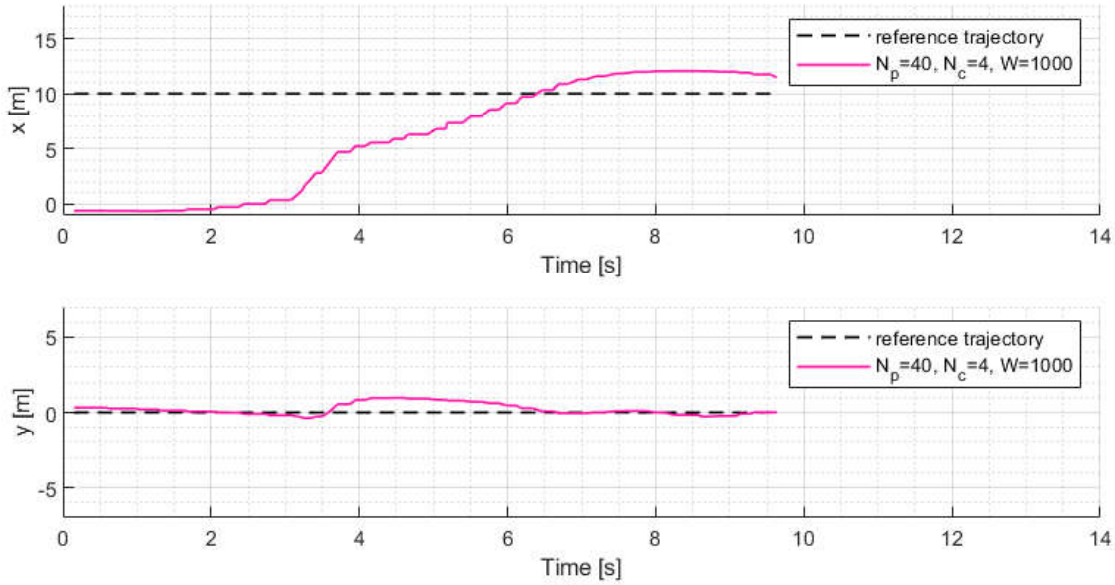

Figure 6.5. Test-2 Output Results for the Reference Waypoint.

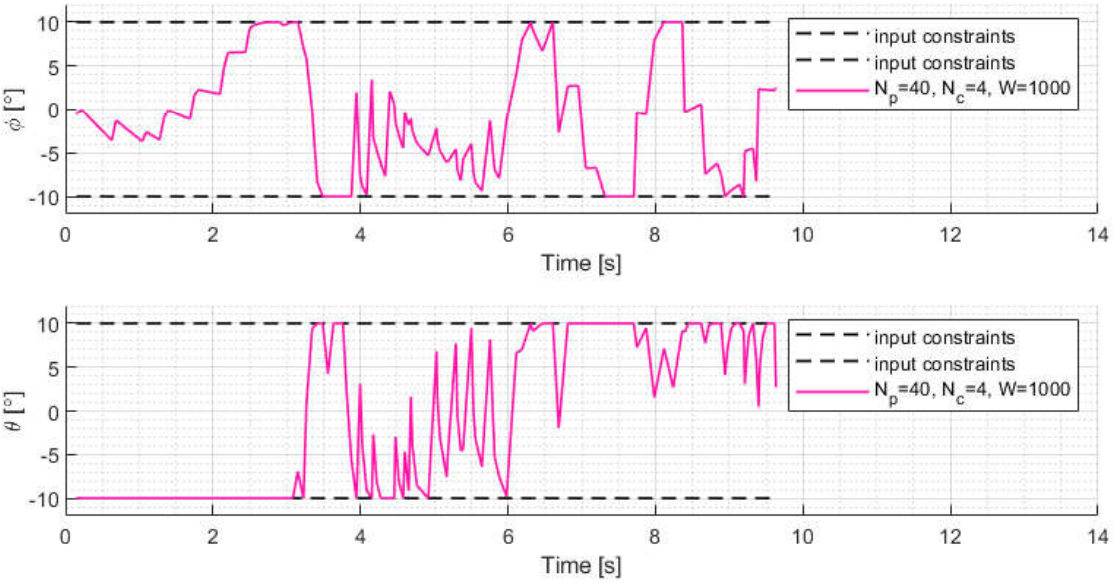

Figure 6.6. Test-2 Input Results for the Reference Waypoint.

6.1.3. Test-3 with the Design Parameters $N_p = 40$, $N_c = 6$, and $W = 1000$

After Test-2 was performed, Test-3 was conducted to decrease chattering observed in the inputs. Therefore, the N_c value was increased, and the reference waypoint test was repeated. In Figure 6.7, the resultant trajectory of the quadcopter on the EFF is shown.

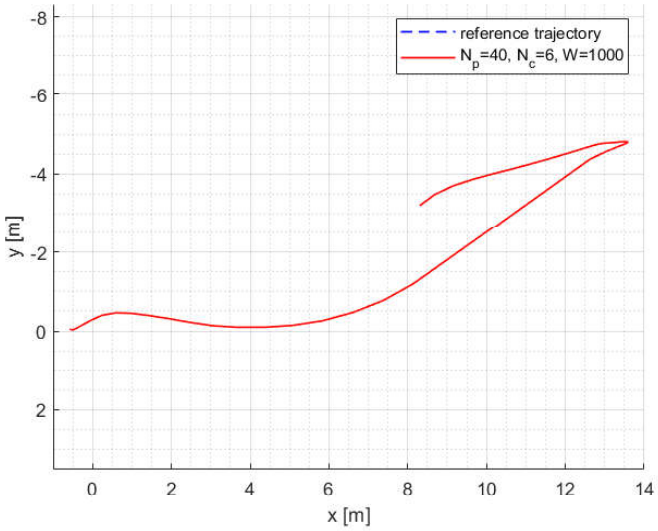

Figure 6.7. Test-3 Results for the Reference Waypoint.

In Figure 6.8, it is observed that the rise time of the controller increases by increasing N_c, and it was also observed that the time required for the optimization solution increased approximately 1.5 times. In Figure 6.9, it is observed that the chattering in θ does not change as it was expected. At $t = 7$, one can see that the actual position of the quadcopter was misestimated. This problem can be explained by the fact that the GPS used in the quadcopter is not of sufficient quality.

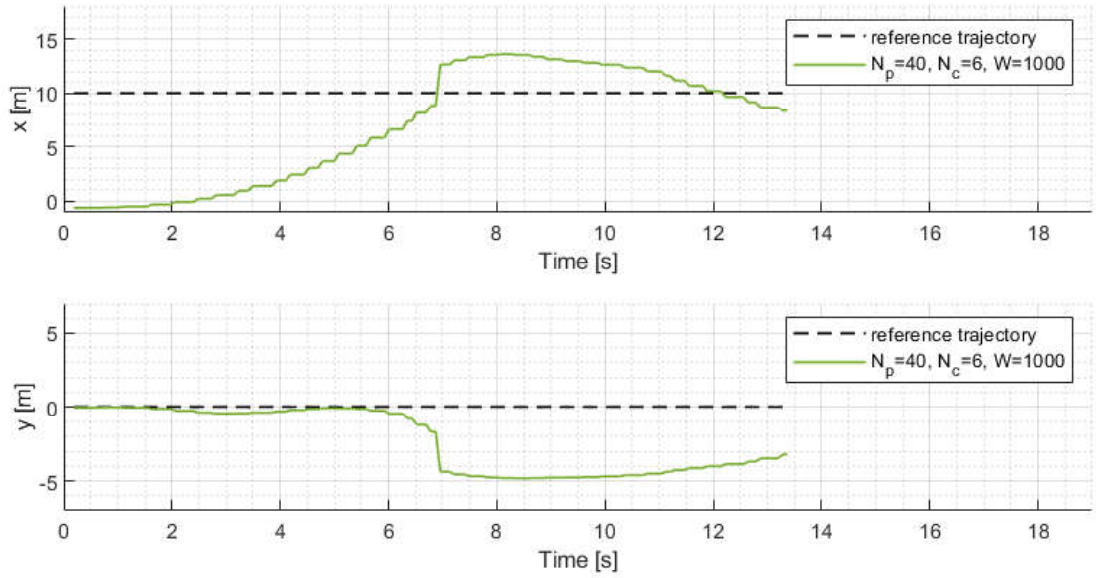

Figure 6.8. Test-3 Output Results for the Reference Waypoint.

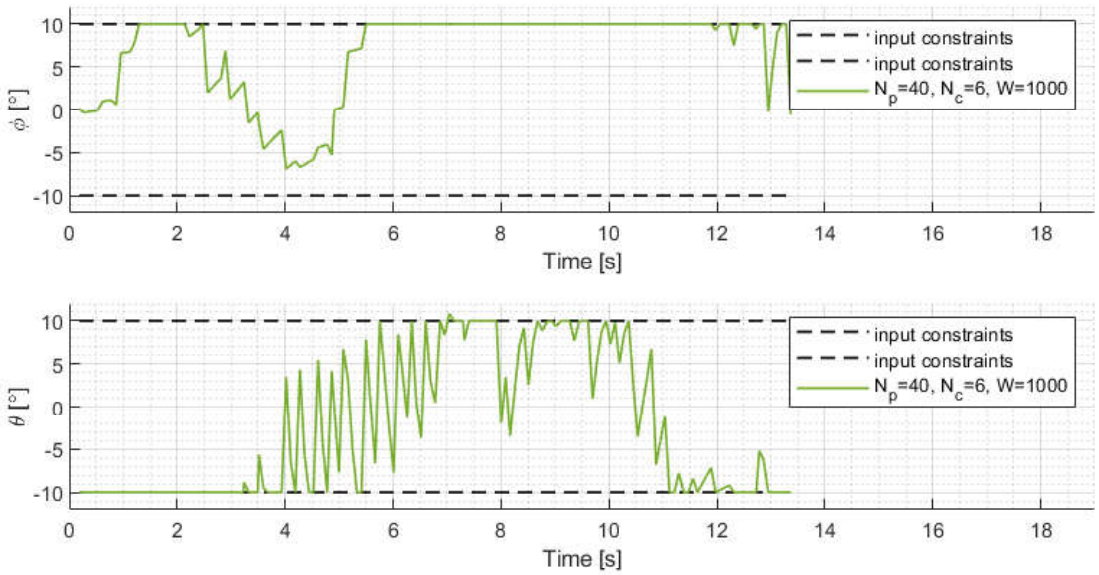

Figure 6.9. Test-3 Input Results for the Reference Waypoint.

6.2. Longitudinal Reference Trajectory

The longitudinal reference trajectory, which was used in simulations tests, was also used in flight tests. The equation is rewritten as

$$r(t) = \begin{bmatrix} x_r(t) \\ y_r(t) \end{bmatrix} = \begin{bmatrix} t \\ 0 \end{bmatrix}. \tag{6.2}$$

6.2.1. Test-1 with the Design Parameters $N_p = 40$, $N_c = 4$, and $W = 10$

Test-1 was conducted to examine the controller performance when the design parameter W is low. In Figure 6.10, the resultant trajectory of the quadcopter on the EFF is shown.

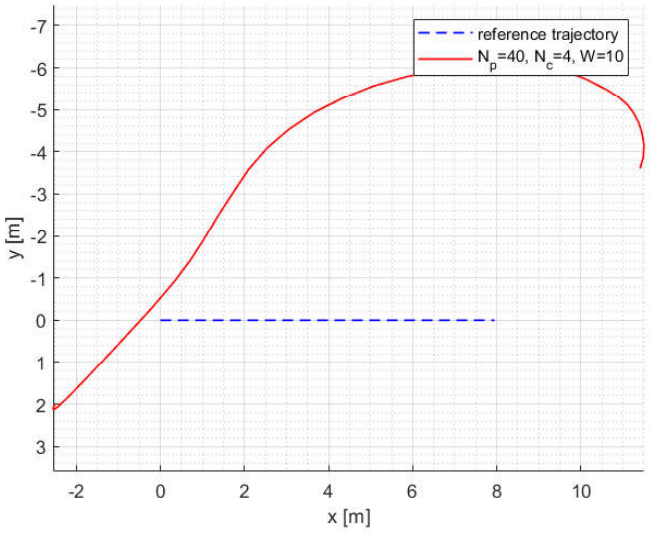

Figure 6.10. Test-1 Results for the Longitudinal Reference Trajectory.

In Figure, 6.11, it is observed that the quadcopter starts to move in the positive y-direction. In Figure 6.12, it is observed that the controller produces negative ϕ input for this reason. As the quadcopter approaches the 0 positions, the generated inputs increase. However, since the design parameter W is low, overshoot is observed in the

movement in the y-direction. The overshoot in the x-direction is lower than in the y-direction. The reason may be that the wind is often effective in the y-direction.

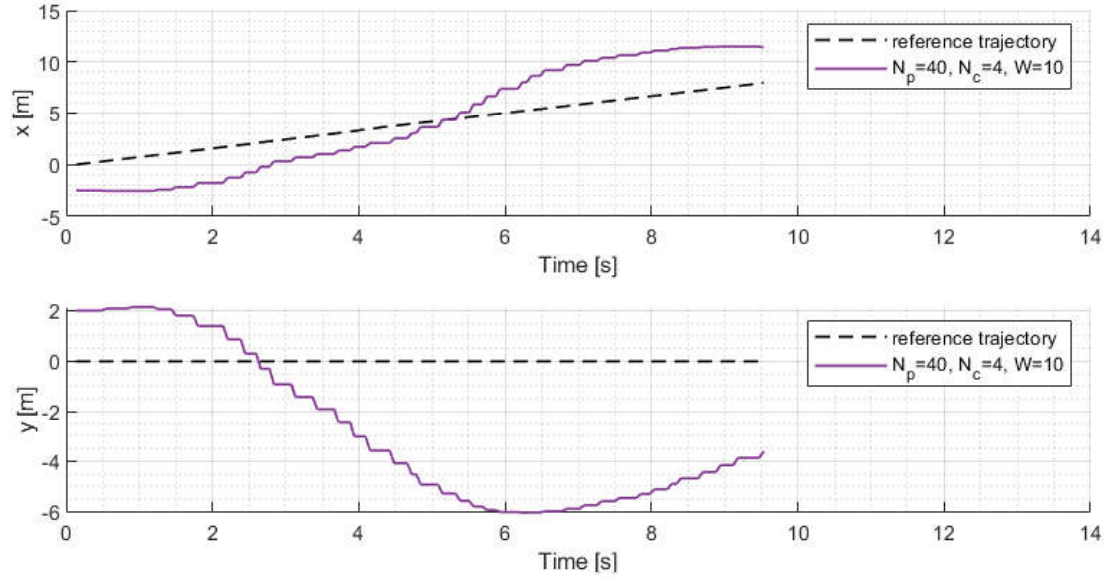

Figure 6.11. Test-1 Output Results for the Longitudinal Reference Trajectory.

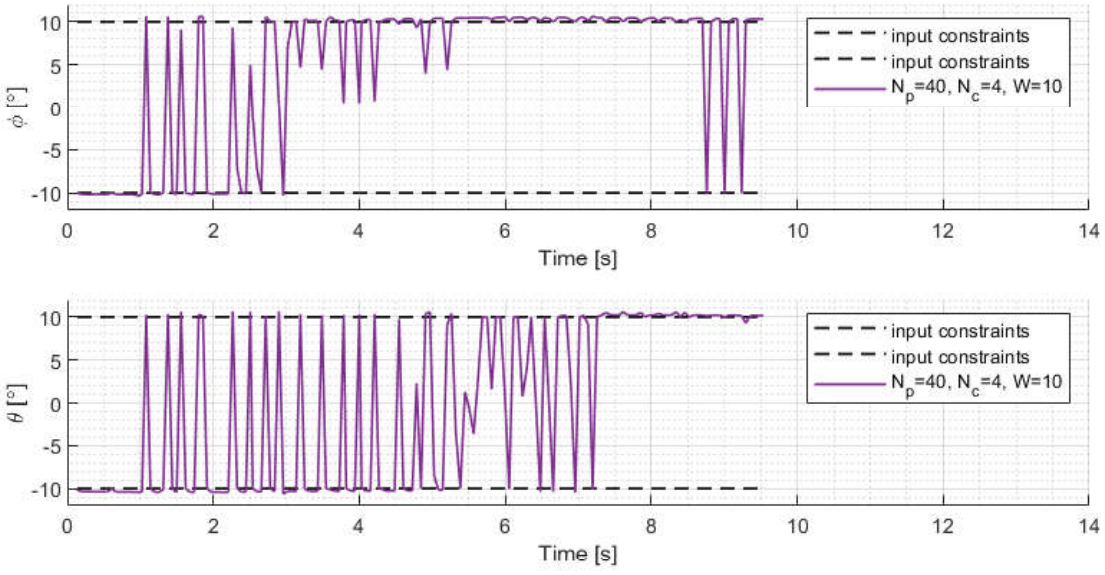

Figure 6.12. Test-1 Input Results for the Longitudinal Reference Trajectory.

6.2.2. Test-2 with the Design Parameters $N_p = 40$, $N_c = 4$, and $W = 1000$

After Test-1 was performed, Test-2 was performed to see the effect of high W on the controller performance, while N_p and N_c kept constant. In Figure 6.13, the resultant trajectory of the quadcopter on the EFF is shown.

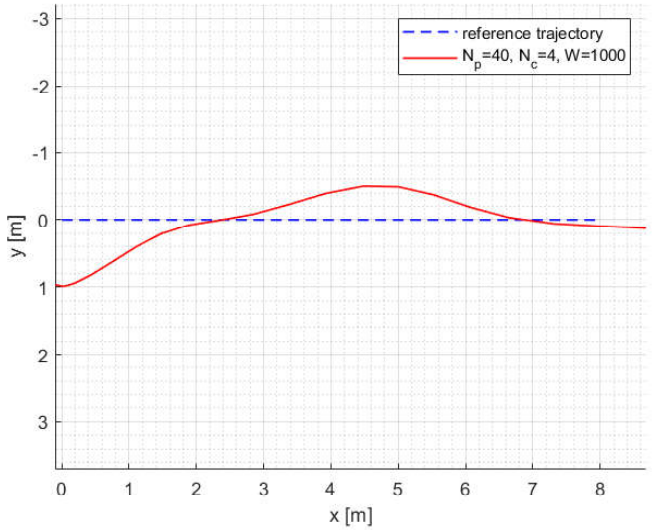

Figure 6.13. Test-2 Results for the Longitudinal Reference Trajectory.

Compared with the low W parameter, it is observed that the high W parameter has better trajectory tracking result. In Figure 6.14, the x and y positions of the quadcopter are shown. Even if the quadcopter started its movement in a positive y position, it was able to follow the trajectory without an overshoot. In Figure 6.15, the generated inputs ϕ and θ angles are shown. The control effort is sufficiently low with respect to the previous results. Also, as it is observed in the simulation results, the high W parameter causes some delay in the generation of the inputs.

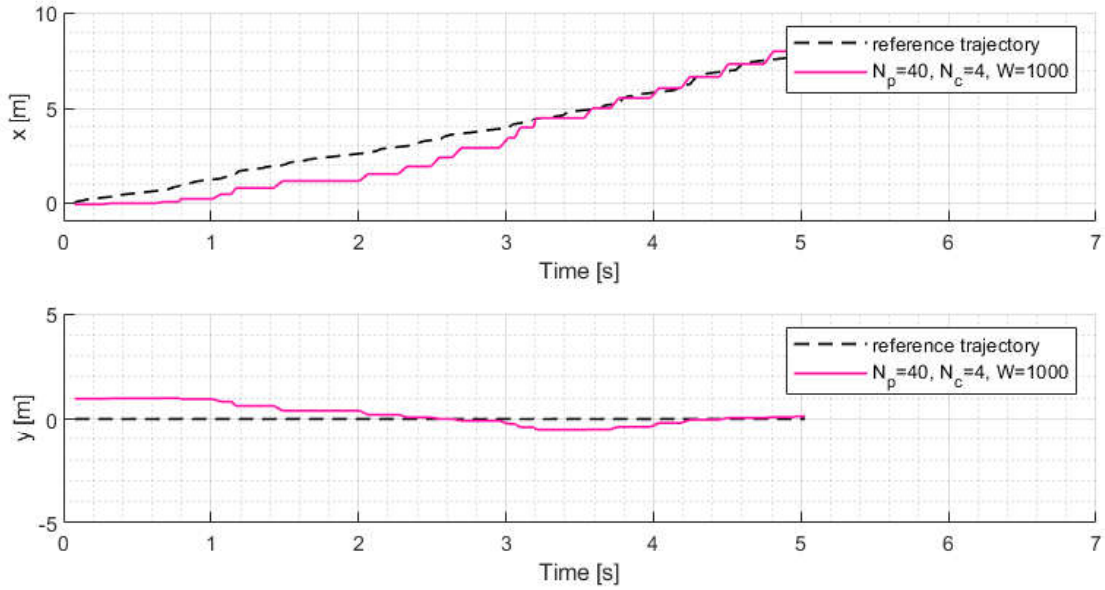

Figure 6.14. Test-2 Output Results for the Longitudinal Reference Trajectory.

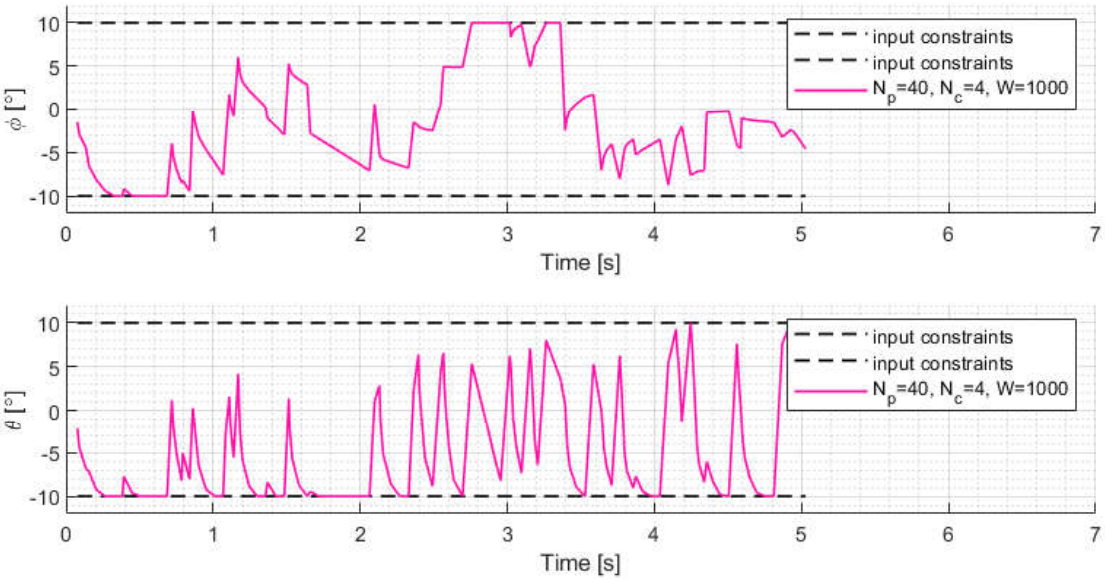

Figure 6.15. Test-2 Input Results for the Longitudinal Reference Trajectory.

6.2.3. Test-3 with the Design Parameters $N_p = 40$, $N_c = 6$, and $W = 1000$

After Test-2 was performed, Test-3 was conducted to observe the effect of high N_c on the controller performance. In Figure 6.7, the resultant trajectory of the quadcopter on the EFF is shown.

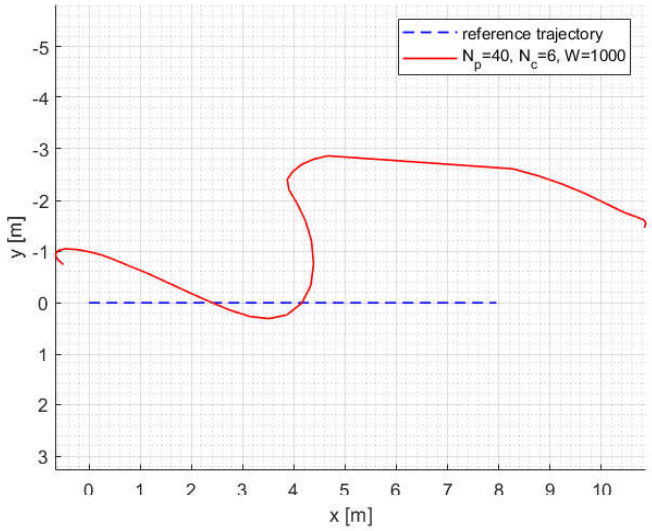

Figure 6.16. Test-3 Results for the Longitudinal Reference Trajectory.

In Figure 6.17, the output results are shown. When compared with the results with the low N_c parameter, it is seen that trajectory errors are higher in this test. However, since the day of this test was very windy, it should be considered that the results were affected by the weather conditions. Increasing the N_c value was tried to reduce the noise in the inputs. In the results in Figure 6.18, it is observed that it does not reduce but decreases the wavelength.

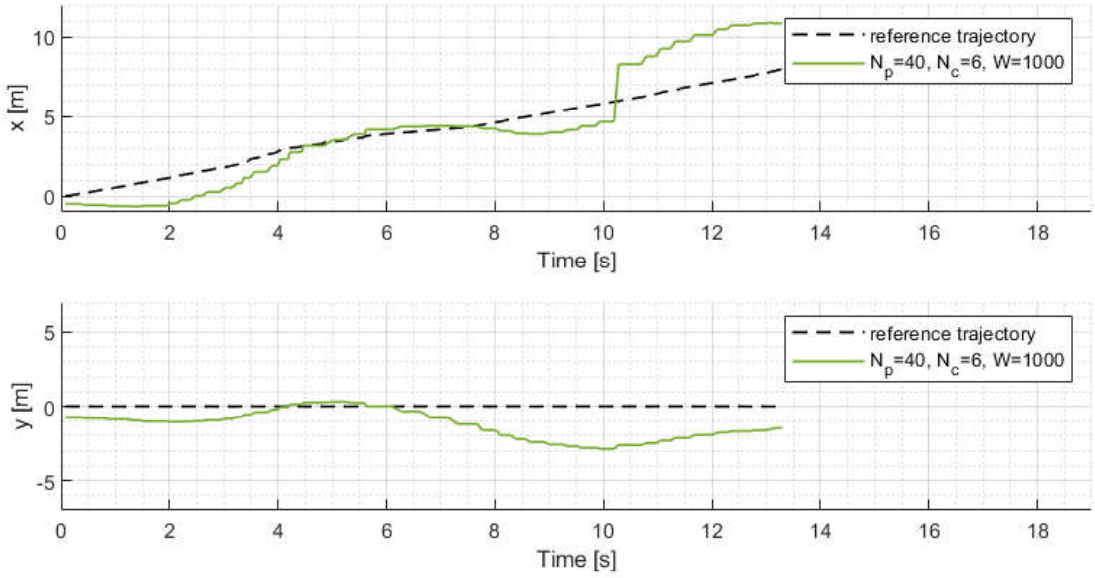

Figure 6.17. Test-3 Output Results for the Longitudinal Reference Trajectory.

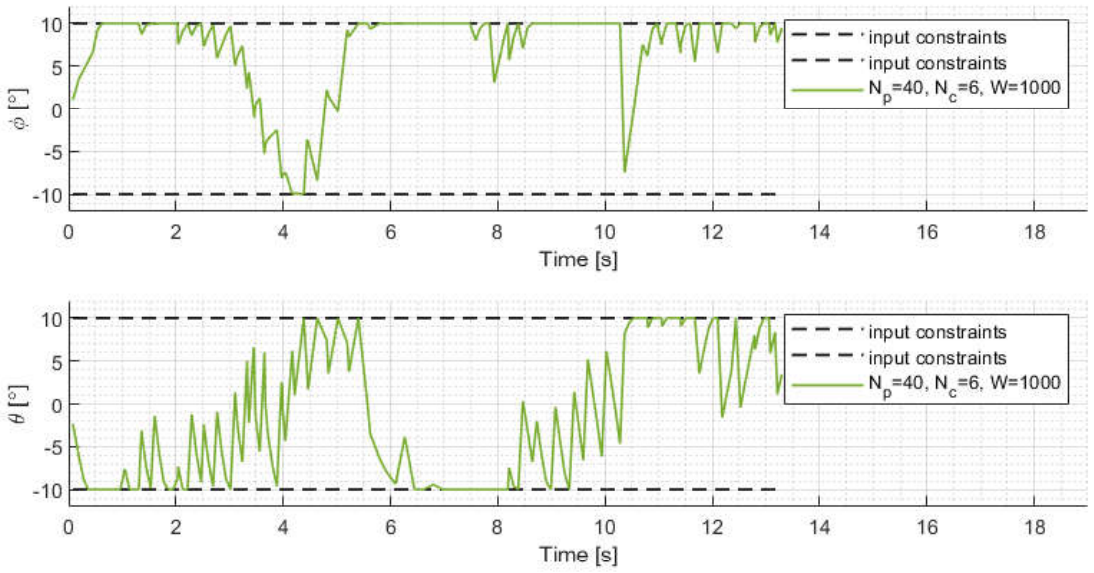

Figure 6.18. Test-3 Input Results for the Longitudinal Reference Trajectory.

6.3. Sinusoidal Reference Trajectory

The sinusoidal reference trajectory, which is used in the simulation tests, was also used in the flight tests. The equation from which the sinusoidal reference trajectory is created as shown below.

$$r(t) = \begin{bmatrix} x_r(t) \\ y_r(t) \end{bmatrix} = \begin{bmatrix} t \\ 3sin(0.4t) \end{bmatrix} \tag{6.3}$$

6.3.1. Test-1 with the Design Parameters $N_p = 40$, $N_c = 4$, and $W = 10$

Test-1 was conducted to examine the controller performance when the design parameter W is low. In Figure 6.19, the resultant trajectory of the quadcopter on the EFF is shown.

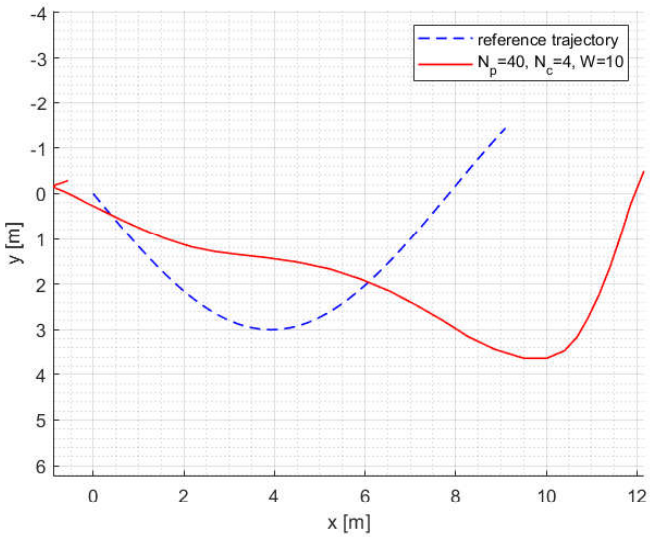

Figure 6.19. Test-1 Results for the Sinusoidal Reference Trajectory.

In Figure 6.20, it is observed that the aircraft started to follow the trajectory with a delay, even the maximum control inputs were produced. The control inputs are shown in Figure 6.21. As in the previous tests with the low W parameter, the control

effort is quite high.

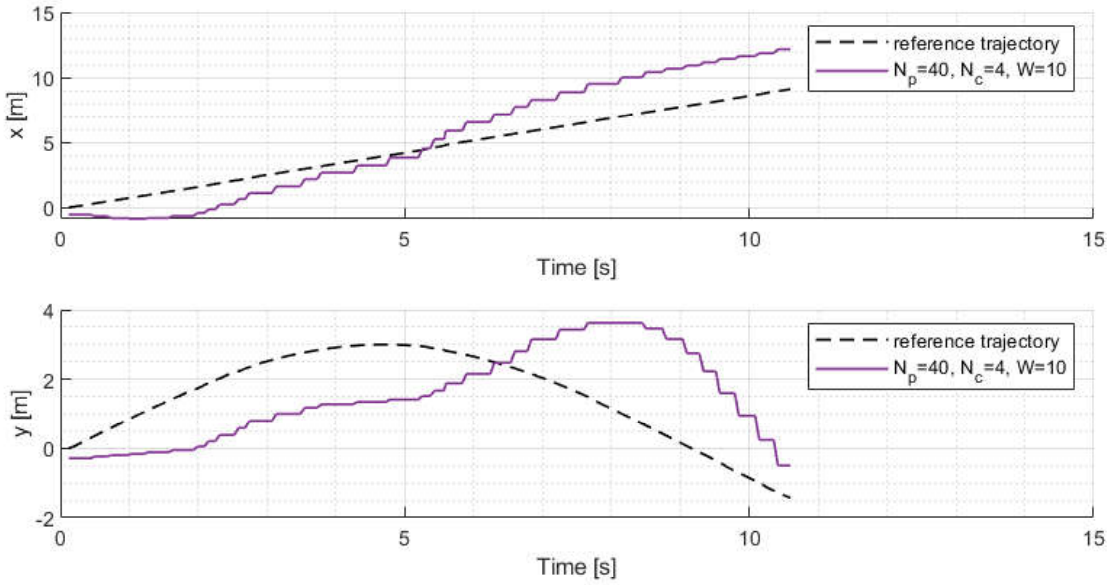

Figure 6.20. Test-1 Output Results for the Sinusoidal Reference Trajectory.

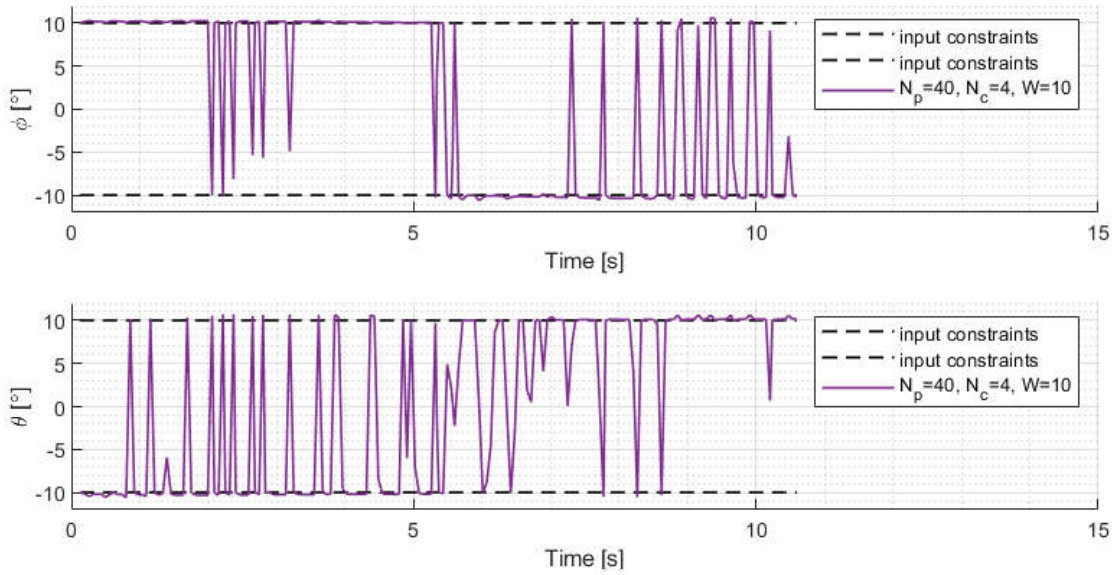

Figure 6.21. Test-1 Input Results for the Sinusoidal Reference Trajectory.

6.3.2. Test-2 with the Design Parameters $N_p = 40$, $N_c = 4$, and $W = 1000$

After Test-1 was performed, Test-2 was performed to see the effect of high W on the controller performance, while N_p and N_c kept constant. In Figure 6.22, the resultant trajectory of the

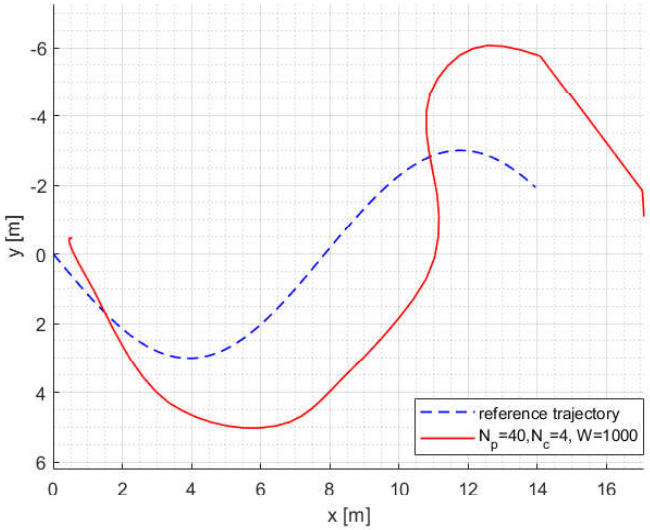

Figure 6.22. Test-2 Results for the Sinusoidal Reference Trajectory.

As it is shown in Figure 6.23, trajectory tracking result in the x-direction is sufficient. However, in the y-direction, a delay caused by the higher W is observed. In Figure 6.24, compared to the results with the low W parameters, it is observed that the control effort is reduced.

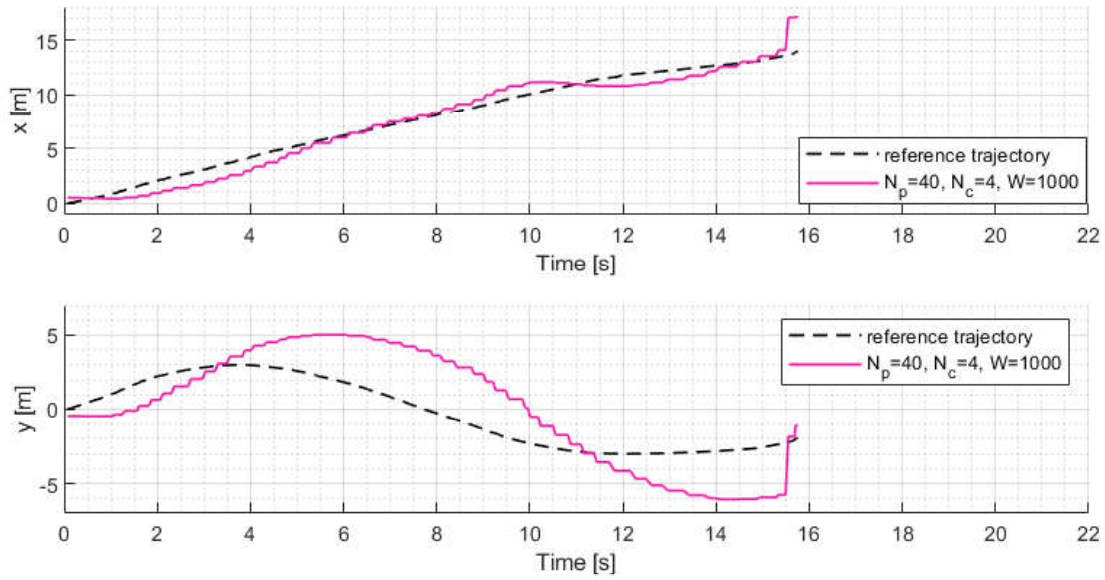

Figure 6.23. Test-2 Output Results for the Sinusoidal Reference Trajectory.

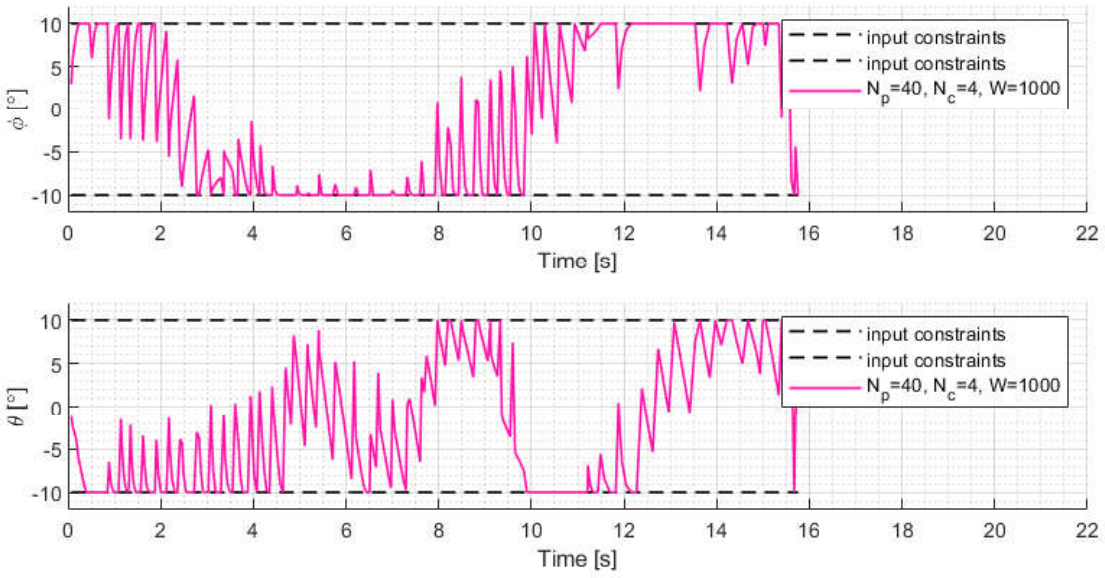

Figure 6.24. Test-2 Input Results for the Sinusoidal Reference Trajectory.

6.3.3. Test-3 with the Design Parameters $N_p = 40$, $N_c = 6$, and $W = 1000$

After Test-2 was performed, Test-3 was conducted to observe the effect of high N_c on the controller performance. In Figure 6.25, the resultant trajectory of the quadcopter on the EFF is shown.

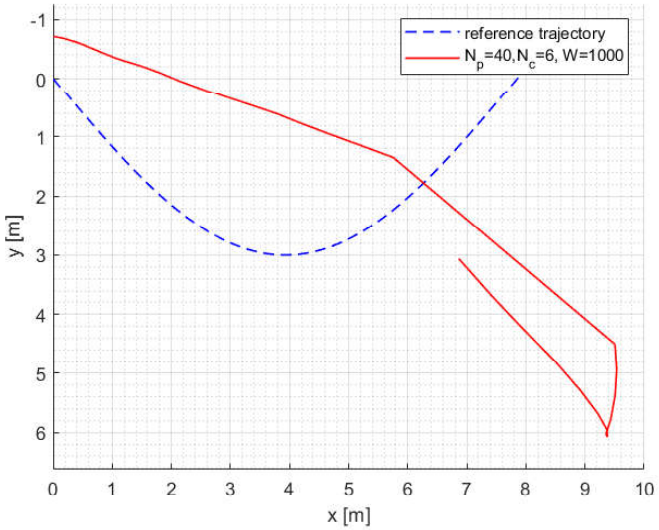

Figure 6.25. Test-3 Results for the Sinusoidal Reference Trajectory.

As observed from the results in Figure 6.26, there was a GPS interruption during the flight. Also in the y-direction, high trajectory error is observed. When the inputs in Figure 6.27 are evaluated, it is observed that although the quadcopter produces the highest θ angle it can produce, the y-direction reference cannot be followed.

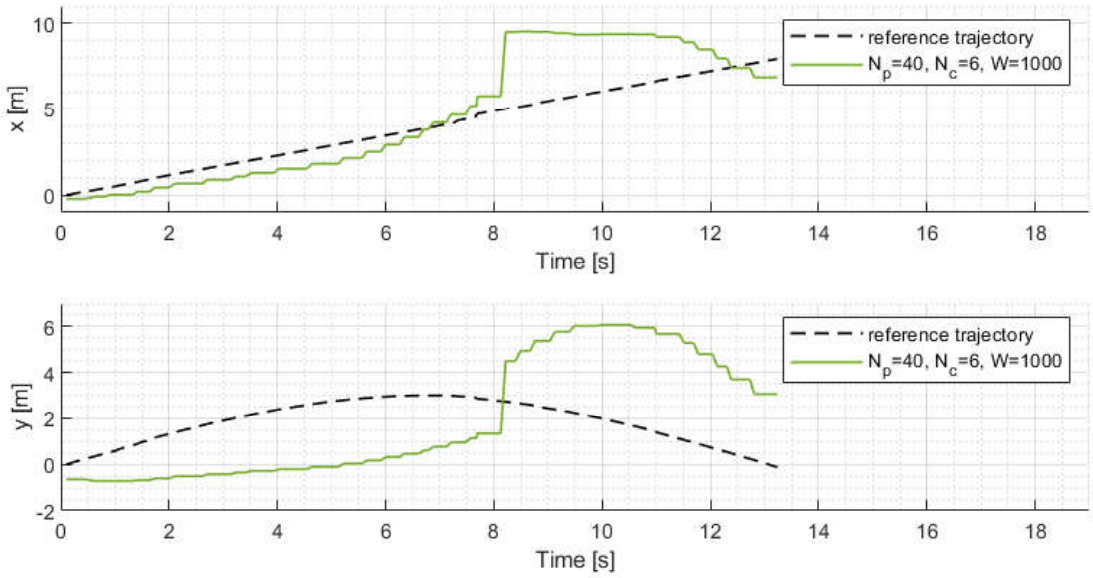

Figure 6.26. Test-3 Output Results for the Sinusoidal Reference Trajectory.

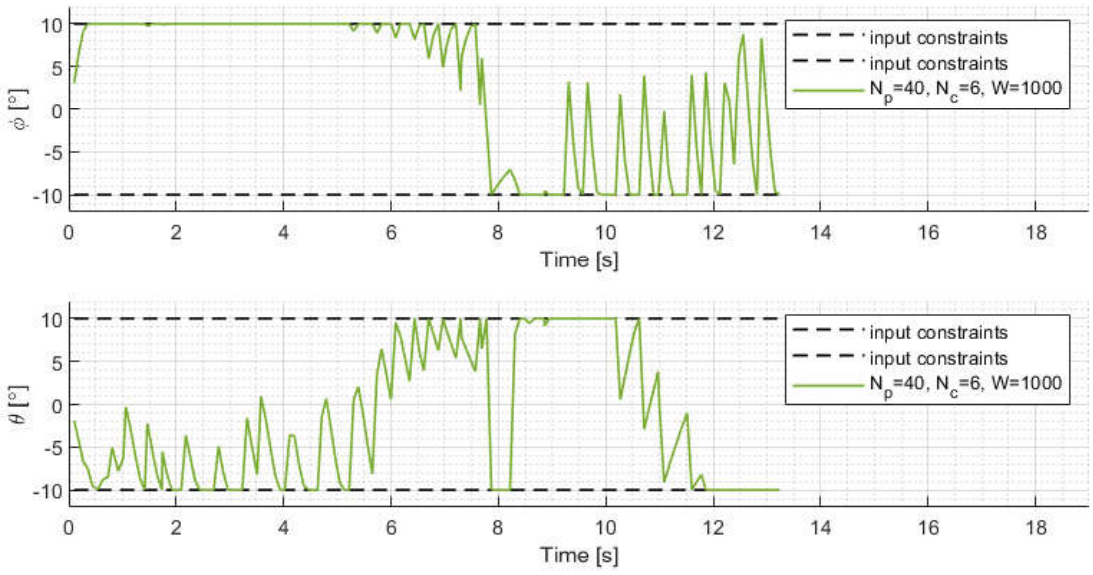

Figure 6.27. Test-3 Input Results for the Sinusoidal Reference Trajectory.

6.4. Circular Reference Trajectory

The circular reference trajectory, which is used in the simulation tests, was also used in the flight tests. The equation is rewritten as

$$r(t) = \begin{bmatrix} x_r(t) \\ y_r(t) \end{bmatrix} = \begin{bmatrix} 3 - 3cos(0.4t) \\ 3sin(0.4t) \end{bmatrix}. \tag{6.4}$$

6.4.1. Test-1 with the Design Parameters $N_p = 40$, $N_c = 4$, and $W = 10$

Test-1 was conducted to examine the controller performance when the design parameter W is low. In Figure 6.28, the resultant trajectory of the quadcopter on the EFF is shown.

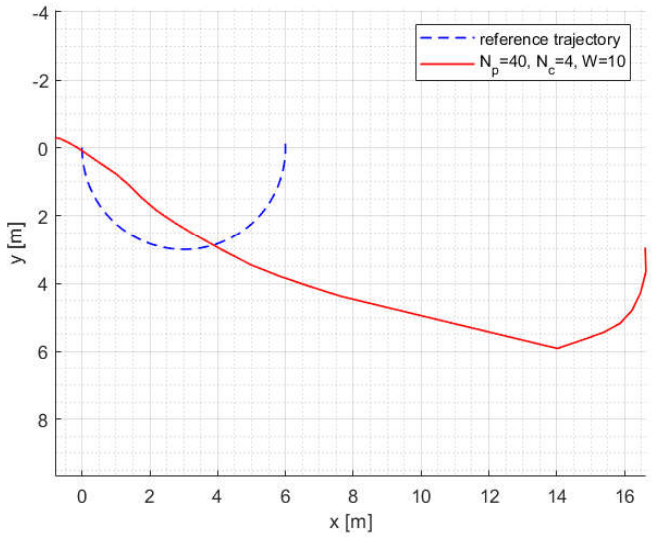

Figure 6.28. Test-1 Results for the Circular Reference Trajectory.

Looking at the position results in Figure 6.29, it is observed that there is a GPS interruption at the time $t = 7.5$. In addition, it is observed that although the controller produces maximum input value for ϕ, it cannot follow the trajectory in the y-direction. The reason for this is estimated to be windy weather, as in the previous tests. In Figure

6.30, the control inputs are shown. It is observed that, due to the low W parameter, the control effort is high.

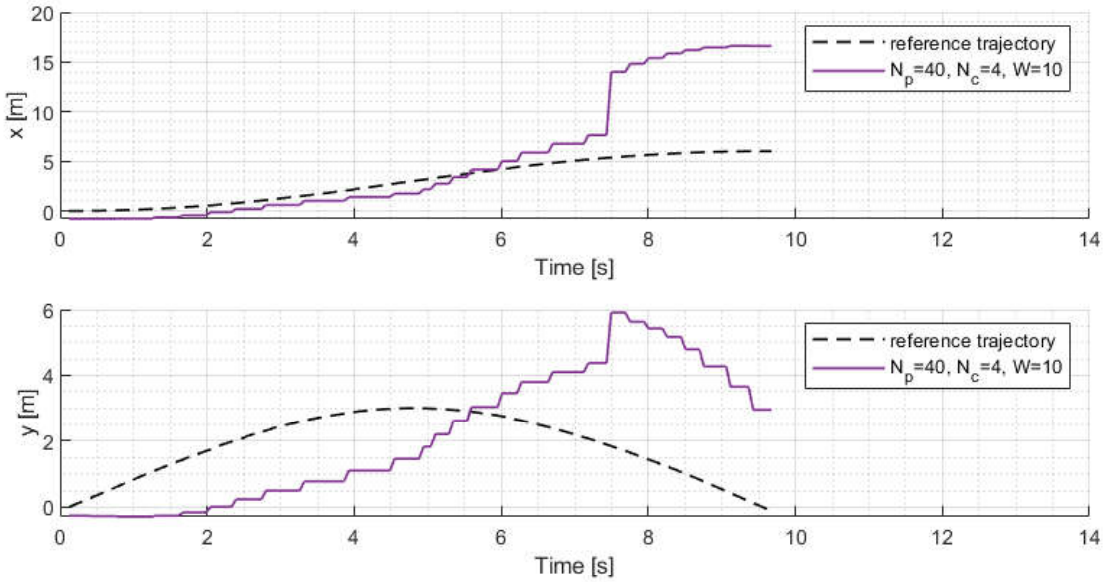

Figure 6.29. Test-1 Output Results for the Circular Reference Trajectory.

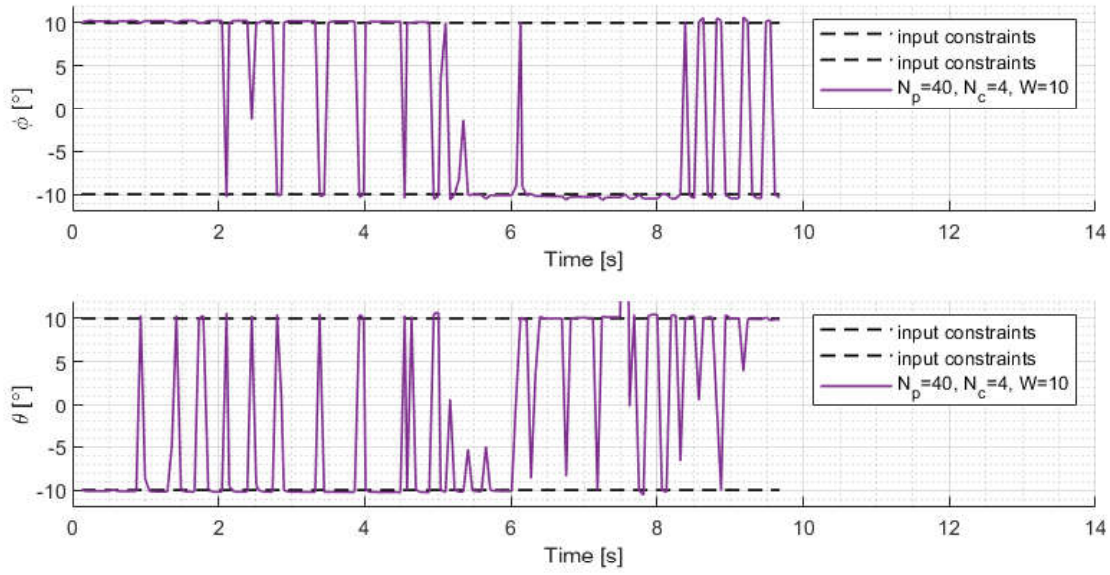

Figure 6.30. Test-1 Input Results for the Circular Reference Trajectory.

6.4.2. Test-2 with the Design Parameters $N_p = 40$, $N_c = 4$, and $W = 1000$

After Test-1 was performed, Test-2 was performed to see the effect of high W on the controller performance, while N_p and N_c kept constant. In Figure 6.28, the resultant trajectory of the quadcopter on the EFF is shown.

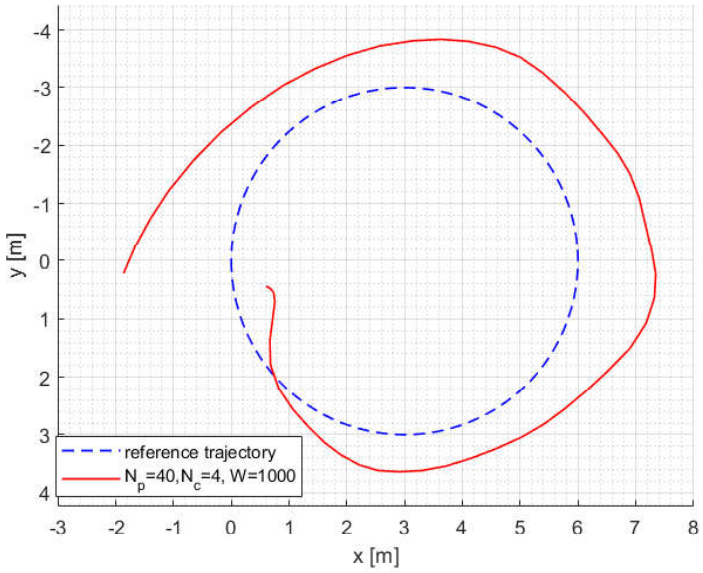

Figure 6.31. Test-2 Results for the Circular Reference Trajectory.

In Figure 6.32, the trajectory results are shown. It is observed that, the position errors are very low. In Figure 6.33, the inputs generated by the controller are shown. It is observed that, the increased W parameter reduces the control effort and causes smoother inputs to be produced. However, the chattering problem in the inputs still persists.

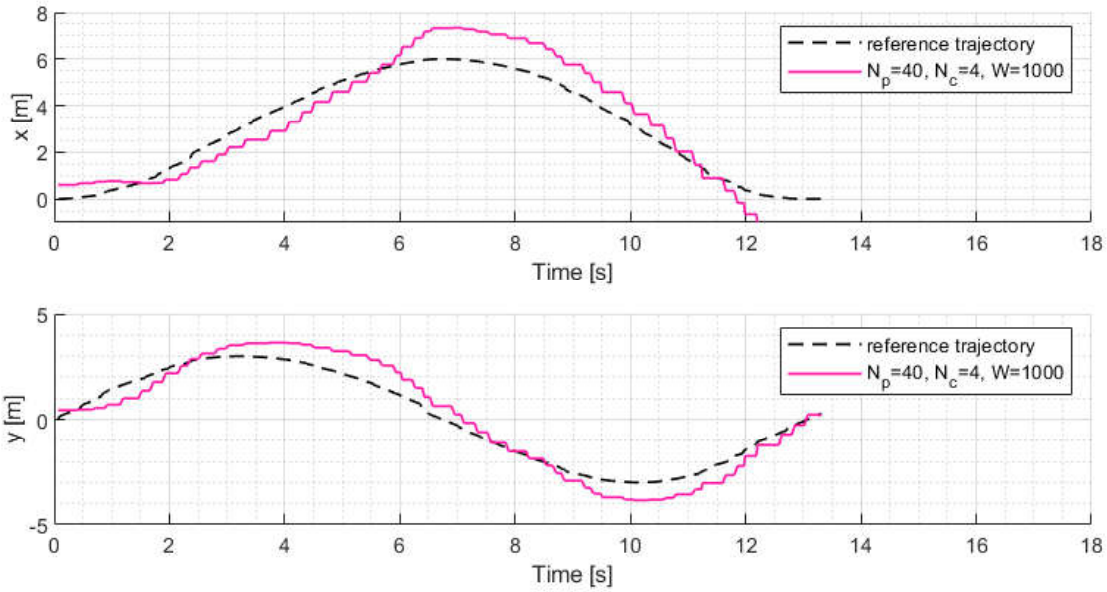

Figure 6.32. Test-2 Output Results for the Circular Reference Trajectory.

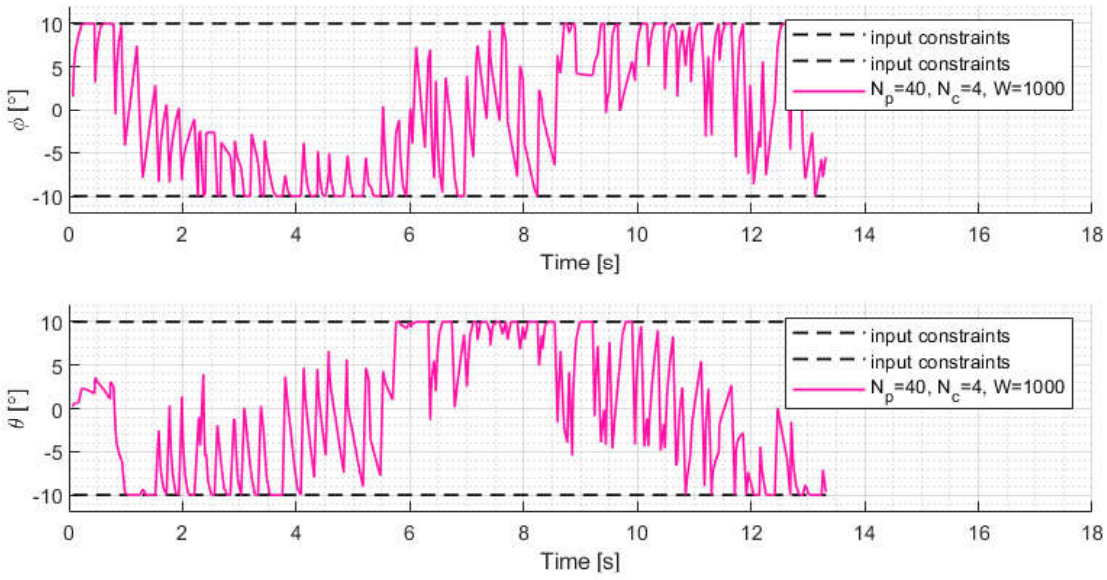

Figure 6.33. Test-2 Input Results for the Circular Reference Trajectory.

6.4.3. Test-3 with the Design Parameters $N_p = 40$, $N_c = 6$, and $W = 1000$

After Test-2 was performed, Test-3 was conducted to observe the effect of high N_c on the controller performance. In Figure 6.34, the resultant trajectory of the quadcopter on the EFF is shown.

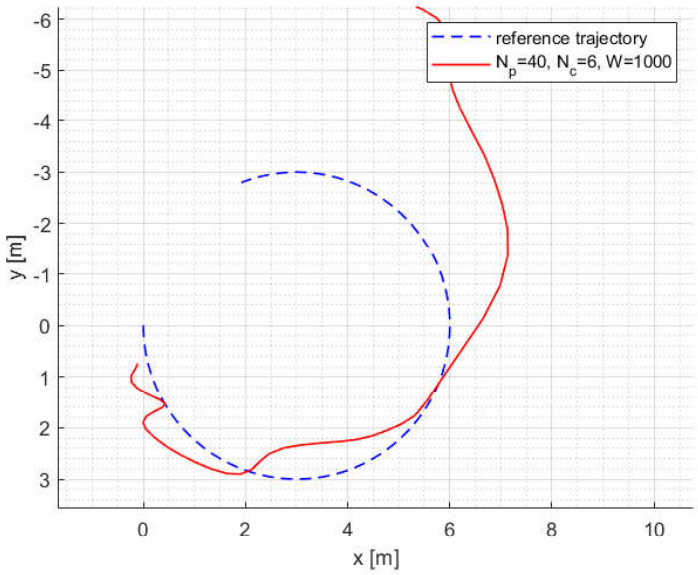

Figure 6.34. Test-3 Results for the Circular Reference Trajectory.

In Figure 6.29, the position results are shown. Compare to the results obtained with the low N_c parameter, increasing the N_c parameter provided trajectory tracking with less errors in the first 10 seconds. The reason for the position errors observed after the 10th second can be interpreted as a disturbance caused by the weather conditions. In Figure 6.36, the inputs produced by the MPC are shown.

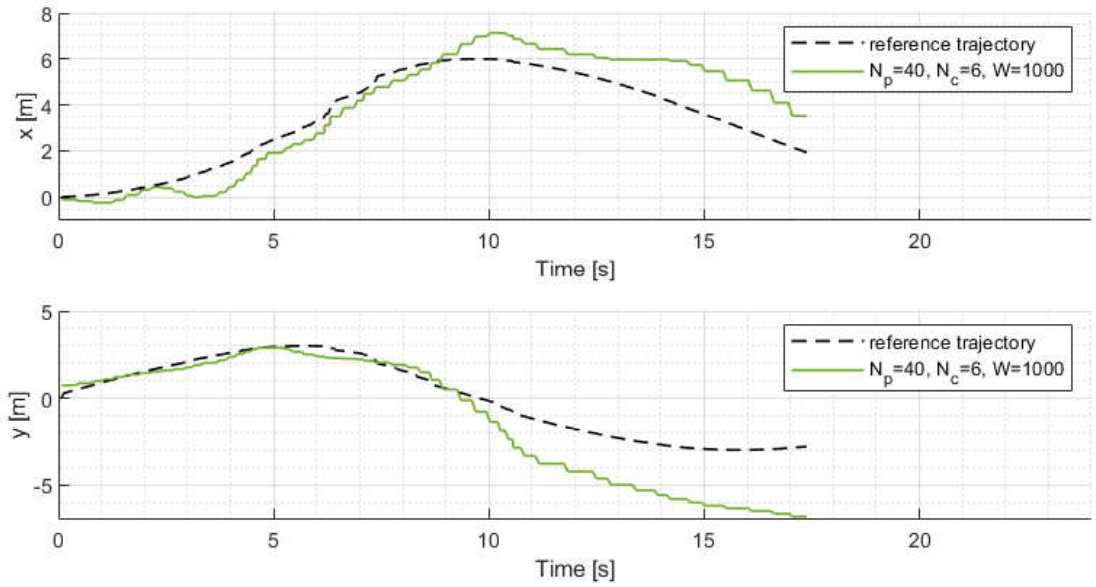

Figure 6.35. Test-3 Output Results for the Circular Reference Trajectory.

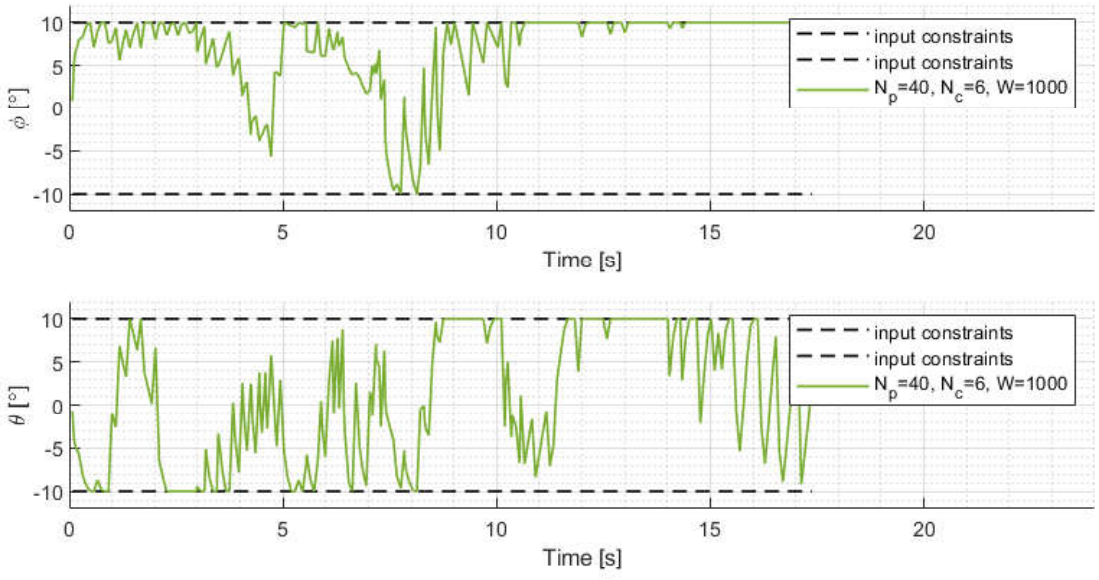

Figure 6.36. Test-3 Input Results for the Circular Reference Trajectory.

6.5. Discussion

As a result, the flight tests were conducted for four different reference trajectories, from simple to complex. They are reference waypoint, longitudinal reference trajectory, sinusoidal reference trajectory, and circular reference trajectory. Three different tests were repeated to see the effect of a different MPC design parameters on the controller performance for each reference trajectory. They are Test-1 with the design parameters N_p as 40, N_c as 4, and W as 10, Test-2 with the design parameters N_p as 40, N_c as 4, and W as 1000, and Test-3 with the design parameters N_p as 40, N_c as 6, and W as 1000. The test results were evaluated by considering GPS interruptions and windy weather conditions.

As observed in the simulation results, it was observed in the test results Test-1 and Test-2 that increasing the input weight (W) reduces the overshoot in the output results and better trajectory tracking performance. The value of the W is directly related to the units of the terms used in the cost function. In this study, trajectory errors with units of meters and the inputs with radians are used in the cost function. For this reason, when the W value was 10, the controller applied a very high control effort to follow the reference trajectory. When the W value was increased to 1000, softer control inputs were sent to the system slightly. In the flight tests, smooth control inputs gave better trajectory tracking performance.

Increasing the number of control horizon (N_c) caused similar results regarding the controller performance in the simulation tests. N_c value was increased in Test-3, considering that it might reduced chattering in the input results observed in the flight tests Test-1 and Test-2. Increasing N_c increased the optimization solution time. Although it did not solve the chattering problem, it has been observed that the wavelength of the noise was reduced.

7. HARDWARE DESIGN

The quadcopter is a mechatronic system in which multiple electronic devices work together. The quadcopter consists of sensors, actuators, and physical components, also called hardware components. In this chapter, the hardware components of the quadcopter platform and quadcopter's performance calculations required to reach the flight objective of the thesis are presented. First, a component list and major systems with related components will be presented. Then, performance calculations performed for component selection will be explained.

7.1. Hardware Implementation

The quadcopter was created by considering the requirements of this study. The components which are used in the quadcopter system are listed in Table 7.1. In this section, hardware components are going to be explained as components and systems that include more than one component such as propulsion system, power supply system and avionics. A diagram shown in Figure 7.1 created to represent hardware components and their connection.

7.1.1. Quadcopter Frame

Quadcopter frames are generally made by plastic or carbon fiber materials. While selecting the frame, first of all the dimensions of the frame should be decided by considering the hardware components to be transported on quadcopter. Especially, it is important to locate selected propellers on the quadcopter without hitting any other components. Therefore, by taking into account the performance calculations causes selection of propeller dimension, a frame which has distance of 450 mm between the mutually positioned electric motor was selected. The calculations cause propeller selection will be explained in the following sections.

Table 7.1. Hardware Component List.

	Hardware Components
1	**Quadcopter Frame**
2	**Propulsion System**
2.1	Propellers
2.2	Electric Motors
2.3	ESCs
3	**Pixhawk**
4	**Raspberry Pi**
5	**Power Supply System**
5.1	Battery
5.2	Voltage Regulators
6	**GPS**
7	**Avionics**
7.1	Telemetry
7.2	RC Receiver and Transmitter
7.3	Buzzer and Safety Switch

7.1.2. Propulsion System

The propulsion system consists of propellers, electric motors, and electric speed controllers (ESC). The propulsion system must produce thrust and provide motion in quadcopter systems. Because the only inputs of the quadcopter dynamic systems are thrusts, selecting the propulsion system components is important. During the selection, the total weight of the quadcopter and the propulsion system's capacity must be compatible.

Electric motors are brushless, and they generally have specifications provided by the producers, including appropriate battery and propeller choices for the engine type. In this case, T-motor U3 electric motor was selected due to its enough thrust generation capability for light quadcopters. It can be used with both 4S and 3S batteries and

propeller pairs with different sizes.

Propellers are made of carbon fiber or plastic materials and are generally preferred with two blades in quadcopter applications. Propellers are designed with an airfoil cross-section to generate aerodynamic lift and drag forces with the airflow. The aerodynamic lift and drag forces cause a thrust and moment in the propeller's axis. The T-motor carbon fiber propellers with the dimension of 12x4 inches was chosen for this application. The reason behind that selection will be explained in detail in the performance calculations section.

An electric speed controller (ESC) is a circuit that controls and regulates the speed of the electric motor. ESCs are generally rated according to the maximum current that they allow. T-motor AIR 40A ESC which can withstand 40A continuously was preferred in this case due to its high maximum current capacity, 60A. In order to understand whether the maximum current capacity of ESC is enough for the selected electric motor and propeller pairs, specifications of the electric motor are used.

7.1.3. Pixhawk

Pixhawk is a board that is commonly used to run open-source flight controllers: Ardupilot and PX4. Pixhawk has the advantage of its straightforward design to connect avionic devices such as GPS, telemetry, radio receiver, buzzer, and safety switch. This thesis, it is aimed to run ArduPilot open-source flight controller code as a low-level controller on the Pixhawk board. Pixhawk board has its inertial measurement unit (IMU) with an accelerometer, gyroscope, and magnetometer for the yaw angle. Pixhawk board can be used for different UAVs, such as fixed wings or rotary wings including tricopter, hexacopter and octocopter designs in addition to quadcopter. In the trajectory tracking application, actual position references are estimated by using the GPS connected on the Pixhawk.

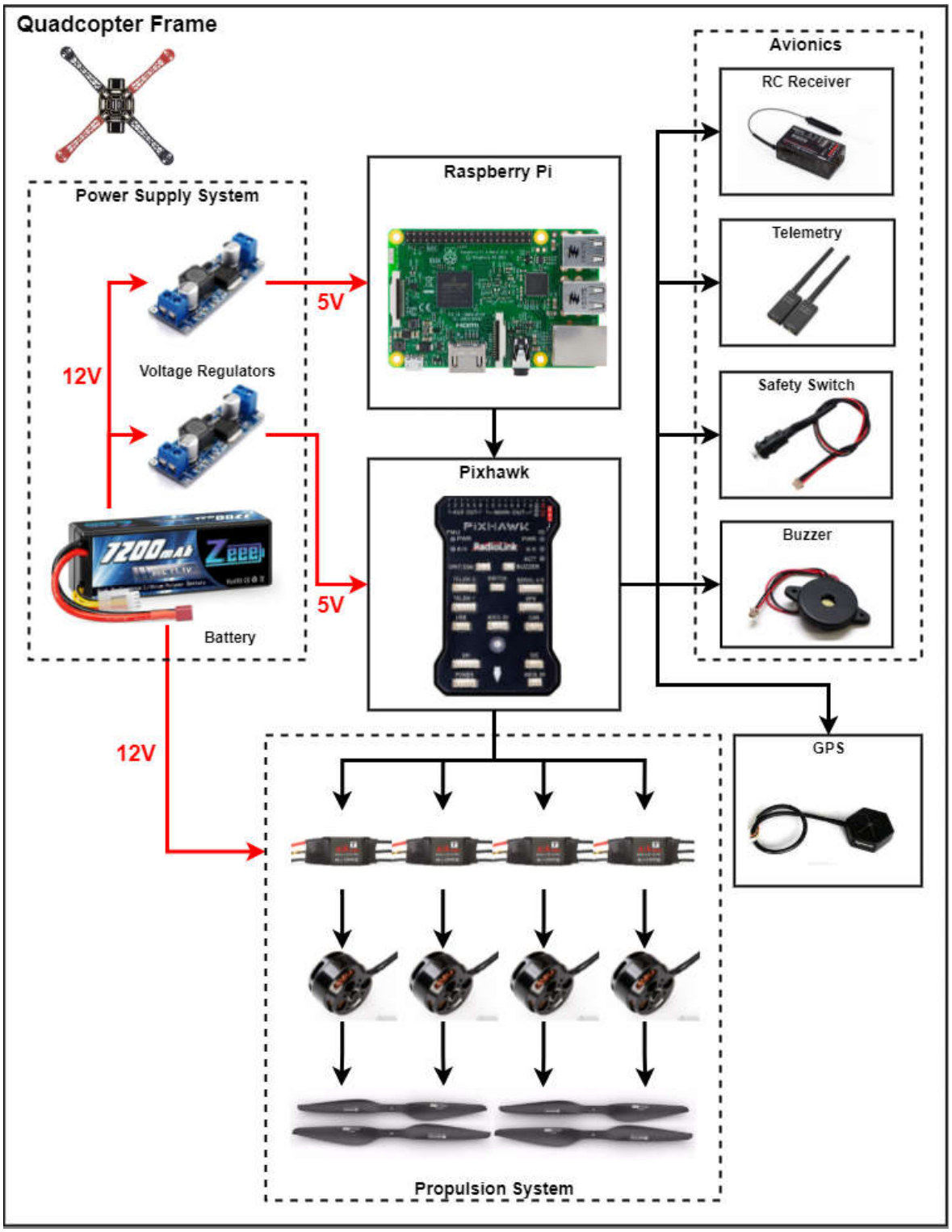

Figure 7.1. Hardware Component's Diagram.

7.1.4. Raspberry Pi

A Raspberry Pi is a small computer that can be used as flight controllers of UAVs. This thesis aims to run a high-level controller on the Raspberry Pi board and construct the serial communication between Raspberry Pi and Pixhawk boards. The communication is supplied with USB to micro USB cable. Because the high-level flight controller requires position information to produce attitude angles as control inputs, Raspberry Pi has to read data from sensors connected on Pixhawk and send produced control inputs to the Pixhawk. In this application, the code generated for MPC was written in Python language and run in the Raspberry Pi.

7.1.5. Power Supply System

Lithium-polymer (LiPo) batteries provide the power supply of the quadcopter system. They are rechargeable batteries of lithium-ion technology using a polymer electrolyte instead of a liquid electrolyte. They separated according to the number of parallel and serial cells. A single cell voltage varies between 4.2 V means fully charged, and 2.7 V means fully discharged. Due to electric motor voltage requirements, 3S1P (11.1V) and 4S1P (14.8V) types are usually preferred for quadcopter applications. In this case 3S1P (11.1V) Leopard LiPo battery was selected. The boards on the quadcopter, such as Pixhawk and Raspberry Pi, require a 5V power supply to operate. For this reason, voltage regulators which convert high voltage to the desired voltage level are required to use in power supply systems.

7.1.6. Global Positioning System (GPS)

A global positioning system (GPS) detects the position in latitude-longitude-altitude positioning format. GPS's working principle is based on a worldwide network of satellites that transmit radio signals from medium earth orbit. In this case, a RadioLink GPS with about 50 centimeters positioning accuracy and can be searched 20 satellites within a few seconds is used. Due to the RadioLink GPS being compatible

with the Pixhawk board and powered by Pixhawk, it is easy to use.

7.1.7. Avionics

Telemetry is a device that provides a two-way data stream. It can supply the communication between a ground station and a UAV. In this case, QGround Control is used as a ground station. QGround Control is flight support for the vehicles running with PX4 or ArduPilot autopilots via MAVLink protocol.

Remote control (RC) receiver and transmitter are used to control UAV remotely. The receiver is connected to the UAV while the transmitter is held in the pilot's hands. RC is important even in autonomous flight because if something goes wrong, it is necessary to take the UAV as one of the manual modes in ArduPilot algorithm.

Buzzer and safety switch are compatible with the Pixhawk board, and they are used to make Pixhawk's responses understandable. The buzzer receives audible warnings, while the safety switch allows electric motor operations. Both of them are powered by the Pixhawk board.

7.2. Component Sizing

Performance calculations are done to design a quadcopter that meets the flight requirements to reach the objective of the thesis. This section consists of the weight calculation directly related to the battery-propeller pair selection and flight time calculations related to the battery capacity selection.

7.2.1. Weight Calculation and Battery-Propeller Selection

During the design process, it is expected from the quadcopter to hover around 50% of the maximum throttle so it can still have extra throttle for maneuverability. Because the battery and propeller pairs determine the ultimate thrust that the quad-

copter can have and the total weight that it can be, weight calculation is vital during the design of the quadcopter. With the components mentioned in Table 7.1, the quadcopter is expected to weigh nearly 1.650 kg. In Table 7.2, many propellers and battery pairs are investigated in terms of the thrust that they can generate with a 50% throttle level. Propeller dimensions, battery types, and thrust values are taken from electric motor T-Motor U3 specifications.

Table 7.2. Battery-Propeller Selection.

Propeller Type	**Battery Type**	**Thrust (%50 Throttle) [g]**	**Total Weight [kg]**
12x4 CF	**3S LiPo**	**350**	**1.4**
13x4.4 CF	**3S LiPo**	400	1.6
14x4.8 CF	**3S LiPo**	550	2.2
11x3.7 CF	**4S LiPo**	460	1.84
12x4 CF	**4S LiPo**	580	2.32
13x4.4 CF	**4S LiPo**	730	2.92

As it can be seen from Table 7.2, it is proper to use 12x4 CF and 13x4.4 propellers with a 3S1P battery and 11x3.7 CF propellers with 4S1P battery. In this thesis, 3S1P battery and 12x4 CF propeller pairs were selected. After the propeller pairs and the battery type is decided, the flight time of the quadcopter can be calculated.

7.2.2. Flight Time Calculations

Flight time calculation depends on the total current used by the hardware components and the battery's capacity. Therefore, one should estimate the total required current to calculate flight time. Electric motor specifications can be used to estimate the required current. The flight time of the quadcopter is calculated as

$$t = \frac{Ca \times 60}{1000 \times Cu} \tag{7.1}$$

where t represents the flight time in $[m]$, Ca represents the battery capacity in $[mAh]$ and Cu represents the total current used by quadcopter in $[A]$. The previous section selected proper battery-propeller pairs for the design weight. With the knowledge of this selection, the required current estimation can be done using the electric motor specifications. On the other hand, battery capacity is added to the table to show the effect of battery capacity on flight time.

Table 7.3. Flight Time Calculations.

Propeller Type	**Battery Type**	**Current [A] (%50 Throttle)**	**Total Current [A] (with +2A)**	**Capacity [mAh]**	**Flight Time [min]**
12x4 CF	**3S LiPo**	2.5	12	3000	15
				5000	25
				7000	35

Finally, it can be observed from the Table 7.3 that choosing the highest capacity causes the highest flight time.

8. CONCLUSION

The quadcopter UAVs are prevalent electromechanical systems due to their advantages in implementing and testing different controller algorithms. This thesis obtains trajectory tracking of a quadcopter by using a linear model predictive controller. MPC is a newly popular method in aviation. It has a preview capacity because it solves an online optimization problem depending on the plant model.

Throughout the thesis, the first nonlinear dynamic model of the quadcopter was derived using Newton-Euler's formalism. Then, the model predictive algorithm was formulated and designed for the trajectory tracking dynamics. From simple to complex, four different reference trajectories were generated to use in simulation tests and actual flight tests. After the controller design, the system was simulated in the MATLAB environment. Some of the design parameters of MPC were determined depending on the physical limitations of the system or calculations. In contrast, the others were selected depending on the simulation tests. The quadcopter system, including the Pixhawk flight computer for the low-level controller and Raspberry Pi flight computer for the designed MPC, was created to validate the simulation results with the actual flight tests. In order to build a working system, required calculations, including the flight time calculations and thrust calculations needed for hover depending on the electric motor and propeller pairs, were done.

The results show that linear MPC can be used in trajectory tracking applications by selecting the proper design parameters. However, due to the model of the linear MPC being highly linearized and decoupled, linear MPC lost some of its preview capacity. For that reason, nonlinear MPC might be considered and tested in future works. Also, while designing the MPC for this thesis, the low-level dynamics of the system were not considered. It is known that ArduPilot's low-level algorithm consists of PID controllers which have their delay problems. Therefore, it might be beneficial to model the low-level dynamics and use them in the control-oriented model while

designing the linear MPC controller. Considering the parts used for hardware, it is obvious that selected GPS is quite inadequate for this application. Real-time kinematic GPS (RTK) system, that provide reliable positioning, can be used to do longer tests by eliminating the GPS drift problem.

REFERENCES

1. Bouabdallah, S. P. T., *Design and Control of Quadrotors with Application to Autonomous Flying*, Doctorate of engineering, Ecole Poltechnique Federale De Lausanne, 2007.

2. Li, J. and Y. Li, "Dynamic Analysis and PID Control for a Quadrotor", *2011 IEEE International Conference on Mechatronics and Automation, ICMA*, pp. 573–578, Beijing, China, 2011.

3. Khan, H. S. and M. B. Kadri, "Position Control of Quadrotor by Embedded PID Control with Hardware in Loop Simulation", *17th IEEE International Multi Topic Conference: Collaborative and Sustainable Development of Technologies, IEEE INMIC - Proceedings*, pp. 395–400, Pakistan, 2015.

4. He, Z. and L. Zhao, "A Simple Attitude Control of Quadrotor Helicopter Based on Ziegler-Nichols Rules for Tuning PD Parameters", *Scientific World Journal*, Vol. 2014, 2014.

5. Lee, K. U., H. S. Kim, J. B. Park and Y. H. Choi, "Hovering Control of a Quadrotor", *International Conference on Control, Automation and Systems*, pp. 162–167, Jeju Island, Korea, 2012.

6. Bao, N., X. Ran, Z. Wu, Y. Xue and K. Wang, "Research on Attitude Controller of Quadcopter Based on Cascade PID Control Algorithm", *Proceedings of the 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference, ITNEC*, Vol. 2018-Janua, pp. 1493–1497, 2018.

7. Bo, G., L. Xin, Z. Hui and W. Ling, "Quadrotor Helicopter Attitude Control Using Cascade PID", *Proceedings of the 28th Chinese Control and Decision Conference, CCDC*, pp. 5158–5163, 2016.

8. Wang, P., Z. Man, Z. Cao, J. Zheng and Y. Zhao, "Dynamics Modelling and Linear Control of Quadcopter", *International Conference on Advanced Mechatronic Systems, ICAMechS*, Vol. 0, Melbourne, Australia, 2016.

9. Szafranski, G. and R. Czyba, "Different Approaches of PID Control UAV Type Quadrotor", *Proceedings of the International Micro Air Vehicles Conference, Summer Edition*, pp. 70–75, Gliwice, Poland, 2011.

10. Ma, J. and R. Ji, "Fuzzy PID for Quadrotor Space Fixed-Point Position Control", *2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control, IMCCC*, pp. 721–726, Heilongjiang, jul 2016.

11. Gautam, D. and C. Ha, "Control of a Quadrotor Using a Smart Self-Tuning Fuzzy PID Controller", *International Journal of Advanced Robotic Systems*, Vol. 10, pp. 1–9, 2013.

12. Ahmad, F., P. Kumar, A. Bhandari and P. P. Patil, "Simulation of the Quadcopter Dynamics with LQR Based Control", *Materials Today: Proceedings*, Vol. 24, pp. 326–332, Elsevier Ltd., India, 2020.

13. Okyere, E., A. Bousbaine, G. T. Poyi, A. K. Joseph and J. M. Andrade, "LQR Controller Design for Quad[U+2010]rotor Helicopters", *The Journal of Engineering*, Vol. 2019, No. 17, pp. 4003–4007, 2019.

14. Setyawan, G. E., W. Kurniawan and A. C. L. Gaol, "Linear Quadratic Regulator Controller (LQR) for AR. Drone's Safe Landing", *Proceedings of 2019 4th International Conference on Sustainable Information Engineering and Technology, SIET*, July 2020, pp. 228–233, Indonesia, 2019.

15. Reyes-Valeria, E., R. Enriquez-Caaldera, S. Camacho-Lara and J. Guichard, "LQR Control for a Quadrotor Using Unit Quaternions: Modeling and Simulation", *Inter-*

national Conference on Electronics, Communication and Computing, pp. 172–178, Pueblo, Mexico, 2013.

16. Khatoon, S., D. Gupta and L. K. Das, "PID & LQR Control for a Quadrotor: Modeling and Simulation", *Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics, ICACCI*, pp. 796–802, 2014.

17. Liu, C., J. Pan and Y. Chang, "PID and LQR Trajectory Tracking Control for an Unmanned Quadrotor Helicopter: Experimental Studies", *Chinese Control Conference, CCC*, Vol. 2016-Augus, pp. 10845–10850, 2016.

18. Islam, M., M. Okasha and M. Mohammad Idres, "Trajectory Tracking in Quadrotor Platform by Using PD Controller and LQR Control Approach", *IOP Conference Series: Materials Science and Engineering*, Vol. 260, IOP, Kuala Lumpur, Malaysia, 2017.

19. Argentim, L. M., W. C. Rezende, P. E. Santos and R. A. Aguiar, "PID, LQR and LQR-PID on a Quadcopter Platform", *2013 International Conference on Informatics, Electronics and Vision, ICIEV*, 2013.

20. Martins, L., C. Cardeira and P. Oliveira, "Feedback Linearization with Zero Dynamics Stabilization for Quadrotor Control", *Journal of Intelligent and Robotic Systems: Theory and Applications*, Vol. 101, No. 1, 2021.

21. Zhao, W. and T. H. Go, "Quadcopter Formation Flight Control Combining MPC and Robust Feedback Linearization", *Journal of the Franklin Institute*, Vol. 351, No. 3, pp. 1335–1355, 2014.

22. Vaidyanathan, S. and A. T. Azar, *An Introduction to Backstepping Control*, Elsevier Inc., 2021.

23. Adiguzel, F. and T. V. Mumcu, "Discrete-Time Backstepping Attitude Control of

a Quadrotor UAV", *2019 International Conference on Artificial Intelligence and Data Processing Symposium, IDAP*, February, İstanbul,Turkey, 2019.

24. Rashad, R., A. Aboudonia and A. El-Badawy, "Backstepping Trajectory Tracking Control of a Quadrotor with Disturbance Rejection", *2015 25th International Conference on Information, Communication and Automation Technologies, ICAT - Proceedings*, Bosnia, 2015.

25. Gros, S., M. Zanon, R. Quirynen, A. Bemporad and M. Diehl, "From Linear to Nonlinear MPC: Bridging the Gap via the Real-time Iteration", *International Journal of Control*, Vol. 93, No. 1, pp. 62–80, 2020.

26. Kamel, M., M. Burri and R. Siegwart, "Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles", *IFAC-PapersOnLine*, Vol. 50, pp. 3463–3469, Elsevier, Zürich, Switzerland, 2017.

27. Bemporad, A. and C. Rocchi, "Decentralized Linear Time-Varying Model Predictive Control of a Formation of Unmanned Aerial Vehicle", *IFAC Proceedings Volumes (IFAC-PapersOnline)*, Vol. 44, pp. 11900–11906, Orlando, USA, 2011.

28. Islam, M., M. Okasha and M. M. Idres, "Dynamics and Control of Quadcopter Using Linear Model Predictive Control Approach", *IOP Conference Series: Materials Science and Engineering*, Vol. 270, IOP, 2017.

29. Ganga, G. and M. M. Dharmana, "MPC Controller for Trajectory Tracking Control of Quadcopter", *2017 International Conference on Circuit ,Power and Computing Technologies, ICCPCT*, India, 2017.

30. Chen, X. and L. Wang, "Cascaded Model Predictive Control of a Quadrotor UAV", *2013 3rd Australian Control Conference, AUCC*, pp. 354–359, Engineers Australia, Pert, Australia, 2013.

31. Amadi, C. A., *Design and Implementation of Model Predictive Control on Pixhawk*

Flight Controller, Master thesis, Stellenbosch University, 2018.

32. Wang, L., *Model Predictive Control System Design and Implementation Using MATLAB*, Springer, United Kingdom, 2009.

APPENDIX A: PYTHON CODE

Algorithm 1 MPC Algorithm.

Call the required libraries.

Connect to the vehicle with DroneKit library.

Define the number of states, inputs and outputs:

$N_s = 4$, $N_i = 2$, $N_o = 2$.

Define MPC design parameters:

$N_p = 40$, $N_c = 4$, $W = 1000$, $T_s = 0.05$, $u_{max} = +10$, $u_{min} =$ -10, $\Delta u_{max} = 100$.

function CONTINUOUS TO DISCRETE(A_m, B_m, C_m, T_s)

Use Euler's Method for discretization.

return A_d, B_d, C_d

end function

function CREATE AUGMENTED MATRICES(A_d, B_d, C_d, N_s, N_o)

Create augmented matrices considering the new state vector $x_a = [\Delta x_m^T \; y_m^T]^T$.

return A_a, B_a, C_a

end function

function CALCULATE PREDICTION OUTPUTS($A_a, B_a, C_a, N_p, N_c, N_y, N_u$)

Calculate the P and H matrices used to describe predicted outputs, Y.

$Y = Px_a + H\Delta u_a$

return Y

end function

function DEFINE CONSTRAINT VECTORS($u_{max}, u_{min}, \Delta u_{max}, N_u, N_c$)

return CC, d

end function

$E = -2(H'H + W)$

---

function DEFINE REFERENCE TRAJECTORY

The reference trajectory is predefined until the N_{sim}.

for $t = 0 : N_{sim}$ **do**

$$R(t) = \begin{bmatrix} x_r(t) \\ y_r(t) \end{bmatrix}$$

end for

return R

end function

Autonomous takeoff with the DroneKit library.

Start the trajectory tracking with MPC.

for $t = 0 : N_{sim}$ **do**

for $k = 1 : n_y$ **do**

Create the reference trajectory vector for current shifted optimization window.

$R_s(:) = R(1 : n_y)$

end for

$F = -2H'(R_s - Px_a(t))$

Update CC and d by using $\Delta u_a(t)$.

function QP SOLVER(E,F,CC,d)

return $\Delta u_a(t)$

end function

Update $u_a(t + \delta t) = u_a(t) + \Delta u_a(t)$

Send attitude commands to the Pixhawk with DroneKit library.

Take local position and velocity information from GPS in EFF.

Use the past and current GPS data to update $x_a(t)$.

end for

Autonomous landing with the DroneKit library.

Close the vehicle with Dronekit Library.
