Treball de Fi de Grau

Master's degree in Automatic Control and Robotics

Model Predictive Controller of a UAV using the LPV approach

MEMÒRIA

Autor: Mark Misin

Director: Dr. Vicenç Puig Cayuela

Convocatòria: January 2020

ETSEIB

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

UPC

Abstract

The Unmanned Air Vehicles (UAVs) are being integrated into our society at an increasing rate. The amount of industries that use drones is getting larger every year. Besides the military sector, which was probably the first adopter of drone technology, they are now also being used in the industries such as search and rescue, delivery service, media, civil engineering, etc.

In fact, airlines now use drones to perform inspections on aircraft. They fly around airplanes in search of cracks and deformations in the structure. They can also perform such inspections inside an airplane wing, in which fuel is stored. That means that the drone is given a trajectory to follow. That path can be generated live from the cameras on board or it can be a programmed track. However, the fact is that there is a path that the drone receives and so it has to regulate its actuators (rotors) in such a way that the drone follows the trajectory. This is what the thesis is about - to design and implement a controller in MATLAB® that makes the UAV follow the coordinates given to it.

The main control strategy used in this thesis will be Model Predictive Control (MPC) that is applied to a drone's mathematical model in the Linear Parameter Varying (LPV) format. Thanks to this format, it will be possible to apply the most basic MPC strategy, which is suitable for linear systems. Firstly, an attempt is made to control the UAV with a single LPV-MPC controller; however, it will be apparent in the thesis that due to strong nonlinearities, the drone was not able to follow the reference coordinates. Therefore, the controller was separated into two separate controllers. The LPV-MPC strategy was used to control the attitude of the UAV and the feedback linearization strategy was used to control the position of the drone.

The validation of the control strategy was performed in MATLAB® in the form of several simulations. Five different tracks were given for the drone to follow. It was then examined how well the UAV followed the given positions, velocities and angles.

Acknowledgements

During the creation of this thesis, I have been stuck several times. Luckily, I have always been able to freely approach Prof. Vicenç Puig Cayuela and ask him for directions. He has been extremely helpful during the entire process. He always replied to my doubts quickly, and in a very helpful and friendly manner. For that, I would like to thank Prof. Vicenç Puig Cayuela for his cooperation. It was a pleasure to work with him.

I would also like to thank Carlos Trapiello Fernández. His master thesis on the UAVs provided me with mathematical models of a drone based on which I could then design an LPV-MPC controller.

Contents

ETSEIB

List of Figures

Chapter 1

Introduction

1.1 Motivation

As we enter into the decade in which drone technology enters into more and more industries, it is imperative that they are reliable in terms of flight. They must be stable enough to withstand disturbances such as wind. They must be robustly safe to minimize the risk of accidents in the human population. Also, they need to be able to track the trajectories given to them with very high precision if it is desired to use them in tight areas such caves in the event of a search and rescue mission.

The main motivation of this thesis is is to contribute in making drones to follow trajectories smoother and with higher precision. That in turn will have a positive impact on the society in terms of various applications such as the inspection of structures from the inside in confined spaces. A drone can follow a trajectory even with a simple Proportional, Integral, Derivative (PID) controller. However, a PID controller is only capable of seeing one sample time ahead. That can make the UAV underdamped resulting it to oscillate in the air dangerously. Therefore, to achieve low error trajectory tracking and smoother flight in sharp turns, more advanced control techniques with higher horizon period should be experimented with. In this thesis, the main attention will be on applying the LPV-MPC controller to a drone mathematical model. The UAV in this thesis is a quadcopter - a drone with four rotors, that are at equal distances from the center of the drones.

1.2 Thesis Objectives

The first goal was to investigate whether one LPV-MPC controller could be applied to the entire mathematical model of the drone. However, it became apparent that due to strong nonlinearities, the drone was not able to follow the reference coordinates. Therefore, the controller was separated into two separate controllers. The LPV-MPC strategy was used to control the attitude of the UAV and the feedback linearization strategy was used to control the position of the drone. The objectives of the thesis are the following:

- To reformulate the mathematical model of the drone into the LPV format.
- To derive the mathematical formulation of the MPC such that a MATLAB® solver called "quadprog" could be applied.
- To implement the LPV-MPC controller to control the drone attitude.
- To implement the position controller to control the drone position in space and integrate it with the LPV-MPC attitude controller.
- To validate the global controller by letting it track various tracks.

1.3 Thesis Structure

The thesis is structured in the following way to achieve the aforementioned goals:

Chapter 2

This chapter establishes the mathematical model of the quadcopter and the reference frames used.

Chapter 3

This chapter reformulates the mathematical model of the drone into the LPV format. It also derives the mathematical formulation of the MPC to make it compatible with the quadprog solver. This will be followed by the implementation of the LPV-MPC controller in MATLAB®. Finally, the position controller will be implemented and integrated with the LPV-MPC attitude controller.

Chapter 4

In this chapter, the global controller is validated - the results are shown and analyzed. The controller will be tested on five different trajectories.

Chapter 5

The final chapter concludes the thesis and summarizes its findings. In addition, future potential research areas from this work will be discussed.

Chapter 2

Quadrotor mathematical model

The MPC strategy uses the model of a system to predict its behaviour into the future based on the length of its horizon period. Based on the model prediction, the optimizer in the MPC minimizes the cost function. The inputs found are then applied to the system. Therefore, it is important that the mathematical model of the system is accurate. If it is not accurate enough, then the obtained inputs, that were obtained from the model will not influence the real system as expected. Its response might become underdamped, overdamped or even unstable. This chapter focuses on defining the coordinate frames used to control the drone and it establishes the mathematical model of a quadcopter.

2.1 The definition of coordinate frames

There will be two reference frames considered: a fixed ground (Earth) reference frame (E-frame) in Figure 2.1 [3] and a Body fixed frame (B-frame) in Figure 2.2 [3]. In Figure 2.1 [3], it can be seen that the E-frame (in green) has the axis N, that points towards the North, the axis E, that points towards the East, and the axis U that is perpendicular to its plane. This last one is the global axis with respect to which the drone flies.

The B-frame in Figure 2.2 [3] is attached to the drone itself. Because of that, it is more suitable to use the E-frame for position measurement and the B-frame for the velocity measurement. It is assumed that the center of the reference frame is put in the center of the mass of the drone [3].

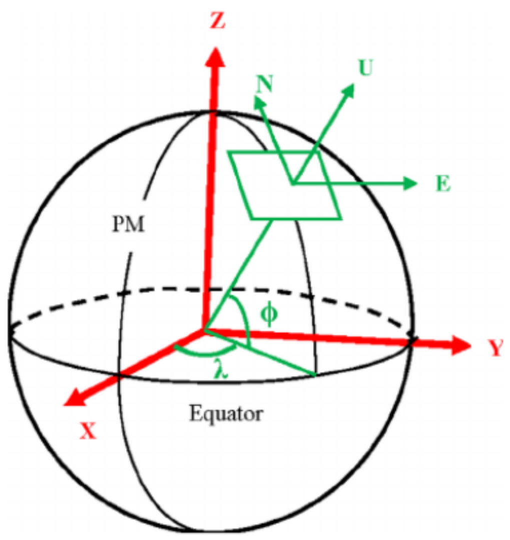

Figure 2.1: Fixed ground reference frame (green) [3]

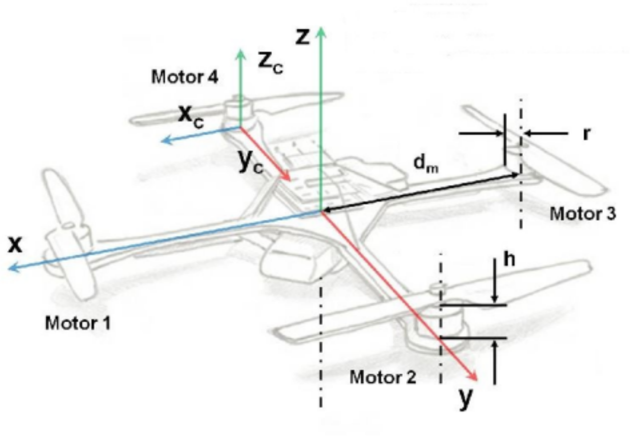

Figure 2.2: The body reference frame attached to the UAV [3]

2.2 The control inputs

In order to understand the input signals of the drone, it is important to clarify how the four rotors of the drone move. In Figure 2.3 [3], it can be seen that the motors 1 and 3 rotate counter-clockwise and the rotors 2 and 4 rotate clockwise. The body axis is positioned in such a way that the positive x-direction points towards motor 1 and the positive y-direction points towards motor 2.

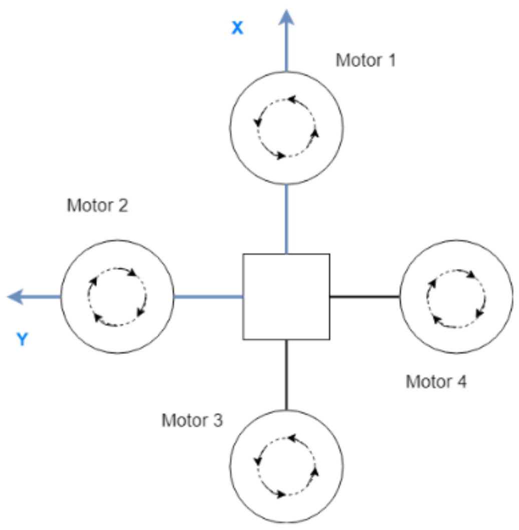

Figure 2.3: The UAV motors, their rotational direction and the B-axis [3]

There are 4 input signals that are introduced into the system: U1, U2, U3 and U4. [3]

Thrust U1 [N] (Throttle)

This input signal is a force that points towards the z-axis of the B-frame. This force is generated by the angular rotation of all the rotors. It does not matter if the rotation of one rotor is faster than the spinning of the other 3 motors. The thrust force that all the rotors generate are summed up resulting in the global thrust force called U1. It can be seen visually in Figure 2.4 [3]. On the left of this figure, it can be clearly seen that the added rotation of each of the rotors contributes to generating the thrust force U1.

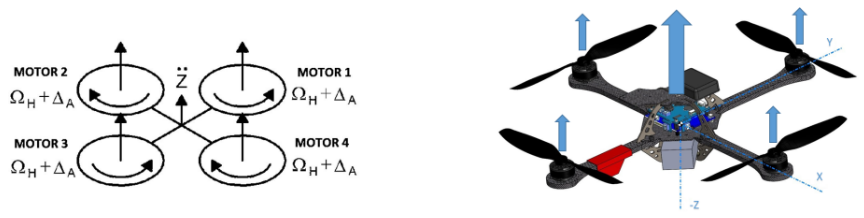

Figure 2.4: Thrust force U1 generated by the 4 rotors of the quadcopter [3]

Moment U2 [N m] (Roll)

The input signal U2 is a torque signal, which is around the x-axis of the body frame. It can be seen visually in Figure 2.5 [3]. In order to produce this input signal, the rotation of the rotors in motors 1 and 3 must be equal; however, the spinning of the motors 2 and 4 must be different. That creates an imbalance in the thrust force around the x-axis, which will create a moment around it, which is the control input signal U2. That moment, which also depends on how far the rotors are from the center of the drone, causes the UAV to rotate around the x-axis of the body frame.

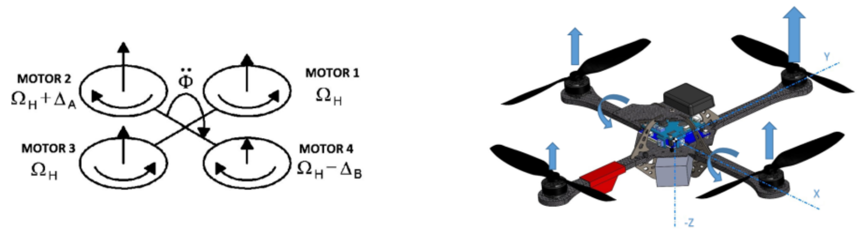

Figure 2.5: Moment U2 generated by the 4 rotors of the quadcopter [3]

Moment U3 [N m] (Pitch)

The input signal U3 is a torque signal, which is around the y-axis of the body frame. It can be seen visually in Figure 2.6 [3]. In order to produce this input signal, the rotation of the rotors in motors 2 and 4 must be equal; however, the spinning of the motors 1 and 3 must be different. That creates an imbalance in the thrust force around the y-axis, which will create a moment around it, which is the control input signal U3. That moment, which also depends on how far the rotors are from the center of the drone, causes the UAV to rotate around the y-axis of the body frame.

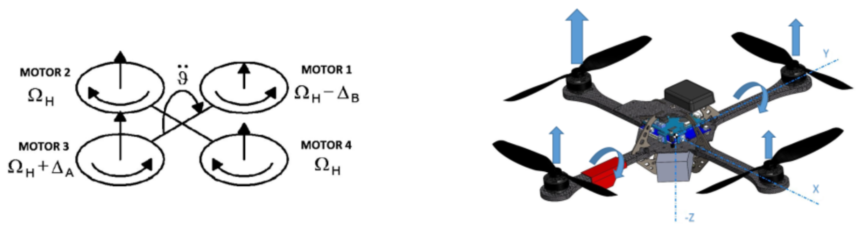

Figure 2.6: Moment U3 generated by the 4 rotors of the quadcopter [3]

Moment U4 [N m] (Yaw)

The input signal U4 is a torque signal, which is around the z-axis of the body frame. It can be seen visually in Figure 2.7 [3]. In order to produce this input signal, the rotation of the rotors in motors 1 and 3 must be equal, and the rotation of the rotors 2 and 4 must be equal; however, the spinning of the motors 1 and 3 must be different from the spinning of the motors 2 and 4. Due to conservation of angular momentum, the Yaw moment, which is the U4 input signal will be generated. Due to that moment, the UAV will start rotating around the z-axis of its body frame.

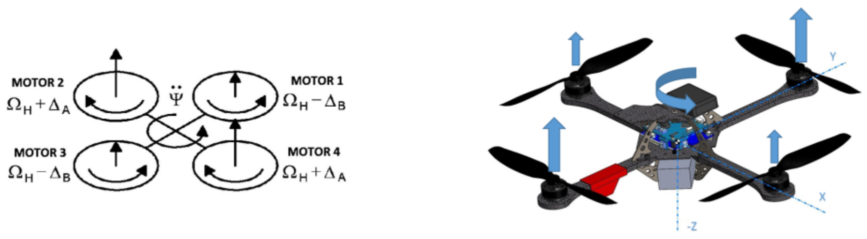

Figure 2.7: Moment U4 generated by the 4 rotors of the quadcopter [3]

In the system of equations 2.1, it is shown how the force U1 and the moments U2, U3 and U4 are related to the acceleration in the z-axis and the angular accelerations of ϕ, θ and ψ, respectively. Here, m, I_x, I_y, I_z are the drone's mass and the values of its angular momentum about the axes specified in their subscript, respectively. The double-dots mean the second time derivative of the variables, which in this case are their acceleration values.

$$\begin{aligned} U1 &= m\ddot{z} \\ U2 &= I_x\ddot{\phi} \\ U3 &= I_y\ddot{\theta} \\ U4 &= I_z\ddot{\psi} \end{aligned} \tag{2.1}$$

In case of a reference tracking problem, the closed loop controller for the UAV gives the input signals U1, U2, U3 and U4 directly to the drone. Based on these inputs, the four rotors of the UAV rotate accordingly. That means that there must be a relationship between the input signals and the angular velocities of the rotors. In the system of equations (2.2) [3], it can be seen very clearly how the control input signals are related to the angular velocities of the rotors, which are denoted as $\Omega_1,\ \Omega_2,\ \Omega_3\ and\ \Omega_4\ [rad \cdot s^{-1}]$ for the motors 1, 2, 3 and 4, respectively. The values of $c_T\ [Ns^2]\ and\ c_Q\ [Nms^2]$ are aerodynamic coefficients of thrust and drag, respectively [1]. The value of $l\ [m]$ is the distance between the center of the quadrotor and the center of a propeller [1]. Finally, the equation (2.3) adds up the rotational velocities of all the rotors [1]. Since the propellers 1 and 3 rotate counter-clockwise and the propellers 2 and 4 rotate clockwise, the motors 1 and 3 have the opposite sign compared to the motors 2 and 4

$$\begin{aligned} U1 &= c_T \cdot (\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ U2 &= c_T \cdot l \cdot (\Omega_4^2 - \Omega_2^2) \\ U3 &= c_T \cdot l \cdot (\Omega_3^2 - \Omega_1^2) \\ U4 &= c_Q \cdot (-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \end{aligned} \tag{2.2}$$

such that

$$\Omega_{total} = -\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4 \tag{2.3}$$

The final step needed to start building the controller for the drone is to obtain its state space equations. It is needed to have the drone's mathematical model in that form because it must only contain first order differentiation. The reason for that is because in the implementation of the controller a MATLAB® ode45 integrator will be used to integrate the system's states in time.

However, the integrator only accepts first order systems; hence, the equations of motion need to be made first order, which will be done in the next section.

2.3 The mathematical model of a quadrotor

A quadcopter has six degrees of freedom - 3 position and 3 attitude dimensions. These are $x,\ y,\ z\ and\ \phi,\ \theta,\ \psi$, respectively. The mathematical model of a UAV has to incorporate all the degrees of freedom. One way to express a mathematical model of a drone is to write it in the B-frame. The system of equations (2.4) describes the drone in the B-frame [3]. The variables u, v and w are x, y and z velocities in the B-frame $[ms^{-1}]$, respectively. The variables p, q and r are the angular velocities of $\phi,\ \theta,\ \psi$ in the B-frame $[rad\ s^{-1}]$, respectively. The Ω is the added rotation of all the rotors that comes from the equation (2.3) [1]. The constant g is the gravitational acceleration on the surface of the Earth, which is 9.81 ms^{-2}. Finally, the constant J_{TP} $[Nms^2]$ is is the total rotational moment of inertia around the propeller axis [3]

$$\begin{aligned}
\dot{u} &= (vr - wq) + g\sin\theta \\
\dot{v} &= (wp - ur) - g\cos\theta\sin\phi \\
\dot{w} &= (uq - vp) - g\cos\theta\cos\phi + \frac{U_1}{m} \\
\dot{p} &= qr\frac{I_y - I_z}{I_x} - \frac{J_{TP}}{I_x}q\Omega + \frac{U_2}{I_x} \\
\dot{q} &= pr\frac{I_z - I_x}{I_y} + \frac{J_{TP}}{I_y}p\Omega + \frac{U_3}{I_y} \\
\dot{r} &= pq\frac{I_x - I_y}{I_z} + \frac{U_4}{I_z}
\end{aligned} \tag{2.4}$$

The system of equations (2.4) is in a convenient form because it only contains first order differentiation [3] . However, the problem with expressing everything in the B-frame is that now all six degrees of freedom states are velocities. However, the trajectory is given in the position values of x, y and z in the E-frame. Therefore, it is needed to have a system of equations in which the translational motion states are in the E-frame position format. The rotational motion states can stay in the B-frame as angular velocities. This Hybrid-frame (H-frame) can be seen in the equation (2.5) [3].

$$\begin{aligned}
\ddot{x} &= (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)\frac{U_1}{m} \\
\ddot{y} &= (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)\frac{U_1}{m} \\
\ddot{z} &= -g + \cos\phi\cos\theta\frac{U_1}{m} \\
\dot{p} &= qr\frac{I_y - I_z}{I_x} - \frac{J_{TP}}{I_x}q\Omega + \frac{U_2}{I_x} \\
\dot{q} &= pr\frac{I_z - I_x}{I_y} + \frac{J_{TP}}{I_y}p\Omega + \frac{U_3}{I_y} \\
\dot{r} &= pq\frac{I_x - I_y}{I_z} + \frac{U_4}{I_z}
\end{aligned} \tag{2.5}$$

The H-frame contains the position variables in the E-frame. However, now the problem is that it has second order differentiation in it. The MATLAB® integrator ode45 needs first order differential equations though. In addition, the H-frame does not contain the angles ϕ, θ, ψ, which are the orientation of a drone in the E-frame. To solve these two problems, the H-frame system of equations can be expanded. The relationship between the E and B-frame can be used to create one large system of equations that contains all the six states in the B and also in the E-frame - in total, 12 states, which are: u, v, w, p, q, r, x, y, z, ϕ, θ, ψ. This system of equations would be first order and therefore suitable for the ode45 integrator. The rotational matrix under the Z-Y'-X'' Euler angles convention that relates the translational velocities in the B-frame (u, v, w) to the translational velocities in the E-frame $(\dot{x}\ \dot{y}\ \dot{z})$ can be seen in the equation (2.6) [2]. The transformation matrix that relates the angular velocities in the B-frame (p, q, r) to the angular velocities in the E-frame $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ can be seen in the equation (2.7) [3]

$$R = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{2.6}$$

$$T = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \tag{2.7}$$

Now, there are all the tools needed to build a global system of equations with all the states from the both frames and that is also first order, suitable for the ode45 integrator.The global open-loop system that will be integrated in this thesis while running the simulations can be seen in the equation (2.8). The system of equations in this configuration allows all the states $(u, v, w, p, q, r, x, y, z, \phi, \theta, \psi)$ to be tracked. Once an initial value is given to them, the equation (2.8) computes their derivatives and then it is possible to know the state values in the next sample time period

$$\begin{aligned}
\dot{u} &= (vr - wq) + g\sin\theta \\
\dot{v} &= (wp - ur) - g\cos\theta\sin\phi \\
\dot{w} &= (uq - vp) - g\cos\theta\cos\phi + \frac{U_1}{m} \\
\dot{p} &= qr\frac{I_y - I_z}{I_x} - \frac{J_{TP}}{I_x}q\Omega + \frac{U_2}{I_x} \\
\dot{q} &= pr\frac{I_z - I_x}{I_y} + \frac{J_{TP}}{I_y}p\Omega + \frac{U_3}{I_y} \\
\dot{r} &= pq\frac{I_x - I_y}{I_z} + \frac{U_4}{I_z} \\
\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} &= R \begin{bmatrix} u \\ v \\ w \end{bmatrix} \\
\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} &= T \begin{bmatrix} p \\ q \\ r \end{bmatrix}
\end{aligned} \tag{2.8}$$

Chapter 3

The controller design

3.1 Introduction to two possible control strategies

In the previous chapter, an open loop system of a UAV in the form of state space equations was derived. This chapter is about designing a suitable control strategy for the drone using the LPV-MPC technique. Two different control strategies will be attempted. The first of them will attempt to apply the LPV-MPC approach to control the entire system. The schematic of the control strategy is shown in Figure 3.1.

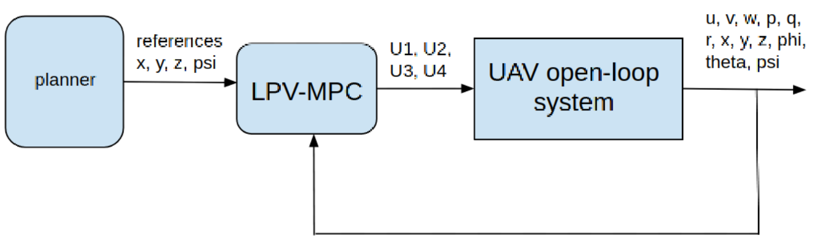

Figure 3.1: Control strategy: only LPV-MPC applied

It will become apparent in this thesis that the strategy presented in Figure 3.1 will fail at controlling the drone because there are hard nonlinearities in its mathematical model. It will make it impossible to extract the angles $\phi\ and\ \theta$ and therefore, the LPV-MPC approach could not be used on the entire system. Nevertheless, the attempt to do it was made as it will be seen in this chapter. Because of the failure of the first method, an alternative approach was used. The schematic of it can be seen in Figure 3.2. In that approach, the controller was split into two subcontrollers. One controller is responsible for the position variables x, y, z. This position controller uses the state feedback linearization method. The outputs it generates are U1, and the angles $\phi\ and\ \theta$. The aforementioned angles are the ones that, together with U1 and ψ, are necessary in order for the UAV to reach its x, y, z reference position. U1 is fed straight as an input into the open loop system. However, the angles $\phi\ and\ \theta$ are then fed into the attitude controller as reference values. This controller is the one that uses the LPV-MPC approach. The reference angles $\phi\ and\ \theta$, together with the angle ψ from the planner, allows the LPV-MPC controller to find the remaining three control actions for the open loop system, which are U2, U3 and U4. It is important to note that the LPV-MPC controller needs time push the state angles towards its reference values. Therefore, the attitude controller must work at a higher frequency compared to the position controller - it has to have higher dynamics. In this thesis, it will be seen that the inner loop (the loop for the attitude controller) works 4 times faster. After integrating the open-loop system, the new angles ϕ, θ and ψ are fed back into the LPV-MPC controller together with the new Ω value at every one-fourth of the sample time. In addition, at every sample time, the open loop system sends the new x, y and z values back into the position controller.

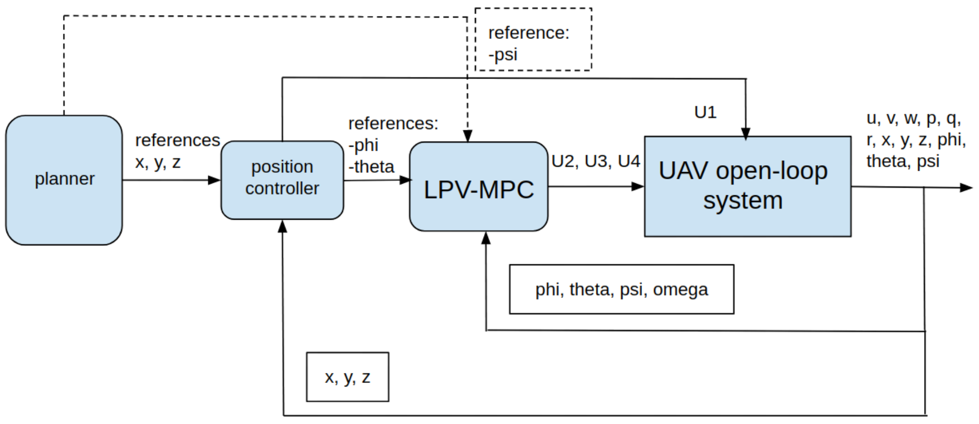

Figure 3.2: Control strategy: LPV-MPC applied in combination with a position controller

3.2 LPV mathematical derivation for the UAV

In order to use the most basic MPC strategy made for linear systems, the nonlinear UAV model must first be linearized or reformulated into the Linear Parameter Varying (LPV) format, which encapsulates the model nonlinearities in a linear structure. The latter approach is used in this thesis.

There are several advantages that the LPV offers compared to linearizing a system. Firstly, linearization becomes less precise as the system moves further away from the operating point. However, this problem does not occur with LPV because it is merely a reformulation of a mathematical nonlinear model into a format that resembles into a linear structure. In other words, all the nonlinearities are encapsulated in A and B matrices of a state space equation.

Secondly, one has to make a stability check every time an operating point in linearizing a system changes. That is because with every point where the system is linearized, the A matrix is different. Thus, its eigenvalues need to be checked to make sure that their real numbers are negative. However, with the LPV approach, one can define a region in the state space and if the stability of the system is proven in the vertices of the region, then the system is stable inside the region as well.

In order to derive an LPV model of the system, it is good to treat the equations concerning the positions separate from the equations that describe the drone's attitude. First, the position equations are dealt with. They are the first three equations in the system of equations (2.5) [3]. However, they are all second order differential equations. To get them into a linear state space format, the system of equations need to be expanded where the the states are $x,\ \dot{x},\ y,\ \dot{y},\ z,\ \dot{z}$. In equation (3.1), a state space system for linear systems is shown. As one can see, it is equivalent to (2.5) [3]. However, since the nonlinearities stay, they are all put in the B matrix. The A matrix is multiplied by the states and the B matrix is multiplied by the input. The states $x,\ \dot{x},\ y,\ \dot{y},\ z,\ \dot{z}$ can be renamed as $x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6$, then their derivatives $\dot{x},\ \ddot{x},\ \dot{y},\ \ddot{y},\ \dot{z},\ \ddot{z}$ will be $\dot{x_1},\ \dot{x_2},\ \dot{x_3},\ \dot{x_4},\ \dot{x_5},\ \dot{x_6}$ - resulting in a first order system of differential equations.

ETSEIB

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{y} \\ \ddot{y} \\ \dot{z} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \\ z \\ \dot{z} \end{bmatrix} + \begin{bmatrix} 0 \\ (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)\frac{1}{m} \\ 0 \\ (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)\frac{1}{m} \\ 0 \\ -\frac{g}{U_1} + \cos\phi\cos\theta\frac{1}{m} \end{bmatrix} U_1 \tag{3.1}$$

In order to have an LPV model for the angles, the last 3 equations of a system of equations (2.5) will be considered [3]. However, these equations are in the B-frame. To control the drone, the LPV model needs to contain the angles and their instantaneous changes in the E-frame. In case of angles, both of the frames are related to each other by the transformation matrix T in equation (2.7) [3].

However, here a simplifying assumption can be made that affects how the drone is controlled, insignificantly. The quadcopter cannot hover in one position if it is tilted in one direction all the time - it would start sliding down diagonally. The goal is to design a controller that stabilizes the quadcopter close to the hovering position. In this case, the angles ϕ and θ are assumed to be zero, which makes the T matrix in the equation (2.7) an identity matrix I. This converts the last three equations in the system of equations (2.5) [3] into a system of equations, in which p, q, r become $\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$, respectively [3]. It can be seen in the equation (3.2) [3], which is also second order

$$\begin{aligned} \ddot{\phi} &= \dot{\theta}\dot{\psi}\frac{I_y - I_z}{I_x} - \frac{J_{TP}}{I_x}\dot{\theta}\Omega + \frac{U_2}{I_x} \\ \ddot{\theta} &= \dot{\phi}\dot{\psi}\frac{I_z - I_x}{I_y} + \frac{J_{TP}}{I_y}\dot{\phi}\Omega + \frac{U_3}{I_y} \\ \ddot{\psi} &= \dot{\phi}\dot{\theta}\frac{I_x - I_y}{I_z} + \frac{U_4}{I_z} \end{aligned} \tag{3.2}$$

Just like it was done with the position variables, to generate an LPV format for the angles, the system of equations (3.2) [3] was expanded. The LPV format is shown in equation (3.3) [3], in which it can be seen that all the nonlinearities are encapsulated in the A matrix. The B matrix only has constant values. The A matrix is multiplied by the states and the B matrix is multiplied by the inputs

$$\begin{bmatrix} \dot{\phi} \\ \ddot{\phi} \\ \dot{\theta} \\ \ddot{\theta} \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{\Omega \cdot J_{TP}}{I_x} & 0 & \dot{\theta} \cdot \frac{I_y - I_z}{I_x} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{\Omega \cdot J_{TP}}{I_y} & 0 & 0 & 0 & \dot{\phi} \cdot \frac{I_z - I_x}{I_y} \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \frac{\dot{\theta}}{2} \cdot \frac{I_x - I_y}{I_z} & 0 & \frac{\dot{\phi}}{2} \cdot \frac{I_x - I_y}{I_z} & 0 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_z} \end{bmatrix} \begin{bmatrix} U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{3.3}$$

In the second control strategy with the position controller, only the equation (3.3) is used in the MPC control [3]. That is because position is handled by the feedback linearization strategy that does not require an LPV model of the system. However, in the first control strategy, where only the LPV-MPC method is used for the entire system, both LPV models are combined into one single state space system of equations as can be seen in equation (3.4). That is used for the MPC in the first control strategy, where only the LPV-MPC method is used to control the entire drone

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{y} \\ \ddot{y} \\ \dot{z} \\ \ddot{z} \\ \dot{\phi} \\ \ddot{\phi} \\ \dot{\theta} \\ \ddot{\theta} \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} A_{LPV-(x-y-z)} & zeros(6,6) \\ zeros(6,6) & A_{LPV-(\phi-\theta-\psi)} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \\ z \\ \dot{z} \\ \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} B_{LPV-(x-y-z)} & zeros(6,3) \\ zeros(6,1) & B_{LPV-(\phi-\theta-\psi)} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{3.4}$$

3.3 General MPC mathematical derivation

In this section, it is shown how the MPC technique was derived to match the MATLAB® quadprog solver that allows to solve a problem with the following structure [4]

$$\min_{x} \frac{1}{2} x^T H x + f^T x \text{ such that } \begin{cases} A \cdot x \leq b \\ \text{Aeq } \cdot x = beq \\ \text{lb } \leq x \leq ub \end{cases} \tag{3.5}$$

In Figure 3.3 [5], a simple intuition into the MPC strategy is presented. The sequence goes from up to down. The sample times go from the sample k to the sample $k + N$. The parameter N is called the prediction horizon. The length of it depends on the system dynamics. The goal of the MPC is to stabilize itself around the reference during the prediction horizon. In the first sub-image, it can be seen that there are many ways to get there. The algorithm determines the error values in each sample time as it can be seen in the second sub-image. It then takes two main features into account - the squared sum of the errors (e) and the squared sum of the change of inputs (δu) as it can be seen in equation (3.6) [5]. The importance of errors and change of inputs can be regulated with weights (w). This entire equation is called a cost function (J)

$$J = \sum_{i=1}^{N} w_e e_{k+i}^2 + \sum_{i=0}^{N-1} w_{\Delta u} \Delta u_{k+i}^2 \tag{3.6}$$

MPC uses a solver that finds a set of change of inputs such that the cost function is minimized. In the third sub-image in Figure 3.3 [5], one can see the predicted path that was generated by the inputs that the solver had previously chosen. However, due to disturbances and uncertainties, the system might end up being slightly off from the prediction in the next sample period. In the third sub-image, it is slightly above the predicted value. Therefore, only the first element of the input vector is chosen - the rest are discarded. Then, the horizon period shifts and it goes from $k + 1$ to $k + N + 1$. A new prediction is made from the most recent position as it can be seen in the forth sub-image. That position might be measured or a combination of measured and predicted position that comes out of a filter such as the Kalman filter.

When dealing with MPC - a distinction must be made, which depends on the objectives. If MPC is used for the purpose of regulation, then that means that the goal is to bring the state values close to zero. In this case, the linear prediction model is

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + Du_k \end{aligned} \tag{3.7}$$

where x is a state variable, u is the input variable, and y is the output vector that is related to a state vector through the C matrix. The D matrix is assumed to be zero as it is with most systems. This model is discretized. In this thesis, the discretization was performed using both, the forward Euler method, and the zero hold order (zoh). In the case of regulation, the cost function of MPC is

$$J = min\ \frac{1}{2}x_{t+N}^T S x_{t+N} + \frac{1}{2}\sum_{k=0}^{N-1}(x_{t+k}^T Q x_{t+k} + u_{t+k}^T R u_{t+k}) \tag{3.8}$$

It is important to point out that in the case of regulation, the cost function does not deal with input changes - the inputs are absolute values. That is because once the states are all zero, the inputs can also be zero assuming that the system is stable. For example, if the objective is to land and stop the drone, then once this goal is met, the UAV's inputs can be equal to zero. In addition, the last element in the horizon period has a different weight matrix, which is called S. Also, the cost function is multiplied by a value of 0.5. That is for convenience. When a gradient of it is taken, then the constant in the cost function becomes 1 [9]. Since the cost function shifts in time, the symbol t is the present time and $t+k$ is k samples from the current present. However, in this thesis, the challenge is not regulation, it is reference tracking. In a tracking problem, in the cost function, the state variable is replaced by the error variable [8] leadind to

$$J = min\ \frac{1}{2}e_{t+N}^T S e_{t+N} + \frac{1}{2}\sum_{k=0}^{N-1}(e_{t+k}^T Q e_{t+k} + u_{t+k}^T R u_{t+k}) \tag{3.9}$$

The error is defined in equation (3.27) [8] as follows

$$e_k = r_k - y_k = r_k - Cx_k. \tag{3.10}$$

In a tracking problem, the inputs must be nonzero to keep the tracking error zero and keep the UAV following the reference so that it could track the desired trajectory. Nonetheless, once the drone reaches the reference value, the change of input can be zero assuming that the system is stable. Therefore, in a tracking problem, the changes of input Δu_k are used, which are defined as follows [8]

$$\Delta u_k = u_k - u_{k-1}. \tag{3.11}$$

The equation (3.11) can be rewritten as [8]

$$u_k = u_{k-1} + \Delta u_k. \tag{3.12}$$

That means that the state space equation (3.7) can be rewritten in the following way as [7]

$$\begin{aligned} x_{k+1} &= Ax_k + B(u_{k-1} + \Delta u_k) \\ y_k &= Cx_k. \end{aligned} \tag{3.13}$$

The next step would be to augment the system where the absolute input one sample in the past u_{k-1} becomes a state as well. The augmented formulation can be written as [8]

$$\begin{gathered} \begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} + \begin{bmatrix} B \\ I \end{bmatrix} \Delta u = \widetilde{A}\widetilde{x_k} + \widetilde{B}\Delta u \\ y_k = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} = \widetilde{C} \begin{bmatrix} x_k \\ u_{k-1}. \end{bmatrix} \end{gathered} \tag{3.14}$$

Once the error term in the cost function is substituted with with the equation (3.27) and the C matrix and the states are replaced with the augmented version from equation (3.14) , the cost function will have Δu instead of an absolute of u [8]

$$\begin{aligned} J = min\ & \frac{1}{2}(r_{t+N} - \widetilde{C}\widetilde{x_{t+N}})^T S(r_{t+N} - \widetilde{C}\widetilde{x_{t+N}}) + \\ & + \frac{1}{2}\sum_{k=0}^{N-1}((r_{t+k} - \widetilde{C}\widetilde{x_{t+k}})^T Q(r_{t+k} - \widetilde{C}\widetilde{x_{t+k}}) + \Delta u_{t+k}^T R \Delta u_{t+k}). \end{aligned} \tag{3.15}$$

Once it is written out, there will be some terms that are constant. From an optimization point of view, constant terms do not affect the results of the optimizer. Therefore, to simplify the equation, they will be crossed out as follows [8]

$$\begin{aligned} J = & \cancel{\frac{1}{2} r_{t+N}^T S r_{t+N}} - r_{t+N}^T S\tilde{C}\tilde{x}_{t+N} + \frac{1}{2}\tilde{x}_{t+N}^T \tilde{C}^T S \tilde{C}\tilde{x}_{t+N} + \\ & + \sum_{k=0}^{n-1}\left[\cancel{\frac{1}{2} r_{t+k}^T Q r_{t+k}} - r_{t+k}^T Q\tilde{C}\tilde{x}_{t+k} + \frac{1}{2}\tilde{x}_{t+k}^T \tilde{C}^T Q \tilde{C}\tilde{x}_{t+k} + \frac{1}{2}\Delta u_{t+k}^T R \Delta u_{t+k}\right]. \end{aligned} \tag{3.16}$$

It is important to note that this form is valid only if the weight matrices are diagonal because then they equal to their transpose values and this form can be achieved.

So far, the horizon period has been described with a summation sign. However, it can also be described by stacking the future reference values, states and change of inputs in one big vector, where each element represents one sample time period. It can be seen in equation (3.17) along with a present state vector denoted as $\tilde{x}_t$, which is written separately and does not form part of the future state values [8]. Also, in the global vector for the horizon period, the reference and the state values start from the period $t+1$ and end at $t+N$; however, the change of inputs start at the period t and end at $t+N-1$. That is because an input in one period affects a state and an output in the next period. In equation 3.18 [8], one can see how the entire cost function is written in that way, where the weight matrices are stacked into big diagonal matrices, which describe the entire horizon period.

$$r = \begin{bmatrix} r_{t+1} \\ r_{t+2} \\ . \\ . \\ r_{t+N} \end{bmatrix} \quad \tilde{x} = \begin{bmatrix} \tilde{x}_{t+1} \\ \tilde{x}_{t+2} \\ . \\ . \\ \tilde{x}_{t+N} \end{bmatrix} \quad \Delta u = \begin{bmatrix} \Delta u_t \\ \Delta u_{t+1} \\ . \\ . \\ \Delta u_{t+N-1} \end{bmatrix} \quad \tilde{x}_t = present \tag{3.17}$$

$$J' = min\frac{1}{2}\tilde{x}^T \begin{bmatrix} \tilde{C}^T Q \tilde{C} & & \\ & \cdot & \\ & & \tilde{C}^T Q \tilde{C} & \\ & & & \tilde{C}^T S \tilde{C} \end{bmatrix} \tilde{x} - r^T \begin{bmatrix} Q\tilde{C} & & \\ & \cdot & \\ & & Q\tilde{C} & \\ & & & S\tilde{C} \end{bmatrix} \tilde{x} +$$
$$+ min\frac{1}{2}\Delta u^T \begin{bmatrix} R & & \\ & \cdot & \\ & & R & \\ & & & R \end{bmatrix} \Delta u = min\frac{1}{2}\tilde{x}^T\overline{\overline{Q}}\tilde{x} - r^T\overline{\overline{T}}\tilde{x} + min\frac{1}{2}\Delta u^T\overline{\overline{R}}\Delta u \tag{3.18}$$

The objective is to write the cost function in the form that only has change of input values and and state values in the present. To remove future state values, a mathematical manipulation is performed as follows [6]

$$\begin{aligned} &\tilde{x}_1 = \tilde{A}\tilde{x}_0 + \tilde{B}\Delta u_0 \\ &\tilde{x}_2 = \tilde{A}\tilde{x}_1 + \tilde{B}\Delta u_1 = \\ &= \tilde{A}^2\tilde{x}_0 + \tilde{A}\tilde{B}\Delta u_0 + \tilde{B}\Delta u_1 \\ &... \\ &\tilde{x}_k = \tilde{A}^k\tilde{x}_0 + [\tilde{A}^{k-1}\tilde{B} \quad \tilde{A}^{k-2}\tilde{B} \quad ... \quad \tilde{B}] \begin{bmatrix} \Delta u_0 \\ \Delta u_1 \\ ... \\ \Delta u_{N-1} \end{bmatrix} \end{aligned} \tag{3.19}$$

where the state values are substituted with the previous state values that only consist of the A and B matrices, the current state value, and the current and future change of input values. The entire state space system for the entire horizon period can be compactly represented as [8]

$$\tilde{x} = \begin{bmatrix} \tilde{B} & & & & \\ \tilde{A}\tilde{B} & \tilde{B} & & & \\ \tilde{A}^2\tilde{B} & \tilde{A}\tilde{B} & \tilde{B} & & \\ . & & & . & \\ \tilde{A}^{N-1}\tilde{B} & . & . & . & \tilde{B} \end{bmatrix} \Delta u + \begin{bmatrix} \tilde{A} \\ \tilde{A}^2 \\ . \\ . \\ \tilde{A}^N \end{bmatrix} \tilde{x}_t = \overline{\overline{C}}\Delta u + \widehat{\widehat{A}}\tilde{x}_t. \tag{3.20}$$

The future state values in the cost function are then replaced by this compact form. As it can be seen in equation (3.21), the future state variables have disappeared from the cost function

$$\begin{aligned} J' &= \frac{1}{2}(\overline{\overline{C}}\Delta u + \widehat{\widehat{A}}\tilde{x}_t)^T\overline{\overline{Q}}(\overline{\overline{C}}\Delta u + \widehat{\widehat{A}}\tilde{x}_t) + \frac{1}{2}\Delta u^T\overline{\overline{R}}\Delta u - r^T\overline{\overline{T}}(\overline{\overline{C}}\Delta u + \widehat{\widehat{A}}\tilde{x}_t) \Rightarrow \\ &\Rightarrow \mathit{ignoring\ constant\ terms} \end{aligned} \tag{3.21}$$

When it is written out, the constant terms are again ignored due to the fact the they do not influence the optimizer results. The final results of the derivation can be seen in the following [8]

$$\begin{aligned} J'' &= \frac{1}{2}\Delta u^T(\overline{\overline{C}}^T\overline{\overline{QC}} + \overline{\overline{R}})\Delta u + \begin{bmatrix} \tilde{x}_t^T & r^T \end{bmatrix} \begin{bmatrix} \widehat{\widehat{A}}^T\overline{\overline{QC}} \\ -\overline{\overline{TC}} \end{bmatrix} \Delta u = \\ &= \frac{1}{2}\Delta u^T\overline{\overline{H}}\Delta u + \begin{bmatrix} \tilde{x}_t^T & r^T \end{bmatrix} \overline{\overline{F}}^T \Delta u = \mathbf{\frac{1}{2}\Delta u^T\overline{\overline{H}}\Delta u + f^T\Delta u} \end{aligned} \tag{3.22}$$

Here, it is shown (in bold) how the entire cost function is put in a form that can be accepted by the MATLAB® quadprog solver. In the most basic case, it needs to receive the matrix $\overline{\overline{H}}$ and the vector f^T. If $\overline{\overline{H}}$ is positive definite, then the solver will find a set of Δu-s that minimizes the cost function. The first element of the set of Δu-s is then used to move the UAV. All the other elements will be discarded and in the next sample time period, the same process is repeated.

ETSEIB

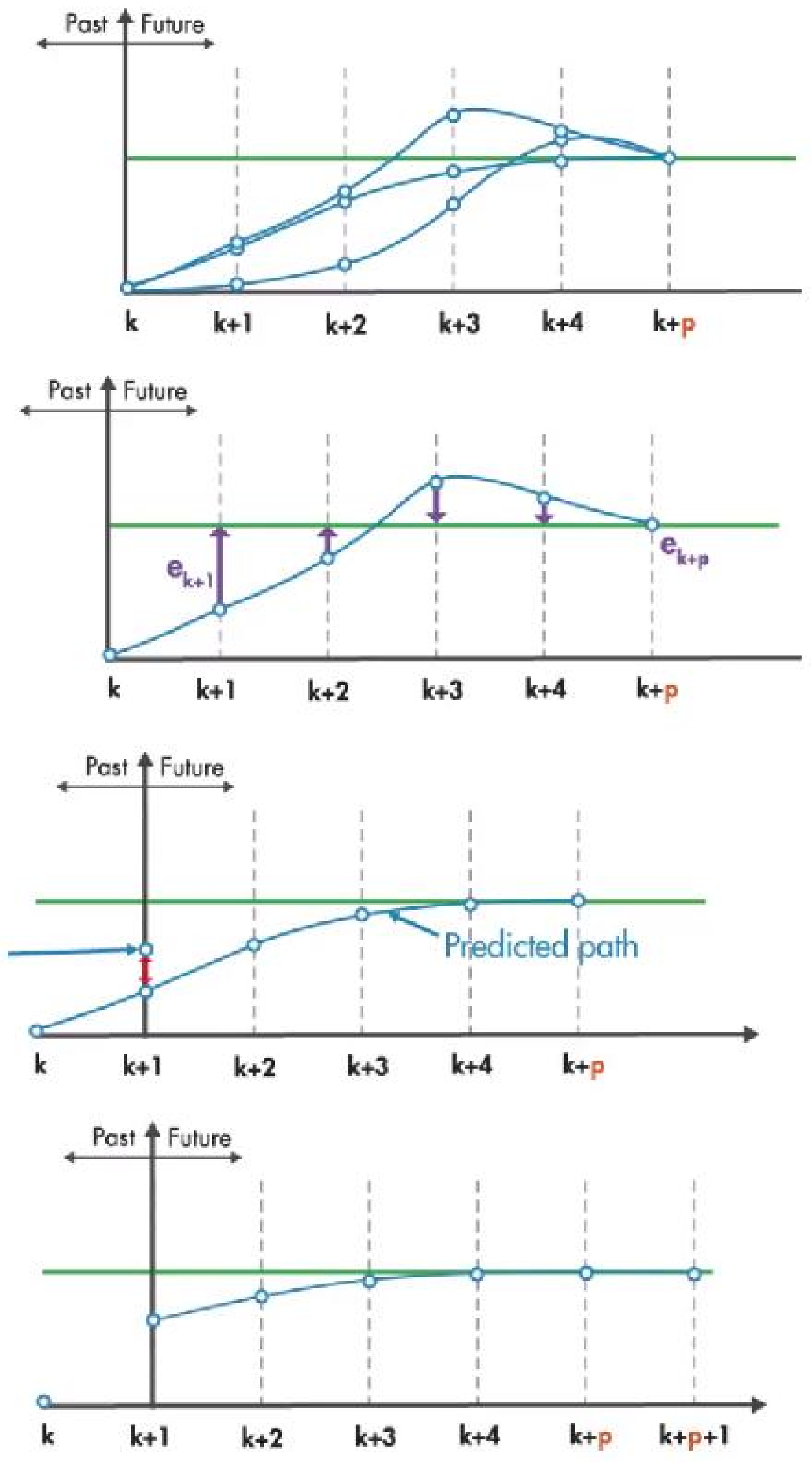

Figure 3.3: MPC intuition [5]

3.4 The architecture of the LPV-MPC controller

In this section, the architecture of the MPC controller is described - its schematic can be seen in Figure 3.4. Since, in the end, a second control strategy in Figure 3.2 was used, the one with the position controller, the MPC architecture described in this section is made to fit this control plan. The dotted line in red is where the control loop goes from one sample time to another. The integration using the MATLAB® ode45 integrator happens in the nonlinear model block, which represents the open loop system. The angular velocities in the B-frame and the angles themselves in the E-frame, together with the total rotational velocity of the rotors from the previous sample time period, are then sent to the continuous LPV block, where the nonlinear model is transformed into an LPV model. The angles in the E-frame are also sent directly to the cost function, because it needs the present angular values as can be seen in equation (3.22) [8].

The LPV model is continuous; however, the controller works discretely. Therefore, the LPV system needs to be discretized. The discrete LPV block takes in the A, B, C and D matrices from the continuous LPV block and discretizes them using the forward Euler method or the zero-order-hold (zoh) method - both are available in the code. The discretized matrices are then used to generate the H and F^T matrices in the H and F^T matrix generation block. They are then sent to the cost function block, which also receives a reference angle vector for the entire horizon period. The matrix H and the vector f^T are then sent to the MATLAB® quadprog optimizer, where a sequence of Δu-s are found that minimizes the cost function. The first element of that sequence is used to calculate the absolute input in the current sample time period. It can be seen in the following

$$U_{t+k} = U_{t+k-1} + \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta u1_{t+k} \\ \Delta u2_{t+k} \\ \Delta u3_{t+k} \\ \Delta u4_{t+k} \end{bmatrix} \tag{3.23}$$

for the horizon period of 4 samples, which is the case in the control strategy in Figure 3.2 It is then fed into the nonlinear model block. The absolute U-s are also used to compute the total rotational velocity of the rotors in the present sample period, which is also sent to the nonlinear model block. Then, the integration happens and the entire process starts all over again.

Finally, in Figure 3.5, it is shown how the reference angle ψ is computed in the planner. The angle is always measured counter-clockwise starting from 3 o'clock. The tail of a red arrow is the current sample time period. The head of that arrow is the next sample time period.

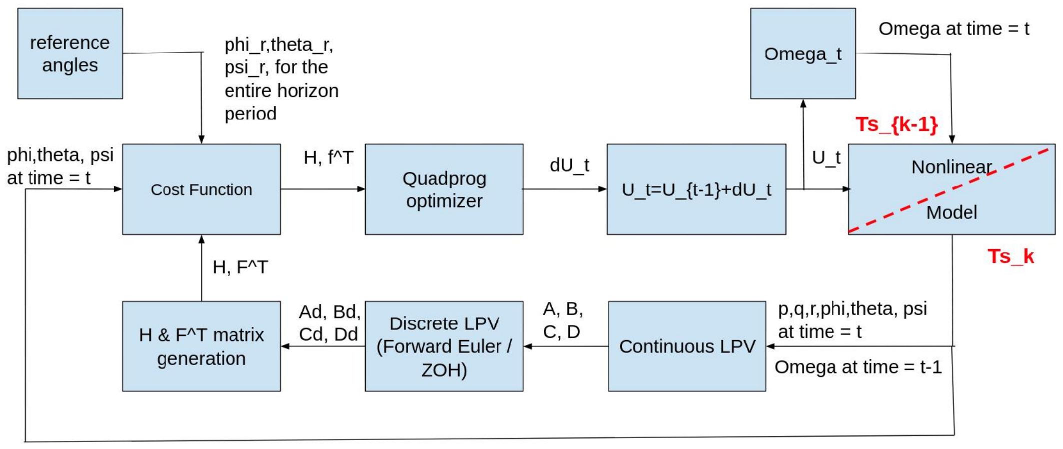

Figure 3.4: MPC controller architecture

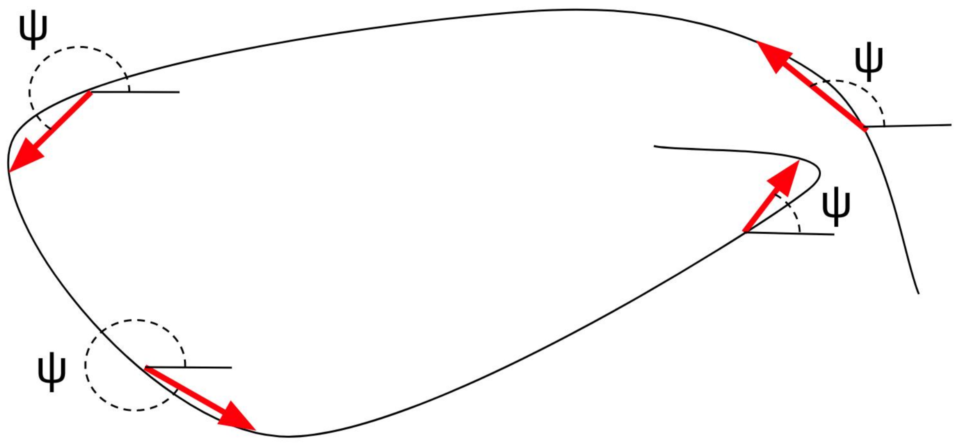

Figure 3.5: The reference angle ψ calculation in the planner

3.5 The position controller

This section describes the feedback linearization method in the position controller that is used in the second control strategy in Figure 3.2. The system of the second order differential equations that govern the UAV's position are [3]

$$\begin{aligned}\ddot{x} &= (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)\frac{U_1}{m}\\ \ddot{y} &= (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)\frac{U_1}{m}\\ \ddot{z} &= -g + \cos\phi\cos\theta\frac{U_1}{m}\end{aligned} \tag{3.24}$$

The system can be written out as a system of first order differential equations as it can be seen in the following [3]

$$\begin{aligned}\dot{x_1} &= x_2\\ \dot{x_2} &= (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)\frac{U_1}{m}\\ \dot{x_3} &= x_4\\ \dot{x_4} &= (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)\frac{U_1}{m}\\ \dot{x_5} &= x_6\\ \dot{x_6} &= -g + \cos\phi\cos\theta\frac{U_1}{m}\end{aligned} \tag{3.25}$$

The system can be written out as a system of first order differential equations as it can be seen in equation (3.25) [3], where $x_1 = x,\ x_2 = \dot{x},\ x_3 = y,\ x_4 = \dot{y},\ x_5 = z,\ x_6 = \dot{z}$ [3].

Besides position reference values x, y, z, the planner also needs to provide the position controller with the reference velocities $\dot{x},\ \dot{y},\ \dot{z}$ that are calculated as follows

$$\begin{aligned}\dot{x^R_{t+1}} &= \frac{x^R_{t+1} - x^R_t}{T_s}\\ \dot{y^R_{t+1}} &= \frac{y^R_{t+1} - y^R_t}{T_s}\\ \dot{z^R_{t+1}} &= \frac{z^R_{t+1} - z^R_t}{T_s}\end{aligned} \tag{3.26}$$

Next, the errors of the position and velocity values are computed. The velocity errors are differentiated one more time to get the acceleration of the error values. The second error derivative is essentially the negative of the acceleration of a position variable, because the reference velocity values are constant during one sample time and so, they become zeros. These errors are calculated as [3].

$$\begin{aligned} e_x &= x_t^R - x_t \quad & \dot{e_x} &= \dot{x_t^R} - \dot{x}_t \quad & \ddot{e_x} &= -\ddot{x}_t = v_x \\ e_y &= y_t^R - y_t \quad & \dot{e_y} &= \dot{y_t^R} - \dot{y}_t \quad & \ddot{e_y} &= -\ddot{y}_t = v_y \\ e_z &= z_t^R - z_t \quad & \dot{e_z} &= \dot{z_t^R} - \dot{z}_t \quad & \ddot{e_z} &= -\ddot{z}_t = v_z \end{aligned} \tag{3.27}$$

The variables $v_x,\ v_y,\ v_z$ are then chosen to be a control action for the linearized state feedback control strategy as [3].

$$\begin{aligned} v_x &= -k_1^x e_x - k_2^x \dot{e_x} \\ v_y &= -k_1^y e_y - k_2^y \dot{e_y} \\ v_y &= -k_1^z e_z - k_2^z \dot{e_z} \end{aligned} \tag{3.28}$$

From equation 3.24 [3], the variables $v_x,\ v_y,\ v_z$ can can also be substituted with the second order differential equations that govern the UAV's position, as it can be seen in the following [3]

$$\begin{aligned} v_x &= -(\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)\frac{U_1}{m} \\ v_y &= -(\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)\frac{U_1}{m} \\ v_z &= -(-g + \cos\phi\cos\theta\frac{U_1}{m}) \end{aligned} \tag{3.29}$$

By choosing negative real poles, the constants in equation (3.28) can be computed [3]. Then, the values $v_x,\ v_y,\ v_z$ are determined. In order to find the angles θ and ϕ for the attitude controller, which it will then use as reference angles, the equations (3.30) [3] and (3.31) [3] are used, respectively. The constants a, b, c and d are calculated as [3]

$$\theta = tan^{-1}(ac + bd) \tag{3.30}$$

$$
\begin{aligned}
&\textbf{if } \left(|\psi_{\mathbf{ref}}| < \frac{\pi}{4} \text{ or } |\psi_{\mathbf{ref}}| > \frac{\mathbf{3}\pi}{\mathbf{4}}\right)\\
&\phi = tan^{-1}\left(\frac{\cos(\theta)(\tan(\theta)d - b)}{c}\right)\\
&\textbf{else}\\
&\phi = tan^{-1}\left(\frac{\cos(\theta)(a - \tan(\theta)c)}{d}\right)
\end{aligned}
\tag{3.31}
$$

$$
a = \frac{v_x}{v_z + g},\ b = \frac{v_y}{v_z + g},\ c = \cos\psi_{ref},\ d = \sin\psi_{ref} \tag{3.32}
$$

$$
U_1 = \frac{(v_z + g)\, m}{\cos\phi\cos\theta} \tag{3.33}
$$

These angles, along with the ψ^R angle from the planner, will serve as reference angles for the LPV-MPC controller. However, the position controller also finds the input U1, that is directly fed into the nonlinear model. The control action U1 can be computed in equation (3.33) [3].

3.6 Implementation of the position and the LPV-MPC controller

In Figure 3.6, one can see the structure of the control strategy code. The code itself is in the Appendix B. The script consists of one MAIN file and six supportive functions. It is approximately shown with arrows in which location in the main file the functions are used. The initial constants function is a library type function in which one can find constants and certain initial values. This function also supports the other functions in this code.

The MAIN file first loads the constants and the initial values. It gets the trajectory, the reference velocities and the reference yaw angle from the trajectory generator (planner). Then, the outer loop begins where the position controller function is used. After that, the LPV-MPC loop starts that uses a function to get the discrete LPV model. The inner loop loops through the code 4 times per 1 loop of the outer loop. In the MPC simplification function, the necessary matrices for the solver are generated. After that, the solver is called. The results are then fed into the nonlinear drone model, where the integration of the open loop system happens. Finally. the results are plotted, which can be seen in the validation chapter.

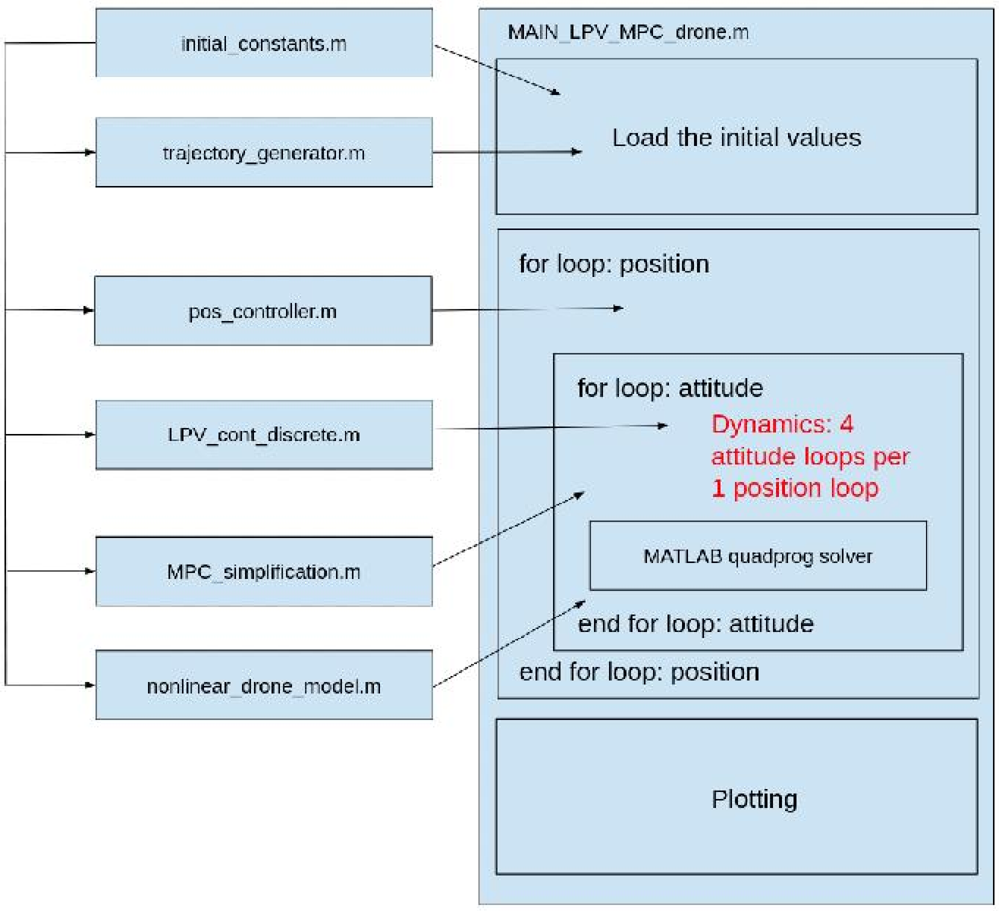

Figure 3.6: The structure of the control strategy code

Chapter 4

Validation of the controller

In this chapter, both of the proposed control strategies in Figures 3.1 and 3.2 will be validated. The testing process will be performed by giving the controllers a trajectory which is a spiral with a radius of 2 meters. Its initial height is 2 meters and the final height is 5 meters. The testing time will be 100 seconds and one sample time period is 0.1 seconds. In both cases, the initial rotational velocity of all the rotors is 3000 $\frac{rad}{s}$. At this rate, the UAV does not produce enough thrust in order to be able to leave the ground. The drone is not tilted in terms of its roll and pitch angles. The initial yaw angle will be 90 degrees counter-clockwise from 3 o'clock. The initial x, y and z coordinates of the UAV are 0, -1, 0 meters, respectively. The weight matrices in MPC (Q, R, S) are all identity matrices. The parameters of the drone in this thesis belong to $AscTec\ Hummingbird$ [3].

In Appendix A, in Figures from A.1 to A.24, four more trajectories were created to test how the proposed position and LPV-MPC controller tracks different trajectories. The paths were created by trying to change the nature of each position $(x,\ y,\ z)$ dimension. Special attention must be given to Figure A.5, in which it can be seen that as the time progresses, the angle ϕ becomes more and more negative. It means that if the test period is very long and the structure of the expanding spiral remains the same (the same initial and final height, and the continuously increasing radius along with the same sample time). Then, it can be expected that the drone becomes unstable at some point.

4.1 Experimenting with a global LPV-MPC controller-spiral

In this section, a global LPV-MPC controller was tested that is presented in Figure 3.1. The results are shown in Figures from 4.1 to 4.4. It is clear that the controller fails to track the trajectory. The x and y dimensions are completely unaffected. The UAV only oscillates up and down along the z-axis. This can be explained by the fact that the angles ϕ and θ remain unaffected. This comes from the fact that the angles could not be separated from highly nonlinear equations of motion and represented as inputs in the LPV model in equation (3.4). In other words, this control architecture was not able to generate reference values for the angles. The hope was that the angles would automatically adjust themselves in the system internally when presented with the position and yaw angle reference values. However, that did not happen and another control strategy was needed. Only the ψ angle properly tracked its reference values, which means that the drone was spinning while going up and down with a much greater magnitude than the amplitude of the spiral in the z-dimension.

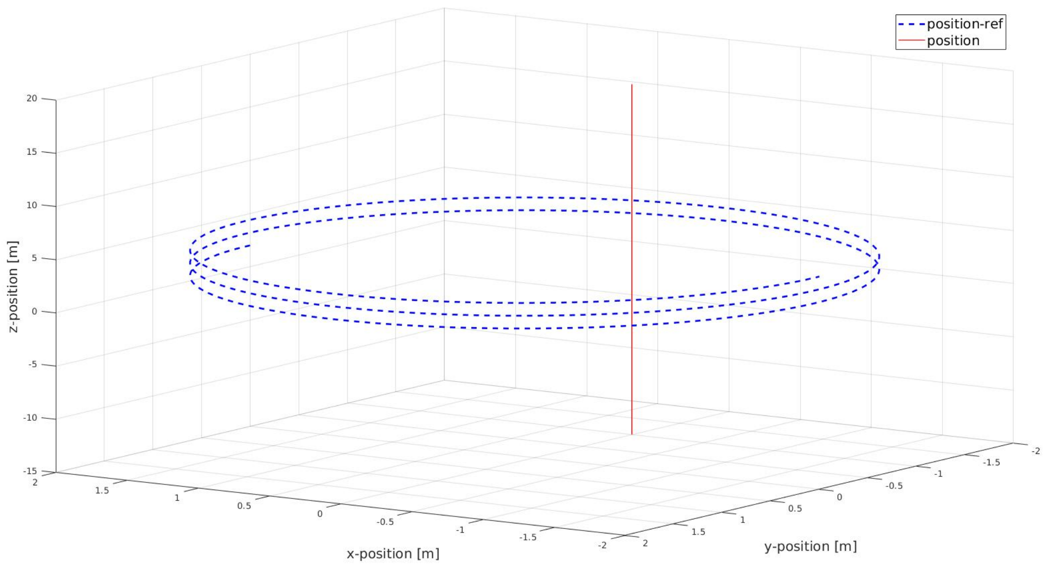

Figure 4.1: Flight trajectory in x, y, z

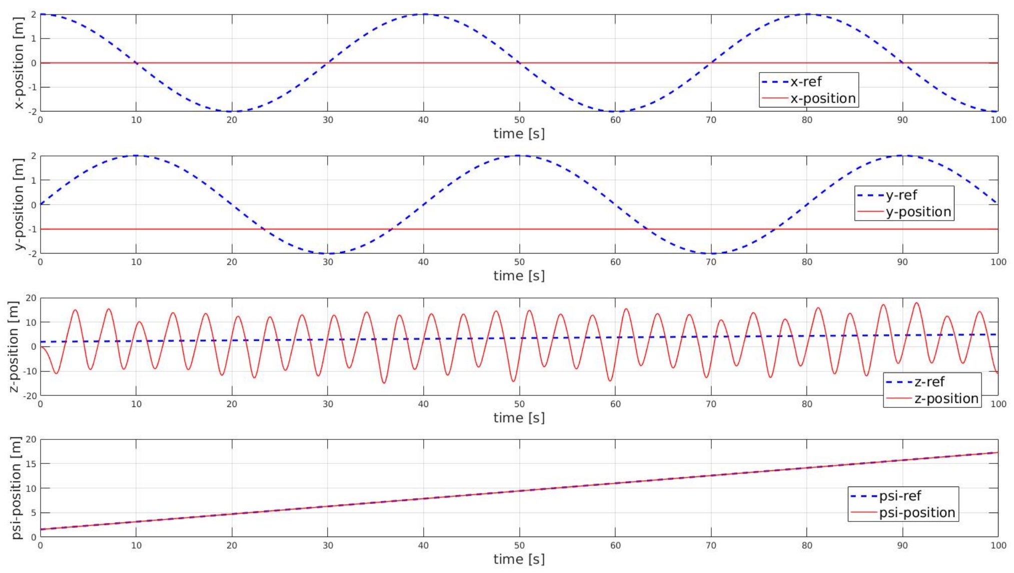

Figure 4.2: x, y, z and ψ values as a function of time

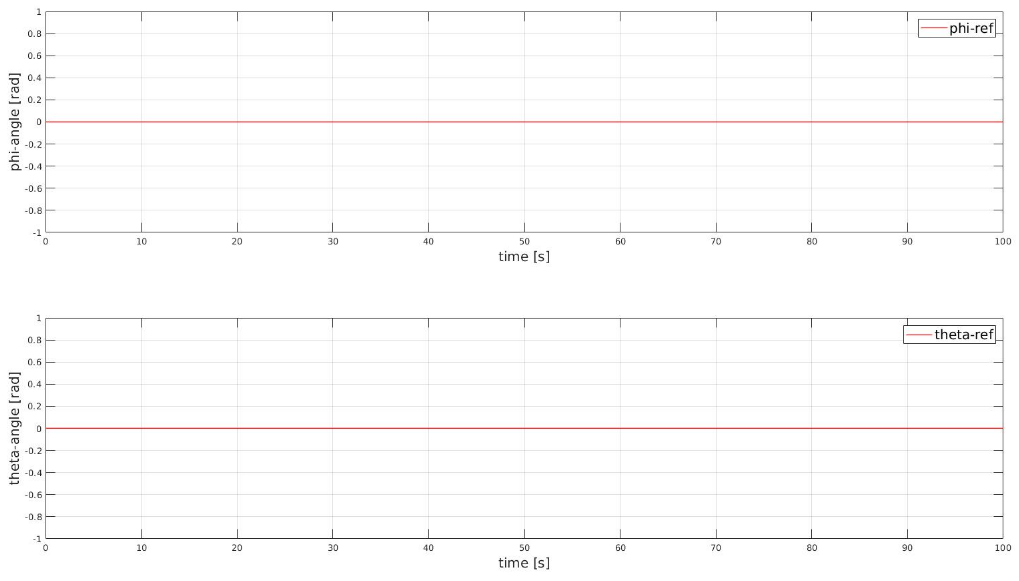

Figure 4.3: ϕ and θ values as a function of time

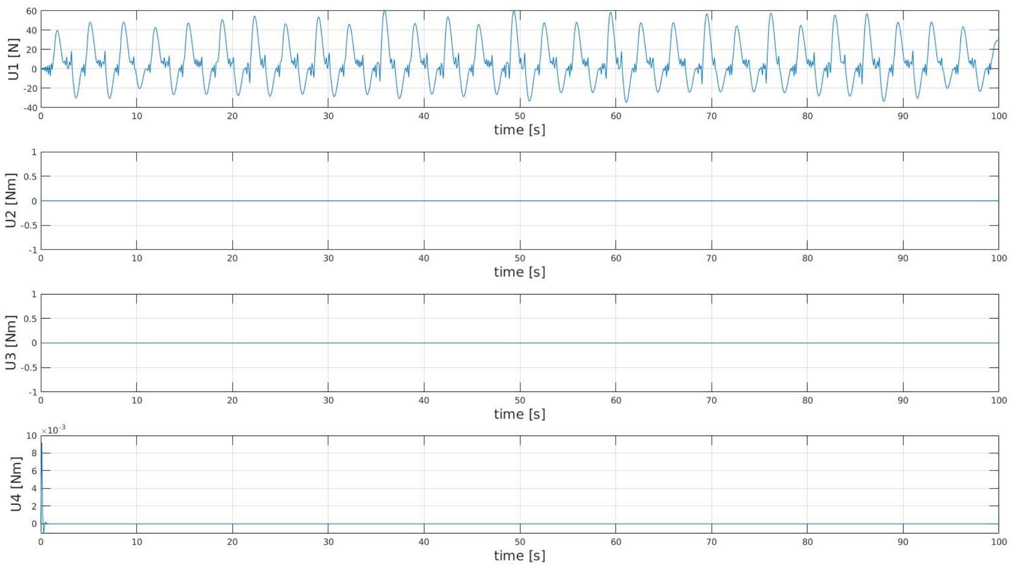

Figure 4.4: U1, U2, U3, U4 values as a function of time

4.2 Validation of the LPV-MPC and position controller - spiral

Since the proposed control strategy in Figure 3.1 failed to meet its objectives, a strategy in Figure 3.2 was implemented to control the UAV. Here, the position and angular variables were decoupled. The position controller, which uses the feedback linearization methodology, computes the necessary U1 input for the drone. However, it also computes the angles ϕ and θ that are necessary for the drone to reach in order for it to be able to reach its target position. These angles along with the ψ angle from the planner are then fed into the LPV-MPC controller as reference values. In order to give the LPV-MPC controller time to adjust the UAV's angles and reach the target orientation, the inner control loop has to work faster - in this case, four times faster. The horizon period for MPC was was also chosen to be 4 samples.

As it can be seen in Figures from 4.5 to 4.11, this control strategy has very high success. All the UAV's six degrees of freedom are tracked with very small errors. In tracking the x, y, z reference velocity values, one can observe strong overshoot at the beginning of the test period. That can be explained by the fact that the drone starts its journey from quite a long distance away from the trajectory. However, once it reaches the path that it needs to follow, the velocities of the UAV stabilize and track the reference values very smoothly.

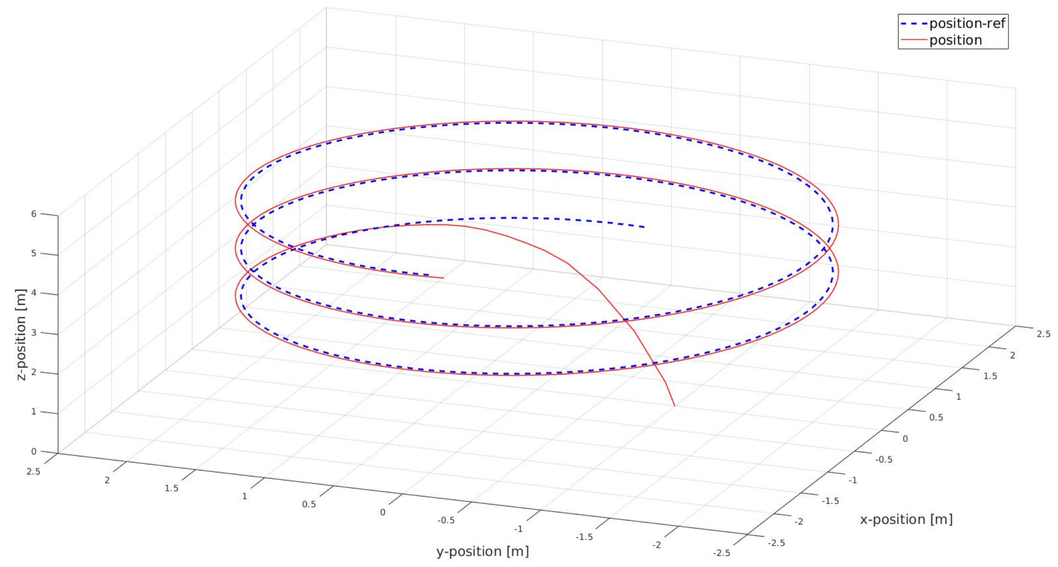

Figure 4.5: Flight trajectory - spiral

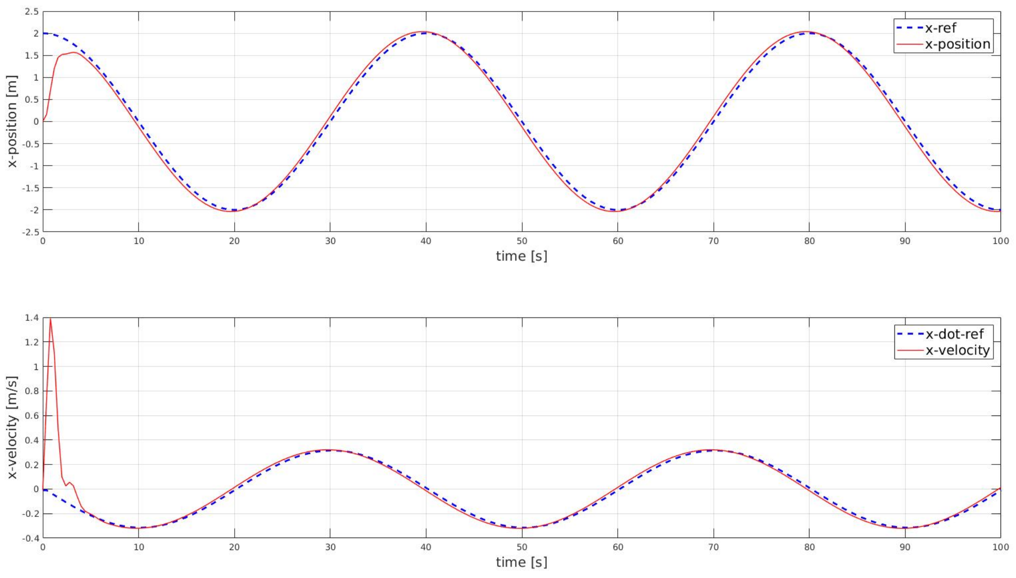

Figure 4.6: x and $\dot{x}$ values as a function of time

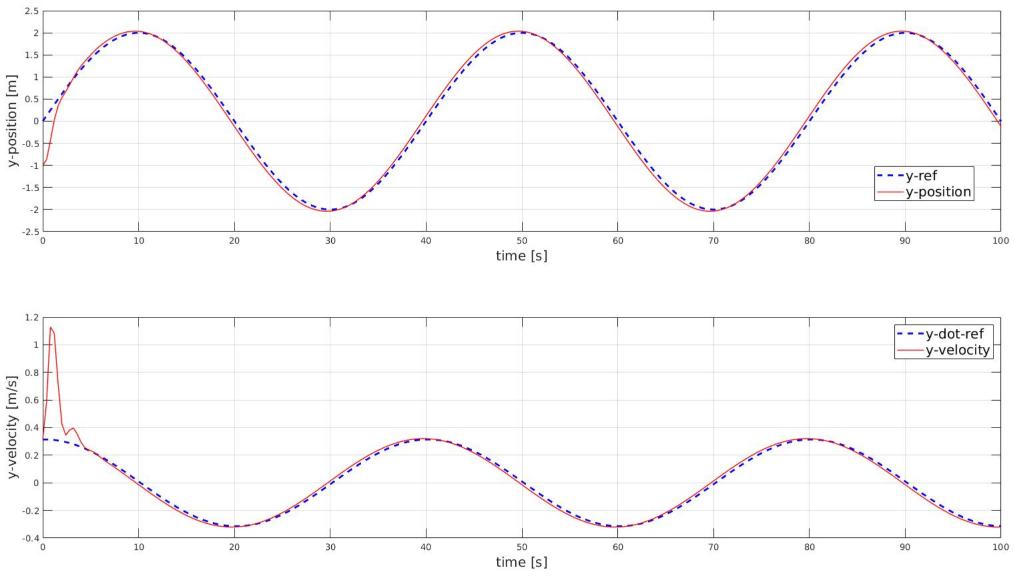

Figure 4.7: y and $\dot{y}$ values as a function of time

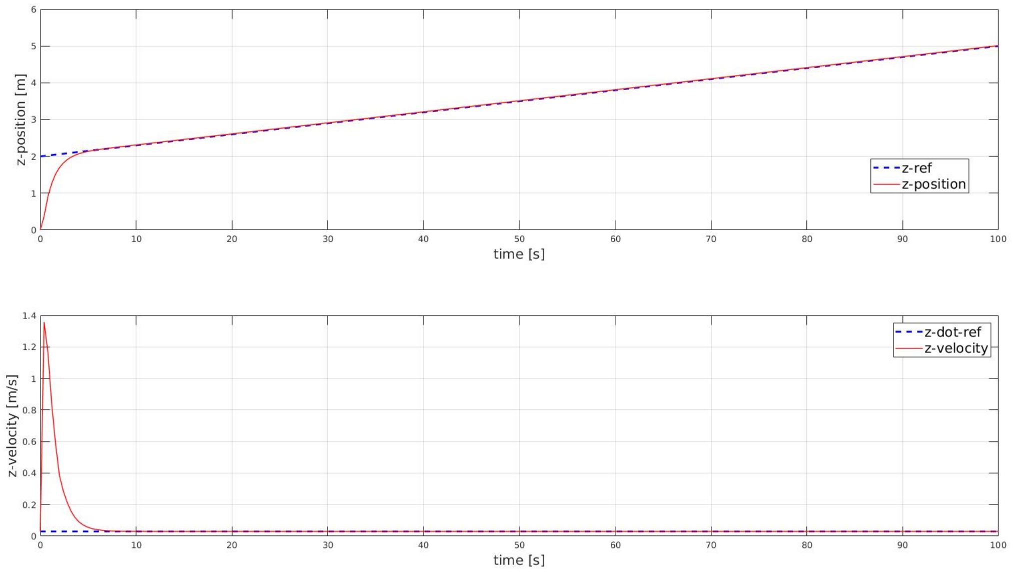

Figure 4.8: z and $\dot{z}$ values as a function of time

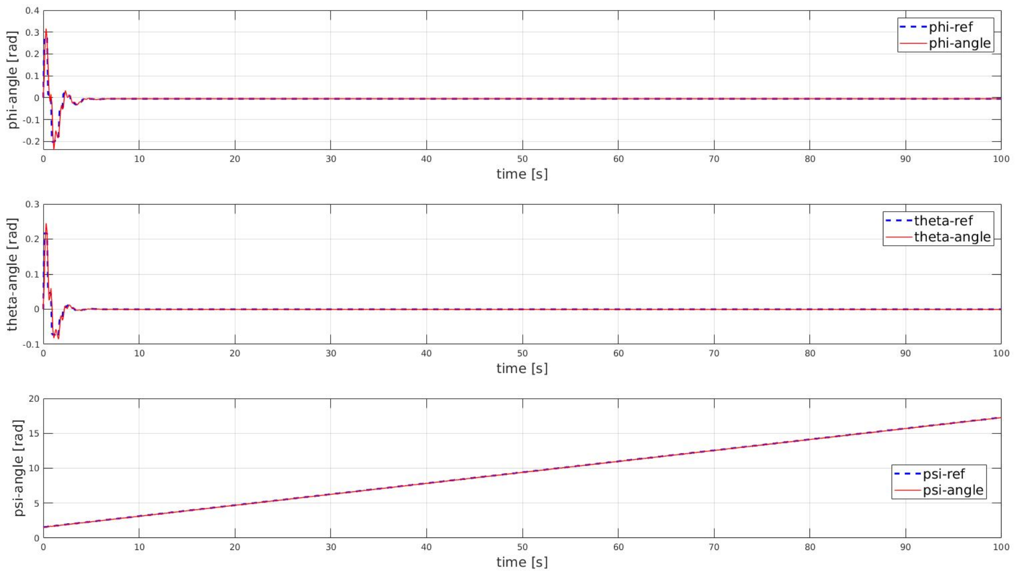

Figure 4.9: ϕ, θ, ψ values as a function of time

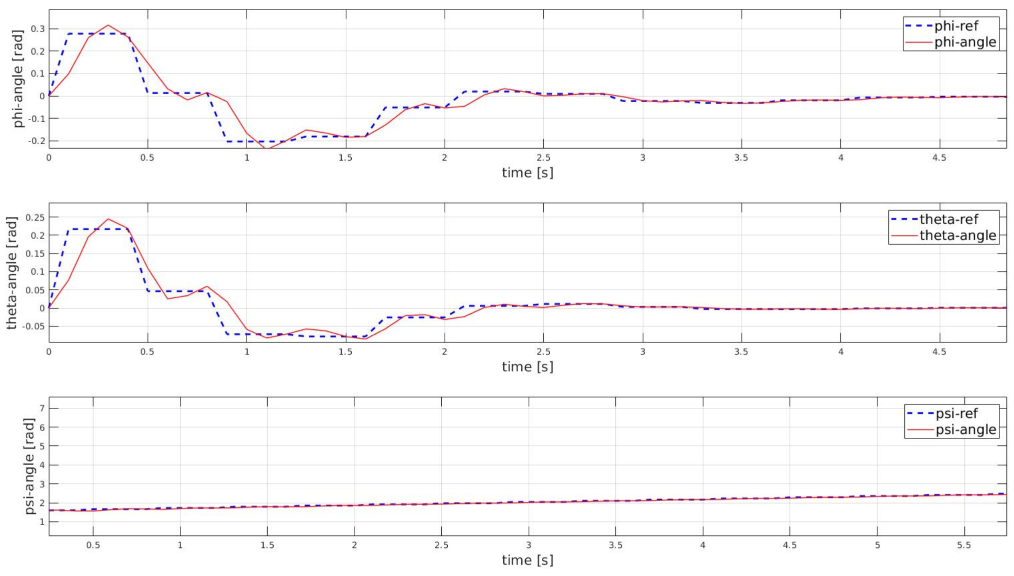

Figure 4.10: ϕ, θ, ψ values as a function of time - zoomed in

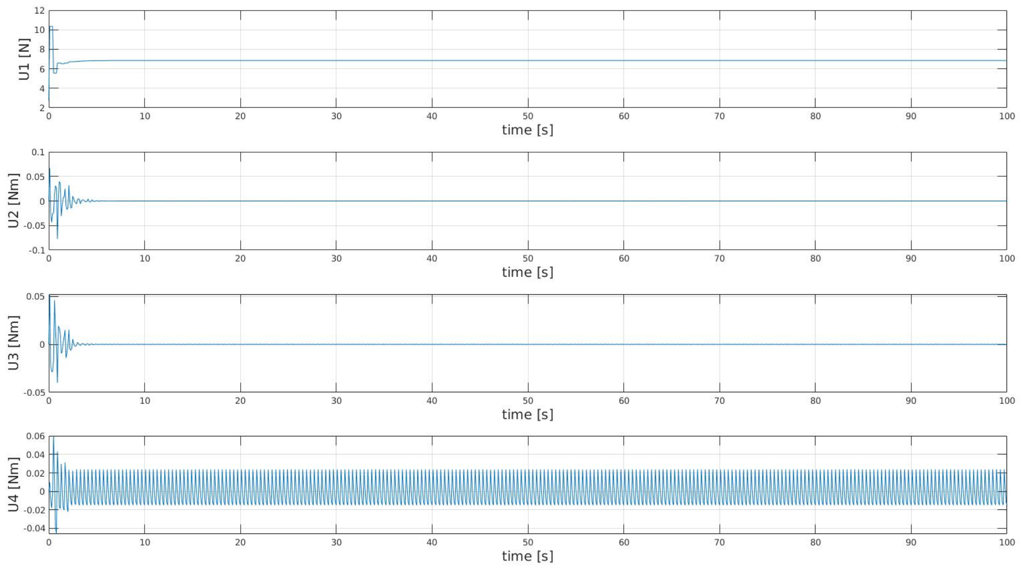

Figure 4.11: U1, U2, U3, U4 values as a function of time

Chapter 5

Conclusion

5.1 Summary of the results

The aim of this Master's thesis was to take the nonlinear mathematical model of a quadcopter and put it in the Linear Parameter Varying (LPV) form in order to be able to use the most basic Model Predictive Control (MPC) strategy, which was developed for linear systems. And then, the goal was to apply the MPC strategy and make the UAV track a given trajectory. It was first attempted to create one global LPV-MPC controller and control the drone that way; however, the attempt was unsuccessful, because there were strong nonlinearities in the LPV model that did not let the pitch and roll angles to be extracted from the model and formulated as control actions. As a result, these angles remained unaffected and unchanged, which in turn meant that the drone's x and y position coordinates were unaltered. The challenge was solved by decoupling the controller into two parts. The position controller, which uses the feedback linearization methodology, was responsible for controlling the position variables, and the attitude controller controlled the angles using the LPV-MPC control strategy. This strategy managed to control the drone with high precision.

5.2 Proposed future research work

The results in this thesis look promising. However, much work remains to be done in this subject. This work did not apply any constraints on the inputs, outputs, nor on the states of the UAV. The parameters of the drone used in this thesis belong to $AscTec\ Hummingbird$ [3]. The work in this thesis could be expanded by taking the maximum state, input and output values of the aforementioned quadcopter and integrating them into the LPV-MPC controller as the bounds for this controller.

This thesis does not deal with any unexpected disturbances either, which introduce uncertainty in model prediction. In addition, it does not take into account the noise that occurs in the sensors, that adds uncertainty in the measured states or outputs. The work could be expanded by introducing the aforementioned challenges into the system, and then modifying the LPV-MPC, and the position controller. Perhaps, a filter such as Kalman filter could be added to the control loop to make the tracking more robust. Along with that, also the weight matrices Q, S, R in MPC could be properly tuned. At this point, they are just identity matrices.

Finally, once all these additions are made, all the results could be tested on a real drone. The MATLAB® script could be translated into a C/C++ code and then loaded on a UAV testing how the control strategy tracks the trajectories from this thesis in real life.

Appendix A

Validation using additional trajectories

A.1 Validation of the LPV-MPC and position controller - expanding spiral

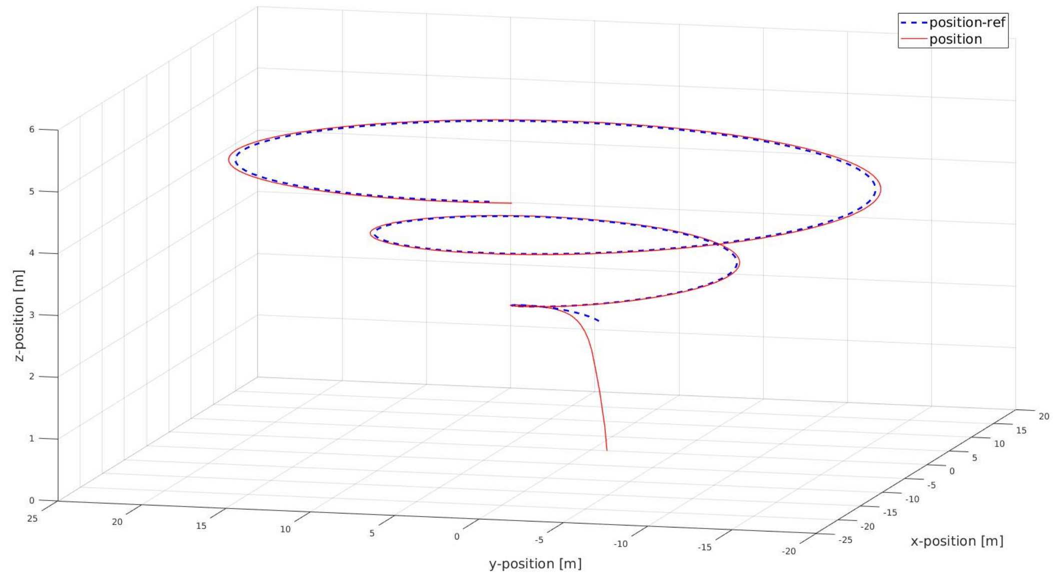

Figure A.1: Flight trajectory - extended spiral

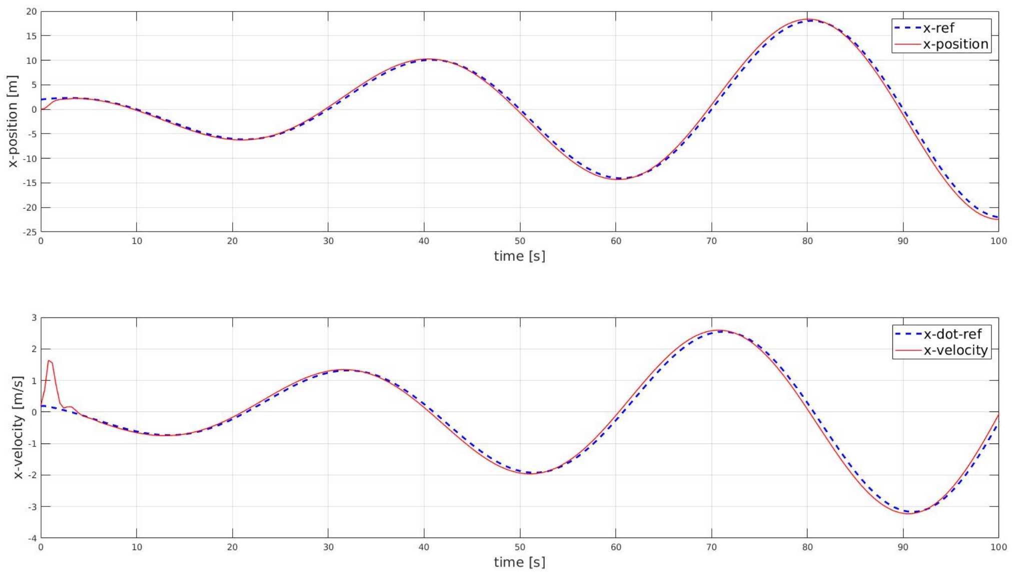

Figure A.2: x and $\dot{x}$ values as a function of time

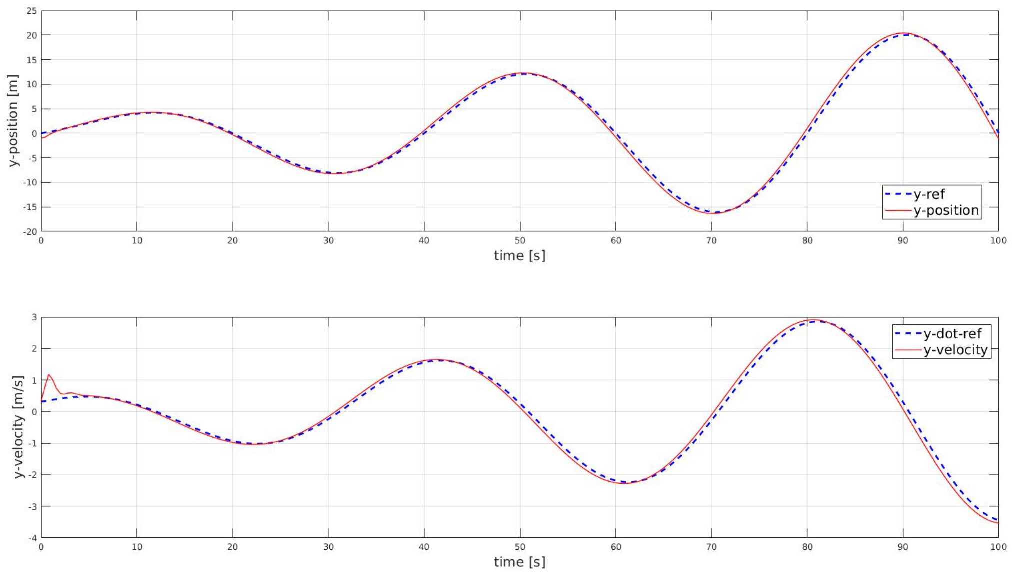

Figure A.3: y and $\dot{y}$ values as a function of time

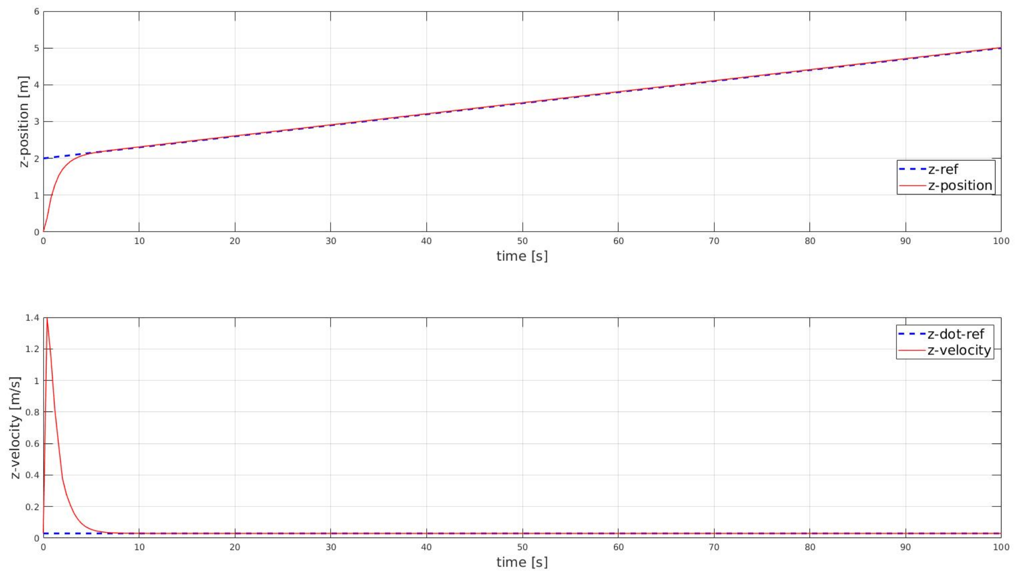

Figure A.4: z and $\dot{z}$ values as a function of time

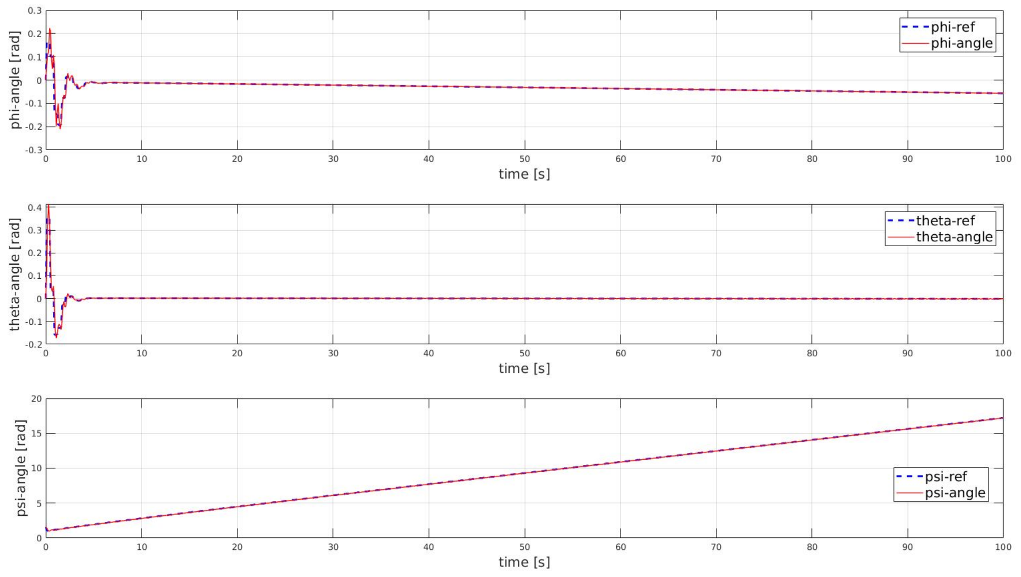

Figure A.5: ϕ, θ, ψ values as a function of time

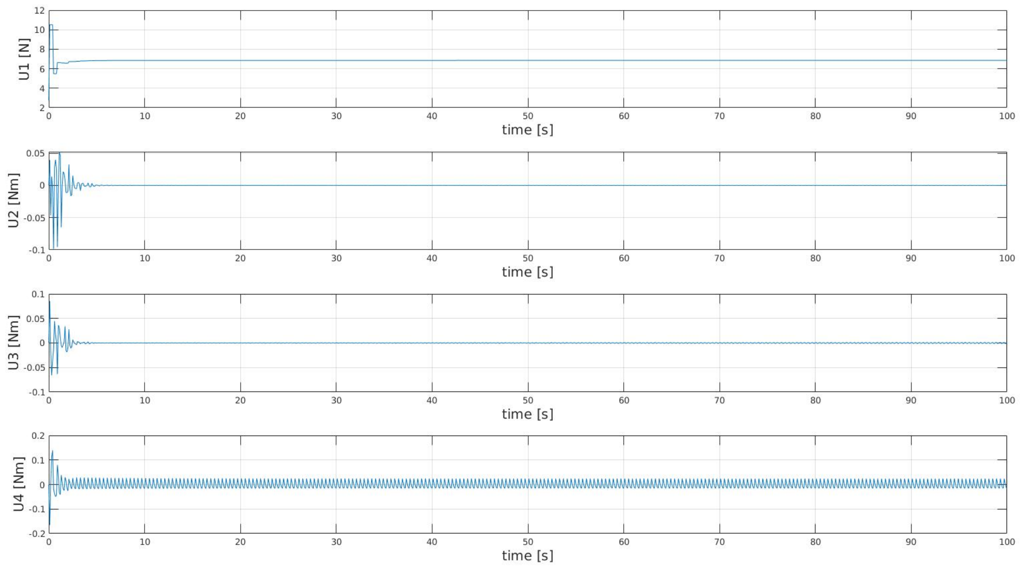

Figure A.6: U1, U2, U3, U4 values as a function of time

A.2 Validation of the LPV-MPC and position controller - straight line in 3D

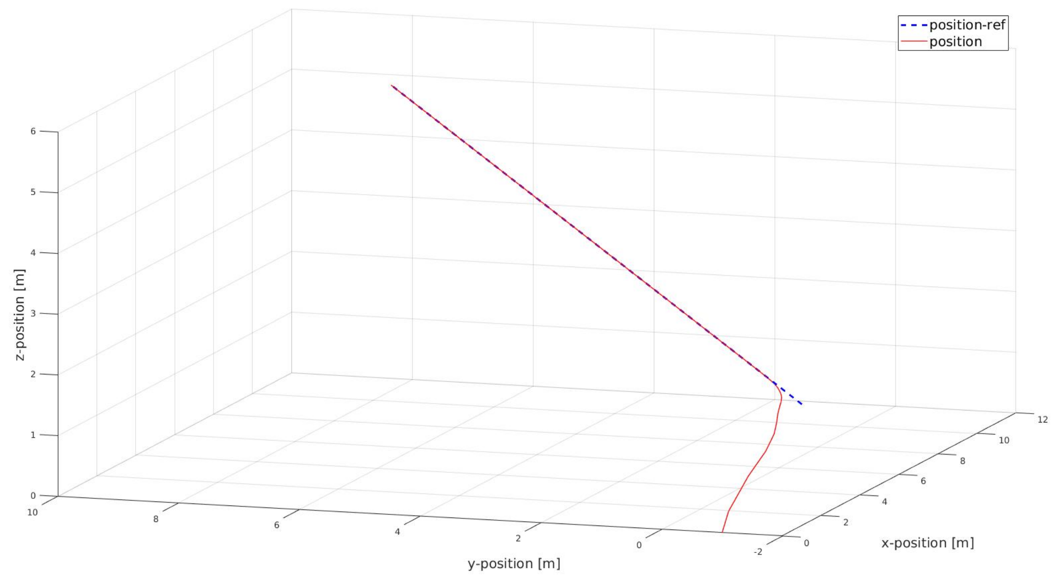

Figure A.7: Flight trajectory - straight line in 3D

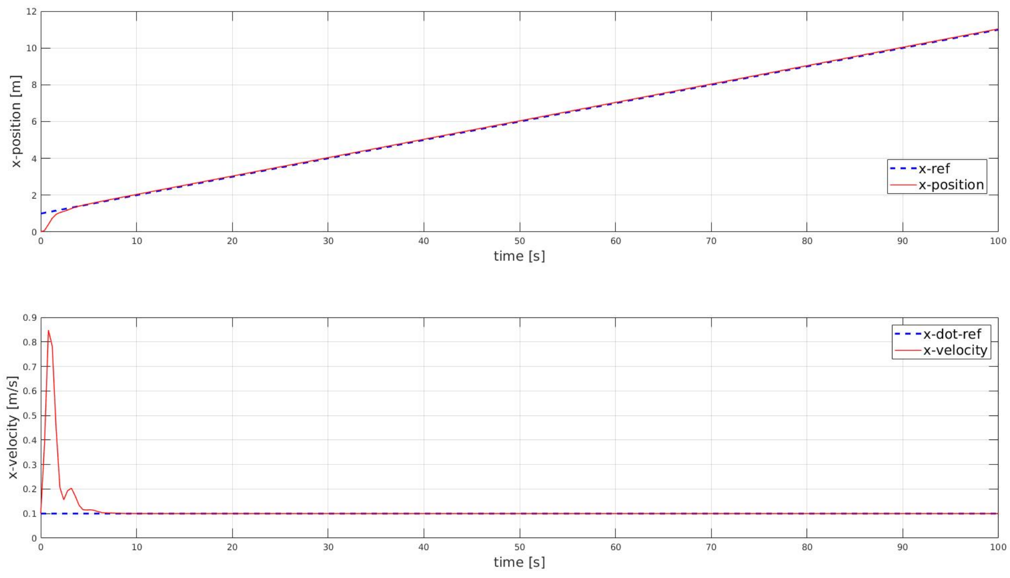

Figure A.8: x and $\dot{x}$ values as a function of time

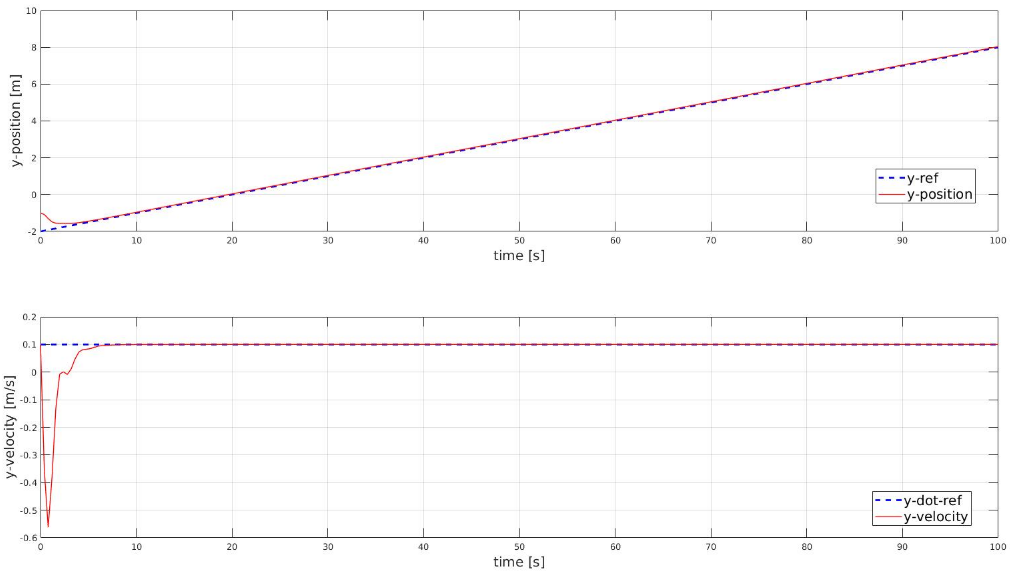

Figure A.9: y and $\dot{y}$ values as a function of time

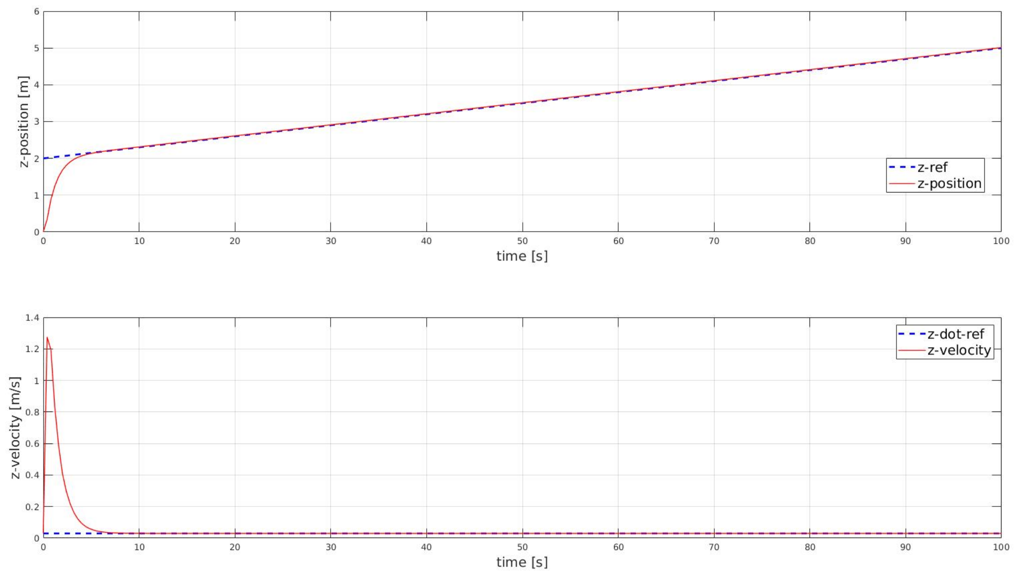

Figure A.10: z and $\dot{z}$ values as a function of time

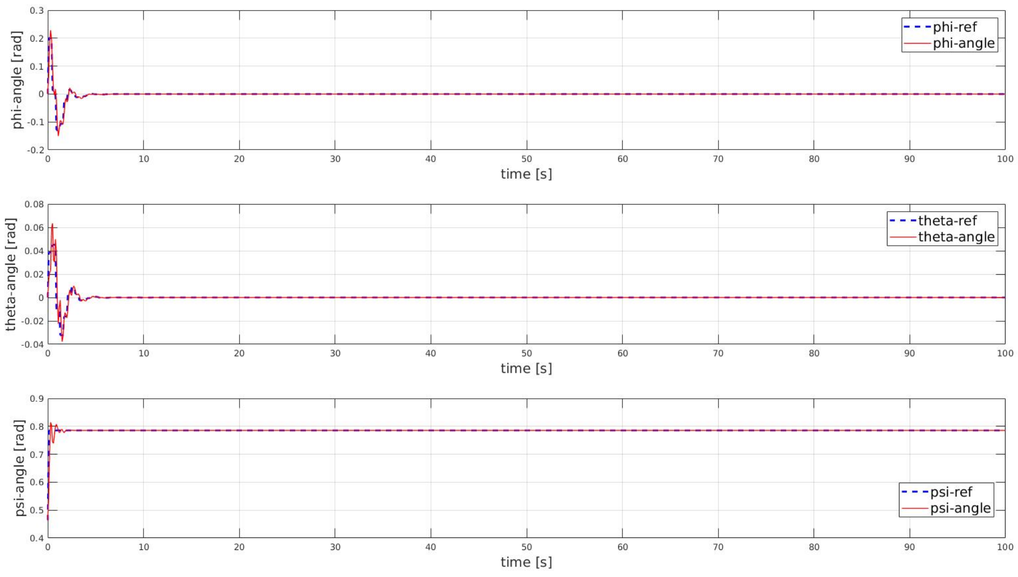

Figure A.11: ϕ, θ, ψ values as a function of time

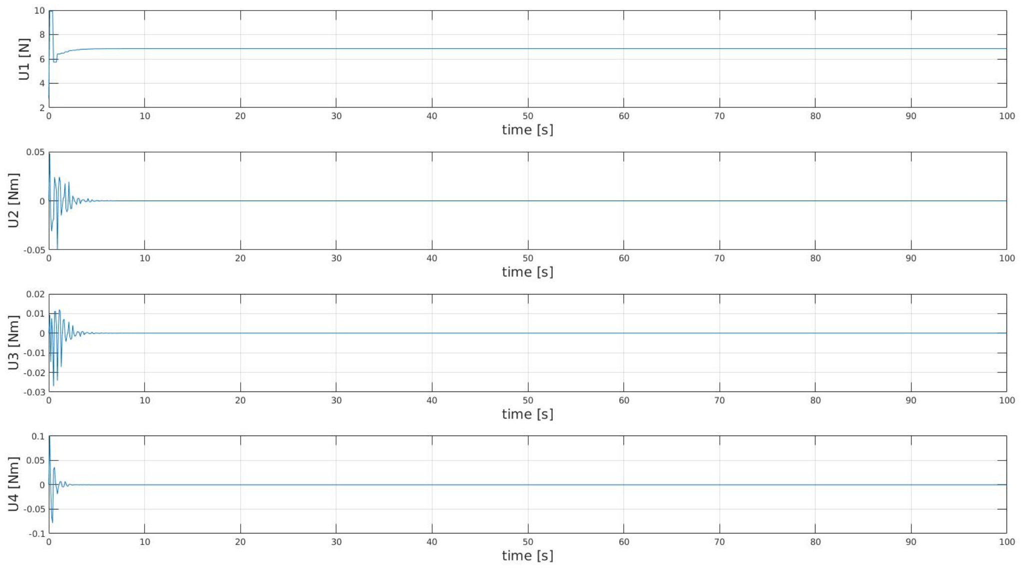

Figure A.12: U1, U2, U3, U4 values as a function of time

A.3 Validation of the LPV-MPC and position controller - wavy line in 3D

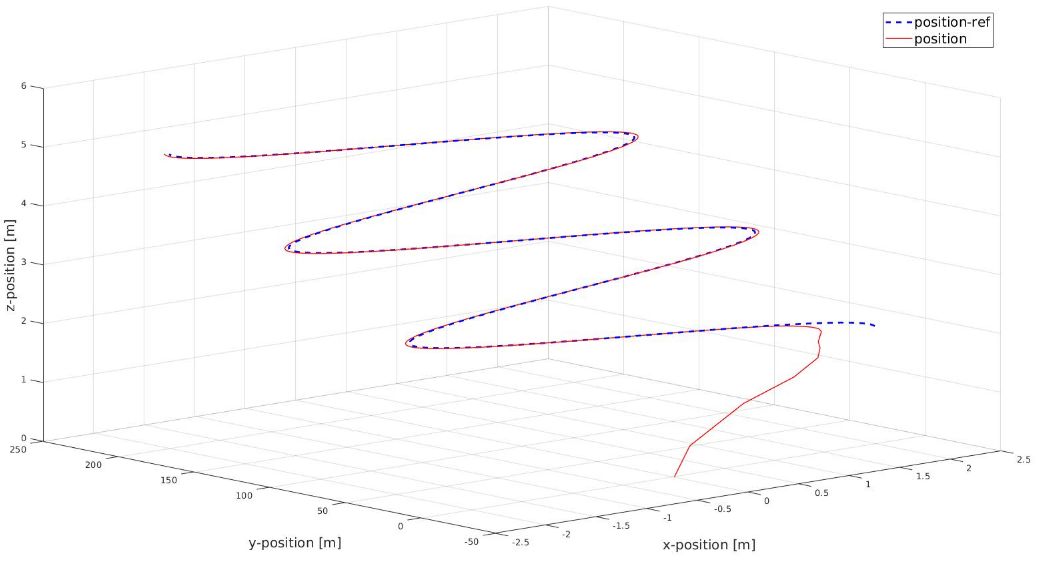

Figure A.13: Flight trajectory - wavy line in 3D

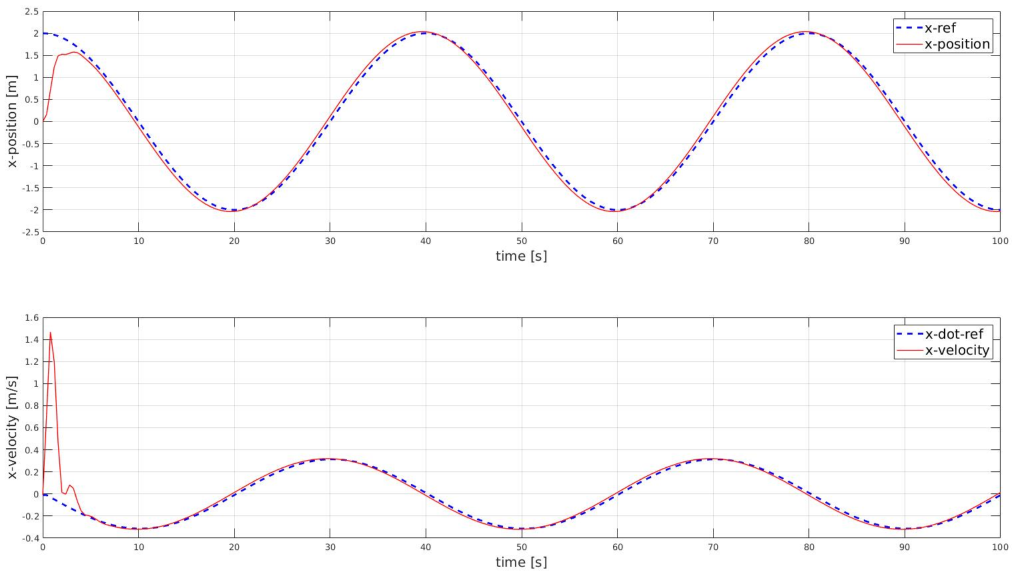

Figure A.14: x and $\dot{x}$ values as a function of time

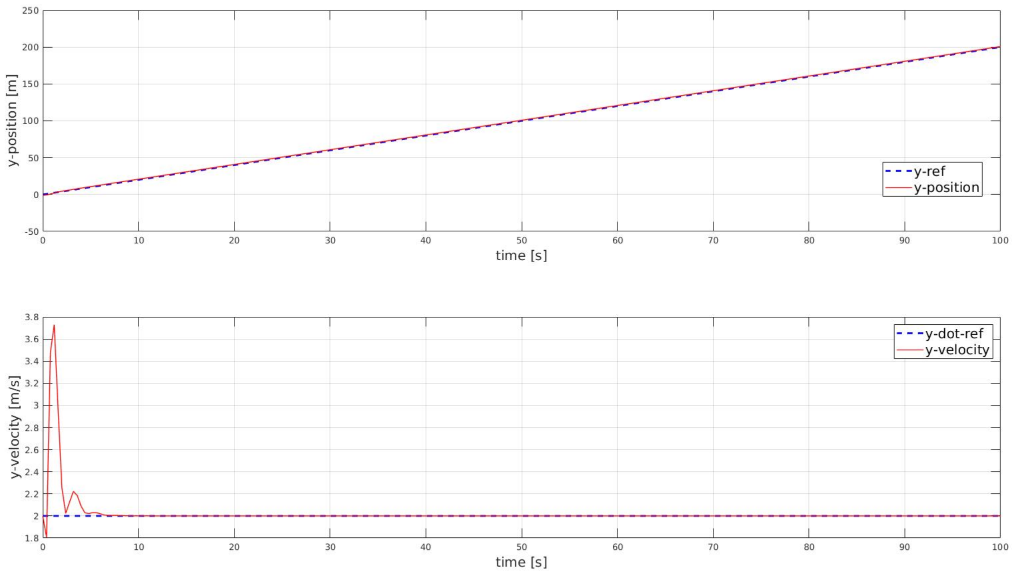

Figure A.15: y and $\dot{y}$ values as a function of time

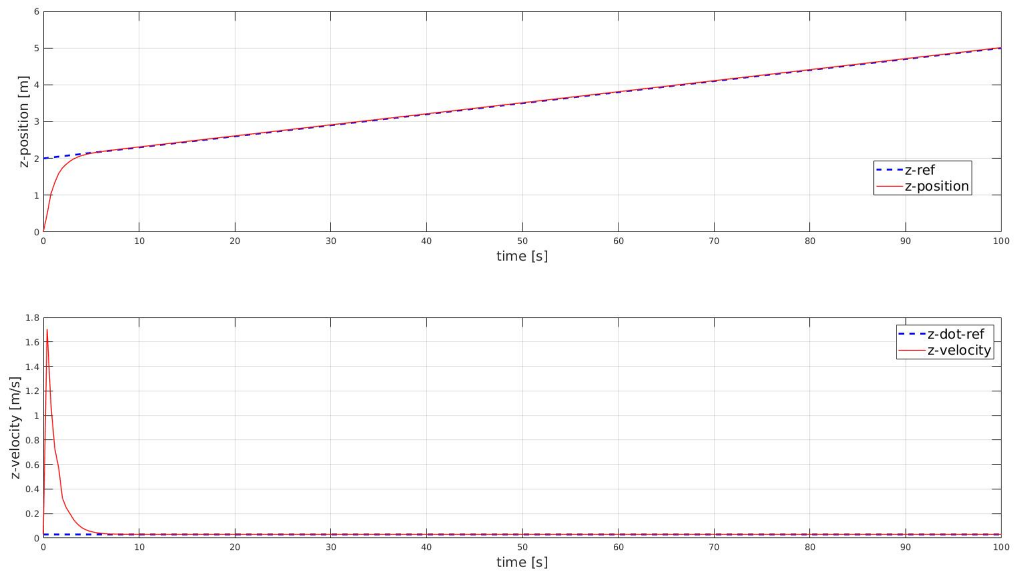

Figure A.16: z and $\dot{z}$ values as a function of time

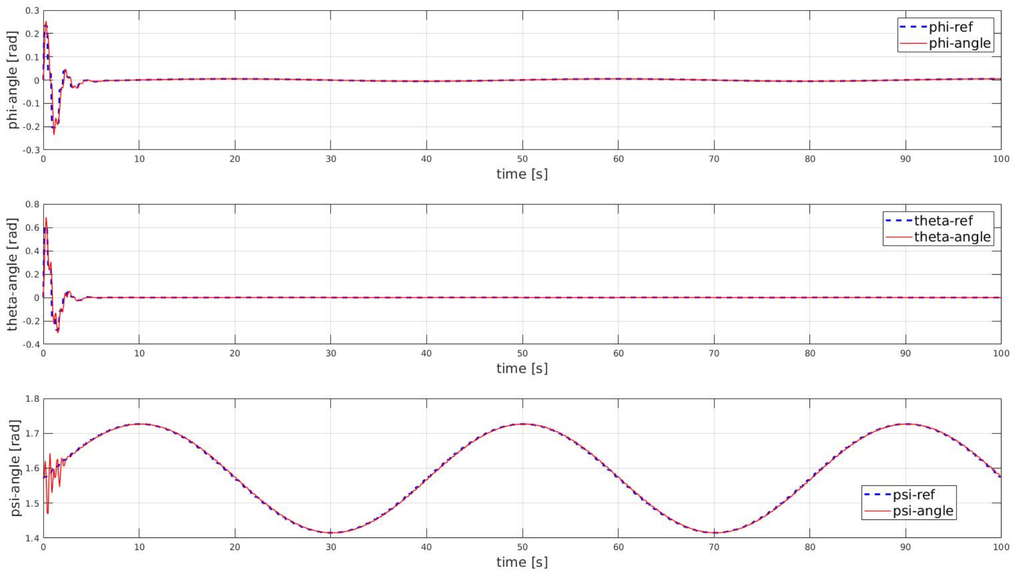

Figure A.17: ϕ, θ, ψ values as a function of time

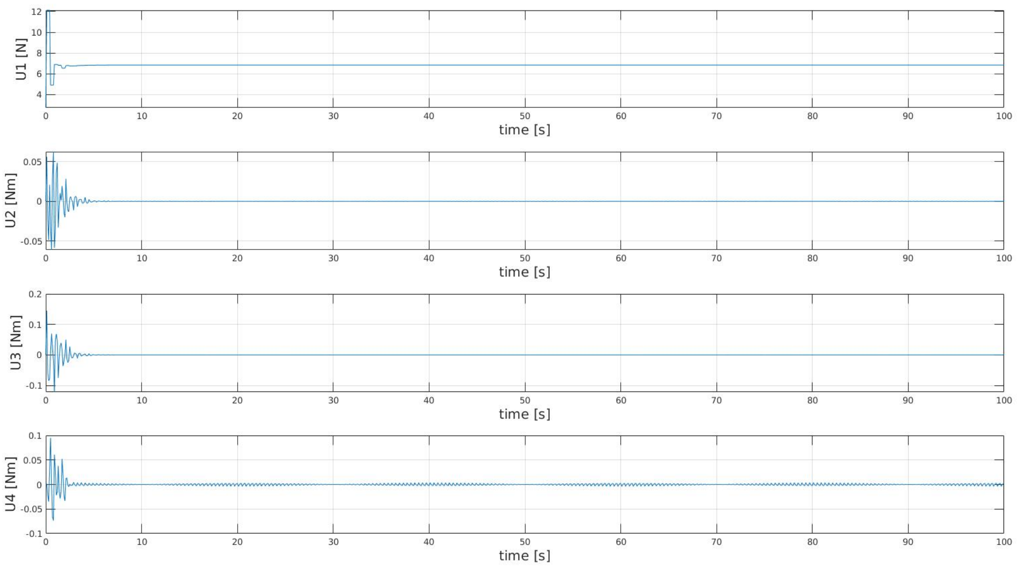

Figure A.18: U1, U2, U3, U4 values as a function of time

A.4 Validation of the LPV-MPC and position controller - the crown

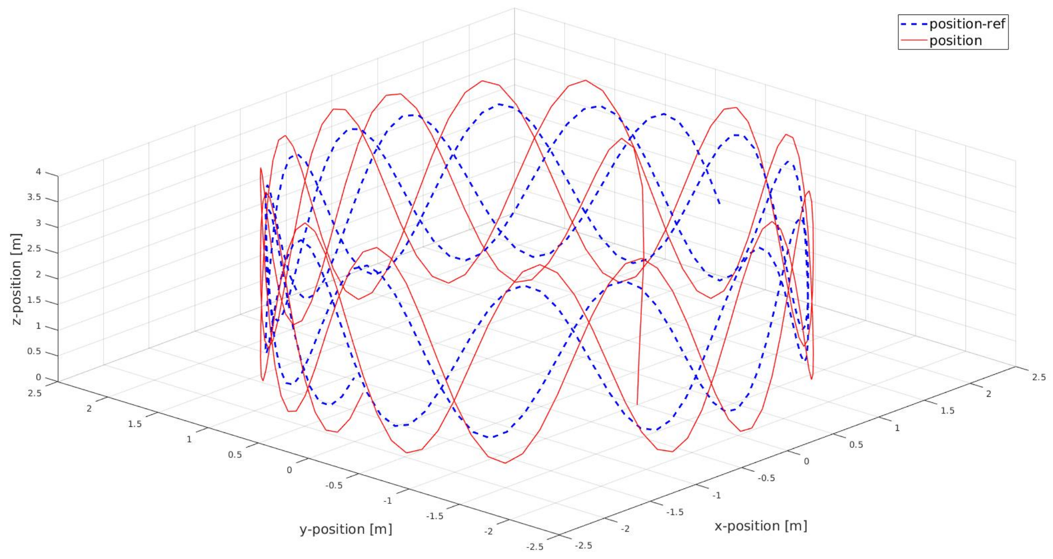

Figure A.19: Flight trajectory - the crown

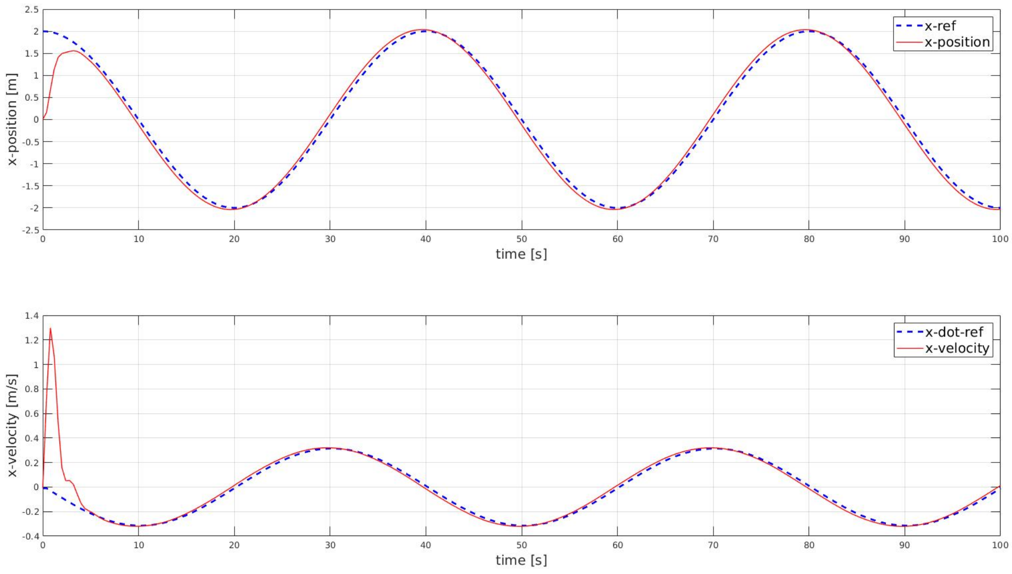

Figure A.20: x and $\dot{x}$ values as a function of time

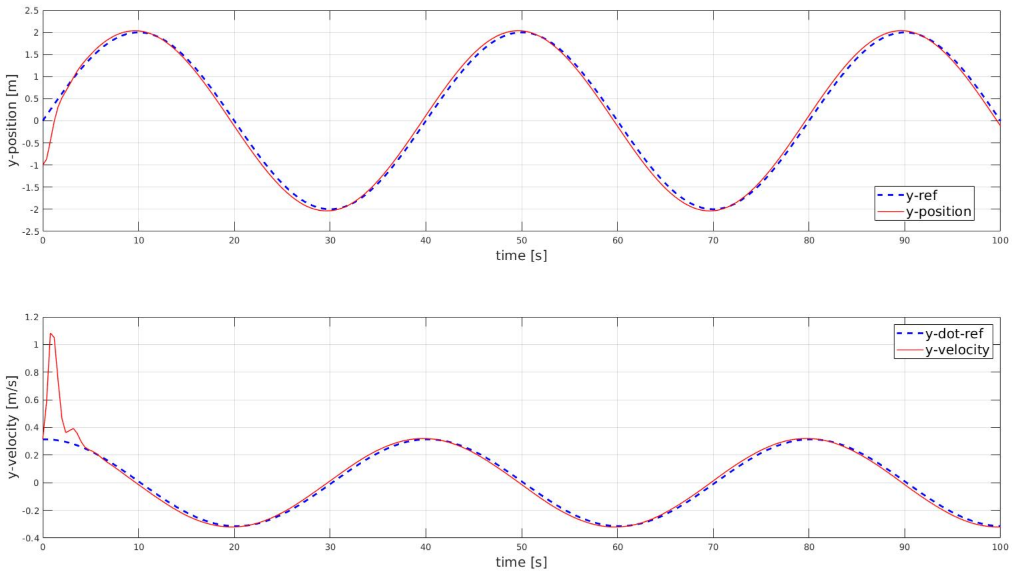

Figure A.21: y and $\dot{y}$ values as a function of time

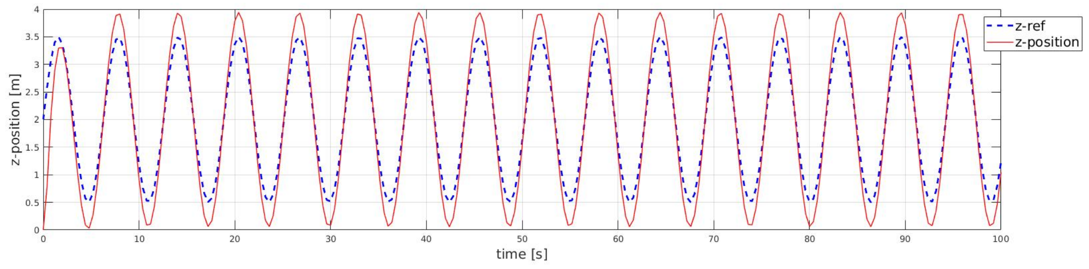
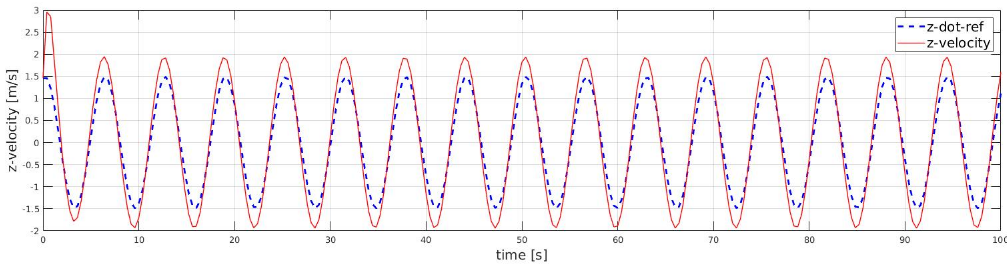

Figure A.22: z and $\dot{z}$ values as a function of time

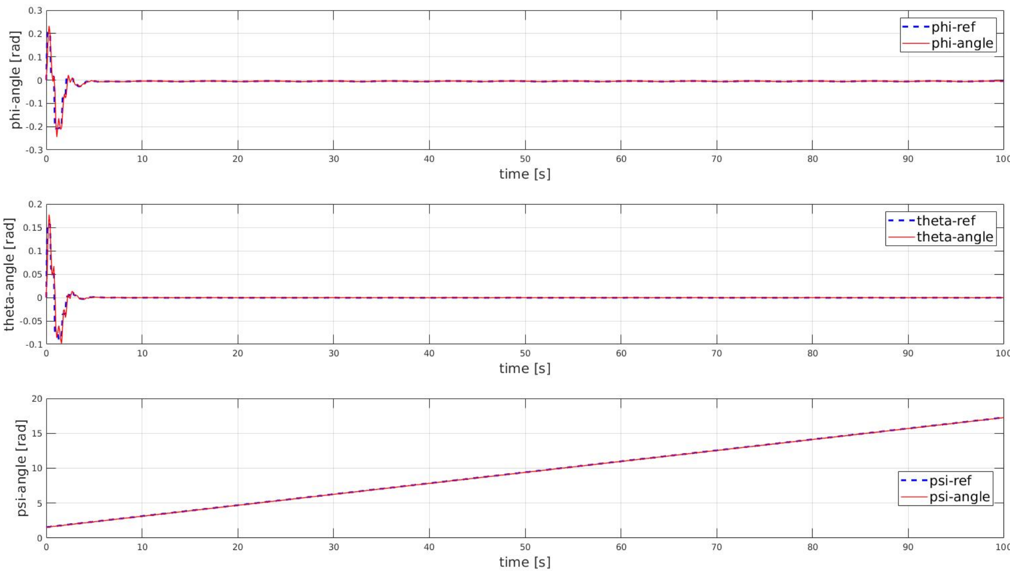

Figure A.23: ϕ, θ, ψ values as a function of time

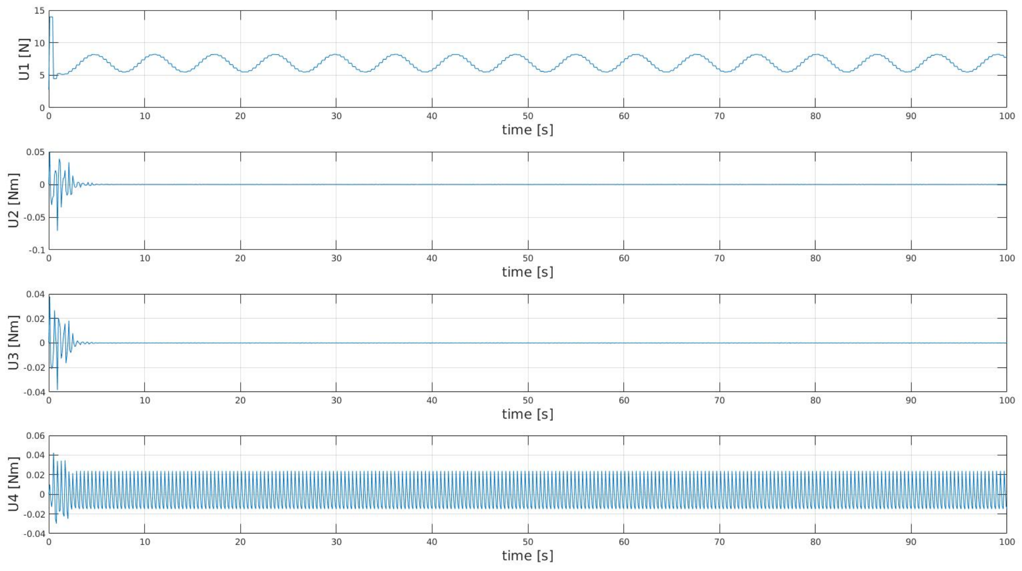

Figure A.24: U1, U2, U3, U4 values as a function of time

Appendix B

The implementation script

B.1 Function: MAIN_LPV_MPC_drone.m

```
clear all
close all
clc

%% Main file for controllering the drone

% The controller consists of the position controller (state feedback
% linearization) - outer loop AND attitude controller (LPV-MPC) -
    inner
% loop with faster dynamics

% The relevant function files to this main file are the following:
    % initial_constants.m
    % LPV_cont_discrete.m
    % MPC_simplification.m
    % nonlinear_drone_model.m
    % trajectory_generator.m
    % pos_controller.m
```

```

%% Load the constant values
constants=initial_constants();
Ts=constants{7};
controlled_states=constants{14}; % number of controlled states in
    this script
innerDyn_length=constants{18}; % Number of inner control loop
    iterations

%% Generate the reference signals
t = 0:Ts*innerDyn_length:100;
t_angles=(0:Ts:t(end))';
r = 2;
f=0.025;
height_i=2;
height_f=5;
[X_ref,X_dot_ref,Y_ref,Y_dot_ref,Z_ref,Z_dot_ref,psi_ref]=
    trajectory_generator(t,r,f,height_i,height_f);
plotl=length(t); % Number of outer control loop iterations

%% Load the initial state vector

ut=0;
vt=0;
wt=0;
pt=0;
qt=0;
rt=0;
xt=0;%X_ref(1,2); % Initial translational position
yt=-1;%Y_ref(1,2); % Initial translational position
zt=0;%Z_ref(1,2); % Initial translational position
phit=0;      % Initial angular position
```

```
thetat=0;  % Initial angular position
psit=psi_ref(1,2);     % Initial angular position

states=[ut,vt,wt,pt,qt,rt,xt,yt,zt,phit,thetat,psit];
states_total=states;

% Assume that first Phi_ref, Theta_ref, Psi_ref are equal to the
    first
% phit, thetat, psit
ref_angles_total=[phit,thetat,psit];
velocityXYZ_total=[X_dot_ref(1,2),Y_dot_ref(1,2),Z_dot_ref(1,2)];
%% Initial drone state

omega1=3000; % rad/s at t = -1 s
omega2=3000; % rad/s at t = -1 s
omega3=3000; % rad/s at t = -1 s
omega4=3000; % rad/s at t = -1 s

ct = constants{11};
cq = constants{12};
l  = constants{13};

U1=ct*(omega1^2+omega2^2+omega3^2+omega4^2); % Input at t = -1 s
U2=ct*l*(omega4^2-omega2^2); % Input at t = -1 s
U3=ct*l*(omega3^2-omega1^2); % Input at t = -1 s
U4=cq*(-omega1^2+omega2^2-omega3^2+omega4^2); % Input at t = -1 s

UTotal=[U1,U2,U3,U4];% 4 inputs

global omega_total
omega_total=-omega1+omega2-omega3+omega4;

```

```
%% Start the global controller

for i_global = 1:plotl-1


    %% Implement the position controller (state feedback
        linearization)

    [phi_ref, theta_ref, U1]=pos_controller(X_ref(i_global+1,2),
        X_dot_ref(i_global+1,2),Y_ref(i_global+1,2),Y_dot_ref(i_global
        +1,2),Z_ref(i_global+1,2),Z_dot_ref(i_global+1,2),psi_ref(
        i_global+1,2),states);


    Phi_ref=phi_ref*ones(innerDyn_length+1,1);
    Theta_ref=theta_ref*ones(innerDyn_length+1,1);
    Psi_ref=psi_ref(i_global+1,2)*ones(innerDyn_length+1,1);

    ref_angles_total=[ref_angles_total;Phi_ref(2:end) Theta_ref(2:end
        ) Psi_ref(2:end)];

    %% Create the reference vector

    refSignals=zeros(length(Phi_ref(:,1))*controlled_states,1);
    % Format: refSignals=[Phi_ref;Theta_ref;Psi_ref;Phi_ref; ... etc]
         x inner
    % loop frequency per one set of position controller outputs
    k_ref_local=1;
    for i = 1:controlled_states:length(refSignals)
        refSignals(i)=Phi_ref(k_ref_local,1);
        refSignals(i+1)=Theta_ref(k_ref_local,1);
        refSignals(i+2)=Psi_ref(k_ref_local,1);
        k_ref_local=k_ref_local+1;
    end


    k_ref_local=1; % for reading reference signals
    hz = constants{15}; % horizon period
    for i =1:innerDyn_length
        %% Generate discrete LPV Ad, Bd, Cd, Dd matrices
        [Ad, Bd, Cd, Dd, x_dot, y_dot, z_dot, phit, phi_dot, thetat, theta_dot, psit, psi_dot]=LPV_cont_discrete(states);
        velocityXYZ_total=[velocityXYZ_total;[x_dot, y_dot, z_dot]];


        %% Generating the current state and the reference vector
        x_aug_t=[phit;phi_dot;thetat;theta_dot;psit;psi_dot;U2;U3;U4];

        k_ref_local=k_ref_local+controlled_states;

        % Start counting from the second sample period:
        % r=refSignals(Phi_ref_2;Theta_ref_2;Psi_ref_2;Phi_ref_3...) etc.
        if k_ref_local+controlled_states*hz-1 <= length(refSignals)
            r=refSignals(k_ref_local:k_ref_local+controlled_states*hz-1);
        else
            r=refSignals(k_ref_local:length(refSignals));
            hz=hz-1;
        end

        %% Generate simplification matrices for the cost function
        [Hdb,Fdbt,Cdb,Adc] = MPC_simplification(Ad,Bd,Cd,Dd,hz);
```

```

%% Calling the optimizer (quadprog)

% Cost function in quadprog: min(du)*1/2*du'Hdb*du+f'du
% f'=[x_t', r']*Fdbt
ft=[x_aug_t',r']*Fdbt;

% Hdb must be positive definite for the problem to have
    finite minimum.
% Check if matrix Hdb in the cost function is positive
    definite.
[~,p] = chol(Hdb);
if p~=0
    disp('Hdb is NOT positive definite');
end

% Call the solver
options = optimset('Display', 'off');
lb=constants{16};
ub=constants{17};
[du,fval]=quadprog(Hdb,ft,[],[],[],[],[],[],[],options);

% Update the real inputs
U2=U2+du(1);
U3=U3+du(2);
U4=U4+du(3);

UTotal=[UTotal;U1,U2,U3,U4];

% Compute the new omegas based on the new U-s.
U1C=U1/ct;
U2C=U2/(ct*l);
```

```
          U3C=U3/(ct*l);
          U4C=U4/cq;

          omega4P2=(U1C+2*U2C+U4C)/4;
          omega3P2=(-U4C+2*omega4P2-U2C+U3C)/2;
          omega2P2=omega4P2-U2C;
          omega1P2=omega3P2-U3C;

          omega1=sqrt(omega1P2);
          omega2=sqrt(omega2P2);
          omega3=sqrt(omega3P2);
          omega4=sqrt(omega4P2);

          % Compute the total omega
          omega_total=-omega1+omega2-omega3+omega4;

          % Simulate the new states
          T = (Ts)*(i-1):(Ts)/30:Ts*(i-1)+(Ts);
          [T,x]=ode45(@(t,x) nonlinear_drone_model(t,x,[U1,U2,U3,U4]),T
              ,states);
          states=x(end,:);
          states_total=[states_total;states];

          imaginary_check=imag(states)~=0;
          imaginary_check_sum=sum(imaginary_check);
          if imaginary_check_sum~=0
              disp('Imaginary part exists - something is wrong');
          end
      end
end

%% Plot the trajectory
```

```

% Trajectory
figure;
plot3(X_ref(:,2),Y_ref(:,2),Z_ref(:,2),'--b','LineWidth',2)
hold on
plot3(states_total(1:innerDyn_length:end,7),states_total(1:
    innerDyn_length:end,8),states_total(1:innerDyn_length:end,9),'r','
    LineWidth',1)
grid on;
xlabel('x-position [m]','FontSize',15)
ylabel('y-position [m]','FontSize',15)
zlabel('z-position [m]','FontSize',15)
legend({'position-ref','position'},'Location','northeast','FontSize'
    ,15)

%% Plot the positions and velocities individually

% X and X_dot
figure;
subplot(2,1,1)
plot(t(1:plotl),X_ref(1:plotl,2),'--b','LineWidth',2)
hold on
subplot(2,1,1)
plot(t(1:plotl),states_total(1:innerDyn_length:end,7),'r','LineWidth'
    ,1)
grid on
xlabel('time [s]','FontSize',15)
ylabel('x-position [m]','FontSize',15)
legend({'x-ref','x-position'},'Location','northeast','FontSize',15)
subplot(2,1,2)
plot(t(1:plotl),X_dot_ref(1:plotl,2),'--b','LineWidth',2)
hold on
```

```matlab
subplot(2,1,2)
plot(t(1:plotl),velocityXYZ_total(1:innerDyn_length:end,1),'r','
    LineWidth',1)
grid on
xlabel('time [s]','FontSize',15)
ylabel('x-velocity [m/s]','FontSize',15)
legend({'x-dot-ref','x-velocity'},'Location','northeast','FontSize'
    ,15)

% Y and Y_dot
figure;
subplot(2,1,1)
plot(t(1:plotl),Y_ref(1:plotl,2),'--b','LineWidth',2)
hold on
subplot(2,1,1)
plot(t(1:plotl),states_total(1:innerDyn_length:end,8),'r','LineWidth'
    ,1)
grid on
xlabel('time [s]','FontSize',15)
ylabel('y-position [m]','FontSize',15)
legend({'y-ref','y-position'},'Location','northeast','FontSize',15)
subplot(2,1,2)
plot(t(1:plotl),Y_dot_ref(1:plotl,2),'--b','LineWidth',2)
hold on
subplot(2,1,2)
plot(t(1:plotl),velocityXYZ_total(1:innerDyn_length:end,2),'r','
    LineWidth',1)
grid on
xlabel('time [s]','FontSize',15)
ylabel('y-velocity [m/s]','FontSize',15)
legend({'y-dot-ref','y-velocity'},'Location','northeast','FontSize'
    ,15)
```

```

% Z and Z_dot
figure;
subplot(2,1,1)
plot(t(1:plotl),Z_ref(1:plotl,2),'--b','LineWidth',2)
hold on
subplot(2,1,1)
plot(t(1:plotl),states_total(1:innerDyn_length:end,9),'r','LineWidth'
    ,1)
grid on
xlabel('time [s]','FontSize',15)
ylabel('z-position [m]','FontSize',15)
legend({'z-ref','z-position'},'Location','northeast','FontSize',15)
subplot(2,1,2)
plot(t(1:plotl),Z_dot_ref(1:plotl,2),'--b','LineWidth',2)
hold on
subplot(2,1,2)
plot(t(1:plotl),velocityXYZ_total(1:innerDyn_length:end,3),'r','
    LineWidth',1)
grid on
xlabel('time [s]','FontSize',15)
ylabel('z-velocity [m/s]','FontSize',15)
legend({'z-dot-ref','z-velocity'},'Location','northeast','FontSize'
    ,15)

%% Plot the angles individually

% Phi
figure;
subplot(3,1,1)
plot(t_angles(1:length(ref_angles_total(:,1))),ref_angles_total(:,1),
    '--b','LineWidth',2)
```

```
hold on
subplot(3,1,1)
plot(t_angles(1:length(states_total(:,10))),states_total(:,10),'r','
    LineWidth',1)
grid on
xlabel('time [s]','FontSize',15)
ylabel('phi-angle [rad]','FontSize',15)
legend({'phi-ref','phi-angle'},'Location','northeast','FontSize',15)

% Theta
subplot(3,1,2)
plot(t_angles(1:length(ref_angles_total(:,2))),ref_angles_total(:,2),
    '--b','LineWidth',2)
hold on
subplot(3,1,2)
plot(t_angles(1:length(states_total(:,11))),states_total(:,11),'r','
    LineWidth',1)
grid on
xlabel('time [s]','FontSize',15)
ylabel('theta-angle [rad]','FontSize',15)
legend({'theta-ref','theta-angle'},'Location','northeast','FontSize'
    ,15)

% Psi
subplot(3,1,3)
plot(t_angles(1:length(ref_angles_total(:,3))),ref_angles_total(:,3),
    '--b','LineWidth',2)
hold on
subplot(3,1,3)
plot(t_angles(1:length(states_total(:,12))),states_total(:,12),'r','
    LineWidth',1)
grid on
```

```
xlabel('time [s]','FontSize',15)
ylabel('psi−angle [rad]','FontSize',15)
legend({'psi−ref','psi−angle'},'Location','northeast','FontSize',15)


%% Plot the inputs

figure
subplot(4,1,1)
plot(t_angles(1:length(states_total(:,10))),UTotal(:,1))
grid on
xlabel('time [s]','FontSize',15)
ylabel('U1 [N]','FontSize',15)
subplot(4,1,2)
plot(t_angles(1:length(states_total(:,10))),UTotal(:,2))
grid on
xlabel('time [s]','FontSize',15)
ylabel('U2 [Nm]','FontSize',15)
subplot(4,1,3)
plot(t_angles(1:length(states_total(:,10))),UTotal(:,3))
grid on
xlabel('time [s]','FontSize',15)
ylabel('U3 [Nm]','FontSize',15)
subplot(4,1,4)
plot(t_angles(1:length(states_total(:,10))),UTotal(:,4))
grid on
xlabel('time [s]','FontSize',15)
ylabel('U4 [Nm]','FontSize',15)
%\\
%\\
```

B.2 Function: initial_constants.m

```
function constants=initial_constants()

    % Constants
     Ix = 0.0034; %kg*m^2
     Iy = 0.0034; %kg*m^2
     Iz  = 0.006; %kg*m^2
    m  = 0.698; %kg
    g  = 9.81; %m/s^2
     Jtp=1.302*10^(-6); %N*m*s^2=kg*m^2
     Ts=0.1; %s

    % Matrix weights for the cost function (They must be diagonal)
    Q=[1 0 0;0 1 0;0 0 1]; % weights for outputs (output x output)
    S=[1 0 0;0 1 0;0 0 1]; % weights for the final horizon outputs (
        output x output)
    R=[1 0 0;0 1 0;0 0 1]; % weights for inputs (input x input)

     ct = 7.6184*10^(-8); %N*s^2
     cq = 2.6839*10^(-9); %N*m^2
     l = 0.171; %m;

     controlled_states=3;
     hz = 4; % horizon period

    % Input bounds:
     lb=[-0.5; -0.5; -0.5];
     ub=[0.5; 0.5; 0.5];

     innerDyn_length=4; % Number of inner control loop iterations

    px=[-1+0j  -2+0j ];
    py=[-1+0j  -2+0j ];
    pz=[-1+0j  -2+0j ];

    constants={Ix Iy Iz m g Jtp Ts Q S R ct cq l controlled_states hz lb ub innerDyn_length px py pz};

end
```

B.3 Function: trajectory_generator.m

```

%% Position trajectory generation
function [X_ref,X_dot_ref,Y_ref,Y_dot_ref,Z_ref,Z_dot_ref,psi_ref]= trajectory_generator(t,r,f,height_i,height_f)

constants = initial_constants();
Ts=constants{7}; %s
innerDyn_length=constants{18}; % Number of inner control loop iterations

alpha=2*pi*f.*t;
d_height=height_f-height_i;

x = r.*cos(alpha);
y = r.*sin(alpha);
z = height_i+d_height/t(end)*t;

% x = (r/10.*t+2).*cos(alpha);
% y = (r/10.*t+2).*sin(alpha);
% z = height_i+d_height/t(end)*t;

% x = r.*cos(alpha);
% y = r.*sin(alpha);
% z = height_i+50*d_height/t(end)*sin(t);

% x = r.*cos(alpha);
% y = 2.*t;
% z = height_i+d_height/t(end)*t;

% x = 2.*t/20+1;
% y = 2.*t/20-2;
% z = height_i+d_height/t(end)*t;

dx=[x(2)-x(1),x(2:end)-x(1:end-1)];
dy=[y(2)-y(1),y(2:end)-y(1:end-1)];
dz=[z(2)-z(1),z(2:end)-z(1:end-1)];

x_dot=dx.*(1/(Ts*innerDyn_length));
y_dot=dy.*(1/(Ts*innerDyn_length));
z_dot=round(dz.*(1/(Ts*innerDyn_length)),8);

psi=zeros(1,length(x));
psi(1)=atan2(y(1),x(1))+pi/2;
psi(2:end)=atan2(dy(2:end),dx(2:end));

for i = 1:length(psi)
    if psi(i)<0
        psi(i)=2*pi-abs(psi(i));
    end
end

for i = 1:length(psi)
```

```
    if i>1
        if abs(psi(i)-psi(i-1))>pi
            psi(i:end)=psi(i:end)+2*pi;
        end
    end
end

X_ref = [t' x'];
X_dot_ref = [t' x_dot'];
Y_ref = [t' y'];
Y_dot_ref = [t' y_dot'];
Z_ref = [t' z'];
Z_dot_ref = [t' z_dot'];
psi_ref = [t' psi'];
end
%\\
%\\
```

B.4 Function: LPV_cont_discrete.m

```

function [Ad, Bd, Cd, Dd, x_dot, y_dot, z_dot, phi, phi_dot, theta,
    theta_dot, psi, psi_dot] = LPV_cont_discrete(states)
   % This is an LPV model concerning the three rotational axis.

   % Get the constants from the general pool of constants
   constants = initial_constants();
   Ix = constants{1}; %kg*m^2
   Iy = constants{2}; %kg*m^2
   Iz  = constants{3}; %kg*m^2
   Jtp=constants{6}; %N*m*s^2=kg*m^2
   Ts=constants{7}; %s
```

```matlab


% Assign the states
% States: [u,v,w,p,q,r,x,y,z,phi,theta,psi]
u = states(1);
v = states(2);
w = states(3);
p = states(4);
q = states(5);
r = states(6);
phi = states(10);
theta = states(11);
psi = states(12);

global omega_total;

%%

% Rotational matrix that relates u,v,w with x_dot,y_dot,z_dot
R_matrix=[cos(theta)*cos(psi), sin(phi)*sin(theta)*cos(psi)-cos(
    phi)*sin(psi), ...
     cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi); ...
     cos(theta)*sin(psi), sin(phi)*sin(theta)*sin(psi)+cos(phi)*
        cos(psi), ...
     cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi); ...
     -sin(theta), sin(phi)*cos(theta), cos(phi)*cos(theta)];

x_dot=R_matrix(1,:)*[u;v;w]; %x_dot
y_dot=R_matrix(2,:)*[u;v;w]; %y_dot
z_dot=R_matrix(3,:)*[u;v;w]; %z_dot

% To get phi_dot, theta_dot, psi_dot, you need the T matrix
```

```

    % Transformation matrix that relates p,q,r with phi_dot,theta_dot
        ,psi_dot
    T_matrix=[1, sin(phi)*tan(theta), cos(phi)*tan(theta); ...
         0, cos(phi), -sin(phi); ...
         0, sin(phi)*sec(theta), cos(phi)*sec(theta)];

    phi_dot=T_matrix(1,:)*[p;q;r]; %phi_dot
    theta_dot=T_matrix(2,:)*[p;q;r]; %theta_dot
    psi_dot=T_matrix(3,:)*[p;q;r]; %psi_dot

    A12=1;
    A24=-omega_total*Jtp/Ix;
    A26=theta_dot*(Iy-Iz)/Ix;
    A34=1;
    A42=omega_total*Jtp/Iy;
    A46=phi_dot*(Iz-Ix)/Iy;
    A56=1;
    A62=(theta_dot/2)*(Ix-Iy)/Iz;
    A64=(phi_dot/2)*(Ix-Iy)/Iz;

    A = [0   A12    0    0      0    0;
         0   0      0    A24    0    A26;
         0   0      0    A34    0    0;
         0   A42    0    0      0    A46;
         0   0      0    0      0    A56;
         0   A62    0    A64    0    0];


    B = [ 0         0       0      ;
          1/Ix      0       0      ;
          0         0       0      ;
```

```
                0          1/Iy   0      ;
                0          0      0      ;
                0          0      1/Iz ];

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

    D=0;


    % Discretize the system

%       % Forward Euler
%       Ad=eye(length(A(1,:)))+Ts*A;
%       Bd=Ts*B;
%       Cd=C;
%       Dd=D;


    % Zero-Order Hold

    % Create state-space
     sysc=ss(A,B,C,D);
     sysd=c2d(sysc,Ts,'zoh');
    Ad=sysd.A;
    Bd=sysd.B;
    Cd=sysd.C;
    Dd=sysd.D;


end
%\\
```

B.5 Function: MPC_simplification.m

```

function [Hdb,Fdbt,Cdb,Adc] = MPC_simplification(Ad,Bd,Cd,Dd,hz)

    % db - double bar
    % dbt - double bar transpose
    % dc - double circumflex

    A_aug=[Ad,Bd;zeros(length(Bd(1,:)),length(Ad(1,:))),eye(length(Bd
        (1,:)))];
    B_aug=[Bd;eye(length(Bd(1,:)))];
    C_aug=[Cd,zeros(length(Cd(:,1)),length(Bd(1,:)))];
    D_aug=Dd; % D_aug is not used because it is a zero matrix

    constants = initial_constants();
    Q = constants{8};
    S = constants{9};
    R  = constants{10};

    CQC=C_aug'*Q*C_aug;
    CSC=C_aug'*S*C_aug;
    QC=Q*C_aug;
    SC=S*C_aug;

    Qdb=zeros(length(CQC(:,1))*hz,length(CQC(1,:))*hz);
    Tdb=zeros(length(QC(:,1))*hz,length(QC(1,:))*hz);
    Rdb=zeros(length(R(:,1))*hz,length(R(1,:))*hz);
    Cdb=zeros(length(B_aug(:,1))*hz,length(B_aug(1,:))*hz);
    Adc=zeros(length(A_aug(:,1))*hz,length(A_aug(1,:)));

    for i = 1:hz
```

```

            if i == hz
                Qdb(1+length(CSC(:,1))*(i-1):length(CSC(:,1))*i,1+length(CSC(1,:))*(i-1):length(CSC(1,:))*i)=CSC;
                Tdb(1+length(SC(:,1))*(i-1):length(SC(:,1))*i,1+length(SC(1,:))*(i-1):length(SC(1,:))*i)=SC;
            else
                Qdb(1+length(CQC(:,1))*(i-1):length(CQC(:,1))*i,1+length(CQC(1,:))*(i-1):length(CQC(1,:))*i)=CQC;
                Tdb(1+length(QC(:,1))*(i-1):length(QC(:,1))*i,1+length(QC(1,:))*(i-1):length(QC(1,:))*i)=QC;
            end

            Rdb(1+length(R(:,1))*(i-1):length(R(:,1))*i,1+length(R(1,:))*(i-1):length(R(1,:))*i)=R;

            for j = 1:hz
                if j<=i
                    Cdb(1+length(B_aug(:,1))*(i-1):length(B_aug(:,1))*i,1+length(B_aug(1,:))*(j-1):length(B_aug(1,:))*j)=A_aug^(i-j)*B_aug;
                end
            end
            Adc(1+length(A_aug(:,1))*(i-1):length(A_aug(:,1))*i,1:length(A_aug(1,:)))=A_aug^(i);
        end
        Hdb=Cdb'*Qdb*Cdb+Rdb;
        Fdbt=[Adc'*Qdb*Cdb;-Tdb*Cdb];
end
%\\
%\\
```

B.6 Function: pos_controller.m

```

function [Phi_ref, Theta_ref, U1]=pos_controller(X_ref,X_dot_ref,
    Y_ref,Y_dot_ref,Z_ref,Z_dot_ref,Psi_ref,states)



%% Load the constants
constants=initial_constants();
m = constants{4}; %kg
g = constants{5}; %m/s^2

%% Assign the states
% States: [u,v,w,p,q,r,x,y,z,phi,theta,psi]

u = states(1);
v = states(2);
w = states(3);
x = states(7);
y = states(8);
z = states(9);
phi = states(10);
theta = states(11);
psi = states(12);

% Rotational matrix that relates u,v,w with x_dot,y_dot,z_dot
R_matrix=[cos(theta)*cos(psi), sin(phi)*sin(theta)*cos(psi)-cos(phi)*
    sin(psi), ...
     cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi); ...
     cos(theta)*sin(psi), sin(phi)*sin(theta)*sin(psi)+cos(phi)*cos(
        psi), ...
    cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi); ...
    -sin(theta), sin(phi)*cos(theta), cos(phi)*cos(theta)];

x_dot=R_matrix(1,:)*[u;v;w]; %x_dot
y_dot=R_matrix(2,:)*[u;v;w]; %y_dot
z_dot=R_matrix(3,:)*[u;v;w]; %z_dot

%% Compute the errors
ex=X_ref-x;
ex_dot=X_dot_ref-x_dot;
ey=Y_ref-y;
ey_dot=Y_dot_ref-y_dot;
ez=Z_ref-z;
ez_dot=Z_dot_ref-z_dot;

%% Compute the the constants K1, K2, and the values vx, vy, vz to
    stabilize the position subsystem
Ax=[0 1;0 0];
Bx=[0;1];
px=constants{19};
Kx=place(Ax,Bx,px);
ux=-Kx*[ex;ex_dot];
vx=-ux;

Ay=[0 1;0 0];
By=[0;1];
py=constants{20};
Ky=place(Ay,By,py);
uy=-Ky*[ey;ey_dot];
vy=-uy;

Az=[0 1;0 0];
```

```
Bz=[0;1];
pz=constants{21};
Kz=place(Az,Bz,pz);
uz=-Kz*[ez;ez_dot];
vz=-uz;

%% Compute phi, theta, U1
a=vx/(vz+g);
b=vy/(vz+g);
c=cos(Psi_ref);
d=sin(Psi_ref);

tan_theta=a*c+b*d;
Theta_ref=atan(tan_theta);

if or(abs(Psi_ref)<pi/4,abs(Psi_ref)>3*pi/4)
    tan_phi=cos(Theta_ref)*(tan(Theta_ref)*d-b)/c;
else
    tan_phi=cos(Theta_ref)*(a-tan(Theta_ref)*c)/d;
end

Phi_ref=atan(tan_phi);
U1=(vz+g)*m/(cos(Phi_ref)*cos(Theta_ref));

%% Check

(cos(Phi_ref)*sin(Theta_ref)*cos(Psi_ref)+sin(Phi_ref)*sin(Psi_ref))/
    m*U1;
(cos(Phi_ref)*sin(Theta_ref)*sin(Psi_ref)-sin(Phi_ref)*cos(Psi_ref))/
    m*U1;
-g+(cos(Phi_ref)*cos(Theta_ref))/m*U1;

```

ETSEIB

```
vx;
vy;
vz;

end
%\\
%\\
```

B.7 Function: nonlinear_drone_model.m

```

function dx = nonlinear_drone_model(t, states, U)
    % In this simulation, the body frame and its transformation is
        used
    % instead of a hybrid frame. That is because for the solver ode45
        , it
    % is important to have the nonlinear system of equations in the
        first
    % order form.

    % Constants
     constants = initial_constants();
     Ix = constants{1}; %kg*m^2
     Iy = constants{2}; %kg*m^2
     Iz  = constants{3}; %kg*m^2
    m  = constants{4}; %kg
     g  = constants{5}; %m/s^2
     Jtp=constants{6}; %N*m*s^2=kg*m^2

    % States: [u,v,w,p,q,r,x,y,z,phi,theta,psi]
     u = states(1);
     v = states(2);
    w = states(3);
    p = states(4);
    q = states(5);
    r = states(6);
    x = states(7);
    y = states(8);
    z = states(9);
    phi = states(10);
    theta = states(11);
    psi = states(12);


    % Inputs:

    U1  = U(1);
    U2  = U(2);
    U3  = U(3);
    U4  = U(4);


    % Rotational matrix that relates u,v,w with x_dot,y_dot,z_dot
    R_matrix=[cos(theta)*cos(psi), sin(phi)*sin(theta)*cos(psi)-cos(
        phi)*sin(psi), ...
        cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi); ...
        cos(theta)*sin(psi), sin(phi)*sin(theta)*sin(psi)+cos(phi)*
            cos(psi), ...
        cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi); ...
        -sin(theta), sin(phi)*cos(theta), cos(phi)*cos(theta)];

    % Transformation matrix that relates p,q,r with phi_dot,theta_dot
        ,psi_dot
    T_matrix=[1, sin(phi)*tan(theta), cos(phi)*tan(theta); ...
```

```
            0, cos(phi), -sin(phi); ...
            0, sin(phi)*sec(theta), cos(phi)*sec(theta)];

    global omega_total

    % The nonlinear equation describing the dynamics of the drone
    dx(1,1)=(v*r-w*q)+g*sin(theta); %u_dot
    dx(2,1)=(w*p-u*r)-g*cos(theta)*sin(phi); %v_dot
    dx(3,1)=(u*q-v*p)-g*cos(theta)*cos(phi)+U1/m; %w_dot
    dx(4,1)=q*r*(Iy-Iz)/Ix-Jtp/Ix*q*omega_total+U2/Ix; %p_dot
    dx(5,1)=p*r*(Iz-Ix)/Iy+Jtp/Iy*p*omega_total+U3/Iy; %q_dot
    dx(6,1)=p*q*(Ix-Iy)/Iz+U4/Iz; %r_dot
    dx(7,1)=R_matrix(1,:)*[u;v;w]; %x_dot
    dx(8,1)=R_matrix(2,:)*[u;v;w]; %y_dot
    dx(9,1)=R_matrix(3,:)*[u;v;w]; %z_dot
    dx(10,1)=T_matrix(1,:)*[p;q;r]; %phi_dot
    dx(11,1)=T_matrix(2,:)*[p;q;r]; %theta_dot
    dx(12,1)=T_matrix(3,:)*[p;q;r]; %psi_dot

end
```

Bibliography

[1] Tommaso Bresciani. "Modelling, Identification and Control of a Quadrotor Helicopter". MA thesis. Lund University, 2008.

[2] John J. Craig. *Introduction to Robotics. Mechanics and Control*. PEARSON, 3rd edition.

[3] Carlos Trapiello Fernández. "CONTROL OF AN UAV USING LPV TECHNIQUES". MA thesis. The Universitat Politècnica de Catalunya - ETSEIB, 2018.

[4] MATLAB®. *Quadratic programming @ONLINE*. URL: `https://es.mathworks.com/help/optim/ug/quadprog.html?lang=en`.

[5] MATLAB®. *Understanding Model Predictive Control, Part 2: What is MPC? @ONLINE*. URL: `https://www.youtube.com/watch?v=cEWnixjNdzs&vl=en`.

[6] The Czech Technical University in Prague (CTU). *Discrete-time optimal control over a finite time horizon as an optimization over control sequences @ONLINE*. URL: `https://moodle.fel.cvut.cz/mod/page/view.php?id=72476`.

[7] The Czech Technical University in Prague (CTU). *Model predictive control (MPC) - regulation @ONLINE*. URL: `https://moodle.fel.cvut.cz/mod/page/view.php?id=72476`.

[8] The Czech Technical University in Prague (CTU). *Model predictive control (MPC) - tracking @ONLINE*. URL: `https://moodle.fel.cvut.cz/mod/page/view.php?id=72476`.

[9] Mark Schmidt. *Deriving the Gradient and Hessian of Linear and Quadratic Functions in Matrix Notation*. February 6, 2019.