Treball de Fi de Màster

Double Master's degree in Industrial Engineering and Automatic Control and Robotics

Control System Design of a 3-Axis Gimbal

MEMÒRIA

Autor: Ariel Medero Borrell

Director: Dr. Vicenç Puig Cayuela

Convocatòria: July 2020

ETSEIB

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

UPC

Abstract

Gimbals are precise devices used for orientation control in the 3D space. They can be used with mounted cameras to maintain horizontal orientation while being hand-held or even mounted on a helicopter, both of them, examples that are more and more common to see in the film industry. They are also widely used for hardware-in-the-loop tests of sensor technology, from testing smartphones gyroscopes to even test mission critical navigation devices for future spacecrafts. Despite their importance, industry still relies in the use of the classical PID controllers for the control of the gimbal axis.

The objective of this master's thesis is to develop a control system using the modern technique of Linear Parameter-Varying (LPV) control for one of such devices, a 3-Axis Gimbal developed by the Ingenia company. In order to develop this control system, in this master's thesis it is explained the whole process required. From obtaining the mathematical model for the mechanical dynamics of the 3-Axis Gimbal, LPV controller design and also fault-tolerant strategies

To obtain the gimbal model, the selected method was to follow the Lagrange formulation for serial-link objects. For the computation of the controllers, it is shown three different ways in which the LPV models in conjunction with the aid of Linear Matrix Inequalities (LMIs) optimization can be used. Then, an inverse kinematics and path generation algorithm suitable for gimbals designed for mounted camera applications was proposed. Finally, a fault tolerant strategy is implemented to help the gimbal control system to deal with partial sensor faults by making an online estimation of the faults and using virtual sensors to reconstruct the non faulty sensor signal.

Acknowledgements

I would genuinely like to thank Prof. Vicenç Puig Cayuela for his valuable guidance during the project. He has given me the necessary advises that have been crucial to simplify or improve the work done. I have always been able to freely approach him, and he replied to my doubts quickly in every occasion, and in a very helpful and friendly manner. It was a pleasure to work with him.

I would also like to thank the team at Ingenia for the time I spent with them. They patiently taught me how their system and their technology for motion control works. Moreover, their team was always open to any question I may asked them and all of them showed genuinely interest in the success of the project and were eager to help.

Finally, I would like to sincerely thank my longtime friend Sergi Cupons Galera for his invaluable help in creating a 3D CAD model of the 3-Axis Gimbal. A key step in the success of the project that I would not had been able to make on my own.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The Ingenia company is dedicated to the design and manufacturing of servodrives for motion control applications. Recently, they developed a brand-new servodrive, named EVEREST XCR, which has as its main novelty in the fact that is uses the EtherCAT communication protocol. A communication protocol that, as they mentioned, is being widely adopted in the motion control industry because of the high bandwidth and communication speeds it can achieve. In order to showcase their products they have designed the 3-Axis Gimbal which can be seen in Figure 1.1. They desire to update this device with its newest EVEREST XCR servodrive, which arose the opportunity for a University-Industry collaboration. Sadly, due to the current situation in the past months much of the hardware implementation part of the project had to be redirected to more theoretical approaches, but the heavy influence of the real device still remains true.

The 3-Axis Gimbal is made up of three moving parts which are able to rotate. This gives the gimbal full control of the three rotational degrees of freedom for spatial orientation, which is the main purpose for gimbals systems. In order to be able to move, each of the three parts that make the gimbal use a FHA-11C-50-D200-EM1 gearmotor by Harmonic Drive. The gearmotors also provide with hall sensors and differential encoders, in addition of which the gimbal counts with three absolute inductive encoders by Zettlex for each of the gimbal axis. Three EVEREST XCR servodrives are in charge of receiving the information from the sensors and driving each of the motors independently. The servodrives have been configured to use the differential encoders of the motors as the sensor source for measuring the angular velocity and the absolute

inductive encoders for measuring the position of the gimbal axes. In order to control the motors, the servodrives implement a classical nested PID cascade control topology. The outer control loop is in charge of controlling the the motor angular position, while the second and inner control loops control respectively the motor angular velocity and the motor current intensity.

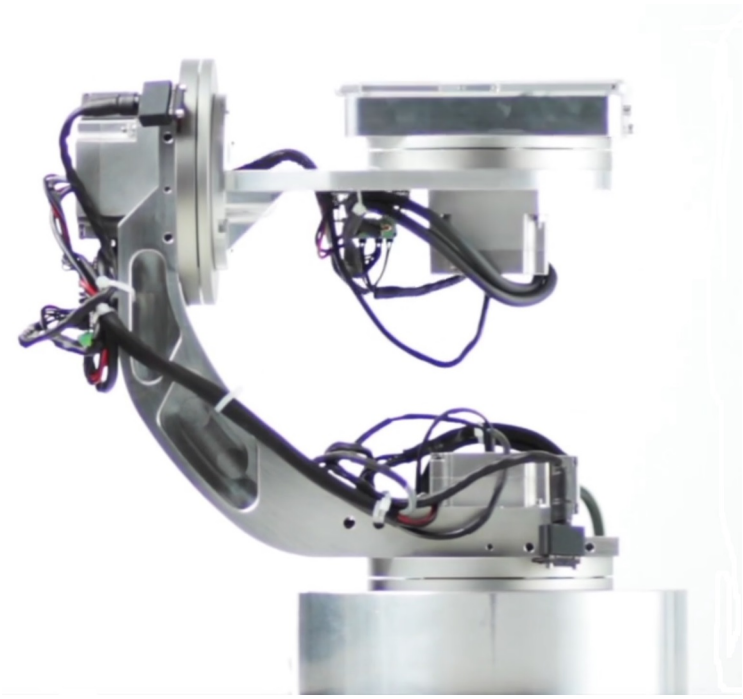

Figure 1.1: Photo of the 3-Axis Gimbal.

1.2 Thesis Objectives

The main objective of this master's thesis is to design a complete control system for the 3-Axis Gimbal. The goal is to demonstrate how complex mechanical systems could benefit from modern non-linear control techniques approaches like the Gain-Scheduled LPV method and computation of the controller and observer parameters based on LMIs techniques. In order to achieve this objective, the following milestones are set for this project:

- To obtain a mathematical model of the 3-Axis Gimbal capable of capturing as much as possible its dynamical behaviour.
- Design a LPV controller architecture that is able to drive the gimbal axis smoothly and without any overshoot.

- Design a High Level Planner to give commands to the gimbal in the Cartesian space and for path generation.
- Implement a fault-tolerant control strategy to deal with partial sensors faults.

1.3 Thesis Structure

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

Chapter 2

This chapter explains how to compute the mathematical model for the mechanical dynamics of the 3-Axis Gimbal by modelling it as robotic serial-link object and using the Lagrange method.

Chapter 3

This chapter shows how the full mathematical model of the 3-Axis Gimbal can be simplified into a viable control model. Then, it is explained how to transform this model into the LPV format and how to deal with singularities in the case they appear in some of the embedded parameters of the LPV model. Finally, using the LPV model and LMIs optimization techniques the controllers for the gimbal axis are computed and then tested in simulation.

Chapter 4

This chapter covers the development of the High Level Planner and how the inverse kinematics and path generation algorithms work, both in conjunction with the planner.

Chapter 5

This chapter proposes a FTC scheme to detect partial sensor faults using online parameter identification techniques and LPV observers. After the fault detection ,it is shown how to reconstruct the fault free sensor signal by using virtual sensors.

Chapter 6

The final chapter concludes the thesis and summarizes its contributions.

Chapter 2

3-Axis Gimbal Mathematical Model

The first step in order to design a controller is to obtain a model of the system that we wish to control. This is specially true when we want to use non-linear techniques for the design of the controller, like the Linear Parameter Varying (LPV) technique, which will be used in this master's thesis. For the LPV controller design, the mathematical model of the system is not only an important tool for validating the design in simulation, but an essential step in order to be able to obtain the controller. How well the mathematical model captures the dynamics of the real system will have a great impact on the final performance. In this chapter, the mathematical model of the 3-Axis Gimbal will be obtained by using the well know Lagrange formulation for serial-link robots.

2.1 3-Axis Gimbal Forward Kinematics

When we look at the 3-Axis Gimbal in Figure 2.1, it could be thought of as a serial-link. This representation is how most industrial robot manipulators are modelled and designed. For this reason, the methodology that will be used to obtain its dynamical model will be based on the very well-known techniques for studying anthropomorphic robotic manipulators [11]. The advantages of this approach is that the whole procedure is very systematic and straightforward to follow even without the need of deep knowledge in theoretical mechanics.

The first step in this approach is to obtain the forwards kinematics of the system [11, Chapter 2]. This can be easily done by computing the parameters that define each link according to the Denavit-Hartenberg Convention. For this, we first need to define the reference frame and

its origin for each of the three links that compose the gimbal.

For selecting the origin of the frames, the DH convention is quite flexible as they do not need to have any explicit relation with the hardware system itself. One interesting aspect of the gimbals is that all the axes of rotation intersect in one point, as shown by the dashed lines in Figure 2.1. Exploiting this fact and the flexibility of the DH convention, all the origin frames have been located at the intersection. With this, all the translational elements of the homogeneous transform matrix will be zero, which will simplify the next steps and computations. This makes sense for this case as gimbals are used for controlling the orientation of its end-effector, so there is no need to track its XYZ position.

With the origins of the reference frames chosen, the next step is to select the right orientation for each one. The DH convention gives the following guidelines:

- Choose axis z_i along the axis of Link Joint $i + 1$, the motor axis of rotation.
- Choose axis x_i along the common normal to axes z_{i-1} and z_i.
- Choose axis y_i so as to complete a right-handed frame.
- For Frame 0, only the direction of axis z_0 is specified, x_o can be randomly selected.

Following this guideline the resulting reference frames can be seen in Figure 2.1.

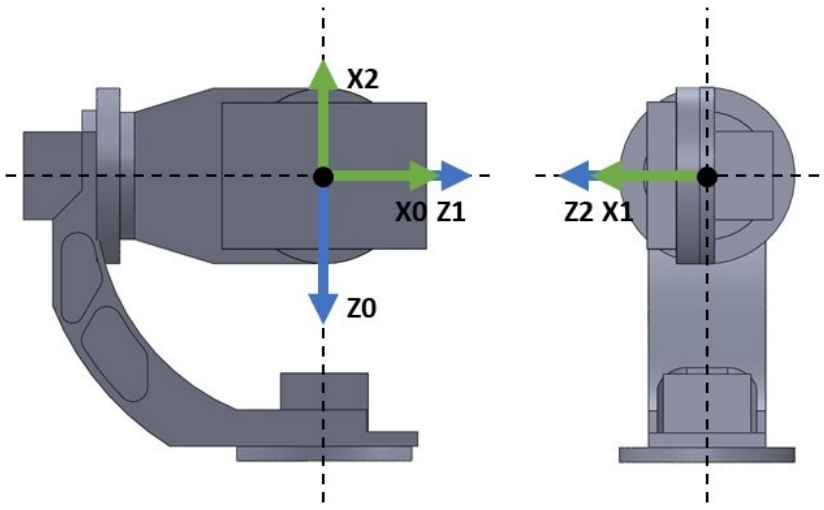

Figure 2.1: Reference frame for each of the three links of the gimbal

With the reference frames selected, then the four DH parameters can be computed. With

these four parameters it is enough to define each link in relation with its previous link. These parameters are defined as follows:

a_i**:** distance along x_i from O_i to the intersection of the x_i and z_{i-1} axes.

d_i**:** distance along z_{i-1} from O_{i-1} to the intersection of the x_i and z_{i-1} axes.

α_i**:** angle between z_{i-1} and z_i measured about x_i.

θ_i**:** angle between x_{i-1} and x_i measured about z_{i-1}.

And the homogeneous transformation matrix given by the DH parameters relating Frame i-1 with Frame i is:

$$A_{i-1}^{i} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

According to the given DH parameters definition and the reference frames chosen for the 3-Axis Gimbal, Table 2.1 presents the parameters that define each of gimbal links.

Link	θ_i	a_i	d_i	α_i
1	$q_1^* + \pi/2$	0	0	$\pi/2$
2	$q_2^* - \pi/2$	0	0	$-\pi/2$
3	q_3^*	0	0	0

* variable

Table 2.1: DH Parameters for the 3-Axis Gimbal.

Notice that because all the origins are in the same point where all the motor axes intersect, all the elements a_i and d_i are zero, meaning that translation between links is not being taken into account, as previously mentioned. Also, the rotation of each link is given by the variables q_i, which by construction have a constant offset as can be seen in links 1 and 2. The homogeneous transformation matrix for each link can be easily obtained by replacing the DH parameters of each link from Table 2.1 into Equation (2.1), and the global transformation matrix from Link 1 to Link 3 can be obtained by concatenating the homogeneous transform matrix from each link

$$T_0^3(q) = A_0^1(q_1)A_1^2(q_2)A_2^3(q_3) \tag{2.2}$$

$$T_0^3(q) = \begin{bmatrix} -c_1s_3 - c_3s_1s_2 & s_1s_2s_3 - c_1c_3 & -c_2s_1 & 0 \\ c_1c_3s_2 - s_1s_3 & -c_3s_1 - c_1s_2s_3 & c_1c_2 & 0 \\ -c_2c_3 & c_2s_3 & s_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Note that c_i and s_i are equal to $\cos(q_i)$ and $\sin(q_i)$, respectively.

2.2 Dynamical Model

The 3-Axis Gimbal, like most mechanical devices, can have its dynamics explained according to Newton's Law of motion, as commonly expressed in Robotics

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + g(q) = \tau \tag{2.4}$$

where

$B(q)$: Global inertia tensor of the gimbal, 3x3 symmetrical matrix.

$C(q,\dot{q})$: Coriolis matrix, 3x3 matrix.

F_v: Viscous friction coefficient, 3x3 diagonal matrix.

$g(q)$: Torques present in the links joints by the effect of gravity, 3x1 vector.

τ: Input torques given by the actuators, 3x1 vector.

From this representation, it could be obtained the feed forward torque needed to accomplish a desired trajectory or as we are most interested in control, simulate how the system will evolve when a torque input is applied according to

$$\ddot{q} = B(q)^{-1}[\tau - C(q,\dot{q})\dot{q} - F_v\dot{q} - g(q)] \tag{2.5}$$

In this section, it will be explained how to compute the terms in Equation (2.4) for the 3-Axis Gimbal, although most of the content of this section is general and will be applicable to

other similar mechanical systems. However, note that for a full demonstration on the equations given, the reader is referred to [11, Chapter 7] as this was the main source to develop this part of the project.

There are many ways in which to obtain the expressions for the terms of the inertia tensor, the Coriolis matrix and the gravity term. For this case though, given that the gimbal can be expressed as a multi-body serial-link system, the Lagrange method is a perfect fit since it is very systematic when a reference frame, like the one according to the DH convention, is used [9].

The first step is to compute the inertia tensor expression, which after developing the Lagrangian equation is:

$$B(q) = \sum_{i=1}^{3} (m_{l_i} J_P^{(l_i)T} J_P^{(l_i)} + J_O^{(l_i)T} R_0^i I_{l_i}^i R_0^{iT} J_O^{(l_i)}) \tag{2.6}$$

The most basics elements in Equation (2.6) are the terms m_{l_i} and $I_{l_i}^i$, which are the mass of Link i and the inertia tensor of the Link i computed in its own center of mass and expressed with respect to its own reference frame, note that both of these elements have a constant value. The rotation matrix R_0^i is used to transform $I_{l_i}^i$ from its the reference frame i-1 to the base reference frame 0. The rotation matrix can be extracted from the first 3x3 subset from Equation (2.1) and for i different from 1 it can be obtained by concatenation, analogue to Equation (2.2).

The elements $J_P^{(l_i)}$ and $J_O^{(l_i)}$ are the translational Jacobian of the center of mass and rotational Jacobian of Link i. Their expressions are:

$$J_P^{(l_i)} = \begin{bmatrix} J_{P_1}^{(l_i)} & \dots & J_{P_i}^{(l_i)} & 0 & \dots & 0 \end{bmatrix} \tag{2.7}$$

$$J_O^{(l_i)} = \begin{bmatrix} J_{O_1}^{(l_i)} & \dots & J_{O_i}^{(l_i)} & 0 & \dots & 0 \end{bmatrix} \tag{2.8}$$

The columns for the translational Jacobian can be computed as:

$$J_{P_j}^{l_i} = z_{j-1} \times p_{l_i} \tag{2.9}$$

where

$$z_{j-1} = R_0^{j-1} z_0 \tag{2.10}$$

$$z_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \tag{2.11}$$

$$p_{l_i} = R_0^i R_i \tag{2.12}$$

Note that R_i are the XYZ coordinates of the center of mass of Link i in its own reference frame. The columns for the rotational Jacobian can be computed as:

$$J_{O_j}^{l_i} = z_{j-1} \tag{2.13}$$

With the global inertia tensor matrix of the gimbal $B(q)$ computed, then, obtaining the elements c_{ij} of the 3x3 Coriolis matrix $C(q, \dot{q})$ is quite straightforward.

$$c_{ij} = \sum_{i=1}^{3} c_{ijk}\dot{q}_k \tag{2.14}$$

$$c_{ijk} = \frac{1}{2}(\frac{\partial b_{ij}}{\partial q_k} + \frac{\partial b_{ik}}{\partial q_j} + \frac{\partial b_{jk}}{\partial q_i}) \tag{2.15}$$

Note that b_{ij} are the elements from the global inertia tensor matrix $B(q)$. The last element that the Lagrange method provide for serial-links allows to easily obtain is the gravitational term $g(q)$

$$g(q) = \begin{bmatrix} g_1(q) \\ g_2(q) \\ g_3(q) \end{bmatrix} \tag{2.16}$$

where

$$g_i(q) = \sum_{j=1}^{3} -m_{l_j} g_0 J_{P_i}^{l_j} \tag{2.17}$$

$$g_0 = \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T \tag{2.18}$$

Note that the element g in Equation (2.18) is the gravity acceleration constant, it is positive as the z axis of the base reference frame 0 is pointing downwards, as can be seen in Figure 2.1.

The last element from the model, according to Equation (2.4), that needs to be determined is the viscous friction matrix F_v. It can not be theoretically obtained and its value need to be determined in the parameter identification phase using input-output data. However, its general form is known, it is a diagonal matrix with constant elements and due to the construction of the 3-Axis Gimbal all of its three elements can be assumed to be equal. This last assumption is based on the fact that all three actuators are exactly the same gear-motor, so the frictions they experience should be the same or at least very similar.

With this, all the elements from Equation (2.4) have been presented and explained how they can be computed for obtaining the mathematical model of the 3-Axis Gimbal. The Lagrange method for serial-link objects despite being a very systematic approach is quite tedious to compute if it is attempted to be solved by hand. For this reason, the best approach is to use the aid of a symbolic software package. In Appendix A, the whole MATLAB® script used to obtain the mathematical model for the gimbal can be seen.

2.3 Open-Loop Simulations and Analysis

After obtaining the mathematical model of the system, it is always interesting to simulate it. This is important to check if the behaviour of the model corresponds with the dynamics of the real system. But, even more, trough open-loop simulations we can also analyze how it responds to certain inputs and be able to infere some interesting properties of the system that may aid in the controller design and specially in the model simplification step. With this objective in mind, the obtained model was simulated in Simulink® by implementing Equation (2.5) as represented in Figure 2.2.

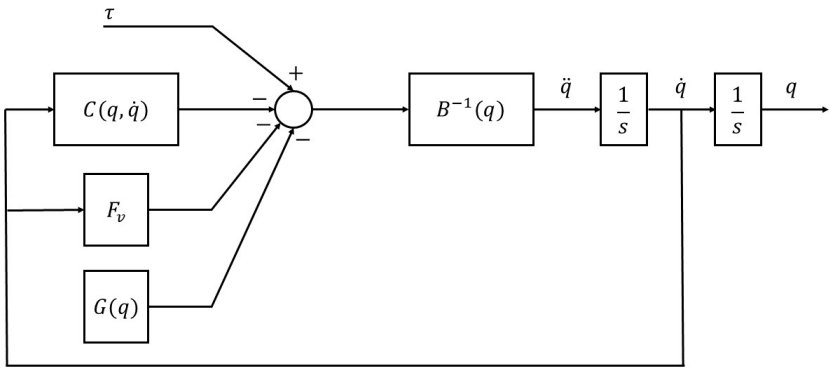

Figure 2.2: Diagram form of Equation (2.5).

In the previous section, the elements that compose the model were computed. However, to be able to execute the simulation, the physical properties of the 3-Axis Gimbal are required. Mainly, the mass m_{l_i} and the inertia tensor $I^i_{l_i}$ of each Link. Sadly, this information was not available within the gimbal documentation. For this reason, in order to be able to do some approximate simulations and controller prototype, the 3-Axis Gimbal was physically modelled in the 3D CAD software SolidWorks®. Using this software it is possible to extract all this information and since the material that the gimbal is made of is known, these values will be quite

close to the real ones. The list of parameters for the physical properties for the 3-Axis Gimbal are summarized in the following tables, all the parameters appear represented in their units according to the international system of units and computed with respect to the reference frame of each Link, shown in Figure 2.1.

Parameter	Value	Description
m_1	2.4292	Mass of the Link
com_{x1}	-0.09056	X coordinate of the center of mass
com_{z1}	0.11474	Z coordinate of the center of mass
I_{xx1}	0.014355	I_{xx} element of the Link inertia tensor
I_{yy1}	0.029371	I_{yy} element of the Link inertia tensor
I_{zz1}	0.017526	I_{zz} element of the Link inertia tensor
I_{xz1}	0.011748	I_{xz} element of the Link inertia tensor

* all non specified parameters have a value equal to zero.

Table 2.2: Physical parameters for Link 1.

Parameter	Value	Description
m_2	1.624	Mass of the Link
com_{x2}	-0.00631	X coordinate of the center of mass
com_{z2}	-0.07338	Z coordinate of the center of mass
I_{xx2}	0.0089560	I_{xx} element of the Link inertia tensor
I_{yy2}	0.0085086	I_{yy} element of the Link inertia tensor
I_{zz2}	0.0019667	I_{zz} element of the Link inertia tensor
I_{xz2}	-0.0006874	I_{xz} element of the Link inertia tensor

* all non specified parameters have a value equal to zero.

Table 2.3: Physical parameters for Link 2.

Parameter	Value	Description
m_3	0.756	Mass of the Link
com_{z3}	0.031	Z coordinate of the center of mass
I_{xx3}	0.00126	I_{xx} element of the Link inertia tensor
I_{yy3}	0.0006552	I_{yy} element of the Link inertia tensor
I_{zz3}	0.0018648	I_{zz} element of the Link inertia tensor

* all non specified parameters have a value equal to zero.

Table 2.4: Physical parameters for Link 3.

The other parameter needed for simulation is the viscous friction coefficient of the actuators. As mentioned at the end of the previous section, this coefficient is shared between all three actuators, so the viscous friction matrix F_v could be thought of as:

$$F_v = f_v \cdot I \tag{2.19}$$

Parameter	Value	Description
f_v	1	Viscous friction coefficient

Table 2.5: Viscous friction coefficient for simulation.

With all the parameters defined, it is finally possible to do the desired open-loop simulations. First, the system was tested in the case that a sinusoidal input was present and in a second test the system was simulated with a constant torque input. This last simulation is not totally realistic, as it would cause the gimbal axes to turn indefinitely, which is not possible in the real system due to construction constraints, but is an interesting test as it gives good insight into the behaviour of the angular velocity of each link.

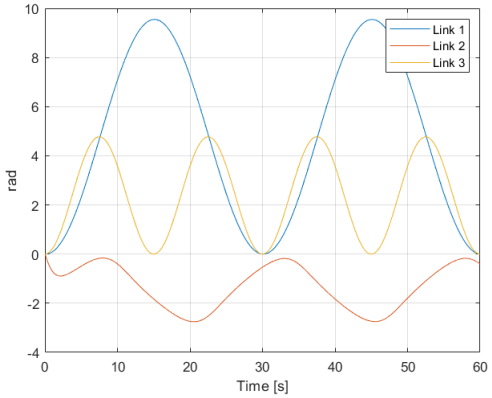

Figure 2.3: Angular position q of the Links in the presence of a sinusoidal torque input.

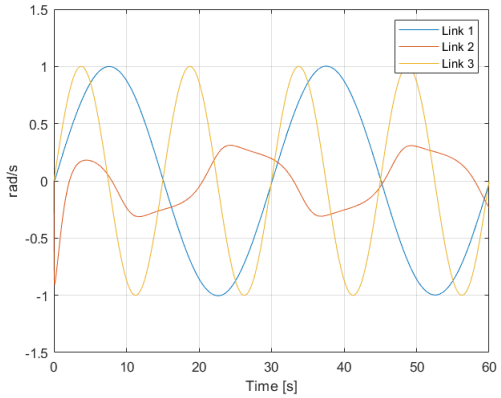

Figure 2.4: Angular velocity $\dot{q}$ of the Links in the presence of a sinusoidal torque input.

It is well know from basic control theory that the output of a linear plant when the input is a sinusoidal signal, is another sinusoidal signal of the same frequency with some offset and a different amplitude. By evaluating Figures 2.3 and 2.4, it can be seen that the response of Links 1 and 3 are clearly sinusoidal, meaning that these two links have a very high level of linearity in their behaviour. The situation is quite different for the Link 2, which its response shows a clear non-linear behaviour.

When analyzing the results presented in Figure 2.5, which shows the angular velocity of each Link when a constant torque is applied, the first conclusion is the same as before, Links 1 and 3 have a quite linear behavior while Link 2 clearly does not. However, in this case a second important conclusion can be reached. It can be seen that despite the erratic behaviour of Link 2, its sudden velocity spikes does not seem to affect greatly the angular velocities of the two other links. This fact indicates that there is a low level of coupling between the links of the gimbal.

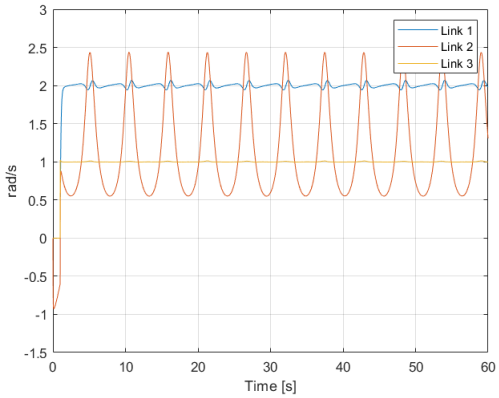

Figure 2.5: Angular velocity $\dot{q}$ of the Links in the presence of a constant torque input.

Chapter 3

The Controller Design

The controller design approach selected for controlling the 3-Axis Gimbal is the Linear Parameter-Varying (LPV) technique. This technique is an excellent tool as it allows using powerful linear control tools (as e.g. LMIs) in the design of controllers for non-linear systems. In this chapter, it will be explained the simplification steps followed to obtain a simplified control model from the full model obtained through the Lagrange method, how this simplified model was transformed into an LPV model and finally the controller synthesis and simulation results.

3.1 Model Simplification

The model obtained in the last chapter trough the Lagrange method for serial-links is a very interesting result for simulation and system analysis purposes. However, its expressions are too large and complex for any attempt of controller design, it is a so called Simulation Oriented Model (SOM). This make it necessary to simplify the model in a way that controller design will be viable, into a Control Oriented Model (COM). Luckily, the open loop simulations gave some interesting insight about the model behaviour that may be of help, mainly:

- Links 1 and 3 present a quasi-linear behaviour.
- The coupling between Links is limited.

From the first conclusion, we can assume that the highest challenge in the control design will come from the controller design for Link 2. However, the second conclusion is the most useful

at this point for model simplification. Using this information, we could attempt to simplify the model by decoupling the elements from the global inertia tensor $B(q)$ and the Coriolis matrix $C(q, \dot{q})$. This simplification can be done by only taking into account the diagonal elements of each matrix. This simplification was tested by simulating the decoupled model

$$\widetilde{B}(q)\ddot{q} + \widetilde{C}(q, \dot{q})\dot{q} + F_v\dot{q} + g(q) = \tau \tag{3.1}$$

against the previously obtained full SOM, note in the equation the tilde on top of $B(q)$ and $C(q, \dot{q})$ indicating that $\widetilde{B}(q)$ and $\widetilde{C}(q, \dot{q})$ are only diagonal matrices. For simulation purposes Equation (3.1) is rewritten as follows

$$\ddot{q} = \widetilde{B}(q)^{-1}[\tau - \widetilde{C}(q, \dot{q})\dot{q} - F_v\dot{q} - g(q)] \tag{3.2}$$

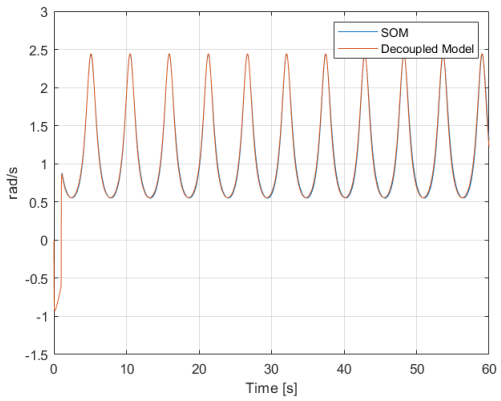

Figure 3.1: Angular velocity $\dot{q}$ of Link 2 in the presence of a constant torque input.

Figure 3.1 shows the simulation comparing the full and the decoupled model, only the angular velocity of Link 2 is shown for clarity purposes as it was seen that this was the most critical of all 3 links. From the simulation results, we can see that the difference with respect the full model are minimal. This means that the decoupling method is more than a valid simplification.

By looking at Equation (3.2), which is the suitable form of the dynamical equation for controller design, it can be seen that $\widetilde{B}(q)^{-1}$ multiplies the torque input vector τ. This means that $\widetilde{B}(q)^{-1}$ will be the B input matrix from an equivalent state-space representation of the system. Note that the system B input matrix will not be time invariant as it will evolve with the angular position of the axes, leading to the following state space representation

$$\ddot{x} = A(q, \dot{q})x + B(q)\tau \tag{3.3}$$

There are several ways to deal with time-varying B input matrices in the LPV controller design literature, as e.g. using an Apkarian filter [1] or using a descriptor representation of the system [5]. However, for the sake of simplicity, a second simplification is proposed by considering the diagonal matrix $\widetilde{B}(q)^{-1}$ to be constant. In order to implement this simplification, a study of the elements of the matrix $\widetilde{B}(q)^{-1}$ was carried. Since the global inertia tensor only depends on q, it is possible to plot its elements and the elements of its inverse. The plots in Figures 3.2 and 3.3 show the first two elements of the diagonal of $\widetilde{B}(q)^{-1}$ in the Z axis while the XY plane represent the variables q_1 and q_2. It can be seen that the variability in both cases is small. With this in mind, only their mean value will be considered in the final simplified model. The third element of the diagonal of $\widetilde{B}(q)^{-1}$ is already a constant value. The final simplified COM is

$$\ddot{q} = \overline{B^{-1}}[\tau - \widetilde{C}(q, \dot{q})\dot{q} - F_v\dot{q} - g(q)] \tag{3.4}$$

where $\overline{B^{-1}}$ is a constant diagonal matrix.

Again, the simplified model was tested against the Lagrange SOM, the results are shown in Figure 3.4. It is quite remarkable how the COM is fully capable of capturing the dynamics of the full model.

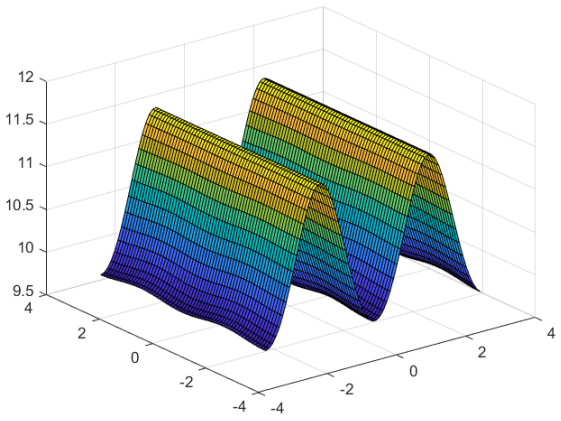

Figure 3.2: First element of the diagonal of $\widetilde{B}(q)^{-1}$.

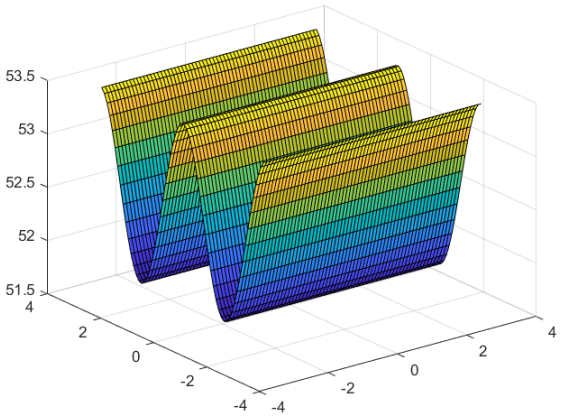

Figure 3.3: Second element of the diagonal of $\widetilde{B}(q)^{-1}$.

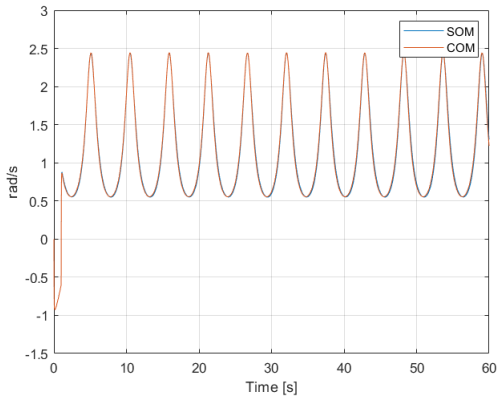

Figure 3.4: Angular velocity $\dot{q}$ of Link 2 in the presence of a constant torque input.

3.2 LPV Modelling

Once a simplified COM is obtained, modelling it as a LPV system is not a complicated task. This is achieved trough the linear parameter embedding technique [8]. In this case, there will not be just one global LPV model but three. This is so because exploiting the fact that the coupling between links is limited, three different controllers will be used, one for each link of the gimbal. Between links, they will share the same plant representation

$$\ddot{x}_i = A_i(q, \dot{q})x_i + B_i\tau_i \tag{3.5}$$

$$\begin{bmatrix} \dot{q}_i \\ \ddot{q}_i \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \theta_j(q, \dot{q}) & \theta_{j+1}(q, \dot{q}) \end{bmatrix} \begin{bmatrix} q_i \\ \dot{q}_i \end{bmatrix} + \begin{bmatrix} 0 \\ b_i \end{bmatrix} \tau \tag{3.6}$$

However, each $i-$Link has small variations and particularities which make them different, causing that all have bee modelled using a slightly different approach.

For Link 1, its plant model is

$$\begin{bmatrix} \dot{q}_1 \\ \ddot{q}_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & \theta_1(q,\dot{q}) \end{bmatrix} \begin{bmatrix} q_1 \\ \dot{q}_1 \end{bmatrix} + \begin{bmatrix} 0 \\ 10.75 \end{bmatrix} \tau_1 \tag{3.7}$$

$$\begin{aligned} \theta_1(q,\dot{q}) = &-10.75 + 0.081625\dot{q}_2 \sin(2q_2) + 0.003251\dot{q}_2 \sin(2q_2) cos(q_3)^2 \\ &- 0.0013887\dot{q}_2 \cos(q_2)^2 + 0.00694\dot{q}_2 \\ &+ 0.003251\dot{q}_3 \sin(2q_3) \cos(q_2)^2 \end{aligned} \tag{3.8}$$

It has only one varying parameter, shown in Equation (3.8). The key issue about LPV control design is that in order to design the controller it is not necessary to take into account all the infinite points within the full range of the non-linearities. Instead, we only need to consider the maximum and minimum value of the non-linear embedded parameters within the considered working region. For this reason, the maximum and minimum values of the embedded parameter $\theta_1(q,\dot{q})$ are presented in Table 3.1. In order to compute the maximum and minimum values of the embedded parameters, the values were found trough optimization, with the $\theta_1(q,\dot{q})$ expression as the objective function. The working region considered, which act as the optimization constraints, was $q_i \in [-\pi,\pi]$ and $\dot{q}_i \in [-2\pi,2\pi]$. This optimization method to determine the maximum and minimum values of the embedded parameters was used for all the others $\theta_i(q,\dot{q})$ parameters as well.

Parameter	Value
minθ_1	-11.2847
maxθ_1	-10.2153

Table 3.1: Link embedded parameter max and min value.

It was seen previously that Link 2 was the one that would be the greatest challenge to control of all three links, since it was the one with the highest level of nonlinear behaviour. Its plant model is as follows

$$\begin{bmatrix} \dot{q}_2 \\ \ddot{q}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \theta_2(q) & \theta_3(q,\dot{q}) \end{bmatrix} \begin{bmatrix} q_2 \\ \dot{q}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 52.5 \end{bmatrix} \tau_2 \tag{3.9}$$

$$\theta_2(q) = \frac{-49.30495\cos(q_2) - 5.27769\sin(q_2)}{q_2} \tag{3.10}$$

$$\theta_3(q,\dot{q}) = -52.5 - 0.015876\dot{q}_3\sin(2q_3) \tag{3.11}$$

. Starting the analysis of its embedded parameters with $\theta_3(q,\dot{q})$, it has a very low level of variation (see Figure 3.6). So, this parameter can be considered to have a constant value. On the other hand, $\theta_2(q)$ have a high degree of variability. In order to be able to embed this term in the LPV model, the solution was to divide it by q_2, as can be seen in Equation (3.10). Because zero is included within the working range of q_2, this can cause a discontinuity around zero in $\theta_2(q)$, as shown in Figure 3.5. To deal with this problem, the solution is to partition the working range of q_2 into $q_2 \in [-\pi, -0.1] \cap [0.1, \pi]$. This partition will later force the design of a Hybrid-LPV controller, dependent on the state of q_2.

Parameter	Value
minθ_2 (q_2 negative)	-17.3492
maxθ_2 (q_2 negative)	485.3176
minθ_2 (q_2 positive)	-495.8560
maxθ_2 (q_2 positive)	16.0752
θ_3	≈-52.5

Table 3.2: Link 2 embedded parameters max and min value.

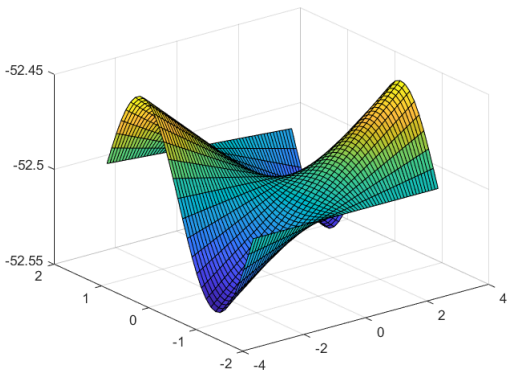

Figure 3.6: Embedded parameter $\theta_3(q,\dot{q})$.

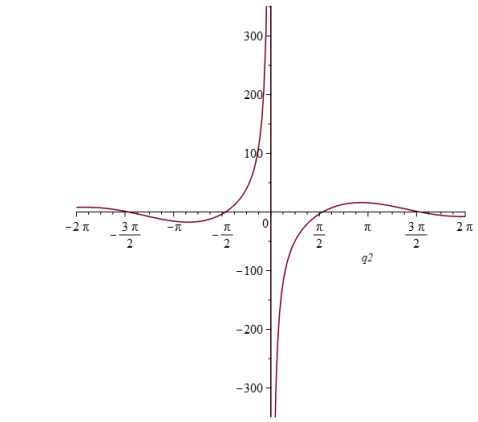

Figure 3.5: Embedded parameter $\theta_2(q)$.

Finally, for Link 3 it can be seen that after applying the simplification from the previous section, the dynamics can be represented by a linear model

$$\begin{bmatrix} \dot{q}_3 \\ \ddot{q}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -536.25 \end{bmatrix} \begin{bmatrix} q_3 \\ \dot{q}_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 536.25 \end{bmatrix} \tau_3 \tag{3.12}$$

This makes sense, since due to the fact that coupling between links have been eliminated in the control model, then Link 3 is basically a motor with a mass attached, which can be represented as a linear system.

3.3 LPV Controller Synthesis

In the model simplification section, it was seen that the control of the 3-Axis Gimbal would be split into three independent controllers. Each controller will control one link from the gimbal, whose state space representations are those of a second order system, as seen in Equation (3.6). In addition to being a second order system, given their state space representation structure, it makes the independent $i-$Link system a perfect fit for the use a Gain-Scheduled PID controller [2]. In order to design this kind of controller, the first step is to augment the state space of the systems with a third state, which is done by adding a new integrator to the model. The augmented state space system for each Link can be expressed as follows

$$\ddot{x}_e = \begin{bmatrix} 0 & 1 & 0 \\ \theta_j(q,\dot{q}) & \theta_{j+1}(q,\dot{q}) & 0 \\ 1 & 0 & 0 \end{bmatrix} x_e + \begin{bmatrix} 0 \\ b_i \\ 0 \end{bmatrix} \tau_i \tag{3.13}$$

To compute the PID coefficients for the controller, it can easily done by computing a state feedback controller K for the augmented system. The relation between this state feedback controller K and the three PID coefficients is given by

$$u = Kx_e = [-k_p - k_d - k_i]x_e \tag{3.14}$$

Then, in order to implement the PID controller, the best approach is to use an I-PD scheme as shown in Figure 3.7. Note that in the presented I-PD scheme there is not any derivative, instead, the derivative term K_d is being feedback directly with the velocity measurement. This is possible because both angular position and angular velocity are being measured. By doing this, the implementation issues associated with derivatives can be avoided.

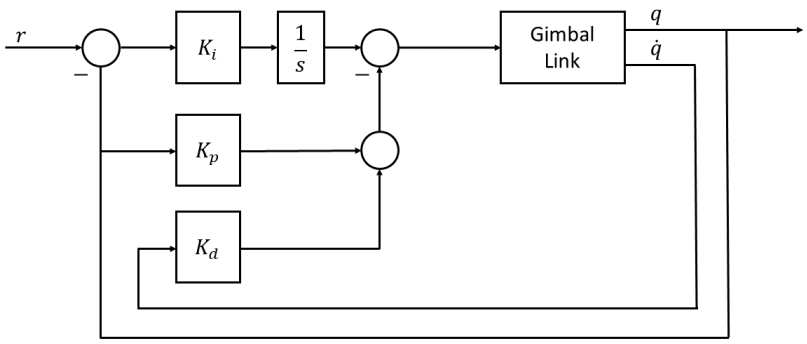

Figure 3.7: I-PD Scheme.

To compute the state feedback controllers K, a different strategy was used for each link. As a result, three different ways to compute the controllers gains are presented in this section. All the controllers were computed by solving a Linear Matrix Inequalities (LMI) problem, since this is the optimal tool for designing LPV gain-scheduling controllers. The results LMIs have

been solved using the YALMIP toolbox [4] and the SeDuMi solver [12]. In order to find a detailed demonstration on the LMI presented in this section the reader is referred to [3].

For Link 1, in Table 3.1 we can find the minimum and maximum values of the embedded parameter $\theta_1(q, \dot{q})$ in its LPV representation. It can be seen that the difference between both values is not really that big. For this reason, the most suitable choice is to design a robust controller instead of a gain-scheduling controller. For this case, since the parameter $\theta_1(q, \dot{q})$ involves many variables would also require many computations at every iteration in the case that a gain-scheduling controller would be used. On the other hand, having a fixed robust controller for Link 1 would avoid those computations while assuring stability across the working region of $\theta_1(q, \dot{q})$. For this reason and given the low level of non-linearity in Link 1, it is not justified the use of a gain-scheduling controller for this case.

To design a robust controller using LMIs and LPV systems, we must use the same controller variable K for all the $i-$vertexes of the LPV polytope. Also, the solution to the LMI problem must guarantee that the Lyapunov matrix P is positive defined. In the case that the latter is fulfilled, then the controller K will guarantee the stability within the working region of the LPV polytope. Since the LPV model for Link 1 have only one embedded parameter $\theta_1(q, \dot{q})$ in this case, then the LPV polytope have only two vertexes. This means that the LMI problem will have a total of three constraints:

$$P > 0 \tag{3.15}$$

$$A_i P + P A_i^T + BW + W^T B + 2\alpha P < 0 \tag{3.16}$$

$$K = WP^{-1} \tag{3.17}$$

The first constraint to guarantee that the Lyapunov matrix must be positive defined, Equation (3.15), and two LMI constraints used for pole placement, each one of them defined at each of the two LPV polytope vertexes. Equation (3.16) defines the LMI constraint used for pole placement. The value of α marks a vertical line that will split the negative plane of the poles space, all the poles of the system will be forced to be to the left of that line. Thus, with the value of α, we are controlling the reaction speed and settling time of the system.

Note that in the LMI set of constraints the Lyapunov matrix P and the auxiliary vector W

are the decision variables. After solving the LMI problem, the state feedback controller for Link 1 is computed according to Equation (3.17).

The design of the controller for Link 2 is more complex. Not only because it has an important non-linear behaviour but also because its embedded parameter $\theta_2(q)$ have a discontinuity in $q_2 = 0$. This forces to split in two the working region of $\theta_2(q)$ for positive and negative values of q_2. Thus, forcing the design of a hybrid LPV gain-scheduling controller. This implies that two sets of controllers, one for each region of q_2, will be computed as two independent set of gain-Scheduling controllers within the same LMI problem. Because both regions will share the same Lyapunov matrix P, if it is found to be positive defined, it will not only guarantee the stability and performance within each region but also while switching between them. However, the LPV controller sets for the positive and negative region of q_2 will be independent between them and will not be scheduled together.

On the other hand, $\theta_3(q, \dot{q})$ was found to be almost constant, this simplifies greatly the control design as assuming $\theta_3(q, \dot{q})$ have a constant value means that for each region of q_2 the LPV polytope have only two vertexes, the minimum and maximum values of $\theta_2(q)$, which can be seen in Table 3.2. Thus, for each region, the gain-scheduling controller set will be formed by only a pair of controllers, which make the scheduling quite simple as it is a simple linear interpolation between them with $\theta_2(q)$ as the scheduling variable.

The set of LMIs used as constraints to compute the controller for Link 2 are

$$P > 0 \tag{3.18}$$

$$A_i P + P A_i^T + B W_i + W_i^T B + 2\alpha P < 0 \tag{3.19}$$

$$M \otimes (A_i P) + M^T \otimes (P A_i^T) + M \otimes (B W_i) + M^T \otimes (W_i^T B^T) < 0 \tag{3.20}$$

$$M = \begin{bmatrix} \sin\vartheta & \cos\vartheta \\ -\cos\vartheta & \sin\vartheta \end{bmatrix} \tag{3.21}$$

Equation (3.19) represents the LMI used to set the desired settling time, it is similar to that used for Link 1. However, note that in this case the auxiliary vector of optimization variables W_i is unique for each of the polytope vertexes, as for each vertex a different K_i will be computed. The LMI presented in (3.20) is used to avoid the oscillations present on the system behaviour.

By selecting the value of ϑ in Equation (3.21) sufficiently small, we can force the system to have all its poles on the negative real axis, thus avoiding the presence of oscillations in the controlled system response. Note that $\otimes$ represents the Kronecker tensor product.

After solving the LMI problem, the controllers K_i can be computed according

$$K_i = W_i P^{-1}. \tag{3.22}$$

The final step is to design the scheduling function. As each region have only two LPV polytope extremes, only one embedded parameter, the scheduling function is a simple linear interpolation

$$h = \frac{\theta_2(q) - \underline{\theta_2}}{\overline{\theta_2} - \underline{\theta_2}} \tag{3.23}$$

Finally, the controller to be applied in real-time is computed according to

$$K = \underline{K}(1 - h) + \overline{K}h \tag{3.24}$$

Note that $\underline{K}$ and $\overline{K}$ represent the controllers computed in the minimum and maximum values of $\theta_2(q)$ respectively, $\underline{\theta_2}$ and $\overline{\theta_2}$, this is done for each of the two hybrid regions independently.

Finally, the design for the controller of Link 3 is the simplest of all, as its simplified model is just a linear system. For the design of the controller only two LMIs are needed,

$$P > 0 \tag{3.25}$$

$$AP + PA^T + BW + W^T B + 2\alpha P < 0 \tag{3.26}$$

As in the previous cases, after solving the LMI optimization problem the controller for Link 3 is computed with

$$K = WP^{-1} \tag{3.27}$$

The MATLAB script used for computing the controllers as well as the function implementing the real-time scheduling of the controller for Link 2 can be found in Appendix B.

3.4 Closed-Loop Simulations

In this last section, the designed controllers are tested in simulation against the full mathematical model of the 3-Axis Gimbal. The results from the first simulation can be found in Figure 3.8. In this simulation, a step reference for angular position was given to each Link. It can be seen that the controllers are able to take each link of the gimbal to their desired position without any problem.

In the controller design section, it was seen that for Link 2 a Hybrid LPV controller had to

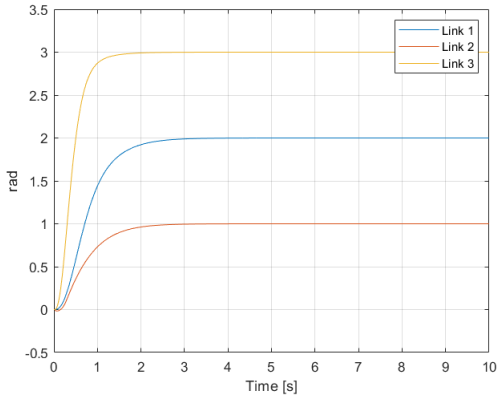

Figure 3.8: Angular position q for all links when a step reference is given.

be implemented. The hybrid element came from the fact the gain-scheduling controllers for the positive and negative region of q_2 act independently within their respective region. To test if the stability is maintained while switching from one region to the other Link 2 was given a sinusoidal reference to follow. The results can be seen in Figure 3.9. Only some small disturbances can be seen when the Link crosses the origin, but in general Link 2 can follow the sinusoidal reference without much trouble and without loosing stability when switching between hybrid regions as the sign of q_2 changes.

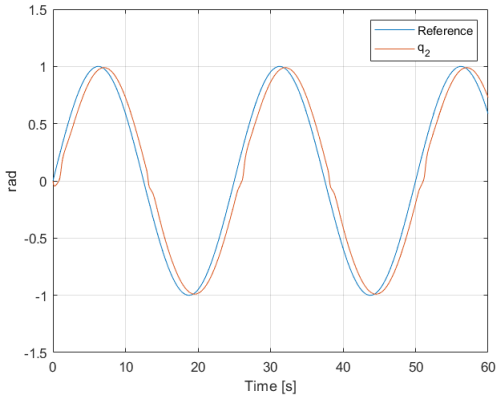

Figure 3.9: Angular position q of Link 2 when a sinusoidal reference is given.

Chapter 4

High Level Planner

In the previous chapter, the low-level control for each of the axis of the gimbal was designed. This level of control is used for giving the individual motors of the 3-Axis Gimbal the torque command required in order to reach the desired angular set-point position. However, since this level of controls makes use of the joint axes coordinates q_i, it is not well suited for controlling the gimbal using setpoints from the 3D space referenced in a Cartesian coordinate frame. For this reason, a higher level of control with the capability to translate the real world coordinates to joint axis coordinates and compute the trajectories between set-points is needed. In this chapter, the inverse kinematics for translating between coordinates frames and the methodology for obtaining the required axis path's is explained. These two tools combined form the so called high level planner. The objective of this planner is to create the trajectories that would make the gimbal transition through a set of real world coordinates in a desired amount of time between transitions.

4.1 Inverse Kinematics

Before starting the discussion on the inverse kinematics, first it should be discussed what the gimbal will actually be used for, as this will dictate the coordinate frame that would be required. The 3-Axis Gimbal is designed to be a stationary camera support with full rotational degrees of freedom. So, its objective is to point the camera to a certain position in the scene while maintaining the camera at a certain tilt angle, normally parallel to the ground plane horizontally.

This means that our desired coordinates space have four coordinates: the X,Y and Z Carte-

sian coordinates, we want the camera to be pointing at and the desired tilt angle. With this defined, now it have to be designed an inverse kinematics algorithm with the goal of transforming from these four coordinates to the gimbal q joint axis coordinates.

Since the problem can be divided into two sub-problems: translating from the Cartesian coordinates X,Y and Z and managing the tilt angle, the inverse kinematics should also be divided in two components.

Representing a Cartesian point using angles is a very old and well known issue, and is solved using the polar coordinate frames, or the spherical coordinate frame for 3D problems as in this case. For carrying the transformation from Cartesian to Spherical frame the MATLAB function $cart2sph()$ is used [6] as follows

$$Azimuth = atan2(y, x) \tag{4.1}$$

$$Elevation = atan2(z, \sqrt{x^2 + y^2}) \tag{4.2}$$

By doing the transformation in this way, the azimuth angle is a positive rotation around the z axis and measured from the x axis, while the elevation is measured from the $x - y$ plane and its sign being set by the z value of the P Cartesian position. The used transformation can be visualized in Figure 4.1. Note that this definition of the spherical angles is not standard. However, for the problem in hand it is the most adequate. This is so, because this definition relates directly the azimuth and elevation angles to the definition of the q_1 and q_2 joint angles of the 3-Axis Gimbal respectively.

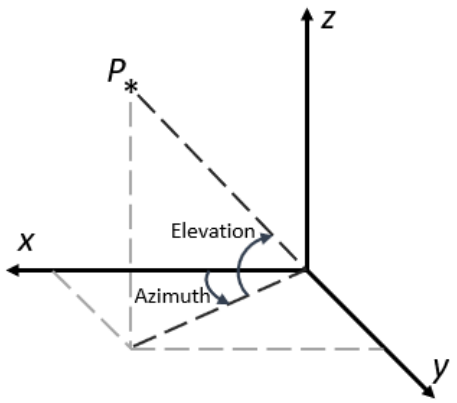

Figure 4.1: Azimuth and Elevation angles representation.

The next step would be to situate the reference frame shown in Figure 4.2 with respect to the gimbal. It could be used the frame 0, shown in Figure 2.1. However, it has two important drawbacks. First, the z axis in that frames point downwards, which makes it not intuitive. Secondly, that frame is not fix as it rotates with q_1. For these reasons, it can be seen that a new fixed frame is needed, which is both with the z axis pointing upwards and fixed to the world. The DH parameters for a new base frame that accomplish these requirements are shown in Table 4.1 and their representation with respect the gimbal in Figure 4.2.

Link	θ_i	a_i	d_i	α_i
Base	$\pi/2$	0	0	π

Table 4.1: DH Parameters for the 3-Axis Gimbal base frame.

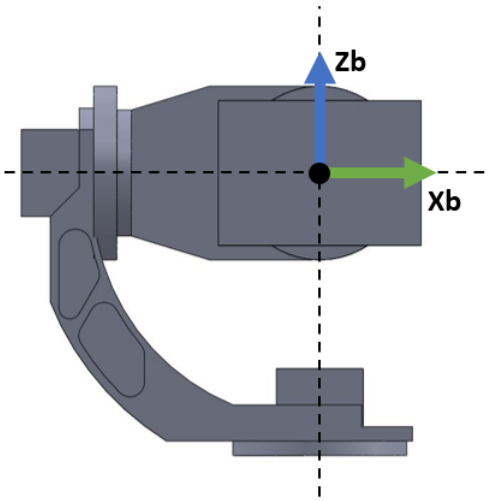

Figure 4.2: Base frame of the 3-Axis Gimbal.

Then, the solution for the inverse kinematics problem for q_1 and q_2 is

$$q_1 = -Azimuth = -atan2(y, x) \tag{4.3}$$

$$q_2 = -Elevation = -atan2(z, \sqrt{x^2 + y^2}) \tag{4.4}$$

The other degree of freedom was the tilt angle of the camera. The tilt angle can be defined as the angle in which the image taken by the camera is rotated with respect to the horizontal ground plane, which can be visualized in Figure 4.3. Taking into account the fact that the camera

is mounted aligned to the 3rd axis of the gimbal, the relationship between the tilt angle and q_3 is quite trivial, it is only needed to consider that their positive direction are opposite

$$q_3 = -Tilt \tag{4.5}$$

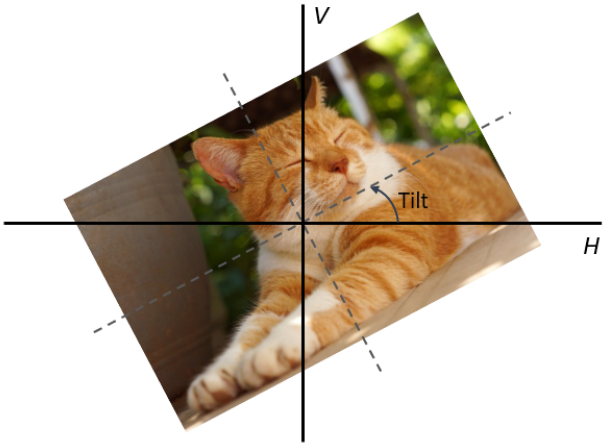

Figure 4.3: Camera image with Tilt angle with respect the Horizontal ground plane.

4.2 Trajectory Generation

After the set of XYZ coordinates, which we would like the gimbal to point at, have been translated into a set of equivalent q joint coordinates, the next step is to create the path that the gimbal should follow from q_o to q_f. Not only that, but the trajectory generation should also take into account the time t_f we would like for the gimbal to take between transitions.

Since the 3-Axis Gimbal will be always at a fixed position and it is expected that its workspace will be collision free, classical approaches to path planning, without considering obstacle avoidance, would be a good solution for this case. Specially, the polynomial interpolation using splines [9] [11, Chapter 4], for its ease of use and good results.

When using splines, a critical decision is to select the order of the spline's polynomial. A frequent choice is to select a polynomial of order 3, since it is the minimum order polynomial, so the simpler, that allows for a smooth path to be generated. The main drawback of using splines of order 3, is that it causes discontinuities to occur in the acceleration at the beginning and at the end of the path. For many applications it may not be an issue, however for the case of a camera gimbal this is not desired. Firstly, in the long run it would cause the actuators to deteriorate. Secondly and most important, this discontinuities between paths may lead to shakiness in the

image, which is the opposite to the objective of using a camera mounted on a gimbal in the first place. When a smooth acceleration during the trajectory is desired, the best decision is to select a spline of order 5, as follows:

$$S(t) = At^5 + Bt^4 + Ct^3 + Dt^2 + Et + F \tag{4.6}$$

$$\dot{S}(t) = 5At^4 + 4Bt^3 + 3Ct^2 + 2Dt + E \tag{4.7}$$

$$\ddot{S}(t) = 20At^3 + 12Bt^2 + 6Ct + 2D \tag{4.8}$$

In order to compute the parameters of the spline, the initial conditions at the start of the trajectory at t_0 are to be selected

$$S(t_0 = 0) = q_0 \tag{4.9}$$

$$\dot{S}(t_0 = 0) = \dot{q}_0 \tag{4.10}$$

$$\ddot{S}(t_0 = 0) = \ddot{q}_0 \tag{4.11}$$

Notice that the initial time is always assumed to be $t_0 = 0$. Also, it is necessary to select the final conditions at the end of the trajectory at t = t_f:

$$S(t = t_f) = q_f \tag{4.12}$$

$$\dot{S}(t = t_f) = \dot{q}_f \tag{4.13}$$

$$\ddot{S}(t = t_f) = \ddot{q}_f \tag{4.14}$$

By substituting these desired conditions into the splines equation, it leads to the following system of equations:

$$\begin{aligned} q_0 &= F \\ \dot{q}_0 &= E \\ \ddot{q}_0 &= 2D \\ q_f &= At_f^5 + Bt_f^4 + Ct_f^3 + Dt_f^2 + Et_f + F \\ \dot{q}_f &= 5At_f^4 + 4Bt_f^3 + 3Ct_f^2 + 2Dt_f + E \\ \ddot{q}_f &= 20At_f^3 + 12Bt_f^2 + 6Ct_f + 2D \end{aligned} \tag{4.15}$$

This system of equation could be simplified by normalizing the path time so its in the interval $\bar{t} \in [0, 1]$ instead of $t \in [0, t_f]$ where

$$\bar{t} = \frac{t}{t_f} \tag{4.16}$$

By using $S(\bar{t})$ instead of $S(t)$ the previous system of equations is transformed into:

$$\begin{aligned} q_f &= F \\ \dot{q}_0 &= E \\ \ddot{q}_0 &= 2D \\ q_f &= A + B + C + D + E + F \\ \dot{q}_f &= 5A + 4B + 3C + 2D + E \\ \ddot{q}_f &= 20A + 12B + 6C + 2D \end{aligned} \tag{4.17}$$

And by solving the system of equations, the value for the spline parameters are as shown in Equation (4.18).

$$\begin{aligned} A &= 6q_f - 3\dot{q}_0 - 6q_0 - \ddot{q}_0/2 + \ddot{q}_f/2 - 3\dot{q}_f \\ B &= 7\dot{q}_f - 15q_f + 8\dot{q}_0 + 15q_0 + 3\ddot{q}_0/2 - \ddot{q}_f \\ C &= \ddot{q}_f/2 + 10q_f - 6\dot{q}_0 - 10q_0 - 3\ddot{q}_0/2 - 4\dot{q}_f \\ D &= \ddot{q}_0/2 \\ E &= \dot{q}_0 \\ F &= q_0 \end{aligned} \tag{4.18}$$

As mentioned before, obtaining a smooth camera movement was the key criteria for selecting an spline of order five. However, in order to enforce this smooth movement, it is required to have continuity in the desired condition between consecutive trajectories. It is done by forcing the initial condition of the path $S_i(\bar{t})$ to be equal to the final conditions of the previous $S_{i-1}(\bar{t})$ path. This can be further simplified by forcing the initial and final conditions for the velocities and acceleration for all paths to be set to zero, $\dot{q}_0 = \ddot{q}_0 = \dot{q}_f = \ddot{q}_f = 0$. With these conditions, the parameters of the spline are:

$$\begin{aligned} A &= 6q_f - 6q_0 \\ B &= -15q_f + 15q_0 \\ C &= 10q_f - 10q_0 \\ D &= 0 \\ E &= 0 \\ F &= q_0 \end{aligned} \tag{4.19}$$

The most interesting aspect about this result is that it is independent of the time, and as so the parameters only depend on the desired initial q_0 and final q_f angular positions. Figure 4.4

shows an example trajectory generated by the spline, it represents a half-turn ($q_0 = 0$ to $q_f = \pi$) of one of the gimbal axis which is desired to last 5 five seconds.

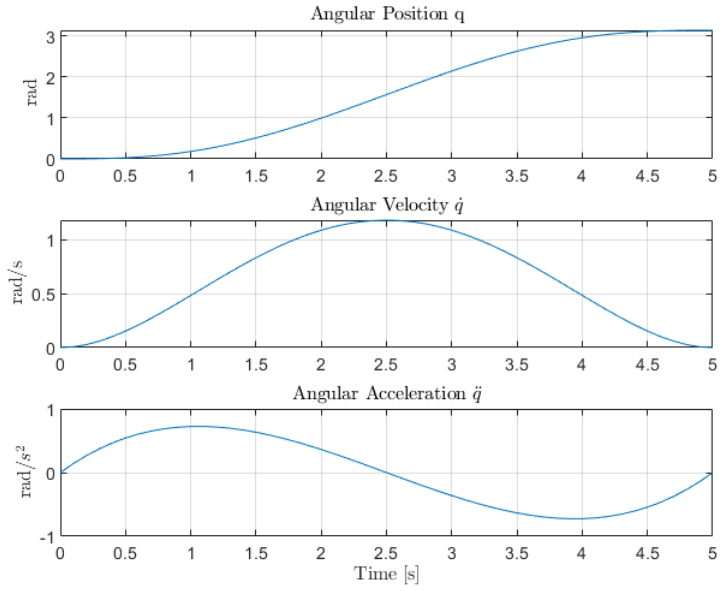

Figure 4.4: Path generated by the 5th order polynomial spline.

4.3 Incorporating the Planner to the 3-Axis Gimbal

Figure 4.5 shows how the planner is interfaced to the closed-loop controlling the angular position of the gimbal axis. The planner inputs are: the set of 3D positions P we want the gimbal to point at, the time instant at which we want the camera to be pointing at the P_i position and the set of $Tilt$ angles for the camera. With this data, the planner can generate the axis reference trajectories that fulfill these constraints as commented in the previous sections. However, two questions remain to be answered. Firstly, at which rate the reference given to the inner controller should be updated? Or what is the same, which should be the sampling time of the path planner? Secondly, which should be the settling time the for the inner controller in order to track the desired trajectory without causing much stress to the actuators?

In order to answer those questions a set of tests have been carried. Table 4.2 shows the sets of coordinates that are being fed to the planner as references. Note that references for the camera $Tilt$ are not being passed, as for the benefit of clarity this section only focuses in the more critical part of the planner, which involves the first two gimbal axes with a higher level of nonlinearity. On the other hand, this could also interpreted as if we desire the camera to remain

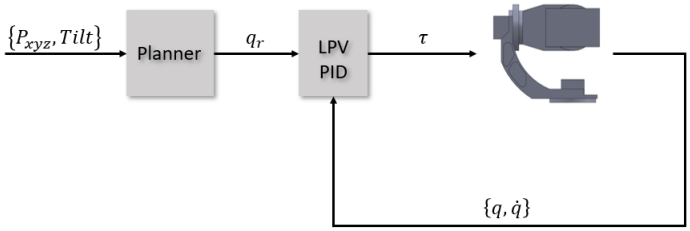

Figure 4.5: 3-Axis Gimbal control system considering the Planner.

parallel to the horizontal plane during the test.

No.	P_i	t_i
0	(1,0,0)	0
1	(1,1,0)	10
2	(0,1,1)	20
3	(1,1,1)	30
4	(-1,-1,1)	45

Table 4.2: Position setpoints for the gimbal end-effector.

In order to understand the relationship between how the planner sampling frequency and the LPV controller settling time requirement affect the 3-Axis Gimbal, two extreme cases are tested. Figure 4.6 shows the result of using a slow settling time, 5 seconds, and a planner sampling frequency of 5 Hz. It can be seen that the performance is not acceptable, caused by the fact that the slow controller cannot cope effectively with nonlinearities, however the trajectory sampling at 5 Hz can be considered to be adequate. Figure 4.7 is the result of a test with a very fast settling time requirement, 0.1 seconds, and a planner sampling frequency of 1 Hz. The resulting path is far from being smooth as the planner frequency is obviously too slow. Moreover, Figure 4.8 shows the control inputs from this last test, the results are clearly not acceptable as it can be seen that each time the planner updates the controller reference huge spikes appears. The magnitude of these spikes in the control signal, caused by the aggressive settling time requirement, in reality only would cause saturation on the actuators and degrade the control performance.

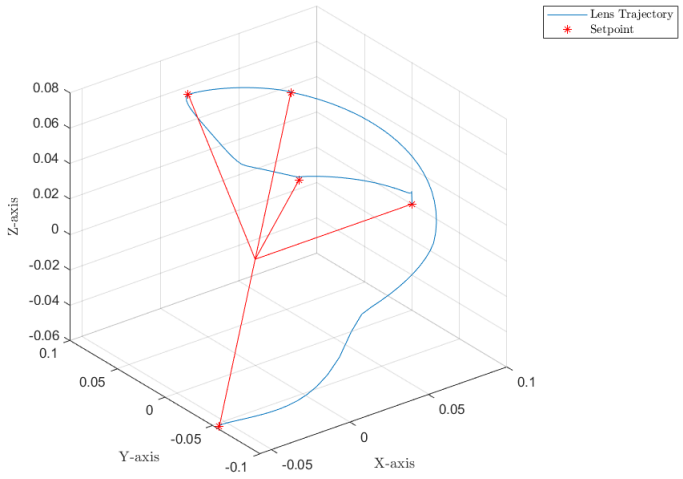

Figure 4.6: Gimbal end-effector path. Controller settling time set at 5 seconds. Planner frequency set at 5 Hz.

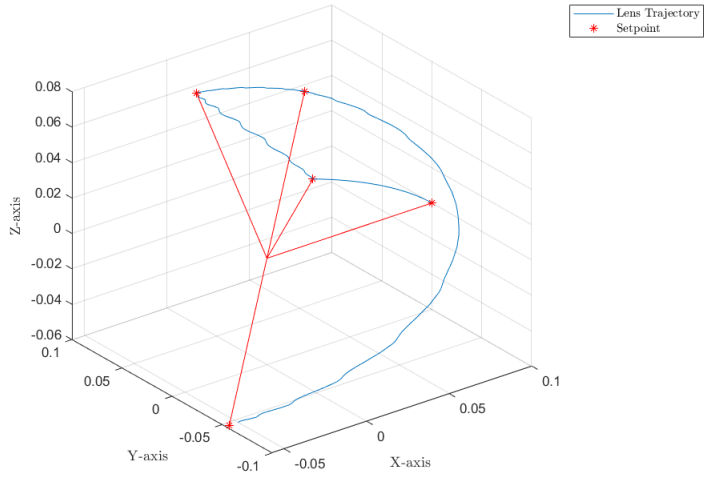

Figure 4.7: Gimbal end-effector path. Controller settling time set at 0.1 seconds. Planner frequency set at 1 Hz.

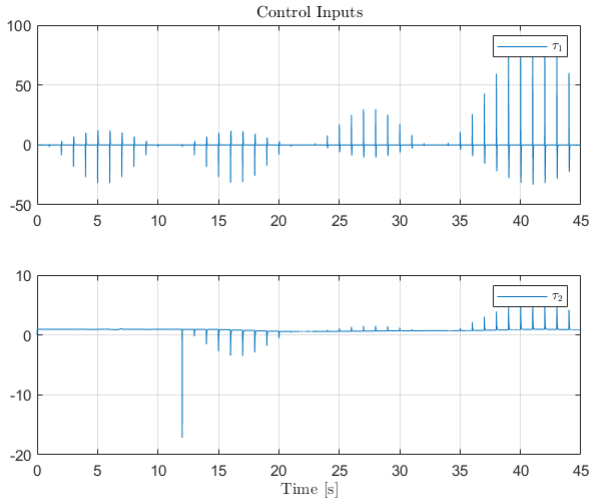

Figure 4.8: Control input. Controller settling time set at 0.1 seconds. Planner frequency set at 1 Hz.

What can be learned from these experiments is that we need a fast planner, at least running at 5 Hz to have good results, and we also need a fast settling time requirement so the controller can cope with the nonlinear effects. However, a value too aggressive would cause an undesired control input characterized by quick sudden spikes of high magnitude, which would quickly lead to actuator saturation. Also, even if those control inputs would be possible, it was seen that when the settling time is lower than the planner sampling rate this leads to a non smooth path resembling a series of steps responses.

With these guidelines in mind and after a series of tests a good trade off was found by setting the LPV controller settling time 3 times slower than the sampling frequency set at 5 Hz. The final result can be seen Figures 4.9, 4.10 and 4.11. The resulting path is shown to be smooth and with a good tracking performance, in addition the control input this time are within an adequate magnitude range.

With the sampling rate defined, at this points all the parameters and elements that form the High Level Planner have been established and the full algorithm step by step as implemented for the 3-Axis Gimbal is summarized in Algorithm 1.

Algorithm 1: High Level Planner Algorithm

Input: P,$Tilt$,T,T_s

Output: q_r

1 Given a set of desired reference Cartesian points P translate them into a set of desired angular positions q_i for Links 1 and 2 using the inverse kinematics Equations (4.3) and (4.4).

2 Do the same for Link 3 from a set of given desired $Tilt$ camera angles using Equation (4.5).

3 For each pair of consecutive points P_i and P_{i+1} compute the splines parameters for the trajectory between them according to Equation (4.19).

4 Generate the trajectory $S(\bar{t})$ between P_i and P_{i+1} by using Equation (4.6) and by normalizing the desired transition time $T_i \in [0, t_f]$ as shown in Equation (4.16).

5 Sample the generated trajectory at a constant rate T_s and feed the sample points $S(\frac{kT_s}{t_f})$ as the q_r reference for the angular position controller of the gimbal axes.

6 Repeat steps **3** to **5** for pairs of consecutive $Tilt_i$ and $Tilt_{i+1}$ desired camera angles to generate the reference path for Link 3.

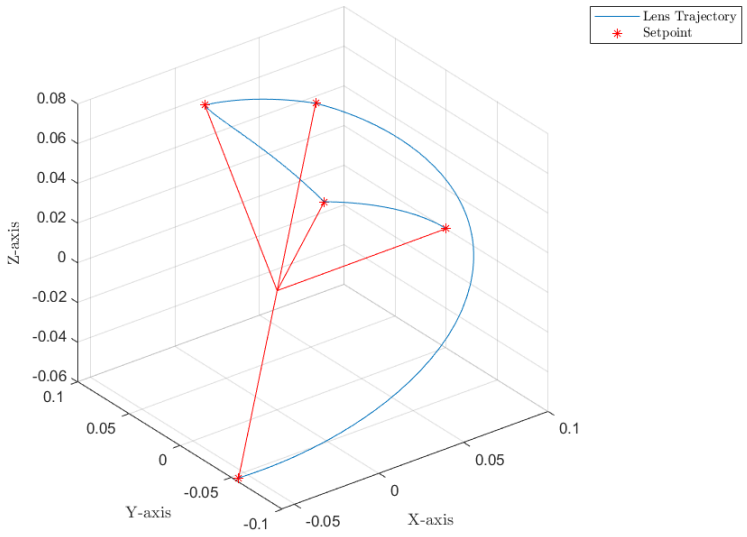

Figure 4.9: Gimbal end-effector path. Controller settling time set at 0.6 seconds. Planner frequency set at 5 Hz.

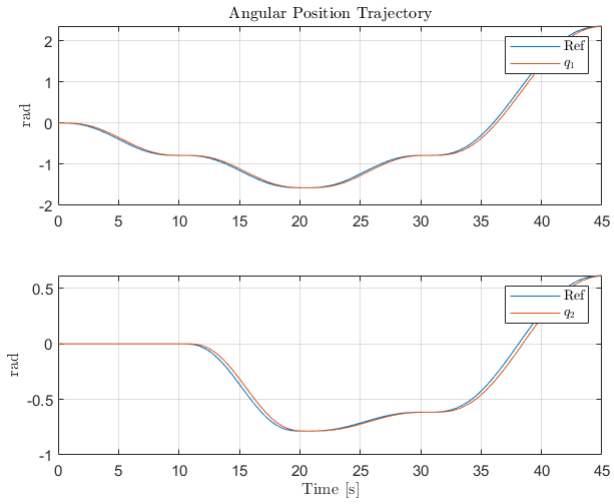

Figure 4.10: Axes tracking the path generated by the planner. Controller settling time set at 0.6 seconds. Planner frequency set at 5 Hz.

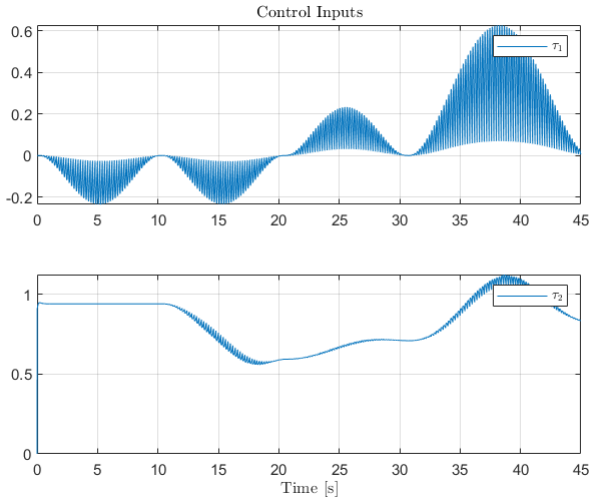

Figure 4.11: Control input. Controller settling time set at 0.6 seconds. Planner frequency set at 5 Hz.

Chapter 5

Sensor Fault Tolerant Strategy

The 3-Axis Gimbal counts with a high number of sensors. Each individual axis of the gimbal have halls effect sensors and a differential encoder attached to the motor plus an absolute encoder attached directly to gimbal body. Not counting the hall effect sensors, which are only used internally for the servodrivers, the gimbal have a total of 6 sensors. Three absolute encoders are used to measure the axes position and three incremental encoders are used to measure the angular velocities. With this number of sensors, some of them are susceptible to suffer a malfunction at some point. However, thanks to the sensor redundancy, there is more than one sensor per axis being used, it makes it possible to use fault tolerant techniques to deal with these malfunctions in the case they appear. In this chapter, a fault tolerant technique to deal with partial sensors faults is presented and applied to the case of the 3-Axis Gimbal.

5.1 Sensor Fault Detection and Virtual Sensor Algorithm

The main objective behind the Fault Tolerant Control (FTC) techniques are to reconfigure the plant in such a way that the nominal controller could still be used in spite of the fault. In other words, for the case of sensor faults, the objective is to hide to the controller the fault occurring in the sensors by reconstructing the signal, which is done using a virtual sensor. Figure 5.1 give an overview of how the virtual sensor based FTC module fits into the greater picture in the control system for the 3-Axis Gimbal.

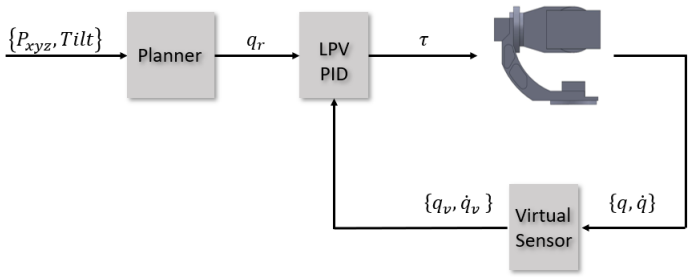

Figure 5.1: 3-Axis Gimbal control system considering the FTC.

When dealing with sensors, the faults can be divided into two groups: total faults and partial faults. Total faults represent a total loss of the sensor signal, as if it were unplugged, whereas partial faults are the rest of faults that may affect the sensor without loosing the signal itself, this may be due to an unexpected change in the sensor gain or some unwanted offset or even drift.

Total sensor fault for the case of the gimbal is considered to be very unlikely, while partial faults, despite having a low chance of occurring, there exists a real possibility of they being a threat. For this reasons the FTC strategy designed in this chapter is for the case of partial sensor faults only.

According to [10], the sensor output can be divided as a part affected by multiplicative faults and a second element representing additive faults. For the case of the 3-Axis Gimbal, the sensor outputs for each of the individual gimbal axes can be expressed according to the mentioned representation as:

$$\begin{bmatrix} q_{yf}(k) \\ \dot{q}_{yf}(k) \end{bmatrix} = \begin{bmatrix} \gamma_1(k) & 0 \\ 0 & \gamma_2(k) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{q}(k) \\ \hat{\dot{q}}(k) \end{bmatrix} + \begin{bmatrix} f_{y1}(k) \\ f_{y2}(k) \end{bmatrix} \tag{5.1}$$

where q_{yf} and $\dot{q}_{yf}(k)$ represent the faulty output from the sensors, $\hat{q}(k)$ and $\hat{\dot{q}}(k)$ represent the states without fault, γ_i represents the multiplicative sensor fault and f_{yi} represents the additive fault. Notice that in the nominal situation, when there is no fault occurring, the values of the multiplicative and additive fault terms would be γ_i=1 and f_{yi}=0 respectively.

The proposed method for achieving FTC has a first and critical requirement: the online fault detection. The strategy for fault detection is to make an online parameter identification of the fault parameters γ_i and f_{yi}. In order to accomplish the parameter identification task, the output representation from Equation (5.1) should be transformed into an equivalent regressor

form. To do it, first we should look at the output from each sensor individually, focusing on the position sensor Equation (5.1) results in:

$$q_{yf}(k) = \gamma_1(k)\hat{q}(k) + f_{y1}(k) \tag{5.2}$$

which can be easily transformed into a regressor equation with γ and f_y as the unknown variables to be determined:

$$q_{yf}(k) = \begin{bmatrix} \hat{q}(k) & 1 \end{bmatrix} \begin{bmatrix} \gamma(k) \\ f_y(k) \end{bmatrix} \tag{5.3}$$

By solving the regressor problem the fault parameters can be obtained. Given their estimation and by manipulating Equation (5.2) it is straightforward to obtain the virtual sensor output y_v equation, and it is a simple static compensation:

$$y_v = \frac{q_{yf} - f_y}{\gamma} \tag{5.4}$$

However, before being able to compute the virtual sensor equation first the regressor problem should be solved for the parameter identification, using q_{yf} and $\hat{q}$ as our known variables. The variable q_{yf} is the output from the faulty sensor, so it is trivial to obtain its value. However, as was mentioned, $\hat{q}$ is the fault free state. Since the objective of the FTC strategy is to reconstruct the faulty sensor signal, to assume the previous knowledge of it for the fault detection may seem contradictory. In practice, the way this is achieved is by using an state observer to estimate it, hence the hat in the formulation.

At this point, all the elements from the FTC scheme have been presented. In order to clarify the pretended FTC strategy a visual summary of it can be seen in Figure 5.2. Comparing the faulty sensor with the fault free state, we can detect the faults by means of the parameter identification from the regressor. From the estimated parameters and the faulty signal, the faulty free signal can be reconstructed by means of the virtual sensor using Equation (5.4). Finally, the outputs from the virtual sensor are used for estimating the fault free states of the system. The observers used, as well as the synthesis of the observers gain L, will be discussed in the next section, for now it will be discussed mainly the fault detection method.

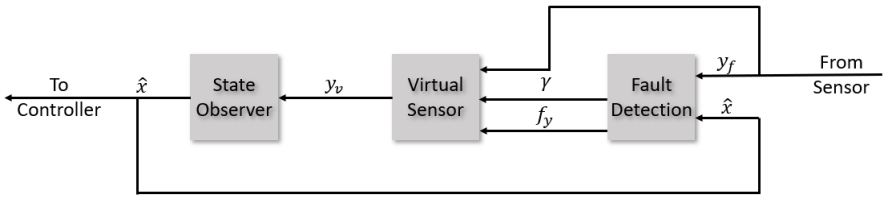

Figure 5.2: FTC scheme diagram representation.

Going back to the parameter identification problem, with the knowledge of both variable q_{yf} and $\hat{q}$, there exists a number of ways to make an online identication of the fault parameters. The classic approach is to solve the regressor problem using the Moore-Penrose pseudoinverse to isolate the vector of unknown variables as follows

$$\begin{bmatrix} \gamma_1(k) \\ f_{y1}(k) \end{bmatrix} = \begin{bmatrix} \hat{q}(k) & 1 \\ \hat{q}(k-1) & 1 \\ \vdots & \vdots \\ \hat{q}(k-n) & 1 \end{bmatrix}^{+} \begin{bmatrix} q_{yf}(k) \\ q_{yf}(k-1) \\ \vdots \\ q_{yf}(k-n) \end{bmatrix} \tag{5.5}$$

A few things should be commented about that equation. First, as can be seen, up to n past data points are required, with n being a tunable variable. Also, the fact that previous data points are required implies that the implementation of the parameter identification strategy must be done in discrete time. Since the parameter identification in this case is also the fault detection method, it is considered as a crucial element and as such it should react at a higher frequency in comparison with the gimbal dynamics. Since the 3-Axis Gimbal representation being used so far is only from the mechanical point of view, its dynamics can be considered as medium to slow in the world of discrete systems, for this reason a sampling frequency of 100Hz have been considered sufficient.

In order to select how many n previous data points should be used, the parameter identification is tested in open-loop with different values of n. For clarification, open-loop in this scenario means that the faulty signals are not being fed to the controller and are only used for testing the parameter identification method. For this test, the fault parameters affecting the sensors are shown in Table 5.1 with a time interval of 10 seconds. Figure 5.3 shows the online

estimation of the parameters. It can be seen that the estimation does indeed converge to the values shown in Table 5.1, proving that the proposed method is a good strategy for fault detection in the case of partial sensor faults. There does not appear to be much of a difference when the numbers of past samples n being used changes, taking this into consideration, the selected value for n is chosen to be $n = 10$. When using a value of $n = 10$, there is a minor improvement in convergence speed, in addition a lower number of data samples also implies less computational burden when applying the required Moore-Penrose pseudoinverse at every iteration.

	0-10	**10-20**	**20-30**	**30-40**	**40-50**	**50-60**
γ	1	1	0.6	1	1	1
f_y	0	0	0	0	0.3	0

Table 5.1: Sensor faults applied

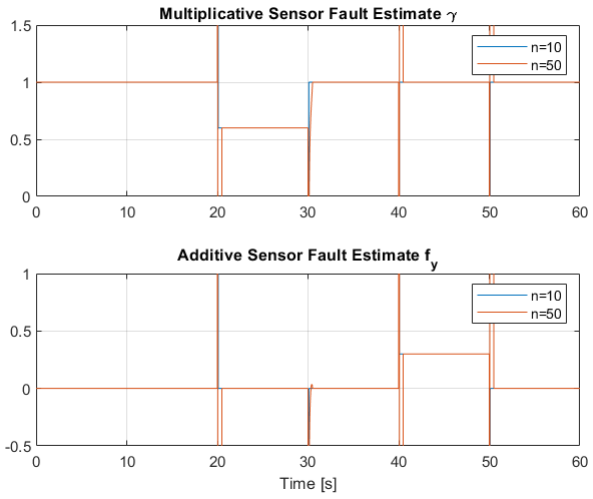

Figure 5.3: Open-loop estimation of γ_i and f_{yi}.

The next step is to test the fault parameter identification in closed-loop. This test implies that observers have already been designed and implemented as the fault detection depends on the state estimation of the angular position q and angular velocity $\dot{q}$. However, as mentioned before, the design of observers will be explained in next sections and for now the focus will be the fault parameter identification and its performance. For this test, it was used the same sequence of sensor faults shown Table 5.1 as in the open-loop case.

Figure 5.4 shows the performance in closed-loop of the FTC scheme using the pseudoin-

verse method for parameter identification of the sensor faults. It can be seen that the improvement in comparison to the case when there is not FTC strategy in place is quite clear. Without FTC scheme, the controller reacts to the faults as if there were a change in reference. It is good that faults do not cause instability issues but the performance losses are far from being desirable anyway. On the other hand, when using the FTC scheme the system is not being affected by the faults at all, which is a great result. However, it can be seen that when the FTC scheme is being used, the angular position does not converge to the reference and there is always a small steady state error present. This is being directly caused by the fault parameter identification, shown in Figure 5.5. The estimation for the additive sensor fault f_y is on point as expected, in the fault free intervals it estimates a value of 0 and quickly converges to the actual fault value when it happens. The problems are in the estimation of the multiplicative sensor fault γ, it is capable of converging quickly as well when a fault occurs. Howeverm in the fault free intervals, it does not converges exactly to 1 as it should. This deviation in the fault free scenario estimation carries on to the others components of the FTC scheme and ends up causing the steady state error. This is not an acceptable result, as trying to compensate for the small chance that a fault may appear would decrease the nominal performance of the whole system.

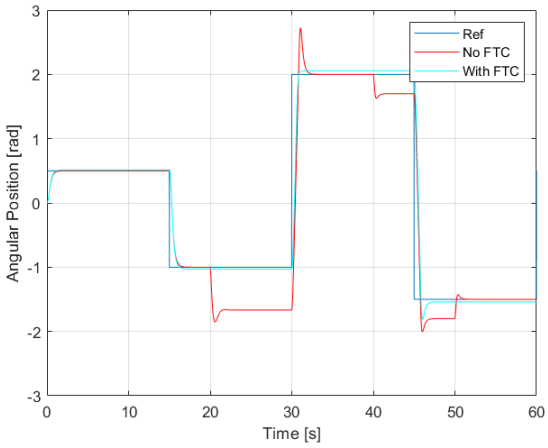

Figure 5.4: Closed-loop performance in the presence of sensor faults.

In order to remedy the steady state error problem, a simple compensation algorithm is proposed. The fact that γ does not converges to 1 when there is no fault happening causes the output from the virtual sensor y_v, Equation (5.4), to be slightly different to what it should be. So, the proposed idea is to obtain a new compensated virtual sensor output y_v^* that is forced to

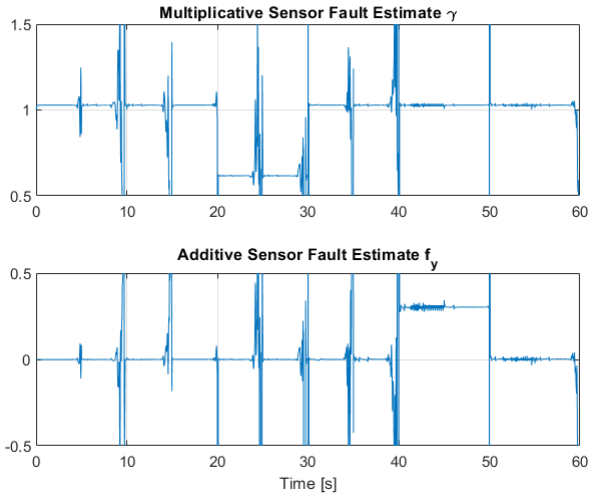

Figure 5.5: Closed-loop estimation of γ_i and f_{yi}.

converge to the sensor when there is no evidence of a fault happening, Equation (5.6).

$$y_v^* = y_v - eR(\gamma) \tag{5.6}$$

Where

$$e = y_v - y_f \tag{5.7}$$

$$R(\gamma) = \begin{cases} 1, & \text{if } |\gamma - 1| \leq 0.03 \\ 0, & \text{otherwise} \end{cases} \tag{5.8}$$

The term e represents the error that is being caused by γ not converging to 1 when it should. While in Equation (5.8), the term $|\gamma - 1|$ can be considered as a parameter estimation based residual, since it should be almost zero in the nominal case and only greater in case of a sensor fault. When this happens, then $R(\gamma)$ deactivates this compensation, as the sensor can no longer be consider to be trusty and the output from the virtual sensor is preferred.

Figure 5.6 shows the closed-loop response of the system when the correction is added to the FTC scheme, the sensor faults applied in this tests are those from Table 5.1. It can be seen that during the initial fault free intervals, the angular position converges to the desired reference, which is an improvement over the original FTC scheme without compensation. Also, when the multiplicative fault occurs, interval 20 to 30 seconds, the system response is not affected by it. Again, a great result fulfilling the expected objectives for the FTC strategy. However, during the second fault scenario, additive sensor fault during the interval 40 to 50 seconds, it can be seen that the system behaviour is far from what it is expected and is definitely not tolerant to the fault. The reason can be found in the estimation of the sensors faults parameters, Figure 5.7. After the first fault occurrence, the estimation is very jagged and not able to converge to a single fault parameter. This new dynamics in the estimation may be attributed directly to the $R(\gamma)$ factor due to rapid consecutive on/off switching of the correction. On the other hand, the reason why this nervous behaviour does not transfer to the output of the virtual sensor is because the estimated values of γ and f_y compensate each other. In conclusion, the addition of the correction to the FTC scheme seemed promising at first, but it is too sensitive to successive faults and does not seem able to cope with additive faults properly.

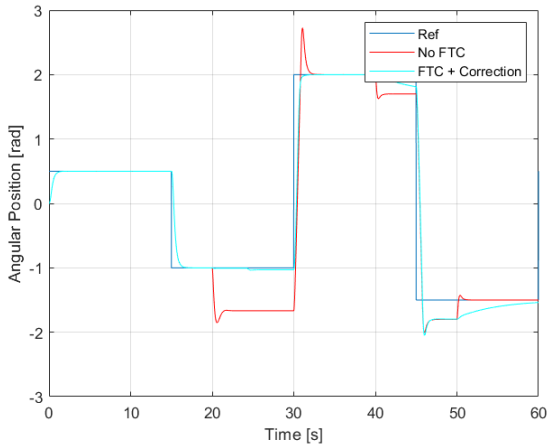

Figure 5.6: Closed-loop performance in the presence of sensor faults.

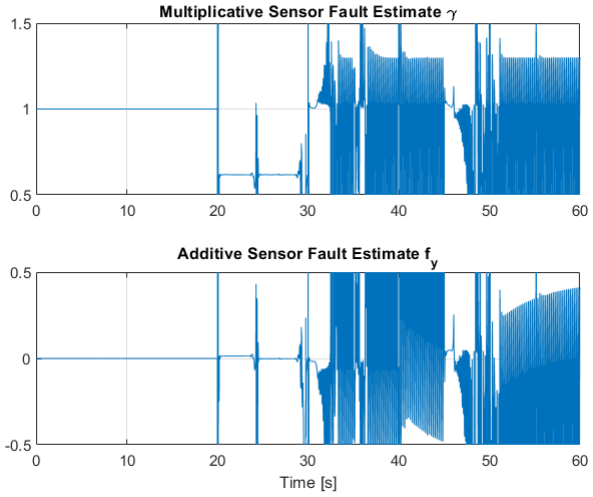

Figure 5.7: Closed-loop estimation of γ and f_y.

The reason why the correction was needed in the first place was because the fault parameter identification failed to converge successfully to the expected value of 1 in the fault free scenario. Then, this offset carried to the virtual sensors producing an steady-state error in the system response. Instead of trying to keep attempting to make improvements to the correction, the best course of action should be to tackle the source of the problem: the parameter identification. With this in mind, a new parameter identification strategy is tested, the Recursive Least Square (RLS) will be used this time to estimate the values of γ and f_y, instead of solving the regressor problem using the pseudoinverse method. The implementation was done using the Simulink block Recursive Least Squares Estimator (RLS) [7] using a forgetting factor λ=0.9. The forget-

ting factor is necessary since the parameters to be identified are not static and by using it the RLS estimation is forced to give greater importance to the past T_0 samples.

$$T_0 = \frac{1}{1-\lambda} \tag{5.9}$$

From Equation (5.9), it can be seen that by selecting λ=0.9, it is given a greater importance to the past $T_0 = 10$ samples, as was the case for the value n selected for the pseudoinverse method. Figure 5.8 shows the response of the system in closed-loop and Figure 5.9 shows the fault parameter estimation. It can be seen that this time the system is fully tolerant to both types of faults and also the steady state error is null. As for the fault parameter identification, the estimation using RLS is as good as in the case of the open loop results, as can be seen in Figure 5.3. In conclusion, the switch to RLS for the fault estimation delivers the best results and great FTC performance, making the system fully tolerant to partial sensor faults without loosing nominal performance.

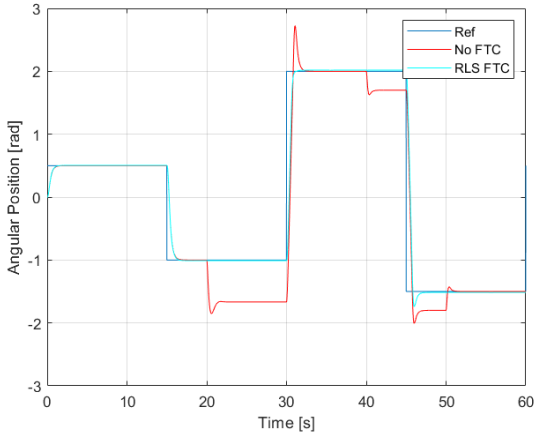

Figure 5.8: Closed-loop performance in the presence of sensor faults.

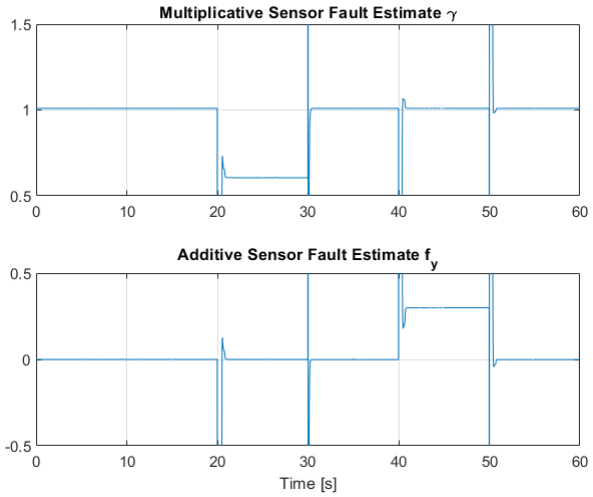

Figure 5.9: Closed-loop estimation of γ and f_y.

5.2 Observer Design for the 3-Axis Gimbal

As seen in the proposed FTC scheme, Figure 5.2, besides the fault detection and virtual sensor the other crucial element is the state observer. Why is that? Firstly, as was mentioned, the state observers are the elements that provide $\hat{q}$, which is the fault free state and elementary input for the fault detection method to work. However, the observer inputs are given by the combined processes of fault detection and the virtual sensors, so all three elements all deeply interlinked. This interconnection between elements means that not every observer works, it has to fulfill some requirements. Although it that may seem contradictory for control, the most important of these requirements is that the observer needs to be slow, or at least needs to be significantly slower than the fault detection process. This can be thought of as if the observer needs to have some inertia that forces it to continue its previous trajectory in the case of a sudden sensor fault, which will cause the sensor signal and state estimation to diverge. These samples in which there is a difference between state estimation and the faulty sensor signal is what will allow the parameter identification to make a good estimation of the faults and consequently construct a useful virtual sensor signal.

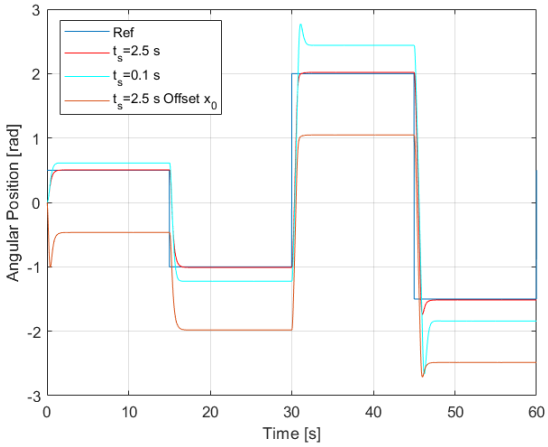

Figure 5.10: State observer-FTC interaction test.

In order to check what effects does have the interactions commented between the state observer speed and the rest of the FTC elements a number of tests have been carried, the results can be seen in Figure 5.10. It compares the performance when the observer settling time is set to $t_s = 2.5$s, when it is using a faster settling time at $t_0 = 0.1$s and also when it is back to $t_s = 2.5$s but the initial condition from the observer and the real system are different. The settling time at $t_s = 2.5$s is the settling time that was used in the previous section, and as before it has proven to have great performance. With the other two cases the situation is quite different. A fast observer causes the parameter identification to converge to a wrong set of fault parameters γ and f_y, and as seen before this causes steady state error in the closed loop system. On the other hand, having an slow observer means that if the initial conditions of the observer and real system differ for enough time this will be seen by the FTC elements as a fault. As seen in the results, this causes a big offset with respect the reference. This last problem can be avoided by given the observer a grace period in which the FTC module will not be acting, so the observer will have time to converge. For the rest of the chapter, it will be assumed that observer and gimbal system start from the same initial conditions.

With the most important requirement already defined, the next step is the design and implementation of the observers itself. The design philosophy behind the observers is the same as in controller design, there will be a individual observer for each of the three axes that form the gimbal. In order to compute the observers it will be used the LPV models obtained in Section 3.2 and the computation of the observers gain will be done using a very similar approach as in Section 3.3 applying optimization based LMI techniques to obtain L. In fact, using the

controller-observer duality property, the same LMI constraints used to compute the controller gain can be used by simply applying the following transformations:

$$A \Longrightarrow A^T = \tilde{A} \tag{5.10}$$

$$B \Longrightarrow C^T = \tilde{B} \tag{5.11}$$

And by transposing the computed gain K it is obtained the observer gain L:

$$L = K^T \tag{5.12}$$

Starting from the first axis of the gimbal with Link 1, lets recall its plant model seen in Equation (3.7) and its embedded parameters from Table 3.1. Due to the small variability in the embedded parameter, the synthesis philosophy for computing the controller for the first link was to obtain a single robust controller for all the space of the LPV polytope. In practice, this is achieved by using only one controller as the variable of the LMI optimization problem that covers both vertexes of the LPV polytope of the Link 1 model. Following the same approach, LMI problem used for computing a robust observer gain are:

$$P > 0 \tag{5.13}$$

$$\tilde{A}_i P + P\tilde{A}_i^T + \tilde{B}W + W^T\tilde{B} + 2\alpha P < 0 \tag{5.14}$$

$$M \otimes (\tilde{A}_i P) + M^T \otimes (P\tilde{A}_i^T) + M \otimes (\tilde{B}W) + M^T \otimes (W^T\tilde{B}^T) < 0 \tag{5.15}$$

Where

$$M = \begin{bmatrix} \sin\vartheta & \cos\vartheta \\ -\cos\vartheta & \sin\vartheta \end{bmatrix} \tag{5.16}$$

And after solving the LMI problem the observer gain is computed as:

$$L = (WP^{-1})^T \tag{5.17}$$

It can be seen that for each of the two vertexes of the LPV polytope, two LMI constraints are applied, plus the Lyapunov feasibility constraint, Equation (5.13), a total of five LMI constraints have been used. By selecting an appropriate value for α, the observer settling time can be chosen and a small value for ϑ is used to bound the imaginary part of the observers poles.

With the observer gain synthesis done, the implementation of the observer is straightforward by using the classical Luenberg observer, Equation (5.19). Since the system matrix A from

Link 1 had a nonlinear embedded parameter, seen in Equation (3.7), this means that for obtaining the A value in real time it would be necessary to use a scheduling approach. However, given the soft nonlinearities of Link 1, the approach used was to approximate A as the result of evaluating the LPV matrix of the model at the middle point of the LPV polytope, resulting in

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -10.745 \end{bmatrix} \tag{5.18}$$

$$\dot{x} = Ax + Bu - L(y - Cx) \tag{5.19}$$

For the synthesis of the observer gain of Link 2 of the gimbal, as in the case of its controller, the situation is a bit more complex due to its heavy nonlinear dynamics and its LPV representation that forces the use of a hybrid observer with two regions depending on the value of the angular position q_2 of Link 2. The LPV model for the second axis is given in Equation (3.9) and the vertexes of the LPV polytope space are given in Table 3.2. The LMI constraints used are the same as in the case of the first axis, with the minor difference that this time each vertex of the LPV polytope space have its own observer gain. Since each of the two hybrid regions have two vertexes, there are nine LMI constraints used after adding the LMI for enforcing Lyapunov feasibility

$$P > 0 \tag{5.20}$$

$$\tilde{A}_i P + P\tilde{A}_i^T + \tilde{B}W_i + W_i^T\tilde{B} + 2\alpha P < 0 \tag{5.21}$$

$$M \otimes (\tilde{A}_i P) + M^T \otimes (P\tilde{A}_i^T) + M \otimes (\tilde{B}W_i) + M^T \otimes (W_i^T\tilde{B}^T) < 0 \tag{5.22}$$

And the individual observer gains are computed as:

$$L_i = (W_i P^{-1})^T \tag{5.23}$$

As for the implementation of the observer, the real-time observer gain L

$$L = \underline{L}(1 - h) + \overline{L}h \tag{5.24}$$

to be used comes from the scheduling function

$$h = \frac{\theta_2(q) - \underline{\theta_2}}{\overline{\theta_2} - \underline{\theta_2}} \tag{5.25}$$

The terms $\underline{L}$ and $\overline{L}$ refer to the observer gains associated to the minimum and maximum vertex of the relative hybrid region. Finally, the observer is again implemented as a Luenberger observer with the minor difference that the A matrix comes from evaluating in real-time the embedded parameter $\theta_2(q_2)$

$$\dot{x} = A(\theta)x + Bu - L(y - Cx) \tag{5.26}$$

Finally, for the design of the observer for Link 3, as in the case of its controller, it is the simplest one. The control model for it is shown in Equation (3.12) and since it is a linear state-space model there are not embedded parameters. The LMI are the same used in the previous axes, and since there are no LPV vertexes only three LMIs are needed to define the constraints. The implementation of the observer is done applying a simple Luenberger observer with the computed gain.

$$P > 0 \tag{5.27}$$

$$\tilde{A}P + P\tilde{A}^T + \tilde{B}W + W^T\tilde{B} + 2\alpha P < 0 \tag{5.28}$$

$$M \otimes (\tilde{A}P) + M^T \otimes (P\tilde{A}^T) + M \otimes (\tilde{B}W) + M^T \otimes (W^T\tilde{B}^T) < 0 \tag{5.29}$$

And the observer gain is:

$$L = (WP^{-1})^T \tag{5.30}$$

The MATLAB script used for computing the observer gains can be found in Appendix C.

5.3 Improved FTC Scheme for the 3-Axis Gimbal

Notice that in the last chapter there have been no comment about the structure and dimension of the system matrix C. It was not mentioned because the design of the observer can be easily adapted to different dimensions of the C matrix. Given that for each axis, both angular position q_i and angular velocity $\dot{q}_i$ states are measured, choices for the matrix C are to have it be $C = I$ or it could be build using only the position sensor, $C = [1\ 0]$. Using only the velocity sensor, $C = [0\ 1]$, is not a possibility because the gimbal system representations then results in a non observable system. Using the model of Link 3 as an example, Equation (3.12), it can be seen that the observability matrix when $C = [0\ 1]$ is not full rank.

$$Ob = \begin{bmatrix} C \\ CA \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -536.2505 \end{bmatrix} \tag{5.31}$$

To begin the design of the observers, the first obvious choice is to use both sensors for estimation, so $C = I$. With this selection of C, in order to compute the observers gains L, the matrix of optimization variables W from the LMI constraints should be selected to be of dimensions 2x2. The results obtained when using this full observer can be seen in Figures 5.11 and 5.12. The results from the position estimation are great. However, the estimation for the angular velocity of Links 2 and 3 are horrible, completely not usable.

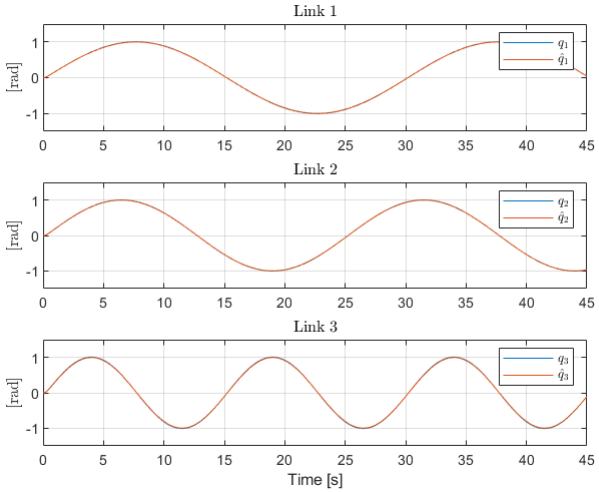

Figure 5.11: Angular position estimation. $C{=}I$.

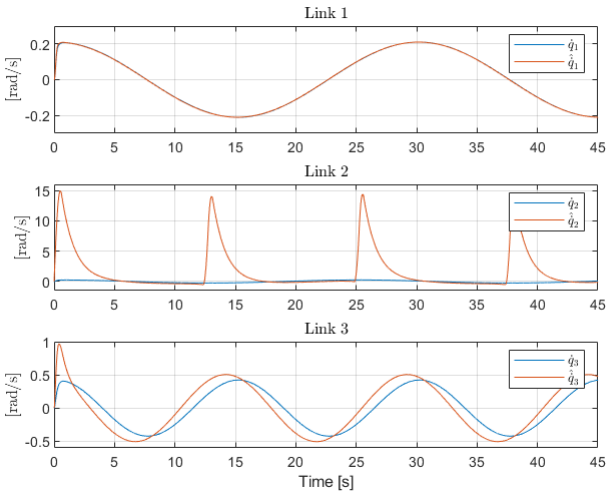

Figure 5.12: Angular velocity estimation. $C{=}I$.

The reason behind this behaviour may be in the observer gains. Taking the third axis as an

example again, its observer gain is:

$$L_3 = \begin{bmatrix} -2.1724 & -1.0000 \\ 0 & 534.0781 \end{bmatrix} \tag{5.32}$$

Clearly the values for the second row presents a great disparity with the rest of the values of the gain matrix, two orders of magnitude greater. Whereas the observer gain for Link 1 presents all values with the same magnitude, as a result it can be seen that its velocity estimation is actually quite good. This points out that the numerical differences within the observers gains is the reason causing the ill estimation problems as suspected. It should be noted that proving different methods to compute the observer gains or also different tuning parameters did not resulted in an improvement with respect to the observed numerical problem

$$L_1 = \begin{bmatrix} -1.0357 & -1.0000 \\ 0 & 8.9771 \end{bmatrix} \tag{5.33}$$

Since the angular velocity estimation $\hat{\dot{q}}$ from the observers when using both sensors cannot be used because how bad it is, a different approach is needed. The observers are computed again, but this time using only the angular position sensor, so C=[1 0]. For the synthesis of these new observers, the only change required in the observer computation design is to adapt the matrix of auxiliary optimization variables W in the LMI constraints formulation to be of dimensions 1x2. The results from the angular velocity estimation are shown in Figure 5.13, results from the position estimation are not shown for the sake of not cluttering since they do not show any improvement in comparison to the previous observer design using $C = I$.

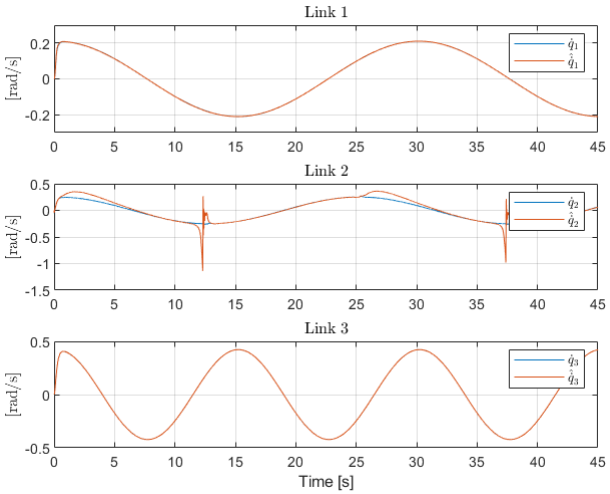

Figure 5.13: Angular velocity estimation. $C = [1\ 0]$.

The results from the estimation of angular velocity show a great improvement in comparison from the previous tests. This time the observer gain matrix for Link 3 is:

$$L_3 = \begin{bmatrix} -1.9399 \\ 0.8465 \end{bmatrix} \tag{5.34}$$

Which this time has values of the same magnitude, as a result the estimation of $\hat{\dot{q}}_3$ this time is very good. The estimation for $\hat{\dot{q}}_2$, on the other hand, is an improvement compared to the results obtained with the previous observer but is still far from perfect and still not suitable for its use in closed-loop. It can be seen that the estimation diverges from the real state at the maximum and minimum point of the velocity curve, which are the points where the angular position is switching from positive to negative and vice versa. This points to the hybrid nature of the observer design for Link 2 as the source of the problems in this case. The estimation is not unstable as can be seen, but the resulting transient dynamics during switching are not admissible.

Since the cause of the problems in the estimation of $\hat{\dot{q}}_2$ was the switching during the change of hybrid regions, the solution is to build a new not hybrid observer. Different techniques for building a nonlinear observer could be used, however taking advantage of the work done until now, there is still a last possibility within the LPV framework to build an observer for the estimation of both states of Link 2. This last opportunity is to compute the observer gain as was done in the case of the first axis, computing just one observer gain that should be robust and give a

good estimation within the whole LPV polytope. To do this, the only required modification to the LMI optimization constraints is to use a global matrix of optimization variables W instead of a different W_i at each of the vertexes of the LPV polytopic space. When using this new robust observer gain for Link 2, it can be seen in Figure 5.14 that this time the estimation is pretty good and is finally usable for feedback in closed-loop.

Due to the fact that the observers using both sensors, $C = I$, were deemed to be not fully

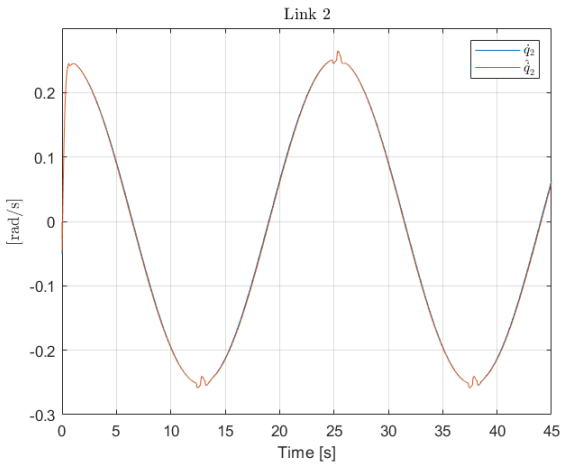

Figure 5.14: Angular velocity estimation. $C = [1\ 0]$.

usable because of the bad estimation of the angular velocities $\hat{\dot{q}}_i$, a new modification to the FTC scheme is proposed. This final scheme can be seen in Figure 5.15. In the diagram, it can be seen that the path for obtaining a fault tolerant estimation of the position $\hat{q}$ is as it was originally proposed. The changes have been done in order to provide the fault tolerant estimation of the angular velocity $\hat{\dot{q}}$. The estimation of the fault free state $\hat{\dot{q}}$ is given by the observers which use only the position information, $C = [1\ 0]$. The signal being used for this does not come from the sensor, as this would easily provoke that a fault in the position sensor will be transformed into a wrong estimation of the angular velocity. Instead, the already fault free position state $\hat{q}$ is used, as this will allow for a fault free estimation of the angular velocity. From there, obtaining the output from the virtual sensor for angular velocity is as in the case of the position sensor: using the estimated $\hat{\dot{q}}$ the fault detection takes place and finally the angular velocity signal can be reconstructed using the virtual sensor Equation (5.4). Finally, the signals used for closing the feedback loop are the fault free position state estimation $\hat{q}$ and the output from the angular velocity virtual sensor $\dot{q}_v$. As a side note, the reason to keep using the observers where $C = I$ is because using both sensors for estimation actually helps to obtain a better position estimation

during a sensor fault.

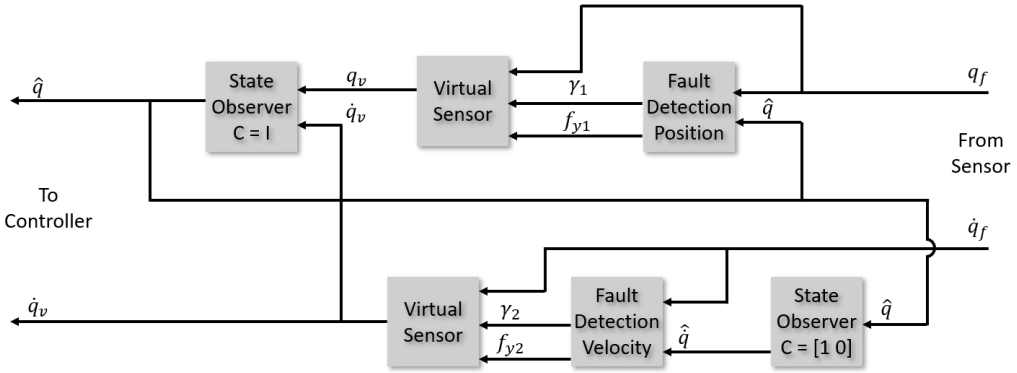

Figure 5.15: Final FTC scheme diagram representation.

5.4 Testing the FTC Performance

In order to test the final proposed FTC scheme, it has been tested in an scenario with multiple occurring partial sensor faults. The fault parameters applied to the sensors are summarized in Tables 5.2 to 5.5. The faults have been only applied to Links 1 and 2 for the sake of clarity in the shown results as they are the most critical axes, in the sense on nonlinearities present, and results for Link 3 do not add any new valuable insight.

	0-10	**10-20**	**20-30**	**30-40**	**40-45**
γ	1	0.7	1	1	0.8
f_y	0	0	0	0.3	0.2

Table 5.2: Sensor faults applied in q_1

	0-10	**10-20**	**20-30**	**30-40**	**40-45**
γ	1	1	1	0.6	0.9
f_y	0	0.2	0	0	0.15

Table 5.3: Sensor faults applied in $\dot{q}_1$

	0-10	**10-20**	**20-30**	**30-40**	**40-45**
γ	1	1	1	0.6	0.9
f_y	0	0.2	0	0	0.15

Table 5.4: Sensor faults applied in q_2

	0-10	**10-20**	**20-30**	**30-40**	**40-45**
γ	1	0.7	1	1	0.8
f_y	0	0	0	0.3	0.2

Table 5.5: Sensor faults applied in $\dot{q}_2$

As can be seen in Figure 5.16, in the case of a sensor fault, the performance of the 3-Axis Gimbal control system will be very degraded. Given that in this test multiple faults have been applied in a short amount of time, the results may seem exaggerated. But it was done to showcase how every one the faults had a great impact in the resulting trajectory of the gimbal end effector. It is shown that without using FTC strategies, the appearance of any of these faults would clearly render the gimbal useless as a camera mount. However, as can be seen in Figure 5.17, when the Fault Tolerant Control scheme is being used the system is able to cope and maintain its performance almost intact, as the virtual sensors are able to reconstruct successfully the sensor signal and thus masking the fault to the controller. The obtained path is very smooth and to the users of the gimbal system the sensor fault would pass unnoticed. With the results shown in this section, the development of the FTC module can be considered as successful.

From Figure 5.18 onward, the results for the fault parameter estimation are presented. It can be seen that the estimation closely match the applied partial sensor faults, shown in the tables at the beginning of this section. The obtained estimation using the Recursive Least Square method proved to give a clean identification without noises and converges to the expected values in the nominal fault free scenario, $\gamma = 1$ and $f_y = 0$. Also, when a fault occurs it quickly converges effectively to real fault occurring in the axis sensor, independently if it is a multiplicative or an additive fault, proving to be robust against both types of failures. In Figure 5.20, it can be seen that in the first interval of 10 seconds, the estimation of the multiplicative fault γ is quite noisy. The reason for this noisy estimation is that for that initial 10 seconds interval the Link 2 does not move, and as it is know in order to have a good estimation performance an exciting

input is needed in the system, which for that interval is not being fulfilled. However, as can be seen in Figure 5.17, it is not translated as an unexpected behaviour in the gimbal response. That is thanks to use of the observer which is successfully estimating that Link 2 is not moving for those 10 seconds, proving that using an observer with $C = I$ is indeed beneficial despite its bad estimation of the angular velocity. As soon as the axis start to moves the estimation for γ performs as expected.

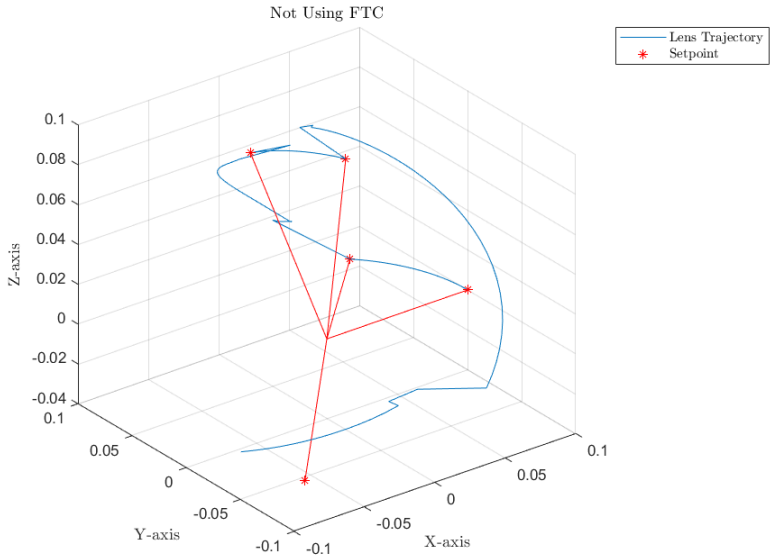

Figure 5.16: Gimbal response in faulty scenario.

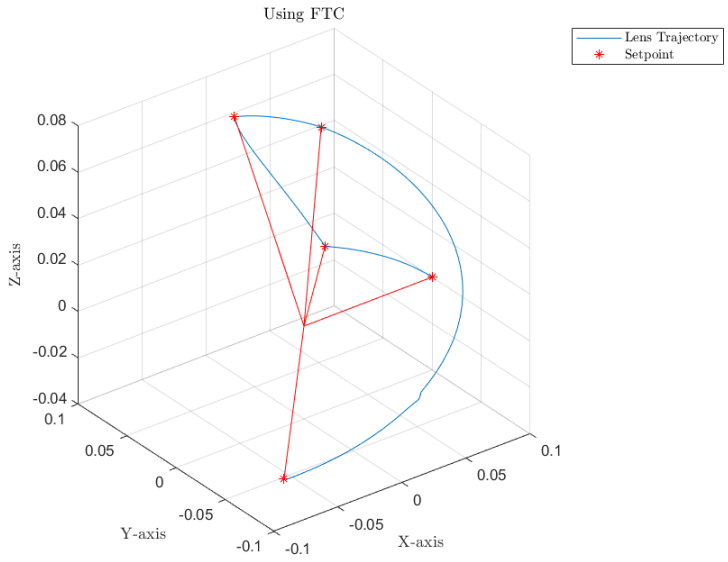

Figure 5.17: Gimbal response in faulty scenario using the FTC scheme.

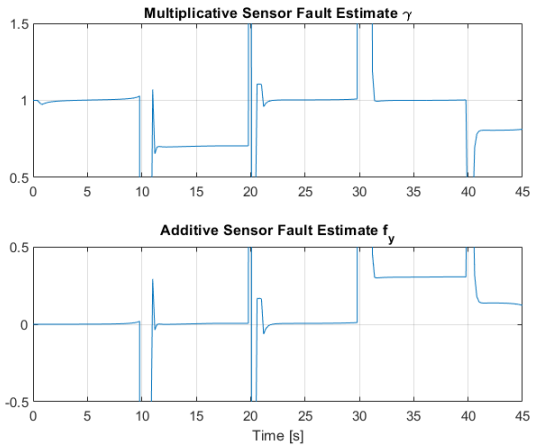

Figure 5.18: Fault parameter estimation for angular position sensor of Link 1 q_1.

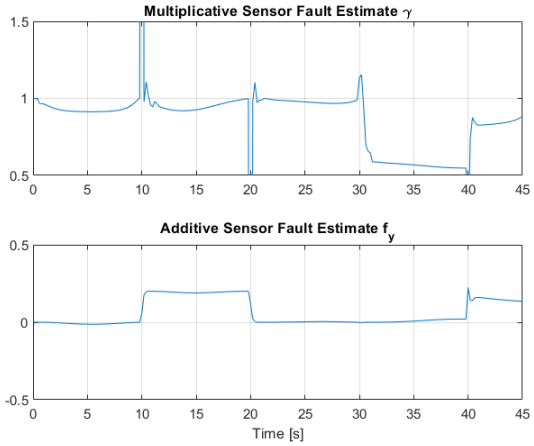

Figure 5.19: Fault parameter estimation for angular velocity sensor of Link 1 $\dot{q}_1$.

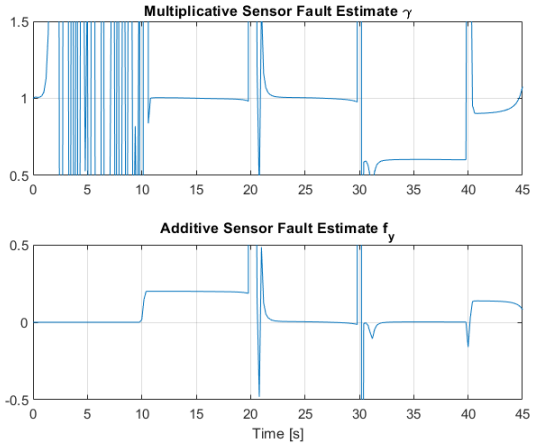

Figure 5.20: Fault parameter estimation for angular position sensor of Link 2 q_2.

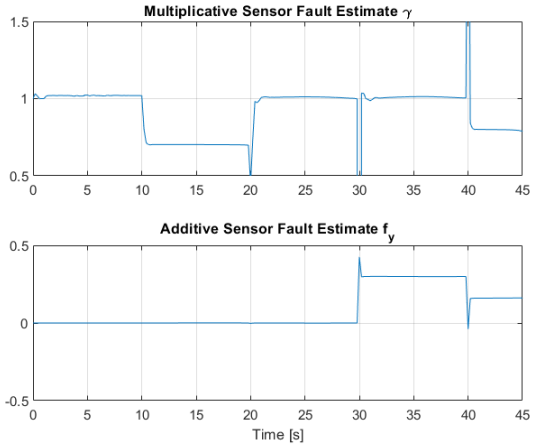

Figure 5.21: Fault parameter estimation for angular velocity sensor of Link 2 $\dot{q}_2$.

Chapter 6

Conclusion

6.1 Summary of the Results

The aim of this Master thesis was to design a full control system for the 3-Axis Gimbal, which with the results obtained it can be considered that this goal was successfully achieved. The first step in the process, and arguably one of the most important results achieved, was being able to obtain a realistic model of the system using the serial-link theory for robotics manipulator. The serial-link theory have been long studied and it plays a crucial role in robotics, the results obtained in this thesis show that it can also extended to gimbal systems, which opens the possibilities for obtaining highly accurate models for commercial gimbal products. Counting with more accurate models is a potential benefit in and of itself but it equally makes it possible so that modern control techniques can be used, as the Gain Scheduling LPV controller implemented in this work. For the synthesis of the controllers, at first, it was shown how the complex model could be simplified and transformed into a suitable LPV model for controller design. Finally, the three controllers for each axis of the gimbal were computed using a different approach for each one, showcasing different examples of uses in which the LPV methodology and LMI are a very adaptable and powerful tool in control problems. Then, in order to be able to give to the 3-Axis Gimbal commands in the real 3D space, and not only in its axes coordinates, an inverse kinematics algorithm suitable for using the gimbal as a camera mount was designed, as well as a simple but effective planner to generate trajectories between 3D Cartesian points. Finally, a fault-tolerant scheme was implemented to give the gimbal robustness against partial sensor faults. Due to observability issues in some states, the initially proposed FTC strategy had to be

adapted to fit the model of the gimbal system representation better. When using the modified FTC scheme the gimbal showed great robustness to sensor faults in the tested scenario.

6.2 Proposed Future Research Work

The results obtained in this master thesis are quite good and promising, but so far have been only implemented and tested in a simulation environment. For this reason most of the future work should go to the implementation in the real hardware device of the control elements designed in this thesis. In fact, there is a critical area related to the hardware implementation that have not been commented in this thesis: the parameter identification of the model elements. The values used for many of the physical properties of the gimbal in this project are accurate, as they where obtained from a CAD model of it. However, friction parameters can only be obtained from parameter identification techniques. It would be also interesting to see how the controllers handle different forms of frictions, as so far only viscous frictions have been considered but in the real gimbal static friction also plays an important role. Maybe the controllers will need to be modified in order to cope with them. Also, the simulation oriented model could be improved in the future by considering the effects of the static frictions.

Other area that could be improved in the future is the High Level Planner. As it is implemented now, it is capable of generating smooth trajectories between Cartesian points but it is still very simple, as it is just a spline generator. It could be upgraded by adding some constraints to the way it moves, for example: forcing it to only move in the plane that contains the initial and the final reference points. In addition to constraints, also objectives could be implemented, as minimizing the linear distance of the trajectories. Also, it would be interesting to study the possibility of generating the trajectories in the Cartesian space directly and not only in the joints coordinate frame as implemented now.

The FTC module proved to be quite successful in the tests carried, however it was seen that observers had trouble estimating the angular velocity. Although a modified scheme worked around this issue, it would be just better to have simpler scheme that works as intended. For this, ways to improve the observability of the system should be found. A possible solution may be to enlarge the model of the system by taking the electrical dynamics into consideration and not only the mechanical aspect as until now. Other approaches may be to test different methods to implement nonlinear observers or also test different LMI during the controller synthesis formulation, as H_2 constraints to build an LPV LQR observer.

6.3 Economic Assessment and Environmental Impact

Due to current situation, most of the project have been developed in the context of social distancing. This caused that most of the project have been done from a design perspective with development validation done exclusively in a simulation environment. For this, to compute the cost of development for the project what should be consider is mainly the development time. Given that the project started on February the length of the project have been 4 and half months. Considering a partial work time of 5 h/day dedicated to the project during working days, the approximate time invested in the project adds up to 450 hours. In Table 6.1, it is shown how the time was divided in the different key aspects of the project and give an economic assessment in 3 different scenarios. Case A is the development cost considering the engineer doing the project is master's student from ETSEIB in an internship. Case B is the development cost considering the engineer doing the project is a master's student from ETSETB (Telecos) in an internship. Case C is the development cost considering the engineer doing the project is working at full time in the company with a gross annual salary of 35,000 €.

Tasks	Development Hours
Modelling	100 h
LPV Controller	125 h
High Level Planer	50 h
FTC strategy	125 h
Technical Report	50 h
Total =	450 h
Cost A: 8€/h	3600 €
Cost B: 10€/h	4500 €
Cost C: 20€/h	9000 €

Table 6.1: Development cost of the project.

For assessing the environmental impact of the project, again, it have to be considered the current situation. As explained, most of the development ave been done from a design perspective, as such the principal tool have been my personal computer. With the time assessment of the entire project and considering that the PC was required for all steps, then an estimation of the CO_2 emission related to the project can be done. For this, an average PC consumption of 50 Watts is assumed and the Spanish 2019 emission factor [1] is used.

$$450\ h \times 0.05\ kW \times 0.241\ \frac{kg\ CO_2}{kWh} = 5.4225\ kg\ CO_2 \tag{6.1}$$

[1] https://canviclimatic.gencat.cat/es/actua/factors_demissio_associats_a_lenergia/

Appendix A

The mathematical model script

A.1 Main script for computing the full 3-Axis Gimbal model

```
%% Forward Kinematics

syms q1
syms q2
syms q3
assume(q1,'real')
assume(q2,'real')
assume(q3,'real')

DH = [q1+pi/2 0 0 pi/2;
      q2-pi/2 0 0 -pi/2;
      q3 0 0 0];

A01 = simplify([cos(DH(1,1)) -sin(DH(1,1))*cos(DH(1,4)) sin(DH(1,1))*
    sin(DH(1,4)) 0;
      sin(DH(1,1)) cos(DH(1,1))*cos(DH(1,4)) -cos(DH(1,1))*sin(DH
          (1,4)) 0;
      0 sin(DH(1,4)) cos(DH(1,4)) 0;
      0 0 0 1]);
```

```
18
19 A12 = simplify([cos(DH(2,1)) -sin(DH(2,1))*cos(DH(2,4)) sin(DH(2,1))*
       sin(DH(2,4)) 0;
20          sin(DH(2,1))  cos(DH(2,1))*cos(DH(2,4)) -cos(DH(2,1))*sin(DH
               (2,4)) 0;
21          0 sin(DH(2,4)) cos(DH(2,4)) 0;
22          0 0 0 1]);
23
24 A23 = simplify([cos(DH(3,1)) -sin(DH(3,1))*cos(DH(3,4)) sin(DH(3,1))*
       sin(DH(3,4)) 0;
25          sin(DH(3,1))  cos(DH(3,1))*cos(DH(3,4)) -cos(DH(3,1))*sin(DH
               (3,4)) 0;
26          0 sin(DH(3,4)) cos(DH(3,4)) 0;
27          0 0 0 1]);
28
29 T03 = A01*A12*A23;
30
31 R01 = A01(1:3,1:3);
32 R12 = A12(1:3,1:3);
33 R23 = A23(1:3,1:3);
34
35 %% Geometric Jacobian (end effector)
36
37 %Transalational part of the jacobian is null
38 JP = zeros(3,3);
39
40 z0 = [0 0 1]';
41 z1 = R01*z0;
42 z2 = R01*R12*z0;
43
44 Jw3 = [z0,z1,z2];
45
```

```
46 JO = Jw3;
47
48 J = [JP;JO];
49
50 %% Inertia Matrix B
51
52 %link mass
53 syms m1 m2 m3
54 assume(m1, 'real')
55 assume(m2, 'real')
56 assume(m3, 'real')
57
58 %center of masses
59 syms com1x com1z
60 assume(com1x, 'real')
61 assume(com1z, 'real')
62 syms com2x com2z
63 assume(com2x, 'real')
64 assume(com2z, 'real')
65 syms com3z
66 assume(com3z, 'real')
67
68 %Inertia Tensor elements
69 syms Ixx1 Iyy1 Izz1
70 syms Ixz1
71 assume(Ixx1, 'real')
72 assume(Iyy1, 'real')
73 assume(Izz1, 'real')
74 assume(Ixz1, 'real')
75 I11 = [Ixx1 0 -Ixz1; 0 Iyy1 0; -Ixz1 0 Izz1];
76
77 syms Ixx2 Iyy2 Izz2
```

```
78 syms Ixz2
79 assume(Ixx2, 'real')
80 assume(Iyy2, 'real')
81 assume(Izz2, 'real')
82 assume(Ixz2, 'real')
83 I12 = [Ixx2 0 -Ixz2; 0 Iyy2 0; -Ixz2 0 Izz2];
84
85 syms Ixx3 Iyy3 Izz3
86 assume(Ixx3, 'real')
87 assume(Iyy3, 'real')
88 assume(Izz3, 'real')
89 I13 = [Ixx3 0 0; 0 Iyy3 0; 0 0 Izz3];
90
91 %position of center of mass in base frame
92 p11 = R01*[com1x 0 com1z]';
93 p12 = R01*R12*[com2x 0 com2z]';
94 p13 = R01*R12*R23*[0 0 com3z]';
95
96 %Transaltional Jacabians
97 Jp11 = simplify([cross(z0,p11) zeros(3,1) zeros(3,1)]);
98 Jp12 = simplify([cross(z0,p12) cross(z1,p12) zeros(3,1)]);
99 Jp13 = simplify([cross(z0,p13) cross(z1,p13) cross(z2,p13)]);
100
101 %Rotational Jacobian
102 Jo11 = [z0 zeros(3,1) zeros(3,1)];
103 Jo12 = [z0 z1 zeros(3,1)];
104 Jo13 = [z0 z1 z2];
105
106 %Inertia matrix
107 R02 = R01*R12;
108 R03 = R02*R23;
109
```

```
110 B1 = simplify(m1*(Jpl1'*Jpl1) + Jol1'*R01*Il1*R01'*Jol1);
111 B2 = simplify(m2*(Jpl2'*Jpl2) + Jol2'*R02*Il2*R02'*Jol2);
112 B3 = simplify(m3*(Jpl3'*Jpl3) + Jol3'*R03*Il3*R03'*Jol3);
113
114 B = simplify(B1 + B2 + B3,100)
115 invB = inv(B);
116
117 %% Coriolis Matrix C
118
119 syms qdot1 qdot2 qdot3
120 assume(qdot1, 'real')
121 assume(qdot2, 'real')
122 assume(qdot3, 'real')
123
124 b11 = B(1,1);
125 b12 = B(1,2);
126 b13 = B(1,3);
127 b21 = B(2,1);
128 b22 = B(2,2);
129 b23 = B(2,3);
130 b31 = B(3,1);
131 b32 = B(3,2);
132 b33 = B(3,3);
133 %Function cijk(bij, bik, bjk, qi, qj, qk) computes:
134 %cijk = simplify(0.5*(diff(bij,qk) + diff(bik,qj)-diff(bjk,qi)))
135 %cijk = cikj
136 c111 = cijk(b11, b11, b11, q1, q1, q1);
137 c112 = cijk(b11, b12, b12, q1, q1, q2);
138 c113 = cijk(b11, b13, b13, q1, q1, q3);
139 c121 = c112;
140 c122 = cijk(b12, b12, b22, q1, q2, q2);
141 c123 = cijk(b12, b13, b23, q1, q2, q3);
```

```
142 c131 = c113;
143 c132 = c123;
144 c133 = cijk(b13, b13, b33, q1, q3, q3);
145 c211 = cijk(b21, b21, b11, q2, q1, q1);
146 c212 = cijk(b21, b22, b12, q2, q1, q2);
147 c213 = cijk(b21, b23, b13, q2, q1, q3);
148 c221 = c212;
149 c222 = cijk(b22, b22, b22, q2, q2, q2);
150 c223 = cijk(b22, b23, b23, q2, q2, q3);
151 c231 = c213;
152 c232 = c223;
153 c233 = cijk(b23, b23, b33, q2, q3, q3);
154 c311 = cijk(b31, b31, b11, q3, q1, q1);
155 c312 = cijk(b31, b32, b12, q3, q1, q2);
156 c313 = cijk(b31, b33, b13, q3, q1, q3);
157 c321 = c312;
158 c322 = cijk(b32, b32, b22, q3, q2, q2);
159 c323 = cijk(b32, b33, b23, q3, q2, q3);
160 c331 = c313;
161 c332 = c323;
162 c333 = cijk(b33, b33, b33, q3, q3, q3);
163
164 c11 = c111*qdot1+c112*qdot2+c113*qdot3;
165 c12 = c121*qdot1+c122*qdot2+c123*qdot3;
166 c13 = c131*qdot1+c132*qdot2+c133*qdot3;
167 c21 = c211*qdot1+c212*qdot2+c213*qdot3;
168 c22 = c221*qdot1+c222*qdot2+c223*qdot3;
169 c23 = c231*qdot1+c232*qdot2+c233*qdot3;
170 c31 = c311*qdot1+c312*qdot2+c313*qdot3;
171 c32 = c321*qdot1+c322*qdot2+c323*qdot3;
172 c33 = c331*qdot1+c332*qdot2+c333*qdot3;
173
```

```
174 C = simplify([c11 c12 c13; c21 c22 c23; c31 c32 c33],100);
175
176 %% Gravity term G
177
178 g0 = [0 0 9.81]';
179
180 g1 = simplify(-m1*g0'*Jp11(:,1)-m2*g0'*Jp12(:,1)-m3*g0'*Jp13(:,1));
181 g2 = simplify(-m1*g0'*Jp11(:,2)-m2*g0'*Jp12(:,2)-m3*g0'*Jp13(:,2));
182 g3 = simplify(-m1*g0'*Jp11(:,3)-m2*g0'*Jp12(:,3)-m3*g0'*Jp13(:,3));
183
184 G = [g1 g2 g3]';
185
186 %% whole model
187
188 syms qdotdot1 qdotdot2 qdotdot3
189 assume(qdotdot1, 'real')
190 assume(qdotdot2, 'real')
191 assume(qdotdot3, 'real')
192 qdotdot = [qdotdot1,qdotdot2,qdotdot3]';
193 qdot = [qdot1,qdot2,qdot3]';
194
195 syms Fv1 Fv2 Fv3
196 assume(Fv1, 'real')
197 assume(Fv2, 'real')
198 assume(Fv3, 'real')
199 Fv = diag([Fv1 Fv2 Fv3]);
200
201 syms tau1 tau2 tau3
202 assume(tau1, 'real')
203 assume(tau2, 'real')
204 assume(tau3, 'real')
205 tau = [tau1,tau2,tau3]';
```

```
206
207 TAU = B*qdotdot+C*qdot+Fv*qdot+G
208 QDD = invB*(tau-C*qdot-Fv*qdot-G)
209
210 %% Solid Work values
211
212 % Link 1
213 vm1 = 2.492;%kg
214 vcom1x = -90.56/1000; %m
215 vcom1z = 114.74/1000;
216
217 vIxx1 = 14354888.77*1e-9; %1e9 = 1/1000^3 conversion de g mm^2 a
      kg m^2
218 vIyy1 = 29370510.38*1e-9;
219 vIzz1 = 17526249.63*1e-9;
220 vIxz1 = 11748431.07*1e-9;
221
222 %Link 2
223 vm2 = 1.624;%kg
224 vcom2x = -6.31/1000; %m
225 vcom2z = -73.38/1000;
226
227 vIxx2 = 8955973.99*1e-9;
228 vIyy2 = 8508639.70*1e-9;
229 vIzz2 = 1966741.10*1e-9;
230 vIxz2 = -687364.95*1e-9;
231
232 %Link 3
233 vm3 = 0.756;%kg
234 vcom3z = 31/1000;
235
236 vIxx3 = 1260000.00*1e-9;
```

```
237 vIyy3 = 655200.00*1e-9;
238 vIzz3 = 1864800.00*1e-9;
239
240 vB = vpa(subs(B,{Ixx1,Iyy1,Izz1,Ixz1,Ixx2,Iyy2,Izz2,Ixz2,Ixx3,Iyy3,
        Izz3,...
241      m1,m2,m3,com1x,com1z,com2x,com2z,com3z},...
242      {vIxx1,vIyy1,vIzz1,vIxz1,vIxx2,vIyy2,vIzz2,vIxz2,vIxx3,vIyy3,
             vIzz3,...
243      vm1,vm2,vm3,vcom1x,vcom1z,vcom2x,vcom2z,vcom3z}));
244
245 vinvB = vpa(subs(invB,{Ixx1,Iyy1,Izz1,Ixz1,Ixx2,Iyy2,Izz2,Ixz2,Ixx3,
        Iyy3,Izz3,...
246      m1,m2,m3,com1x,com1z,com2x,com2z,com3z},...
247      {vIxx1,vIyy1,vIzz1,vIxz1,vIxx2,vIyy2,vIzz2,vIxz2,vIxx3,vIyy3,
             vIzz3,...
248      vm1,vm2,vm3,vcom1x,vcom1z,vcom2x,vcom2z,vcom3z}));
249
250 vC = vpa(subs(C,{Ixx1,Iyy1,Izz1,Ixz1,Ixx2,Iyy2,Izz2,Ixz2,Ixx3,Iyy3,
        Izz3,...
251      m1,m2,m3,com1x,com1z,com2x,com2z,com3z},...
252      {vIxx1,vIyy1,vIzz1,vIxz1,vIxx2,vIyy2,vIzz2,vIxz2,vIxx3,vIyy3,
             vIzz3,...
253      vm1,vm2,vm3,vcom1x,vcom1z,vcom2x,vcom2z,vcom3z}));
254
255 vG = vpa(subs(G,{m2,m3,com2x,com2z,com3z},{vm2,vm3,vcom2x,vcom2z,
        vcom3z}));
256
257 Fv = diag([1,1,1]);
258
259 TAU = vB*qdotdot+vC*qdot+Fv*qdot+vG
260 QDD = vinvB*(tau-vC*qdot-Fv*qdot-vG)
261
```

```
262 function [coef_cijk] = cijk(bij, bik, bjk, qi, qj, qk)
263     coef_cijk = simplify(0.5*(diff(bij,qk) + diff(bik,qj)-diff(bjk,qi
        )));
264 end
```

Appendix B

The controller design script

B.1 Script for Computing the Gimbal Link Controllers

```
%% 3er link

st = 1.5;
gamma = st/4;
alpha = 1/gamma;

M3 = [0 1 0;0 -536.2505 0;1 0 0];
B3 = [0 536.2505 0]';

P = sdpvar(3,3);
W3 = sdpvar(1,3);
options = sdpsettings('solver','sedumi');

F = [P>=0];
F = [F,M3*P+P*M3'+B3*W3+W3'*B3'+2*alpha*P<=0];
optimize(F,[],options)

Psol = value(P);
W3sol = value(W3);
```

```
K3 = W3sol/Psol


%% 1er Link

st = 3;
gamma = st/4;
alpha = 1/gamma;

M11 = [0 1 0;0 -10.21 0;1 0 0];
M12 = [0 1 0;0 -11.28 0;1 0 0];
B1 = [0 10.75 0]';

P = sdpvar(3,3);
W1 = sdpvar(1,3);
options = sdpsettings('solver','sedumi');

F = [P>=0];
F = [F,M11*P+P*M11'+B1*W1+W1'*B1'+2*alpha*P<=0];
F = [F,M12*P+P*M12'+B1*W1+W1'*B1'+2*alpha*P<=0];
optimize(F,[],options)

Psol = value(P);
W1sol = value(W1);
K1 = W1sol/Psol


%% 2do Link

st = 2;
gamma = st/4;
alpha = 1/gamma;
```

```
52
53 theta = pi*5/180;
54 M = [sin(theta) cos(theta);-cos(theta) sin(theta)];
55
56 M210 = [0 1 0;-17.3492 -52.5 0;1 0 0]; %min
57 M220 = [0 1 0;485.3176 -52.5 0;1 0 0]; %max
58 M211 = [0 1 0;-495.8560 -52.5 0;1 0 0]; %min
59 M221 = [0 1 0;16.0752 -52.5 0;1 0 0]; %max
60 B2 = [0 52.5 0]';
61
62 P = sdpvar(3,3);
63 W210 = sdpvar(1,3);
64 W220 = sdpvar(1,3);
65 W211 = sdpvar(1,3);
66 W221 = sdpvar(1,3);
67 options = sdpsettings('solver','sedumi');
68
69 H1 = kron(M,M210*P)+kron(M',P*M210')+kron(M,B2*W210)+kron(M',W210'*B2
       ');
70 H2 = kron(M,M220*P)+kron(M',P*M220')+kron(M,B2*W220)+kron(M',W220'*B2
       ');
71 H3 = kron(M,M211*P)+kron(M',P*M211')+kron(M,B2*W211)+kron(M',W211'*B2
       ');
72 H4 = kron(M,M221*P)+kron(M',P*M221')+kron(M,B2*W221)+kron(M',W221'*B2
       ');
73
74 F = [P>=0];
75 F = [F, M210*P+P*M210'+B2*W210+W210'*B2'+2*alpha*P<=0];
76 F = [F, M220*P+P*M220'+B2*W220+W220'*B2'+2*alpha*P<=0];
77 F = [F, M211*P+P*M211'+B2*W211+W211'*B2'+2*alpha*P<=0];
78 F = [F, M221*P+P*M221'+B2*W221+W221'*B2'+2*alpha*P<=0];
79 F = [F,H1<=0,H2<=0,H3<=0,H4<=0];
```

```
optimize(F,[],options)

Psol = value(P);
W210sol = value(W210);
W220sol = value(W220);
W211sol = value(W211);
W221sol = value(W221);

Z.K210 = W210sol/Psol
Z.K220 = W220sol/Psol
Z.K211 = W211sol/Psol
Z.K221 = W221sol/Psol

Z.theta0min = -17.3492;
Z.theta0max = 485.3176;
Z.theta1min = -495.8560;
Z.theta1max = 16.0752;
```

B.2 Scheduling Function for the Controller of Link 2

```
function K = L2Sched(q2,Z)

    if sign(q2)==1
        if q2<=0.1
            q2 = 0.1;
        end
        theta = (-49.3049*cos(q2)-5.2777*sin(q2))/q2;
        h1 = (theta-Z.theta1min)/(Z.theta1max-Z.theta1min);
        h2 = 1-h1;
        K = Z.K211*h2 + Z.K221*h1;

    elseif sign(q2)==-1
        if abs(q2)<=0.1
```

```
14              q2 = -0.1;
15          end
16          theta = (-49.3049*cos(q2)-5.2777*sin(q2))/q2;
17          h1 = (theta-Z.theta0min)/(Z.theta0max-Z.theta0min);
18          h2 = 1-h1;
19          K = Z.K210*h2 + Z.K220*h1;
20      else
21          K = Z.K211;
22      end
23
24  end
```

Appendix C

The observers design script

C.1 Script for Computing the Observer Gains

```
%% 2do Link full

st = 5;
gamma = st/4;
alpha = 1/gamma;

M210 = [0 1;-17.3492 -52.5]'; %min
M220 = [0 1;485.3176 -52.5]'; %max
M211 = [0 1;-495.8560 -52.5]'; %min
M221 = [0 1;16.0752 -52.5]'; %max
B2 = eye(2);


theta = pi*5/180;
M = [sin(theta) cos(theta);-cos(theta) sin(theta)];

P = sdpvar(2,2);
W210 = sdpvar(2,2,'full');
W220 = sdpvar(2,2,'full');
```

```
20 W211 = sdpvar(2,2,'full');
21 W221 = sdpvar(2,2,'full');
22 options = sdpsettings('solver','sedumi');
23
24 H1 = kron(M,M210*P)+kron(M',P*M210')+kron(M,B2*W210)+kron(M',W210'*B2
       ');
25 H2 = kron(M,M220*P)+kron(M',P*M220')+kron(M,B2*W220)+kron(M',W220'*B2
       ');
26 H3 = kron(M,M211*P)+kron(M',P*M211')+kron(M,B2*W211)+kron(M',W211'*B2
       ');
27 H4 = kron(M,M221*P)+kron(M',P*M221')+kron(M,B2*W221)+kron(M',W221'*B2
       ');
28
29 F = [P>=0];
30 F = [F, M210*P+P*M210'+B2*W210+W210'*B2'+2*alpha*P<=0];
31 F = [F, M220*P+P*M220'+B2*W220+W220'*B2'+2*alpha*P<=0];
32 F = [F, M211*P+P*M211'+B2*W211+W211'*B2'+2*alpha*P<=0];
33 F = [F, M221*P+P*M221'+B2*W221+W221'*B2'+2*alpha*P<=0];
34 F = [F,H1<=0,H2<=0,H3<=0,H4<=0];
35 optimize(F,[],options)
36
37 Psol = value(P);
38 W210sol = value(W210);
39 W220sol = value(W220);
40 W211sol = value(W211);
41 W221sol = value(W221);
42
43 L210 = W210sol/Psol;
44 L220 = W220sol/Psol;
45 L211 = W211sol/Psol;
46 L221 = W221sol/Psol;
47
```

```
48 Z.L210 = L210';
49 Z.L220 = L220';
50 Z.L211 = L211';
51 Z.L221 = L221';
52
53
54 %% 3er link full
55
56 st = 2.5;
57 gamma = st/4;
58 alpha = 1/gamma;
59 beta = alpha+1;
60
61 M3 = [0 1;0 -536.2505]';
62 B3 = eye(2);
63 C3 = eye(2);
64
65 theta = pi*5/180;
66 M = [sin(theta) cos(theta);-cos(theta) sin(theta)];
67
68 P = sdpvar(2,2);
69 W3 = sdpvar(2,2,'full');
70 options = sdpsettings('solver','sedumi');
71 F = [P>=0];
72 F = [F,M3*P+P*M3'+B3*W3+W3'*B3'+2*alpha*P<=0];
73 F = [F,-M3*P-P*M3'-B3*W3-W3'*B3'-2*beta*P<=0];
74 H1 = kron(M,M3*P)+kron(M',P*M3')+kron(M,B3*W3)+kron(M',W3'*B3');
75 F = [F,H1<=0];
76 optimize(F,[],options)
77
78 Psol = value(P);
79 W3sol = value(W3);
```

```
80 L3 = W3sol/Psol;
81 Z.L3 = L3'
82
83 L3'
84
85
86 %% 1er Link
87 st = 5;
88 gamma = st/4;
89 alpha = 1/gamma;
90
91 theta = pi*5/180;
92 M = [sin(theta) cos(theta);-cos(theta) sin(theta)];
93
94 M11 = [0 1;0 -10.21]';
95 M12 = [0 1;0 -11.28]';
96 B1 = eye(2);
97
98 P = sdpvar(2,2);
99 W1 = sdpvar(2,2,'full');
100 options = sdpsettings('solver','sedumi');
101
102 F = [P>=0];
103 F = [F,M11*P+P*M11'+B1*W1+W1'*B1'+2*alpha*P<=0];
104 F = [F,M12*P+P*M12'+B1*W1+W1'*B1'+2*alpha*P<=0];
105 H1 = kron(M,M11*P)+kron(M',P*M11')+kron(M,B1*W1)+kron(M',W1'*B1');
106 H2 = kron(M,M12*P)+kron(M',P*M12')+kron(M,B1*W1)+kron(M',W1'*B1');
107 F = [F,H1<=0,H2<=0];
108 optimize(F,[],options)
109
110 Psol = value(P);
111 W1sol = value(W1);
```

```
112 L1 = W1sol/Psol;
113 Z.L1 = L1'
114
115 %% 3er link q3
116
117 st = 2.5;
118 gamma = st/4;
119 alpha = 1/gamma;
120 beta = alpha+1;
121
122 M3 = [0 1;0 -536.2505]';
123 B3 = [1 0]';
124 C3 = eye(2);
125
126 theta = pi*5/180;
127 M = [sin(theta) cos(theta);-cos(theta) sin(theta)];
128
129 P = sdpvar(2,2);
130 W3 = sdpvar(1,2,'full');
131 options = sdpsettings('solver','sedumi');
132 F = [P>=0];
133 F = [F,M3*P+P*M3'+B3*W3+W3'*B3'+2*alpha*P<=0];
134 % F = [F,-M3*P-P*M3'-B3*W3-W3'*B3'-2*beta*P<=0];
135 H1 = kron(M,M3*P)+kron(M',P*M3')+kron(M,B3*W3)+kron(M',W3'*B3');
136 F = [F,H1<=0];
137 optimize(F,[],options)
138
139 Psol = value(P);
140 W3sol = value(W3);
141 L3 = W3sol/Psol;
142 Z.L3q = L3'
143
```

```
144 L3'
145
146 %% 1er Link q1
147 %st = 0.2;
148 gamma = st/4;
149 alpha = 1/gamma;
150
151 theta = pi*5/180;
152 M = [sin(theta) cos(theta);-cos(theta) sin(theta)];
153
154 M11 = [0 1;0 -10.21]';
155 M12 = [0 1;0 -11.28]';
156 B1 = [1 0]';
157
158 P = sdpvar(2,2);
159 W1 = sdpvar(1,2,'full');
160 options = sdpsettings('solver','sedumi');
161
162 F = [P>=0];
163 F = [F,M11*P+P*M11'+B1*W1+W1'*B1'+2*alpha*P<=0];
164 F = [F,M12*P+P*M12'+B1*W1+W1'*B1'+2*alpha*P<=0];
165 H1 = kron(M,M11*P)+kron(M',P*M11')+kron(M,B1*W1)+kron(M',W1'*B1');
166 H2 = kron(M,M12*P)+kron(M',P*M12')+kron(M,B1*W1)+kron(M',W1'*B1');
167 F = [F,H1<=0,H2<=0];
168 optimize(F,[],options)
169
170 Psol = value(P);
171 W1sol = value(W1);
172 L1 = W1sol/Psol;
173 Z.L1q = L1'
174
175 %% 2do Link q2
```

```
176
177 st = 2.5;
178 gamma = st/4;
179 alpha = 1/gamma;
180
181 M210 = [0 1;-17.3492 -52.5]'; %min
182 M220 = [0 1;485.3176 -52.5]'; %max
183 M211 = [0 1;-495.8560 -52.5]'; %min
184 M221 = [0 1;16.0752 -52.5]'; %max
185 B2 = [1 0]';
186
187
188 theta = pi*45/180;
189 M = [sin(theta) cos(theta);-cos(theta) sin(theta)];
190
191 P = sdpvar(2,2);
192 W210 = sdpvar(1,2,'full');
193 W220 = sdpvar(1,2,'full');
194 W211 = sdpvar(1,2,'full');
195 W221 = sdpvar(1,2,'full');
196 options = sdpsettings('solver','sedumi');
197
198 H1 = kron(M,M210*P)+kron(M',P*M210')+kron(M,B2*W210)+kron(M',W210'*B2
      ');
199 H2 = kron(M,M220*P)+kron(M',P*M220')+kron(M,B2*W220)+kron(M',W220'*B2
      ');
200 H3 = kron(M,M211*P)+kron(M',P*M211')+kron(M,B2*W211)+kron(M',W211'*B2
      ');
201 H4 = kron(M,M221*P)+kron(M',P*M221')+kron(M,B2*W221)+kron(M',W221'*B2
      ');
202
203 F = [P>=0];
```

```
204 F = [F, M210*P+P*M210'+B2*W210+W210'*B2'+2*alpha*P<=0];
205 F = [F, M220*P+P*M220'+B2*W220+W220'*B2'+2*alpha*P<=0];
206 F = [F, M211*P+P*M211'+B2*W211+W211'*B2'+2*alpha*P<=0];
207 F = [F, M221*P+P*M221'+B2*W221+W221'*B2'+2*alpha*P<=0];
208 F = [F,H1<=0,H2<=0,H3<=0,H4<=0];
209 optimize(F,[],options)
210
211 Psol = value(P);
212 W210sol = value(W210);
213 W220sol = value(W220);
214 W211sol = value(W211);
215 W221sol = value(W221);
216
217 L210 = W210sol/Psol;
218 L220 = W220sol/Psol;
219 L211 = W211sol/Psol;
220 L221 = W221sol/Psol;
221
222 Z.L210q = L210'
223 Z.L220q = L220'
224 Z.L211q = L211'
225 Z.L221q = L221'
```

Bibliography

[1] Pierre Apkarian, Pascal Gahinet, and Greg Becker. “Self-scheduled Hinf control of linear parameter-varying systems: a design example”. In: (1995).

[2] F. Bianchi and R. Mantz. “Robust LPV Gain Scheduled PID Control”. In: (2004).

[3] Guang-Ren Duan and Hai-Hua Yu. *LMIs in Control Systems: Analysis, Design and Applications*. CRC Press, 2013.

[4] J. Löfberg. “YALMIP: A toolbox for modeling and optimization in MATLAB”. In: (2004).

[5] I. Masubuchi, T. Akiyama, and M. Saeki. “Synthesis of output feedback gain-scheduling controllers based on descriptor LPV system representation”. In: *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*. Vol. 6. 2003, 6115–6120 Vol.6.

[6] Mathworks. *MATLAB Documentation: cart2sph @ONLINE*. URL: `https://es.mathworks.com/help/matlab/ref/cart2sph.html`.

[7] Mathworks. *MATLAB Documentation: Recursive Least Squares Estimator @ONLINE*. URL: `https://es.mathworks.com/help/ident/ref/recursiveleastsquaresestimator.html`.

[8] Kamyar Mehran. “Takagi-Sugeno Fuzzy Modeling for Process Control”. In: (2008).

[9] Jan Rosell. *Robotics: Kinematics, Dynamics and Control @ONLINE*. URL: `https://sir.upc.edu/projects/kinematics_dynamics_control/`.

[10] Damiano Rotondo, Fatiha Nejjari, and Vicenç Puig. “A virtual actuator and sensor approach for fault tolerant control of LPV systems”. In: *Journal of Process Control* (2014).

[11] B. Siciliano et al. *Robotics: Modelling, Planning and Control*. Springer, 2009.

[12] J. F. Sturm. “Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones”. In: (1999).