POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING

MSC DEGREE IN MECHATRONIC ENGINEERING

MSC DEGREE THESIS

Development of the SmartGimbal Control System for the SmartBay Platform

Supervisor:
Prof. Paolo MAGGIORE

Candidate:
Annachiara GRECO

DECEMBER 2019

Abstract

The final work focuses on the development of the SmartGimbal control system that is a camera holder. The thesis is the outcome of a collaboration with Digisky Srl, offering high technology and low cost solutions in the avionic system field. One of the Digisky's proper product is the SmartBay platform, a wing pylon enabled to host any type of sensors. The SmartGimbal is one of the sensors conceived by Digisky, compatible with the SmartBay platform, holding a camera to achieve the purpose of aerial monitoring, surveillance, fire detection and medical emergencies.
Proceeding according to the Model-Based approach, four main parts of the thesis can be identified. First, the mechanics and the dynamics of the gimbal system are analyzed to derive a mathematical model of the gimbal system and its dynamic equations, also by applying the Denavit-Hartenberg convention from robotic.
In the second part, the control system of the gimbal, that is able to rotate around the azimuth and elevation axes, is developed. From the simulations performed through *Matlab* and *Simulink* environments, both the maximum torque and angular speed values that the DC motors should provide to the system are evaluated. Eventually, the DC motors, already mounted on the system, are directly controlled by means of PI controllers.
Finally the control algorithms are converted into C code to be executed on the microcontroller, placed on the Arduino Mega 2560 board. In this manner, the two control algorithms can run on the microcontroller and the control system can be tested on the real existing system. Before the validation phase on the real gimbal system, as suggested by the model-based design approach, a Software-in-the-Loop and a Processor-in-the-Loop simulations are performed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays the aerial shots provided by cameras are very useful and employed instruments to achieve different purposes like surveillance, earth monitoring, disasters prediction. The Digisky Srl offers this kind of services thanks to the development and enhancement of the SmartBay platform and its own sensors, installed outside the aircraft.

1.1 Digisky and the SmartBay project

This thesis grows in collaboration with the Digisky company whose aim is to offer high technology solutions in avionic systems and advanced aerial monitoring projects. Starting from the ICT and automotive environments, the innovations are moved into the general aviation and UAV industry offering the possibility of low-cost integrated systems easy to install and applicable in several different fields. From design to prototyping phase, the Digisky's products can be employed in precision farming, fire detection, remote medical emergencies systems and plant surveillance.
The SmartBay platform is a Digisky's proper integrated system of sensors and softwares, able to host any type of peripherals and to establish a smart connection between the mission aircraft and external devices.

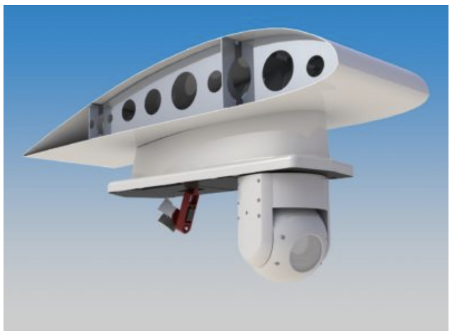


Figure 1.1: SmartBay wing pylon

SmartBay, shown in figure 1.1, is a wing pylon, i.e. a rigid structural element generally used to keep up elements outside the fuselage or the wings. Some of the SmartBay technical features are listed below:

- wing pod of low aerodynamic impact
- 3 slots carrying up to 40 kg payloads
- embedded IMU and RTK-GPS
- universal payload 'plug&play'

SmartBay compatible payload sensors include cameras with gyro stabilization, chemical sensors for monitoring the air quality, audio sensors to check noise areas and a wide range of sensors produced by other manufactures.

1.2 The SmartGimbal

The SmartGimbal is one of the Digisky's proprietary sensors, and it supports a camera located at the center of the gimbal mechanism. This complex electromechanical system, which is a two-axis tracking system, is also called Inertially Stabilized Platform (ISP) and has been widely studied by the researchers in the recent years. In fig.1.2 the *SmartGimbal* sensor is shown and its own technical characteristics are listed below:

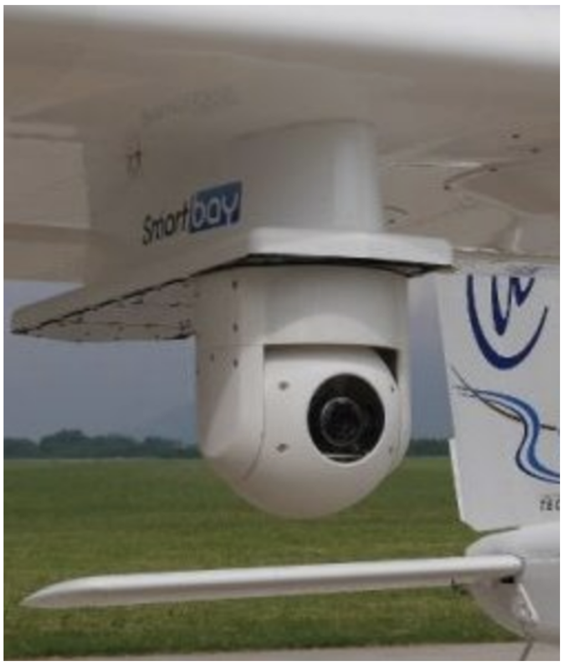


Figure 1.2: *SmartGimbal* sensor

- fast IMU embedded
- tilt range $0° - 180°$
- yaw range $0° - 360°$
- full remotely controlled from ground or aircraft
- yaw/tilt compensation speed up to 55°/s
- camera holder weighing up to 3 kg

From a strictly mechanical point of view, a gimbal is a cardan joint employed to stabilize any object. In absence of the 2-axis gimbal, the video camera would be hooked to the aircraft and then integral to its motion and oscillations; the results would be highly inaccurate images and videos.
As shown in the next figure, the mechanical system is made of two joints: the first one is external and allows the *pan* rotation of the gimbal; the second one is the inner joint which permits the *tilt* rotation. While a continuous rotation of the pan joint is guaranteed, as mentioned in the gimbal technical specifications, the tilt angle is constrained between 0° and 180°.

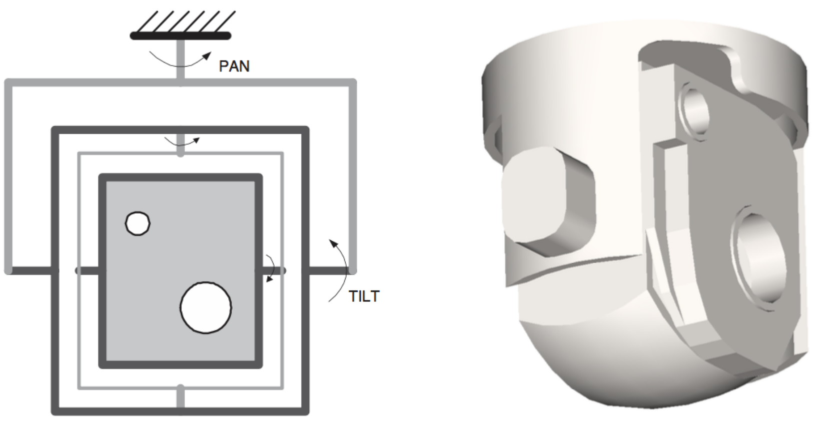


Figure 1.3: Gimbal mechanical structure

Each one of the motion axis has its own DC motor, guaranteeing the motion of the camera in the desired direction. A control architecture must be defined to move the gimbal accordingly to a reference input command and to compensate the aircraft oscillations and disturbances (for example due to the wind) on the azimuth/elevation axis. In the control architecture definition the other sensors mounted on the 2-axis or near the video camera are also considered. There are two digital encoders, to exactly know the pan and

tilt angles in local reference frame and the IMU (Inertial Measurement Unit) sensors to know the actual angular position of the camera in the absolute reference frame.
The other important problem related to the gimbal motion control is the image stabilization done by a specific programming technique, but this matter it's out of the scope of the thesis.

1.3 Outline of the thesis

The thesis is structured as follows:

- **Chapter 2:** the model-based design approach is explained since it will be applied in each step of the work;
- **Chapter 3:** the kinematics of the Gimbal system will be studied, i.e. its mechanical structure. The roto-translation matrix of the gimbal will be obtained by applying the DH conventions coming from robotics. In the present chapter, the jacobians of the gimbal will be found out since they will be used in chapter 4;
- **Chapter 4:** by means of the Lagrange equations and the jacobians, the dynamic model of the gimbal system is derived. The obtained dynamic model will be used in chapter 6 as plant of the controller;
- **Chapter 5:** the dynamic model of the gimbal is expressed in terms of state-space representation matrices to be linearized around some linearization points. The same is done for the two independent DC-motor driving the pan and tilt axes;
- **Chapter 6:** in the present chapter the control system is developed following two different control techniques, the Loop-Shaping and the LQR. The gimbal is chosen as plant of the controller which is designed starting from the definition of a few specifications given in both time and frequency domains. The aim is to identify the maximum values of torque and angular velocities the DC-motors should provide to the plant. Then, the possibility of a new motor-reducer-encoder group different from the assembled one is evaluated (in chapter 9). A comparison between the results taken from the two control strategies is made;
- **Chapter 7:** the two available DC-motors are controlled by means of the PID control techniques, from which two distinct controllers are designed. The plants are now the DC-motors and a linear control system is first considered and simulated. Eventually the system is converted into a non-linear one by introducing Simulink's blocks modeling the typical non linearities (actuator saturations, backlash, ...). In the second scenario, it is only verified that, despite all the introduced non linearities, the control system still gives acceptable simulation results;
- **Chapter 8:** the control algorithm developed in chapter 7 is here converted into C code using the Simulink's *code generation* tool. Indeed the final goal is to integrate the provided code in Arduino, in order to execute it on the Arduino Mega 2560

microcontroller platform. Before exporting the control algorithm on actual board, a SIL (Software in the Loop) and a PIL (Processor in the Loop) simulations are performed, following the V-model of the MBD approach. Once these steps are completed, the Arduino board is configured to be properly linked to the external peripherals;

- **Chapter 9:** collecting the data coming from the simulation of chapter 6, a DC motor survey is conducted, aimed at finding two DC-motors which satisfy the maximum torque and angular speed values required by the gimbal system. Furthermore, a survey is conducted in order to search for compatible reducers and control shields, both within the Faulhaber's and Maxson's catalogues.

- **Chapter 10:** suggestions for future works are presented.

Chapter 2

Model-based design

2.1 Model-based design overwiev

Today there are smart systems whose functionality and capabilities increase making these systems increasingly complex. The model-based design technique is a typical approach when dealing with embedded software and control problems, allowing simplifications in complex design problem, avoiding project or automation problems, decreasing the effort and removing the hand-coding errors, ensuring shorter times in the development process. The reference V-model of the Model-based design approach is shown in the next figure.

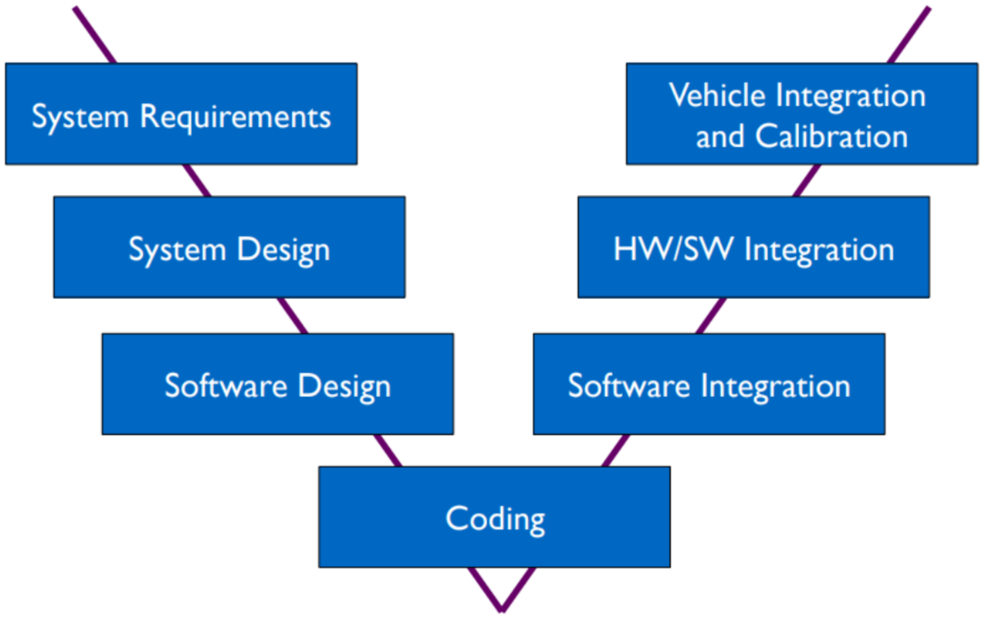


Figure 2.1: Model-Based design V-model

This method is appreciated and employed in the engineering fields since it permits faster achievements of systems design and facilitates data analysis and system validation. *MathWorks* company created software tools in order to improve the MBD, providing modeling and simulations environments to test and improve (or redefine) the model. Since the tests and validations are done endlessly, and not at the end of the project, the main errors are found and corrected before the hardware testing. Among the many advantages of the MBD method there is the possibility to create reusable projects with the purpose of saving time, reduced costs, flexibility and better quality of the product, guaranteeing the optimization of the process. Moreover the software tools, like as *Simulink*, are provided with code generation to automatically generate the code from the system model, for the implementation on the chosen micro controller.

2.2 The V-model

The V-model is a development model proper of the software field. It is so called because of the V-shape, which proves the connection between each phase of the software development life cycle and its own testing phase. It is a well organized model, in which each testing phase can be implemented from the documentation of the previous one. Four main steps characterizing the V-model can be identified: the model in the loop (MIL), the software in the loop (SIL), the processor in the loop (PIL) and finally the hardware in the loop (HIL) step.

- MIL (Model in the loop): It consists of plant modeling and control system design, starting from the requirements definition.

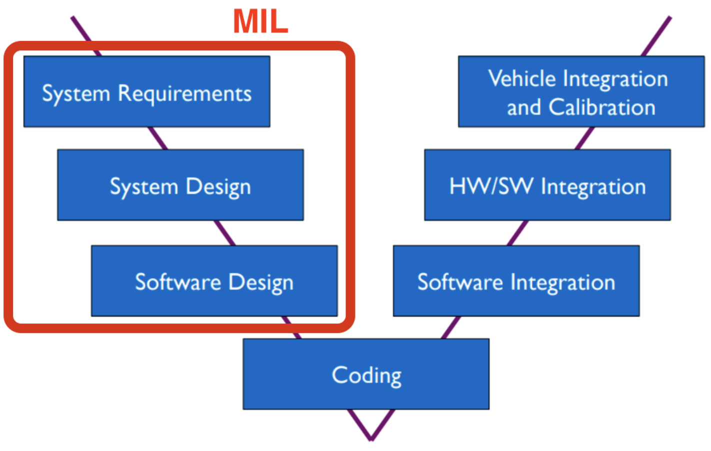


Figure 2.2: Model in the loop step

It is a simulation only phase, in fact both plant and controller run on the PC. In the actual step is allowed to design, simulate and improve the project until the given requirements aren't satisfied. In this way the planner can be sure that the system is working properly. Taken *Matlab* as design environment, its tool is used as simulation one (i.e.*Simulink* in which the model entirely exists).

- SIL (Software in the loop):
  Here is provided the C/C++ code, by means of the code generation. The simulation still entirely runs on the PC: while the controller is translated in an executable C/C++ code, the plant exists on the native simulation language.

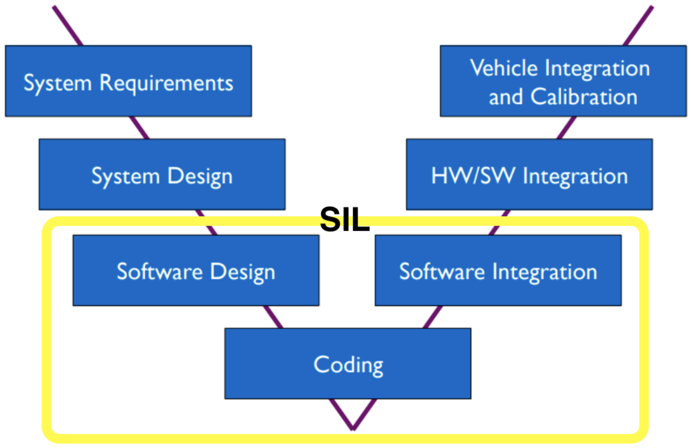


Figure 2.3: Software in the loop step

  Then, while the controller block is replaced with the *S-function* block extracted after the code generation process, the plant exists in terms of Simulink's block. The aim of the SIL step is to verify that the generated code works as well as the original designed model. The validation takes place on PC and it's not real time, the logic signal are simply exchanged between controller and plant and the design optimization is required to ensure that the simulation time do not become too high.

- PIL (Processor in the loop):
  In the actual phase, an hardware/software integration is performed since the controller is validated on the target hardware. The plant runs in simulation environment instead the controller (that is an executable) runs on the hardware. In this project the target hardware is the Arduino Mega 2560 board.

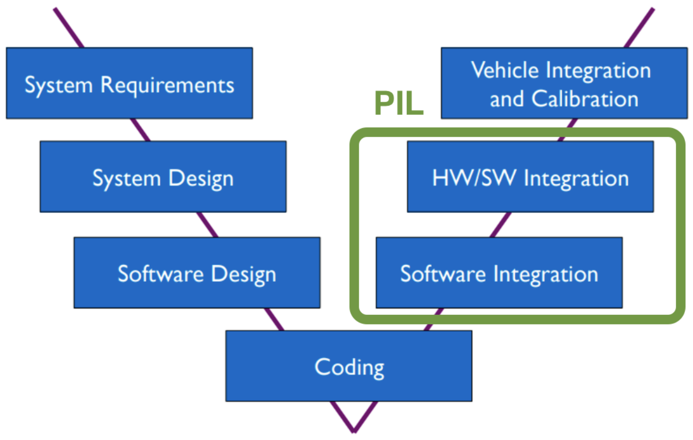


Figure 2.4: Processor in the loop step

- HIL (Hardware in the loop):
  It is the last step in which the simulation takes place in real-time, then in the real physical domain. The controller runs on the HW target, while the plant is emulated on a dedicated hardware. The exchanged signals between controller and plant are physical ones, and it's a good method to test the influences of the hardware on the real-time performances.

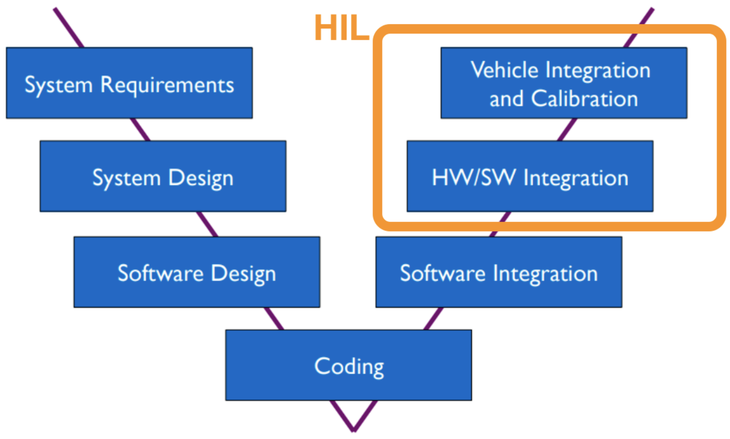


Figure 2.5: Hardware in the loop step

Chapter 3

Kinematics

The kinematics purpose is to study the motion of a body, without taking into account the causes that generate the motion (i.e. forces and moments acting on the body). Derive suitable kinematics model of the bodies is essential to understand the behavior of an entity, especially in the case of the robotic manipulators. It's possible to recognize two different kind of kinematics: *forward and inverse kinematics.* The first one has always a solution and the calculation to achieve the motion equation is more simpler than in the case of the inverse kinematics, of which the solution could take a long time and be very expansive. In fact, there are two typical approaches to the inverse kinematics problem: the analytical or the numerical one, that are described later.

3.1 Forward Kinematics

The two-axis tracking system is a mechanical system, conceived as an ideal rigid body, to be more exact as a robotic *manipulator*. In fact, is possible to imagine the gimbal as a kinematic chain which consists of intermediate rigid bodies (*links*) connected through *joints*, bounded to a base at one end and featuring a gripper ($end - effector$) at the other end. The gimbal system is made of two revolute joints, each one driven by a DC-motor.

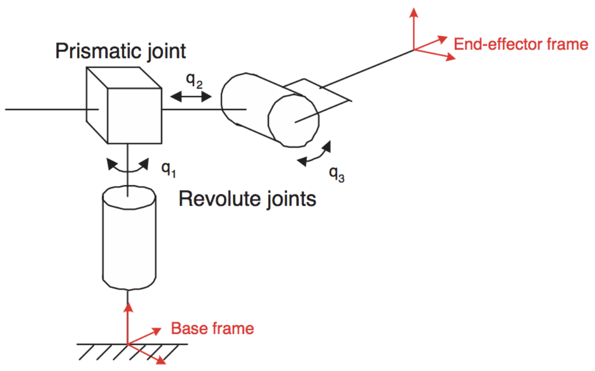


Figure 3.1: A generic open kinematic chain

Like all rigid bodies, the gimbal moves in the space by means of the two possible types of motions that are translation and rotation (or roto-translation if the two motions are combined). As known, any body is fully described in space by its own position and orientation, the so called *pose* of a body, with respect to an inertial reference frame. Being the manipulator made of links and joints is indispensable to identify the reference frames of each one of them and find the relationships between their different orientations. To obtain the equation of motion of the gimbal the forward kinematic, the homogeneous transformation matrices and the DH conventions are used.

The following are the guidelines in defining the kinematics of any rigid body. Of a generic manipulator having an open kinematic chain body, can be identified the links enumerated from 0 to n starting from the base. Also the joints, whether they are revolute or prismatic, are numerated from 1 to n, remembering that the i-th joint attaches link $i-1$ to i. Then a reference frame is assigned to each link, considering the reference frame 0 as corresponding to the one of the base. Since each joint adds one degree of freedom to the entire system, the variable associated to this motion is called *joint variable*, or *generalized coordinate* if it is independent. The i-th joint variables are represented by q_i, which is the rotation angle for revolute joint ($q_i = \theta_i$) or the displacement for prismatic one ($q_i = d_i$).
The aim of the kinematic study is to translate the position of a point from the reference frame $i-1$ into the frame i, by applying the homogeneous transformation matrices. Let A_i is the transformation matrix, then it is dependent by only joint coordinate q_i and it has the following structure:

$$A_0^1 = \begin{bmatrix} R_0^1 & \boldsymbol{d}_0^1 \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.1}$$

According to (3.1) the matrix A_0^1 describes the coordinates of a point in frame 1 with respect to frame 0, in fact it consists of the rotation matrix R and the translation vector $\boldsymbol{d}$. So, it's possible to describe the pose of the gripper with respect to the base frame through the roto-translation matrix T_0^n, defined as:

$$T_0^n = \prod_{i=1}^{n} A_i \tag{3.2}$$

3.1.1 Gimbal Forward Kinematics

Since the gimbal is a 2DOF system and the allowed motions are the elevation and azimuth rotations, it is conceived as a manipulator made of two revolute joints driven by two independent DC motors. With these premises the gimbal structure becomes comparable to a spherical arm manipulator, as shown in figure 3.2. However is useful to take into consideration a further variable (d_3) corresponding to the distance of the camera from the tracked object and associated to a prismatic joint (which is not really controllable by any DC motor).

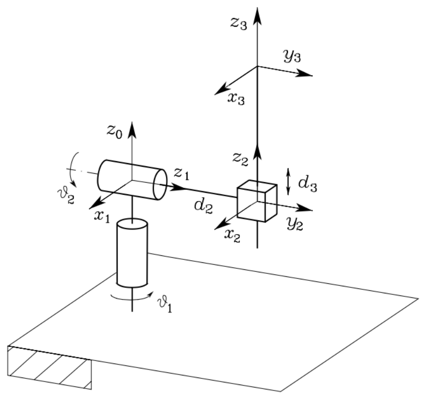


Figure 3.2: Typical spherical arm manipulator

In the figure the reference frames are jet located and the joint variables are also chosen: in the next section all these datas will be derived by means of the Denavit-Hartenberg conventions.

3.1.2 Denavit-Hartenberg convention

The Denavit-Hartenberg (DH) convention, is a widely used technique in robotic systems to choose the reference frames of the manipulator links and to achieve the kinematic equations of an open kinematic chain structure. According to [2], to obtain an overall description of the manipulator kinematics, it's better to focus on the single relationships existing between two consecutive links, identifying the frames of each link (from 0 to n if the manipulator has $n + 1$ links connected through n joints). Finally the total description of the frame n with respect to frame 0 will be obtained by applying the following calculation:

$$T_0^n(\boldsymbol{q}) = A_1^0(\boldsymbol{q}_1)A_2^1(\boldsymbol{q}_2)\dots A_n^{n-1}(\boldsymbol{q}_n) \tag{3.3}$$

Starting from the expression (3.3) is needed to define the frame of each link and evaluate the coordinate transformation between them. The DH convention is for this purpose a systematic method to refer to concerning the reference frames choice.

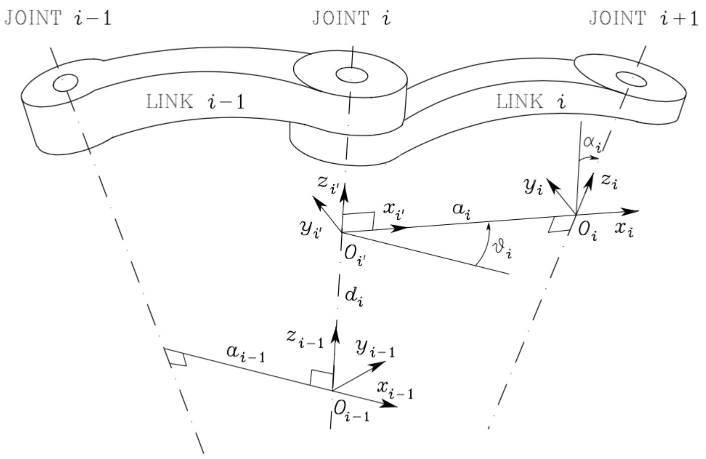


Figure 3.3: Denavit-Hartenberg parameters

DH convention algorithm: Referring to figure 3.3, assume that joint i connects link $i-1$ to link i: the *DH convention algorithm* is applied to fix the frame of link i.

1. The origin of i-th frame, which is O_i, is always on the motion axis of joint $i+1$ and is located at the intersection point of axis z_i with the minimum distance segment between z_i and z_{i-1} axis;
2. The z_i axis is along the motion axis of joint z_{i+1} and its direction is defined accordingly to the right-hand rule, indicating the positive direction of the motion;
3. The axis x_i is located to be orthogonal to both z_i and z_{i-1} and its own direction is arbitrary (in general it points toward the next joint);
4. The j_i axis simply must complete the right-hand rule in defining the actual reference frame i.

Some clarifications:

- If two consecutive motion axis are parallel the origin O_i is set on a desired point by the user, rather on the arm;
- The end-effector frame doesn't have a subsequent RF, then O_n is placed in a point of choice, generally on the end-effector. Moreover the only condition to observe is that x_n axis must be normal to z_{n-1}, while z_n is not conditioned and j_n always must complete the right-hand rule;

- The base reference frame, then the 0 RF, is not preceded by others: in this case the only required condition is related to the z_0 axis, which must follow the motion axis z_1. The origin O_0 and the axis x_0 are devoid of any rule.

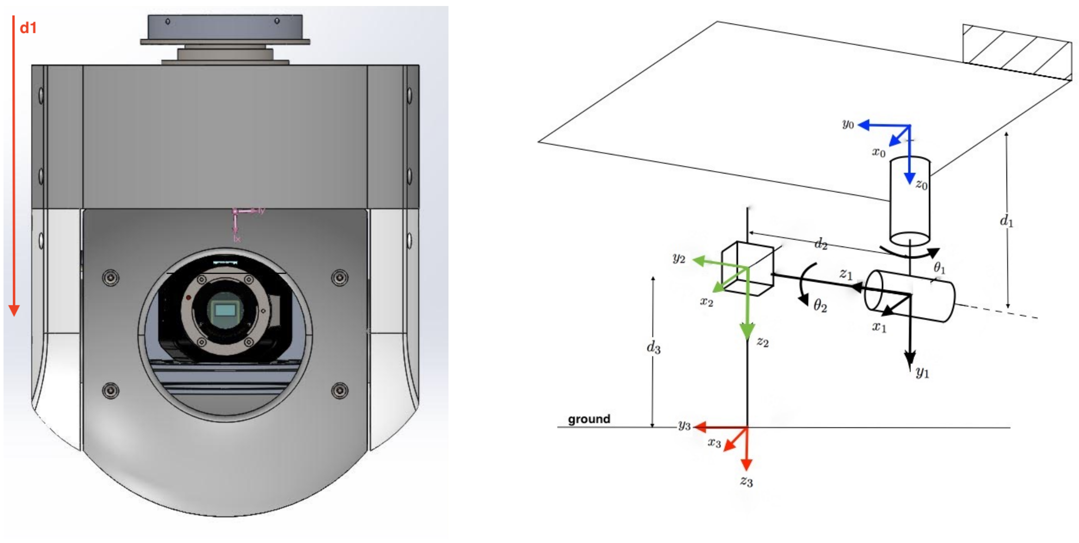
Once defined the reference frames of each manipulator link, the pose of the i^{th} frame with respect to the following and the preceding ones must be established, then the DH parameters are introduced.
As general to move between two frames, 6 parameters are needed (3 rotation variables and as much of translation). The DH convention catch on to express the relative positions between the RFs in a common manner and to reduce the number of the required parameters. In fact only 4 parameters are necessaries, the so-called *DH parameters*. The four DH parameters, also said *minimal* variables, emerge in fig. 3.3 :

- a_i represents the translation measured along axis x_i and it corresponds to the minimal signed distance between z_i and z_{i-1} axis along the common normal;
- d_i is the distance measured along z_{i-1} axis, identifying the translation between the origin O_{i-1} and the intersection of x_i and z_{i-1};
- α_i is the rotation angle around x_i axis, such that k_i and k_{i-1} coincide and the positive direction is counter-clockwise;
- θ_i is the rotation angle around k_{i-1} axis, such that x_{i-1} and x_i coincide.

Three of these parameters are geometric ones (then constants) while one of these depends on the relative motion between two successive links and then is time dependent.

3.1.3 DH rules applied to the Gimbal

The Denavit-Hartenberg rules from robotics are applied to the gimbal system.



(a) Gimbal CAD drawing (b) DH Gimbal parameters

Figure 3.4: Gimbal CAD model and approximated model

With reference to figure 3.4 the DH parameters are chosen and listed below:

$link$	$\theta_i[rad]$	$a_i[m]$	$\alpha_i[rad]$	$d_i[m]$
1	θ_1	0	$\frac{\pi}{2}$	d_1
2	θ_2	0	$-\frac{\pi}{2}$	d_2
3	0	0	0	d_3

Table 3.1: DH parameters of the Gimabl

Some simplifications are adopted to make the calculations simpler and more immediate:

- the gimbal model is approximated to a spherical arm manipulator;
- the joints 1 and 2 are two real revolute joints, each one autonomously driven by a DC motor;

- joint 3 is a prismatic one, which only allows to consider the distance (d_3) between camera and ground: it's not really controllable;
- the distance d_2 is set to 0, like as the two revolution axis intercept in a point;
- the segment d_1 represents the distance between the base and the motion axis of joint 1 and its value is taken from the CAD drawing in fig. 3.4;
- θ_1 and θ_2 correspond to the tilt and pan rotation angles, respectively. From technical datas, the system features continuous rotation around the azimuth axis and a pan rotation bounded between 0°and 180°.

Once fixed the links, end-effector, base and ground *RFs*, simply by applying (3.1) and (3.3) the overall coordinate transformations is evaluated:

$$T_3^0(\boldsymbol{q}) = \begin{bmatrix} R_3^0 & d_3^0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} c_1c_2 & -s_1 & c_1s_2 & c_1s_2d_3 + d_1 \\ s_1c_2 & c_1 & s_1s_2 & s_1s_2d_3 \\ -s_2 & 0 & c_2 & c_2d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Note the end-effector pose which coincides with the three rows of the 4-th column, and the square rotation matrix [3x3] made of the first three columns and rows.
The final matrix is obtained as the result of: [1]

$$T_3^0(\boldsymbol{q}) = T_1^0(\boldsymbol{q}_1)T_2^1(\boldsymbol{q}_2)T_3^2(\boldsymbol{q}_3) =$$
$$= \underbrace{\begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{T_1^0} \underbrace{\begin{bmatrix} c_2 & 0 & s_2 & 0 \\ 0 & 1 & 0 & 0 \\ -s_2 & 0 & c_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{T_2^1} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{T_3^2}$$

3.1.4 Joint and operational spaces

Once computed the T_i^j coordinate transformation matrices and the total one, the kinematic functions can be obtained. First of all the *joint space* must be distinguished from the *operational space*:

1. The joint variables q_i (θ_i or d_i) are defined into the joint space and the joint variables vector is denoted as $\boldsymbol{q}$:
   $\Longrightarrow \boldsymbol{q} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ d_3 \end{bmatrix}$ with q_1=θ_1, q_2=θ_2, q_3=d_3;

[1]From now on short forms of mathematical expressions relating to *sin* and *cos* will be used: s_1 or c_1 mean $sin\theta_1$ and $cos\theta_2$ respectively.

2. The operational space, also called *task space*, is reachable by the end-effector of the manipulator: in this space the task of the gripper is fixed. Here is defined the pose vector $\boldsymbol{p}$ made of 6 parameters:
   $\Longrightarrow \boldsymbol{p} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\alpha} \end{bmatrix}$ where $\boldsymbol{x}$ coincides with the position and $\boldsymbol{\alpha}$ represents the orientation.
3. The last definition is related to the *workspace*, i.e. the points that the origin of the grip can reaches. To identify this space only $\boldsymbol{x}$ is needed. Then:
   $\Longrightarrow \boldsymbol{p} = \boldsymbol{p}\,(\theta_1, \theta_2, d_3)$ with $\boldsymbol{p}$ provided by the 4-th column of T_3^0.

Considering the structural and mechanical limitations of the gimbal structure, that is:

$$\begin{cases} -\infty < \theta_1 < \infty \\ 0 < \theta_2 < \pi \\ d_{min} < d_3 < \infty \end{cases}$$

the two vectors of interest will be

$$\boldsymbol{q} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ d_3 \end{bmatrix} = \textit{joint space} \tag{3.5}$$

$$\boldsymbol{p} = \begin{bmatrix} c_1 s_2 d_3 + d_1 \\ s_1 s_2 d_3 \\ c_2 d_3 \end{bmatrix} = \textit{workspace} \tag{3.6}$$

From the above defined vectors, if the joint variables are known, the direct kinematics functions are achieved: the three cartesian components are extract depending on the q_i parameters as follow.

$$\begin{cases} x(t) = c_1 s_2 d_3 + d_1 \\ y(t) = s_1 s_2 d_3 \\ z(t) = c_2 d_3 \end{cases} \tag{3.7}$$

3.2 Inverse Kinematics

Since the manipulator task is defined in the operational space/workspace while the control actions are designed as functions of the joint variables, a transformation from the pose vector to the joint vector is required. To this purpose the inverse kinematics functions must be obtained translating the motion requirements (designated to the gripper) into actions which allow the desired behavior of the manipulator.
Unfortunately the problem is not easy to solve for a number of reasons, like as:

- In general the equations are non linear, then is not always allowed to find a closed-form solution;

- More than one solution to the problem could exists, depending on the DOF of the manipulator and on the non-null DH parameters;
- If the manipulator structure is redundant infinite solutions could exist.

Sometimes the closed-form solutions of (3.7) cannot be analytically computed, then to solve the inverse kinematics problem several numerical techniques should be applied. Among all of them, the widely used method is the *Jacobian representation*.

3.2.1 Jacobians

The Jacobians are matrices mapping the existing relation between joint and end-effector velocities. They are of two types:

- Linear jacobian;
- Angular jacobian.

Being $\dot{\boldsymbol{p}}(q)$ the velocities vector of the gripper depending on the joint variables, and $\dot{\boldsymbol{q}}$ the velocities vector of the joint:

$$\begin{aligned}\dot{\boldsymbol{p}}(t) &= \boldsymbol{J}_L\dot{\boldsymbol{q}}\\ \dot{\boldsymbol{v}}(t) &= \boldsymbol{J}_A\dot{\boldsymbol{q}}\end{aligned} \tag{3.8}$$

Notice the two different jacobian matrices in (3.8):

- The Linear jacobian $\boldsymbol{J}_L$, also called *task jacobian*, which is directly extracted from the coordinate transformation matrix T_3^0;
- The Angular jacobian $\boldsymbol{J}_A$, which is a matrix made of other two matrix terms: the *Analytical jacobian* and the *Geometric jacobian*.

Linear Jacobian

Suppose of dealing with a manipulator having n DOF. Its own overall coordinate transformation will be expressed by means of T_n^0 matrix:

$$T_n^0(\boldsymbol{q}) = \begin{bmatrix} R_n^0 & p_n^0 \\ 0 & 1 \end{bmatrix}, \boldsymbol{q} = \begin{bmatrix} q_1 \\ \dots \\ q_n \end{bmatrix}$$

To obtain the linear jacobian is required to isolate the d_n^0 vector and evaluate its own time derivative.Then the jacobian matrix will appear as:

$$\boldsymbol{J}_L = \begin{bmatrix} \frac{\partial p_1}{\partial q_1} & \frac{\partial p_1}{\partial q_2} & \cdots & \frac{\partial p_1}{\partial q_n} \\ \frac{\partial p_2}{\partial q_1} & \frac{\partial p_2}{\partial q_2} & \cdots & \frac{\partial p_2}{\partial q_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial p_n}{\partial q_1} & \frac{\partial p_n}{\partial q_2} & \cdots & \frac{\partial p_n}{\partial q_n} \end{bmatrix} \tag{3.9}$$

$\implies \boldsymbol{J}_L \in R^{m \times n}$

Geometric Jacobian

The Geometric Jacobian of a manipulator is simply identified through the $\boldsymbol{J}$ entity. It is a $[6 \times n]$ matrix used to make explicit the relation between the gripper linear velocities $\dot{\boldsymbol{p}}_{\boldsymbol{e}}$ and the angular ones with respect the joint velocities $\dot{\boldsymbol{q}}$. The geometric jacobian is defined as:

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_P \\ \boldsymbol{J}_O \end{bmatrix} \tag{3.10}$$

The $\boldsymbol{J}$ is made of two other matrices: $\boldsymbol{J}_P$ (coinciding with $\boldsymbol{J}_L$) represents the *position jacobian*, while $\boldsymbol{J}_O$ represents the *orientation jacobian*. Concerning the $\boldsymbol{J}_O$ an observation is necessary: can be defined two different types of velocities vectors, that are

- analytical velocities

$$\boldsymbol{v} = \begin{bmatrix} \dot{\boldsymbol{p}}_e \\ \dot{\boldsymbol{\alpha}} \end{bmatrix} \dot{\boldsymbol{q}}$$

- angular velocities

$$\boldsymbol{p} = \begin{bmatrix} \dot{\boldsymbol{p}}_e \\ \boldsymbol{\omega} \end{bmatrix} \dot{\boldsymbol{q}}$$

While $\dot{\boldsymbol{p}}_e$ is the same vector in both analytical and angular matrices, the angular velocities ($\dot{\boldsymbol{\alpha}}$ and $\boldsymbol{\omega}$) don't represent the same quantities. In fact, $\boldsymbol{\omega}$ is a real vector, instead $\dot{\boldsymbol{\alpha}}$ is not. The reason of this lies in the derivative of the rotation matrix R_n^0.
In order to compute the geometric jacobian of a generic manipulator made of n links, it's better to distinguish the cases of revolute or prismatic joints and follow the guidelines below. The geometric jacobian is expressed as:

$$\boldsymbol{J} = \begin{bmatrix} J_1 & J_2 & \dots & J_n \end{bmatrix} = \begin{bmatrix} J_{P1} & J_{P2} & \dots & J_{Pn} \\ J_{O1} & J_{O2} & \dots & J_{On} \end{bmatrix}$$

The J_i expression depends on the joint type:

$$\boldsymbol{J}_i = \begin{bmatrix} J_{Pi} \\ J_{Oi} \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}_{i-1} \times (\boldsymbol{d}_0^n - \boldsymbol{d}_0^{i-1}) \\ \boldsymbol{z}_{i-1} \end{bmatrix} \text{if the joint is } \textit{revolute} \tag{3.11}$$

$$\boldsymbol{J}_i = \begin{bmatrix} J_{Pi} \\ J_{Oi} \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}_{i-1} \\ 0 \end{bmatrix} \text{ if the joint is } \textit{prismatic} \tag{3.12}$$

Notice that the jacobian computation is quite easy if the homogeneous transformation matrix T_n^0 is already obtained, in fact all the vectors could be find inside it:

- $\boldsymbol{z}_{i-1}$ vector corresponds to the first three terms of the 3-th column of T_0^{i-1};
- $\boldsymbol{d}_0^n$ vector is the pose vector, i.e the first three terms of the 4-th column of T_0^n;
- $\boldsymbol{d}_0^{i-1}$ as before, are the first three terms of the 4-th column of T_0^{i-1}

3.3 Gimbal Jacobians

With reference to (3.11) and (3.12), the linear and geometric jacobians of the gimbal system are obtained in the actual section and will be used in the next chapter to achieve the equations of motion of gimbal system. Let approximate the gimbal system to a spherical manipulator, the overall jacobian will be:

$$J = \begin{bmatrix} J_{P1} & J_{P2} \\ J_{O1} & J_{O2} \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}_0 \times (\boldsymbol{p}_2 - \boldsymbol{p}_0) & \boldsymbol{z}_1 \times (\boldsymbol{p}_1 - \boldsymbol{p}_0) \\ \boldsymbol{z}_0 & \boldsymbol{z}_1 \end{bmatrix} \tag{3.13}$$

being

$$\boldsymbol{z}_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \ \boldsymbol{z}_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

3.3.1 Gimbal Linear Jacobians

From (3.9) the linear jacobian is computed, by means of the coordinate transformation matrix T_3^0:

$$T_3^0(\boldsymbol{q}) = \begin{bmatrix} R_3^0 & d_3^0 \\ 0 & 1 \end{bmatrix}$$

Isolating the pose vector $\boldsymbol{p}$ from T_3^0 and computing the time derivative of its terms:

$$\boldsymbol{p} = \begin{bmatrix} c_1 s_2 d_3 + d_1 \\ s_1 s_2 d_3 \\ c_2 d_3 \end{bmatrix} \implies \begin{cases} x(t) = c_1 s_2 d_3 + d_1 \\ y(t) = s_1 s_2 d_3 \\ z(t) = c_2 d_3 \end{cases} \implies \begin{cases} \dot{x}(t) = -s_1 s_2 d_3 \dot{q}_1 - c_1 c_2 d_3 \dot{q}_2 \\ \dot{y}(t) = c_1 s_2 d_3 \dot{q}_1 + s_1 c_2 d_3 \dot{q}_2 \\ \dot{z}(t) = -s_2 d_3 \dot{q}_2 \end{cases}$$

If $\dot{\boldsymbol{p}}(t) = \boldsymbol{J}_L \dot{\boldsymbol{q}}$,then:

$$\boldsymbol{J}_L = \begin{bmatrix} J_{L1} \\ J_{L2} \end{bmatrix} = \begin{bmatrix} -d_3 s_1 s_2 & d_3 c_1 c_2 \\ d_3 c_1 s_2 & d_3 s_1 c_2 \\ 0 & -d_3 s_2 \end{bmatrix} \in R^{3\times 2} \tag{3.14}$$

where

$$J_{L1} = \begin{bmatrix} -d_3 s_1 s_2 \\ d_3 c_1 s_2 \\ 0 \end{bmatrix} \quad , \quad J_{L2} = \begin{bmatrix} d_3 c_1 c_2 \\ d_3 s_1 c_2 \\ -d_3 s_2 \end{bmatrix}$$

3.3.2 Gimbal Geometric Jacobians

As said, the geometric jacobian consist of the analytical and orientation terms.

J_P evaluation

Considering the 2 DOF of the gimbal system, which can rotate around the azimuth and the elevation axis, can be identified the two rotation vectors:

$$\begin{cases} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \text{ for azimuth rotation} \\ \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T \text{ for elevation rotation} \end{cases}$$

Then the analytical matrix is:

$$\boldsymbol{J}_P = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{3.15}$$

J_O evaluation

The orientation jacobian strongly depends on the chosen angles representation (RPY, Euler angles, ...). A relation exists between the eulerian and angular velocities: the first can be transformed in the second ones by means of a transformation matrix. Identified the eulerian velocities as

$$\boldsymbol{\omega}_e = \begin{bmatrix} 0 & \dot{\theta}_2 & \dot{\theta}_1 \end{bmatrix}^T = \begin{bmatrix} 0 & \dot{q}_2 & \dot{q}_1 \end{bmatrix}^T$$

the angular velocities are:

$$\boldsymbol{\omega} = T_{RPY} \boldsymbol{\omega}_e$$

Being $\theta_1 = \theta_z$ and $\theta_2 = \theta_y$, the transformation matrix T_{RPY} is defined as:

$$T_{RPY} = \begin{bmatrix} c_{\theta_y} c_{\theta_z} & -s_{\theta_z} & 0 \\ c_{\theta_y} s_{\theta_z} & c_{\theta_z} & 0 \\ -s_{\theta_y} & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_1 c_2 & -s_1 & 0 \\ c_2 s_1 & c_1 & 0 \\ -s_2 & 0 & 1 \end{bmatrix} \tag{3.16}$$

By applying (3.16):

$$\boldsymbol{\omega} = T_{RPY} \boldsymbol{\omega}_e = \begin{bmatrix} c_1 c_2 & -s_1 & 0 \\ c_2 s_1 & c_1 & 0 \\ -s_2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \dot{q}_2 \\ \dot{q}_1 \end{bmatrix} = \begin{bmatrix} -s_1 \dot{q}_2 \\ c_1 \dot{q}_2 \\ \dot{q}_1 \end{bmatrix}$$

If $\boldsymbol{\omega} = J_O \dot{\boldsymbol{q}}$, then J_O will be:

$$J_O = \begin{bmatrix} J_{O1} & J_{O2} \end{bmatrix} \text{ with } J_{O1} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \; J_{O2} = \begin{bmatrix} -s_1 \\ c_1 \\ 0 \end{bmatrix} \tag{3.17}$$

According to (3.13), the **geometric jacobian** of the gimbal system is:

$$J = \begin{bmatrix} -d_3 s_1 s_2 & d_3 c_1 c_2 \\ d_3 c_1 s_2 & d_3 s_1 c_2 \\ 0 & -d_3 s_2 \\ 0 & -s_1 \\ 0 & c_1 \\ 1 & 0 \end{bmatrix} \tag{3.18}$$

Chapter 4

Dynamics

The *dynamics* describes how forces and moments acting on a system can cause the motion variation of the system itself. Since the forces computation is very useful in choosing actuators, or transmission, or in designing the joints, a *dynamic model* of the mechanical system is required when dealing with a control problem. Define the dynamic model means find the system motion equations, in the *joint space*, through two main tecnhiques: the Langrange and Newton-Euler formulations. In the thesis the Lagrange equations will be derived, also if the Newton-Euler approach is a more precise and computationally more efficient than the used one.
References of the actual chapter are [2], [6].

4.1 Lagrange equations of a generic manipulator

The *Lagrange Formulation* provides a set of motion equations through a systematical method. Let assume a manipulator made of n-link, having n DOF: the *generalized coordinates* of each link must be identified as q_i, with $i = 1, \dots, n$. Once the generalized coordinates vector is established, the *Lagrangian* expression of the system is defined as:

$$\mathcal{L} = \mathcal{T} - \mathcal{U} \tag{4.1}$$

Note that the Lagrangian consists of two terms: T and U representing the *kinetics* and *potential energy*, respectively. However the lagrangian is employed to obtain the set of the motion equations, as below:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \mathcal{F}_i \qquad i\text{=1,...,n} \tag{4.2}$$

The expression (4.2) is computed for each i-th link; the entity $\boldsymbol{\mathcal{F}}_i$ represents the *generalized force* vector for the generalized coordinate q_i. In the generalized forces vector can be found the contributions due to the non-conservative and the friction forces, since the manipulator interacts with the environment.

Summing the contributions of all the links, the matrix form of (4.2) is obtained:

$$\frac{d}{dt}\left(\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \dot{\boldsymbol{q}}}\right)^T - \left(\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{q}}\right)^T = \boldsymbol{\mathcal{F}} \tag{4.3}$$

Each matrix will have as many rows as links. From (4.1), the computations of the system kinetic and potential energies are necessaries.

4.1.1 Kinetic energy

Assuming a n-link and n-DOF electromechanical system, its total kinetic energy is:

$$\mathcal{T} = \sum_{i=1}^{n}(\mathcal{T}_{li} + \mathcal{T}_{mi}) \tag{4.4}$$

being $\mathcal{T}_{li}$ the kinetic energy associated to the i-th link, while $\mathcal{T}_{mi}$ is the kinetic energy of the i-th motor moving the i-th link.
As regard the link kinetic energy is the sum of **translational** and **rotational** terms. Then the total kinetic energy of each link is summarized as:

$$\mathcal{T}_{li} = \underbrace{\frac{1}{2}m_{li}\dot{\boldsymbol{q}}^T\boldsymbol{J}_P^{(l_i)T}\boldsymbol{J}_P^{(l_i)}\dot{\boldsymbol{q}}}_{translational} + \underbrace{\frac{1}{2}\dot{\boldsymbol{q}}^T\boldsymbol{J}_O^{(l_i)T}\boldsymbol{R}_i\boldsymbol{I}_{l_i}^i\boldsymbol{R}_i{}^T\boldsymbol{J}_O^{(l_i)}\dot{\boldsymbol{q}}}_{rotational} \tag{4.5}$$

The same employed approach for $\mathcal{T}_l$ can be done for $\mathcal{T}_m$, which will appear as below:

$$\mathcal{T}_{mi} = \underbrace{\frac{1}{2}m_{mi}\dot{\boldsymbol{q}}^T\boldsymbol{J}_P^{(m_i)T}\boldsymbol{J}_P^{(m_i)}\dot{\boldsymbol{q}}}_{translational} + \underbrace{\frac{1}{2}\dot{\boldsymbol{q}}^T\boldsymbol{J}_O^{(m_i)T}\boldsymbol{R}_{mi}\boldsymbol{I}_{m_i}^{m_i}\boldsymbol{R}_{m_i}^T\boldsymbol{J}_O^{m_i}\dot{\boldsymbol{q}}}_{rotational} \tag{4.6}$$

Replacing (4.5) and (4.6) in (4.4) the total kinetic energy of the system is obtained, as:

$$\mathcal{T} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}(\boldsymbol{q})\dot{q}_i\dot{q}_j = \frac{1}{2}\dot{\boldsymbol{q}}^T\boldsymbol{H}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{4.7}$$

where $\boldsymbol{H}(\boldsymbol{q})$ is the **inertia matrix**, defined as a symmetric and positive $[n \times n]$ matrix. Usually it also is configuration-dependent, like as the inertia of a body is.

4.1.2 Potential energy

As known, the potential energy of a body is due to its position with respect to a force field. In the case of a n-link manipulator actuated through motors, again the total potential energy is given by two contributions:

$$\mathcal{U} = \sum_{i=1}^{n}(\mathcal{U}_{li} + \mathcal{U}_{mi}) \tag{4.8}$$

where $\mathcal{U}_{li}$ is the link potential energy and $\mathcal{U}_{mi}$ is the motor potential energy. In detail:

$$\mathcal{U}_{l_i} = -m_{li}\boldsymbol{g}_0^T\boldsymbol{p}_{l_i}\ , \mathcal{U}_{m_i} = -m_{mi}\boldsymbol{g}_0^T\boldsymbol{p}_{m_i} \tag{4.9}$$

where $\boldsymbol{g}_0^T$ represents the gravitational vector, having as component $\boldsymbol{g}_0^T = \begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$. Moreover $\boldsymbol{p}_{l_i}$ and $\boldsymbol{p}_{m_i}$ express the position of link and actuator, respectively.
Substituting the above relations in (4.8), the total potential energy appears as:

$$\mathcal{U} = -\sum_{i=1}^{n}(m_{l_i}\boldsymbol{g}_0^T\boldsymbol{p}_{l_i} + m_{m_i}\boldsymbol{g}_0^T\boldsymbol{p}_{m_i}) \tag{4.10}$$

Under the assumption of rigid link, only the gravity force is taking into account. Obviously if the mechanical structure features elastic properties, also the elastic force term must be added to (4.10).

4.1.3 Manipulator equations of motion

Once computed the total kinetic and potential energy, $\mathcal{T}$ and $\mathcal{U}$ respectively, according to (4.1) the lagrangian is obtained. Then the motion equations can be get by partially deriving (with respect to q_i) or computing the time derivative of the lagrangian, as in (4.2). In matrix form:

$$\boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\mathcal{F}} \tag{4.11}$$

From now on, since the forces acting on the manipulator are the actuator torques, the generalized forces vector $\boldsymbol{\mathcal{F}}$ will be replaced with the torque vector $\boldsymbol{\tau}$. By substituting (4.7) and (4.10) into (4.2) and by evaluating each of this terms, the overall equation of motion for each link is obtained, as below:

$$\sum_{j=1}^{n} h_{ij}(\boldsymbol{q})\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}(\boldsymbol{q})\dot{q}_k\dot{q}_j + g_i(\boldsymbol{q}) = \boldsymbol{\tau}_i \tag{4.12}$$

being c_{ijk} the so-called *Christoffel symbols*, defined as:

$$c_{ijk} = \frac{\partial h_{ij}}{\partial q_k} - \frac{1}{2}\frac{\partial h_{jk}}{\partial q_i} \tag{4.13}$$

The previously introduced Christoffel symbols represent the elements of the $\boldsymbol{C}$ matrix, which will be given a physical meaning later. Then the matrix form of the motion equation of a generic robot manipulator will take the following form:

$$\boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{4.14}$$

Also the $\boldsymbol{\tau}$ vector deserves clarifications: it represents the *non conservative forces* vector, made of the contributions of actuator torque ($\boldsymbol{\tau}_a$) and friction force that do work at the joint. The friction torques are expressed by means of $\boldsymbol{F}_v$ $(n \times n)$ and $\boldsymbol{F}_s$ $(n \times n)$ matrices, coinciding with the viscous and static friction respectively.

Furthermore the contribution of the external forces acting on the manipulator gripper must be taken into account, due to the interaction of the end-effector with the environment. The latter effect is included into the Lagrange equations through $\boldsymbol{J}^T\boldsymbol{h}_e$, in which $\boldsymbol{h}_e$ identifies the moments and forces applied by the gripper on the workspace. In light of this, the motion equation in matrix form becomes:

$$\boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}_v\dot{\boldsymbol{q}} - \boldsymbol{F}_s sgn(\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}_a - \boldsymbol{J}^T\boldsymbol{h}_e \tag{4.15}$$

4.1.4 Physical meaning of the terms

According to (4.12) and (4.15) can be identified:

- *inertial terms*:
  - h_{ii} represent the moment of intertia at joint i in the actual manipulator configuration;
  - h_{ij} coincides with the effect of the j-th joint acceleration on joint i;
- *quadratic terms*
  - $c_{ijj}\dot{q}_i^2$ is the *centrifugal* term due to the velocity of joint j on joint i;
  - $c_{ijk}\dot{q}_j\dot{q}_k$ is the *Coriolis* term: it shows the effect of the joint j and k velocities on joint i;
- *friction terms*:
  - f_{vii} is the element of the viscous friction matrix;
  - f_{sii} is the element of the static friction matrix;
- g_i is the gravitational contribution on joint i, depending on the manipulator configuration.

4.2 Dynamics of the Gimbal

In this section the motion equation of the gimbal system will be provided, also by using some of the results obtained in the previous chapter and accordingly to the above pointed out theory.

Guidelines for deriving the gimbal lagrange equation

1. Compute the $\boldsymbol{H}(\boldsymbol{q})$ matrix:
   - find out the coordinate transformation matrix T_3^0;
   - obtain the jacobians of the gimbal;

- use the linear and angular jacobians to evaluate the total kinetic energy $\mathcal{T}$ and the $\boldsymbol{H}(\boldsymbol{q})$ matrix;

2. Calculate the $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ matrix consistently to (4.13), by means of Christoffel symbols;

3. Obtain $\boldsymbol{g}(\boldsymbol{q})$:

   - evaluate the total potential energy, as in (4.10);
   - $\boldsymbol{g}_i(\boldsymbol{q}_i)$ will be the result of the partial derivative of $\mathcal{U}$ with respect to the joint variables q_i;
   - once known all the i-th component of $\boldsymbol{g}(\boldsymbol{q})$, assemble the gravitational vector.

4. Write the motion equations, also in matrix form, as in (4.14).

4.2.1 Simplifying conditions concerning the Gimbal

As said in the previous chapters, the dynamics equations of the gimbal will be derived starting from the approximation of the gimbal to a spherical arm robot manipulator. Then the reference figure is the following one:

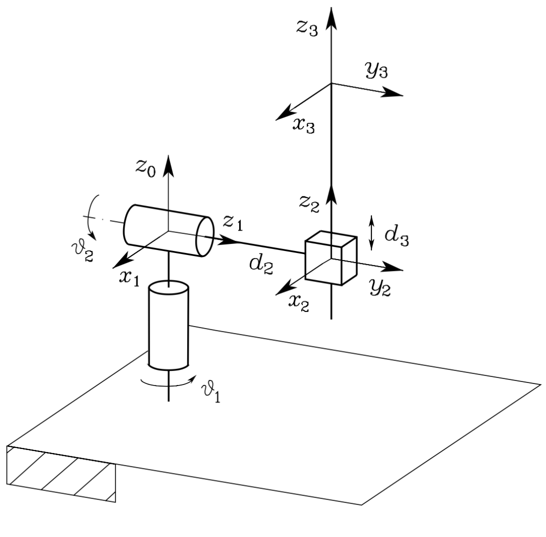


Figure 4.1: Shperical arm manipulator reference

The explained lagrangian procedure will be implement, but earlier the simplifying assumptions of the study are listed:

- the gimbal mechanical structure is considered as rigid one;
- assume that the joint rotational axis are orthogonal;

- let the center of mass of the links between each joints closed by the rotational axis: the overall structure is well balanced;
- also if the inertia matrices are not diagonal, because of the links are not symmetric, they are considered as diagonal ones in order to decrease the number of unknowns during the computation of the dynamic problem;
- in the presence of small angles variations (regarding the elevation motion, for example) it's possible to approximate $\sin\theta_i = \theta_i$ and $\cos\theta_i = 1$
- neglect the static friction term, then $\boldsymbol{F}_s sgn(\dot{\boldsymbol{q}})$ is null; consider the viscous friction as linear means dealing with a diagonal $\boldsymbol{F}_v$ matrix:
$$\boldsymbol{F}_v = diag\begin{bmatrix} f_{v1} & f_{v2} & \dots & f_{vn}\end{bmatrix}$$
- during the calculation of the total kinetic energy, force the linear velocity jacobians as null, due to the fact that the gimbal structure only rotates and doesn't translate: if the manipulator has n links, for i=1, ...,n
$$\boldsymbol{J}_{vi} = 0$$
- concerning to purely rotational systems, the gravity force doesn't affect in any way the dynamics of the system itself, due to the center of mass which features a constant position:
$$\boldsymbol{g}(\boldsymbol{q}) = 0$$
- during the evaluation of the generalized forces vector the term $-\boldsymbol{J}^T\boldsymbol{h}_e$ is not contemplate: since the gripper interacts with the environment, the external forces acting on the system are the aerodynamic moments essentially. In the simulation phase they will be modeled as sinusoidal disturbances affecting the measurements of the gimbal position (provided by the encoder):
$$\boldsymbol{\tau} = \boldsymbol{\tau}_a$$
where $\boldsymbol{\tau}_a$ represents the actuating torques generated by means of the DC-motors.

4.2.2 Lagrange dynamic equations of the Gimbal

Taking into account the premises made in the previous section, the dynamic equations of the gimbal are obtained. The orientation jacobians, computed in chapter 3 through (3.17), are listed below for better readability of the text:

$$J_{O1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \tag{4.16}$$

$$J_{O2} = \begin{bmatrix} 0 & -s_1 \\ 0 & c_1 \\ 1 & 0 \end{bmatrix} \tag{4.17}$$

The jacobians are the essential elements to achieve the dynamic equation of motion, in matrix form:

$$\boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}_v\dot{\boldsymbol{q}} = \boldsymbol{\tau}_a \tag{4.18}$$

Step 1. Define the generalized coordinate vector

Are considered as *generalized coordinates* the states of the mechanical system, i.e a set of coordinates able to univocally describe the analyzed system. As states are chosen the angular positions $(q_1(t), q_2(t))$ and the angular velocities $(\dot{q}_1(t), \dot{q}_2(t))$ of the gimbal. As result, the generalized coordinates vector, also called *state vector*, will be a $[4 \times 1]$ vector:

$$\boldsymbol{q} = \begin{bmatrix}\theta_1 & \theta_2 & \dot{\theta}_1 & \dot{\theta}_2\end{bmatrix}^T \tag{4.19}$$

where $\theta_1 = q_1$ (azimuth angle), $\theta_2 = q_2$ (elevation angle), $\dot{\theta}_1 = q_3$, $\dot{\theta}_2 = q_4$.

Step 2. Find out the inertia matrix $\boldsymbol{H}(\boldsymbol{q})$

Since the gimbal system features 2 links, the overall inertia matrix is given by:

$$H = H_1 + H_2 \tag{4.20}$$

being H_1 and H_2 the inertia matrices dealing with link 1 and 2 respectively. Moreover Γ_1 and Γ_2 coincide with the inertia matrix of each link, assumed to be diagonal as known from section 4.2.1.
For the first link, which can rotate around the azimuth axis:

$$\begin{aligned} H_1 &= (\boldsymbol{J}_{O1})^T \Gamma_1 \boldsymbol{J}_{O1} = \\ &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Gamma_{1x} & 0 & 0 \\ 0 & \Gamma_{1y} & 0 \\ 0 & 0 & \Gamma_{1z} \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} = \\ &= \begin{bmatrix} \Gamma_{1z} & 0 \\ 0 & 0 \end{bmatrix} \end{aligned} \tag{4.21}$$

The same is done for the second link, allowed to rotate around the elevation axis:

$$\begin{aligned} H_2 &= (\boldsymbol{J}_{O2})^T \Gamma_2 \boldsymbol{J}_{O2} = \\ &= \begin{bmatrix} 0 & 0 & 1 \\ -s_1 & c_1 & 0 \end{bmatrix} \begin{bmatrix} \Gamma_{2x} & 0 & 0 \\ 0 & \Gamma_{2y} & 0 \\ 0 & 0 & \Gamma_{2z} \end{bmatrix} \begin{bmatrix} 0 & -s_1 \\ 0 & c_1 \\ 1 & 0 \end{bmatrix} = \\ &= \begin{bmatrix} \Gamma_{2z} & 0 \\ 0 & {s_1}^2\Gamma_{2x} + {c_1}^2\Gamma_{2y} \end{bmatrix} \end{aligned} \tag{4.22}$$

Replacing H_1 and H_2 expressions into (4.20):

$$H = \begin{bmatrix} \Gamma_{1z} + \Gamma_{2z} & 0 \\ 0 & {s_1}^2\Gamma_{2x} + {c_1}^2\Gamma_{2y} \end{bmatrix} \tag{4.23}$$

Step 3. Compute the $\boldsymbol{C}(q,\dot{q})$ matrix

In order to achieve the goal of the actual step, the Christoffel symbols must be evaluated. The $\boldsymbol{C}(q,\dot{q})$ is a $[2\times 2]$ matrix, and its own elements are given by:

$$\begin{cases} c_{11} = c_{111}\dot{q}_1 + c_{112}\dot{q}_2 \\ c_{12} = c_{121}\dot{q}_1 + c_{122}\dot{q}_2 \\ c_{21} = c_{211}\dot{q}_1 + c_{212}\dot{q}_2 \\ c_{22} = c_{221}\dot{q}_1 + c_{222}\dot{q}_2 \end{cases} \tag{4.24}$$

According to (4.24) the $\boldsymbol{C}(q,\dot{q})$ matrix will be:

$$\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \tag{4.25}$$

The elements of the $\boldsymbol{C}(q,\dot{q})$ matrix are computed as:

$$\begin{cases} c_{111} = \frac{1}{2}\frac{\partial h_{11}}{\partial q_1} = 0 \\ c_{112} = c_{121} = \frac{1}{2}\frac{\partial h_{11}}{\partial q_2} = 0 \\ c_{122} = \frac{\partial h_{12}}{\partial q_2} - \frac{1}{2}\frac{\partial h_{22}}{\partial q_1} = (-s_1c_1\Gamma_{2x} + s_1c_1\Gamma_{2y})\dot{q}_1 \\ c_{211} = \frac{\partial h_{21}}{\partial q_1} - \frac{1}{2}\frac{\partial h_{11}}{\partial q_2} = 0 \\ c_{212} = c_{221} = \frac{1}{2}\frac{\partial h_{22}}{\partial q_1} = (s_1c_1\Gamma_{2x} - s_1c_1\Gamma_{2y})\dot{q}_1 \\ c_{222} = \frac{\partial h_{22}}{\partial q_2} = 0 \end{cases} \tag{4.26}$$

Note the presence of the inertia matrix H elements in (4.26). Replacing the results of (4.26) into (4.24), then:

$$\begin{cases} c_{11} = 0 \\ c_{12} = s_1c_1(\Gamma_{2y} - \Gamma_{2x})\dot{q}_1\dot{q}_2 \\ c_{21} = -s_1c_1(\Gamma_{2y} - \Gamma_{2x})\dot{q}_1\dot{q}_2 \\ c_{22} = s_1c_1(\Gamma_{2x} - \Gamma_{2y})\dot{q}_1^2 \end{cases}$$

As already mentioned, the $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ matrix is made of quadratic velocities terms (c_{22}) and explains the contribution of the velocities of the j-th joint on joint i (c_{12}, c_{21}). Anyhow is clear that a strong nonlinearity characterizes the matrix.
Once obtained the values of the Christoffel symbols,the Coriolis matrix will appear as:

$$\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} 0 & s_1c_1(\Gamma_{2y} - \Gamma_{2x}) \\ -s_1c_1(\Gamma_{2y} - \Gamma_{2x}) & s_1c_1(\Gamma_{2x} - \Gamma_{2y}) \end{bmatrix} \tag{4.27}$$

Step 4. Define the viscous friction matrix $\boldsymbol{F}_v$

The viscous friction is modeled as linear, then $\boldsymbol{F}_v$ will be a diagonal $[2 \times 2]$ matrix:

$$\boldsymbol{F}_v = \begin{bmatrix} f_{v1} & 0 \\ 0 & f_{v2} \end{bmatrix}$$

where f_{v1} and f_{v2} coincide with the friction coefficients dealing with link 1 and 2, respectively. In detail:

- concerning with the 1-st link rotating around the azimuth axis, determine the f_{v1} coefficient means consider the bronze-steel coupling. For these materials the standard viscous friction coefficients are in the range $0.15 \div 0.20$: considering the worst friction case is chosen $f_{v1} = 0.20$;
- about the 2-nd link, the viscous friction coefficient is closed to zero: is choosen $f_{v2} = 0.002$.

Because of the clarifications above, the friction matrix is the following one:

$$\boldsymbol{F}_v = \begin{bmatrix} f_{v1} & 0 \\ 0 & f_{v2} \end{bmatrix} = \begin{bmatrix} 0.20 & 0 \\ 0 & 0.002 \end{bmatrix} \tag{4.28}$$

Step 5. Consider the generalized force vector $\boldsymbol{\tau}_a$

Since each link of the gimbal is actuated by a DC-motor, the generalized force vector consist of a $[2 \times 1]$ vector:

$$\boldsymbol{\tau}_a = \begin{bmatrix} \boldsymbol{\tau}_{a1} \\ \boldsymbol{\tau}_{a2} \end{bmatrix} \tag{4.29}$$

where $\boldsymbol{\tau}_{a1}$ and $\boldsymbol{\tau}_{a2}$ are the torques generated through the DC motors at each link in order to move the gimbal system, given an input command.
The dynamic of the DC motors will be exhaustively treated in the next chapter.

Step 6. Write the dynamic equation

Collect all the results provided during the previous steps and write down the dynamic equations of the gimbal system. In this scenario, two differential motion equations are generated by means of the lagrange formulation, each one of them describes the dynamics of the given link. Based on (4.18), the dynamic equations are joined in the matrix form as:

$$\underbrace{\begin{bmatrix} \Gamma_{1z} + \Gamma_{2z} & 0 \\ 0 & {s_1}^2\Gamma_{2x} + {c_1}^2\Gamma_{2y} \end{bmatrix}}_{H} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & s_1c_1(\Gamma_{2y} - \Gamma_{2x}) \\ -s_1c_1(\Gamma_{2y} - \Gamma_{2x}) & s_1c_1(\Gamma_{2x} - \Gamma_{2y}) \end{bmatrix}}_{C} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0.20 & 0 \\ 0 & 0.002 \end{bmatrix}}_{F_v} = \underbrace{\begin{bmatrix} \boldsymbol{\tau}_{a1} \\ \boldsymbol{\tau}_{a2} \end{bmatrix}}_{\tau_a} \tag{4.30}$$

From (4.30) is possible to extract the two differential equations of link 1 and 2:

$$\begin{cases} (\Gamma_{1z}+\Gamma_{2z})\ddot{q}_1 + s_1c_1(\Gamma_{2y}-\Gamma_{2x})\dot{q}_1\dot{q}_2 + f_{v1}\dot{q}_1 = \tau_{a1} \\ (s_1^2\Gamma_{2x}+c_1^2\Gamma_{2y})\ddot{q}_2 + s_1c_1(\Gamma_{2x}-\Gamma_{2y})\dot{q}_1\dot{q}_2 + f_{v2}\dot{q}_2 = \tau_{a2} \end{cases} \tag{4.31}$$

being

$$\boldsymbol{q} = \begin{bmatrix} q_1 & q_2 & \dot{q}_1 & \dot{q}_2 \end{bmatrix}^T = \begin{bmatrix} \theta_1 & \theta_2 & \dot{\theta}_1 & \dot{\theta}_2 \end{bmatrix}^T \implies \ddot{q}_1 = \ddot{\theta}_1, \ddot{q}_2 = \ddot{\theta}_2$$

where θ_1 and θ_2 are the azimuth and elevation angles, $\dot{\theta}_1$ and $\dot{\theta}_2$ are the angular velocities, $\ddot{\theta}_1$ and $\ddot{\theta}_2$ are the angular accelerations.
Some thoughts concerning the obtained motion equations:

- the equations are strongly non-linear: the presence of the product $(\dot{q}_1\dot{q}_2)$ is a non-linear element;
- the equations are of the time-variant type due to the presence of c_1,s_1 that are varying during time;
- in order to design a suitable controller of the gimbal system, a linearization is required, because the controller works with linear time-invariant systems (LTI systems). Also a state-space description of the mechanical system is needed.

4.3 Dynamics of the actuators

Actuators and *sensors* are two basic components of a manipulator: presently the operating principles of the electric actuators will be introduced and their dynamics will be obtained.
The topic is then related to the ***joint actuating systems***, generally made of:

1. *power supply*, which aim is to supply the whole actuating system by helpfully converting the alternating voltage (from the distribution) into direct one, appropriate to the requests of the power amplifier;
2. *power amplifier*, which controls and modulates the power flow coming from the power supply. Its task is to deliver the amount of energy needed and required by the actuators, also taking into account the portion of supply which will be loosed due to dissipation phenomenas;
3. *motors or servomotors*, that can be of two types: electrics if the manipulator has small dimensions, otherwise the actuators are usually of hydraulic kind;
4. *transmission*, inevitably located between the joints and their actuators due to the fact that the joint requires low speed and high torques to complete the given task. On the contrary, the servomotors provides low torques and very high speed, then a *gear* is inserted between motor and joint. For sure adding new components to the chain means keep in mind also the presence of backlash and friction dissipations, from a control problem of view.

4.3.1 Electric servomors

The electric DC motors are the widely used servomotors in robotic field. The dynamics equations can be obtained by means of the equivalent circuit of an electric motor, that is shown below:

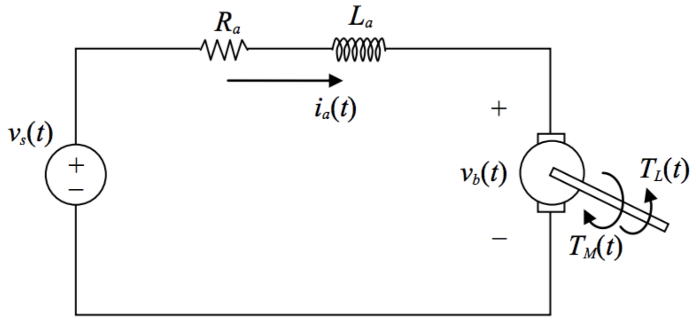


Figure 4.2: Equivalent circuit of a DC motor

Referring to 4.2 are defined:

- R_a, the armature resistance;
- L_a, the inductance, that can be neglected;
- i_a, the armature current;
- $v_s(t)$, voltage source;
- $v_b(t)$, back electromotive force, proportional to the angular motor speed ω;
- $T_M(t)$ is the torque provided by the motor, while $T_L(t)$ is the load required torque;
- if the motor is supplied through a direct voltage $\implies v_s(t) = V_s, i_a(t) = I_a$, and also the torques become constant entities when the steady-state is reached.

Let's assume the model of the circuit of figure 4.2, and a constant supply, then the Kirchhoff's Voltage Law is applied to the scheme:

$$R_a I_a + L_a \frac{di_a}{dt} + V_b = V_s \tag{4.32}$$

As said, the back electromotive force is proportional to the angular velocity, by means of the K_b back emf constant

$$V_b = K_b \omega \tag{4.33}$$

While the torque is proportional to the armature current, through the torque constant K_m. The operating principle of a DC motor is expressed by:

$$\tau = K_m I_a \tag{4.34}$$

where $\tau = T$, represents the torque provided by the motor actuating the joint: it's not possible to directly control the torque, but by means of control action is allowed to control the input voltage V_s. From (4.32),(4.33),(4.34), the i-th torque actuating each joint can be expressed as:

$$\tau_i = \frac{K_{mi}}{R_i}(V_i - K_{bi}\omega_i) \tag{4.35}$$

Besides, is possible to prove that $K_b = K_m$ if both are in the SI, then (4.32) (dynamic equation of the DC motor) can be rewrite as:

$$R_a I_a + L_a \frac{di_a}{dt} + K\omega = V_s \tag{4.36}$$

When the DC motor drives a payload (as in the gimbal case, where the motors are needed to move the rotational axis of the camera), the inertia of the motor and the friction coefficient are to be taken into account when extracting the mechanical equation, as below:

$$J_m \frac{d\omega}{dt} + B_m \omega = \tau \tag{4.37}$$

By using (4.36) and (4.37), given $\omega = \dfrac{d\theta}{dt}$, the system of equations describing the mechanical and electric behavior of the DC motor is:

$$\begin{cases} L_a \dfrac{dI_a}{dt} + R_a I_a = V_s - K\dfrac{d\theta}{dt} \\ J_m \dfrac{d^2\theta}{dt} + B_m \dfrac{d\theta}{dt} = K I_a \end{cases} \tag{4.38}$$

Starting from (4.38) the state-space matrices of the motor will be derived, in the next chapter.

Chapter 5

State-space representation and Linearization

The modern control theory needs the so called *state-space* representation to know the dynamics of the system to be controlled, i.e. the dynamics of the plant. Differently from the *transfer function* description of a system, the state-space representation allows to not consider the plant as a black-box, because an internal knowledge is possible. For this reason the state-space model of the gimbal will be obtained in the actual chapter. Then the model will be linearized, in order to design a suitable controller for the gimbal LTI system.

5.1 State-space model

The state-space model is a mathematical method to represent the dynamic of a real physical system, by means of inputs, outputs and *state variables* definitions. All these system variables are related thanks to n differential equations, also permitting to dealing with multiple inputs multiple outputs (MIMO) systems.

5.1.1 State of the system

The state of the system ($x_i(t)$) is an internal variable of the plant and evolves during time. A system usually has n state variables, depending on the n order of the system itself: all the state variables $x_i(t)$, with $i = 1, \ldots, n$ are collected in the **state vector** $x(t)$. By definition the state vector consists of the *minimal* variables that can univocally determine the overall state of the system, at each given time. In fact, at time t the state vector is the amount of the system information until t and, together with u, is able to determine the system behavior for all instants $t_i > t$.
The state variables of the gimbal system are the rotation angles and the angular velocities of link 1 and 2. So, the state vector appears as:

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T = \begin{bmatrix} \theta_1 & \theta_2 & \dot{\theta}_1 & \dot{\theta}_2 \end{bmatrix}^T \tag{5.1}$$

5.1.2 State-space equations

The mathematical model of the system provides n first-order differential equations, also called *state equations*. The latter, together with the *output equation* determine the *state-space description* of a system, which is defined as:

$$\begin{cases} \dot{\boldsymbol{x}}\boldsymbol{(t)} = \boldsymbol{A}(t)\boldsymbol{x}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C}(t)\boldsymbol{x}(t) + \boldsymbol{D}(t)\boldsymbol{u}(t) \end{cases} \tag{5.2}$$

where, if the system has p inputs, q outputs and n state variables:

- $\boldsymbol{x}(t)$ is the state vector, according to (5.1) $\implies \boldsymbol{x}(t) \in \mathbb{R}^n$;
- $\boldsymbol{u}(t)$ is the system input vector, also called control vector $\implies \boldsymbol{u}(t) \in \mathbb{R}^p$;
- $\boldsymbol{y}(t)$ is the system output vector $\implies \boldsymbol{y}(t) \in \mathbb{R}^q$;
- $\boldsymbol{A}(t)$ is the $[n \times n]$ state matrix $\implies dim(\boldsymbol{A}) = \mathbb{R}^{n\times n}$;
- $\boldsymbol{B}(t)$ is the $[n \times p]$ input matrix $\implies dim(\boldsymbol{B}) = \mathbb{R}^{n\times p}$;
- $\boldsymbol{C}(t)$ is the $[q \times n]$ output matrix $\implies dim(\boldsymbol{C}) = \mathbb{R}^{q\times n}$;

Moreover, after the linearization process around an equilibrium point, the 4 matrices become constant. In this condition the system is said to be linear time invariant (LTI) and the state-space representation is the following one, according to (5.2):

$$\begin{cases} \dot{\boldsymbol{x}}\boldsymbol{(t)} = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t) \end{cases} \tag{5.3}$$

The overall gimbal holder will be considered as plant of the controller, then:

- $\boldsymbol{x}(t)$ is a $[4 \times 1]$ vector $\implies n = 4$;
- $\boldsymbol{u}(t)$ is a $[2 \times 1]$vector, i.e. the inputs are the torques provided by the two DC motors $\implies p = 2$;
- $\boldsymbol{y}(t)$ is a $[4 \times 1]$ vector $\implies q = 4$;
- $\boldsymbol{A}(t)$ is the $[4 \times 4]$ state matrix;
- $\boldsymbol{B}(t)$ is the $[2 \times 4]$ input matrix;
- $\boldsymbol{C}(t)$ is the $[4 \times 4]$ output matrix;

Also if the system has 4 outputs, the really controllable outputs are two, i.e. the gimbal tilt and pan angles.

5.2 Gimbal linearization

The real physical systems are usually non linear and time-variant. Neverthless there are many more powerful control techniques for LTI systems than for non-linear time-variant ones: for this reason the latter are approximated around an equilibrium point by means of suitable linear models, the so called *linearized models*. The choice of the equilibrium point is not univocal, because more than one equilibrium point should exists.
For the motion control purpose the gimbal system is split into two subsystems: the first is able to rotate around the azimuth axis and the second one rotates around the elevation axis. This decision is legitimized because the actuators, that are the two DC-motors, independently work by each other and they are controlled by two different controllers.

5.2.1 Section rotating around azimuth axis

From (4.31), the motion equation describing the dynamic of the system rotating around the azimuth axis is isolated:

$$(\Gamma_{1z} + \Gamma_{2z})\ddot{q}_1 + s_1c_1(\Gamma_{2y} - \Gamma_{2x})\dot{q}_1\dot{q}_2 + f_{v1}\dot{q}_1 = \tau_{a1} \tag{5.4}$$

With reference to (5.1) and (5.3):

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \implies \dot{\boldsymbol{x}}(t) = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}$$

To obtain the state space matrices, the dynamic system must be rewrite as:

$$\begin{cases} \dot{x}_1 = \dot{q}_1 = x_3 = f_1 \\ \dot{x}_2 = \dot{q}_2 = x_4 = f_2 \\ \dot{x}_3 = \ddot{q}_1 = f_3 \\ \dot{x}_4 = \ddot{q}_2 = f_4 \\ y = x_1 = g \end{cases} \tag{5.5}$$

In (5.5) the real controlled output variable coincides with pan angle, then θ_1. Besides to both state variables and output variable is associated a function (f_i and g). To reach matrices of the state equation, the jacobians of f_i with respect to x_i or u_i must be evaluated; while to obtain the matrices of the output equation, the jacobians of g with respect to x_i and u_i should be calculated. Then:

$$\boldsymbol{A}(t) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \frac{\partial f_1}{\partial x_4} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \frac{\partial f_2}{\partial x_4} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \frac{\partial f_3}{\partial x_4} \\ \frac{\partial f_4}{\partial x_1} & \frac{\partial f_4}{\partial x_2} & \frac{\partial f_4}{\partial x_3} & \frac{\partial f_4}{\partial x_4} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ A_{31} & A_{32} & A_{33} & A_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.6}$$

In details:

$$\begin{aligned}
f_3 &= -\frac{x_3 x_4(\Gamma_{2y}-\Gamma_{2x})(\sin x_1 \cos x_1)}{\Gamma_{1z}+\Gamma_{2z}} - \frac{x_3 f_{v1}}{\Gamma_{1z}+\Gamma_{2z}} + \frac{\tau_1}{\Gamma_{1z}+\Gamma_{2z}}\\
A_{31} &= \frac{\partial f_3}{\partial x_1} = -\frac{x_3 x_4(\Gamma_{2y}-\Gamma_{2x})(2\cos^2(x_1)-1)}{\Gamma_{1z}+\Gamma_{2z}}\\
A_{32} &= \frac{\partial f_3}{\partial x_2} = 0\\
A_{33} &= \frac{\partial f_3}{\partial x_3} = -\frac{x_4(\sin x_1 \cos x_1)(\Gamma_{2y}-\Gamma_{2x})}{\Gamma_{1z}+\Gamma_{2z}} - \frac{f_{v1}}{\Gamma_{1z}+\Gamma_{2z}}\\
A_{34} &= \frac{\partial f_3}{\partial x_4} = \frac{x_3(\sin x_1 \cos x_1)(\Gamma_{2y}-\Gamma_{2x})}{\Gamma_{1z}+\Gamma_{2z}}
\end{aligned} \tag{5.7}$$

In an analogous way:

$$\boldsymbol{B}(t) = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} \\ \frac{\partial f_2}{\partial u_1} \\ \frac{\partial f_3}{\partial u_1} \\ \frac{\partial f_4}{\partial u_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{\Gamma_{1z}+\Gamma_{2z}} \\ 0 \end{bmatrix} \tag{5.8}$$

Concerning the output matrix:

$$\boldsymbol{C}(t) = \begin{bmatrix} \frac{\partial g}{\partial x_1} & \frac{\partial g}{\partial x_2} & \frac{\partial g}{\partial x_3} & \frac{\partial g}{\partial x_4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.9}$$

As always, the forward matrix $\boldsymbol{D}(t) = 0$.
Now, in order to obtain state space matrices of the time-invariant type, an equilibrium point must be chosen and the just computed linear matrices should be calculated for the second time in the equilibrium point:

$$\boldsymbol{A} = \left.\frac{\partial f_i}{\partial x_i}\right|_{x_i=\overline{x}_i}, \boldsymbol{B} = \left.\frac{\partial f_i}{\partial u_j}\right|_{u_j=\overline{u}_j}, \boldsymbol{C} = \left.\frac{\partial g}{\partial x_i}\right|_{x_i=\overline{x}_i}, \boldsymbol{D} = \left.\frac{\partial g}{\partial u_j}\right|_{u_j=\overline{u}_j}$$

where $i = 1, \dots, 4$ for the state variables and $j = 1, 2$ for the input commands. The $\overline{x}_i$ and $\overline{u}_j$ terms represent the given values of the equilibrium point.
As equilibrium point is chosen:

$$\begin{cases}
\overline{x}_1 = k\frac{\pi}{2} \implies \cos x_1 = 0 \ , \ \sin x_1 = 1\\
\overline{x}_2 = \frac{\pi}{4} \implies \cos x_2 = \frac{\sqrt{2}}{2} \ , \ \sin x_2 = \frac{\sqrt{2}}{2}\\
\overline{x}_3 = 0 \implies \text{initial velocity null on azimuth axis}\\
\overline{x}_4 = 0 \implies \text{initial velocity null on elevation axis}
\end{cases} \tag{5.10}$$

Replacing the equilibrium point in (5.6),(5.8),(5.9) the final state space matrices are computed:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \dfrac{f_{v1}}{\Gamma_{1z}+\Gamma_{2z}} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ \dfrac{1}{\Gamma_{1z}+\Gamma_{2z}} \\ 0 \end{bmatrix}, \boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{D} = 0$$

5.2.2 Section rotating around the elevation axis

As done in the previous section, from (4.31) the dynamic equation of the rotation around the elevation axis is extracted:

$$(s_1^2\Gamma_{2x} + c_1^2\Gamma_{2y})\ddot{q}_2 + s_1c_1(\Gamma_{2x} - \Gamma_{2y})\dot{q}_1\dot{q}_2 + f_{v2}\dot{q}_2 = \tau_{a2} \tag{5.11}$$

The relations of (5.5) are still valid, with the only difference that now the real output is $y(t) = x_2 = g$. Again, as equilibrium point is chosen the same of (5.10) and $\boldsymbol{A}(t)$ matrix is then obtained deriving f_i with respect to the x_i state variables, instead the matrix $\boldsymbol{A}$ by evaluating $\boldsymbol{A}(t)|_{x_i=\overline{x}_i}$. The state function f_4 is:

$$\begin{aligned} f_4 = & -\frac{x_3x_4(\sin x_1 \cos x_1)(\Gamma_{2x} - \Gamma_{2y})}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} - \frac{x_3(\sin x_1 \cos x_1)(\Gamma_{2x} - \Gamma_{2y})}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} \\ & - \frac{f_{v2}x_4}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} + \frac{\tau_2}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1}; \end{aligned} \tag{5.12}$$

Then, $\boldsymbol{A}(t)$ matrix is:

$$\boldsymbol{A}(t) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \tag{5.13}$$

By computing the partial derivative of f_4 with respect to the partial derivative of the x_i state variables:

$$
\begin{aligned}
A_{41} = \frac{\partial f_4}{\partial x_1} &= \frac{-x_3 x_4(\Gamma_{2x} - \Gamma_{2y})(2\cos^2 x_1 - 1)(\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1)}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} + \\
&+ \frac{x_3 x_4(\sin x_1 \cos x_1)(\Gamma_{2x} - \Gamma_{2y})(2\sin x_1 \cos x_1 \Gamma_{2x} - 2\cos x_1 \sin x_1 \Gamma_{2y})}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} - \\
&- \frac{x_3(\Gamma_{2x} - \Gamma_{2y})(2\cos^2 x_1 - 1)(\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1)}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} - \\
&- \frac{x_3(\sin x_1 \cos x_1)(\Gamma_{2x} - \Gamma_{2y})(2\sin x_1 \cos x_1 \Gamma_{2x} - 2\cos x_1 \sin x_1 \Gamma_{2y})}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} - \\
&- \frac{x_4 f_{v2}(2\sin x_1 \cos x_1 \Gamma_{2x} - 2\cos x_1 \sin x_1 \Gamma_{2y})}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} - \frac{\tau_2(2\sin x_1 \cos x_1 \Gamma_{2x} - 2\cos x_1 \sin x_1 \Gamma_{2y})}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} \ ; \\
A_{42} = \frac{\partial f_4}{\partial x_2} &= 0 \ ; \\
A_{43} = \frac{\partial f_4}{\partial x_3} &= \frac{-x_4(\Gamma_{2x} - \Gamma_{2y})(\sin x_1 \cos x_1)}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} - \frac{(\Gamma_{2x} - \Gamma_{2y})(\sin x_1 \cos x_1)}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} \ ; \\
A_{44} = \frac{\partial f_4}{\partial x_4} &= -\frac{x_3(\Gamma_{2x} - \Gamma_{2y})(\sin x_1 \cos x_1)}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} - \frac{f_{v2}(\Gamma_{2x} - \Gamma_{2y})(\sin x_1 \cos x_1)}{\Gamma_{2x}\sin^2 x_1 + \Gamma_{2y}\cos^2 x_1} \ ;
\end{aligned}
\tag{5.14}
$$

Replacing in (5.14) the values of the chosen equilibrium point, the final $\boldsymbol{A}$ matrix is:

$$
\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & A_{44} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{f_{v2}}{\Gamma_{2x}} \end{bmatrix} \tag{5.15}
$$

Always referring to (5.8) and (5.9), the remaining state matrices are extracted:

$$
\boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{\Gamma_{2x}} \end{bmatrix} \ ; \boldsymbol{C} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \ ; \boldsymbol{D} = 0 \tag{5.16}
$$

5.2.3 Linearized model of the Gimbal and its SS description

At this point is possible to collect all the results of the previous section and write down the state-space equations of the Gimbal system, represented by its own linearized model. The next state matrices are used in *Matlab* and *Simulink* environments in order to design a suitable controller for the gimbal mechanical system. The state-space description of the gimbal electromechanical system is given by:

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t) \end{cases}$$

where:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\dfrac{f_{v1}}{\Gamma_{1z}+\Gamma_{2z}} & 0 \\ 0 & 0 & 0 & -\dfrac{f_{v2}}{\Gamma_{2x}} \end{bmatrix} ; \boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ \dfrac{1}{\Gamma_{1z}+\Gamma_{2z}} \\ \dfrac{1}{\Gamma_{2x}} \end{bmatrix} ; \boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ; \boldsymbol{D} = 0$$

The geometric parameters of the overall gimbal structure are acquired by means of the CAD drawing: are listed below and will be needed for the control design purpose.

Parameter	*Symbol*	*Value[SI]*
Azimuth axis mass	m_1	4.87[Kg]
Elevation axis mass	m_2	1.95[Kg]
Azimuth moments of inertia	Γ_{1x}	0.07079363[Kgm^2]
	Γ_{1y}	0.03039091[kgm^2]
	Γ_{1z}	0.0817408[Kgm^2]
Elevation moments of inertia	Γ_{2x}	0.01174741[Kgm^2]
	Γ_{2y}	0.008261759[kgm^2]
	Γ_{2z}	0.01162105[Kgm^2]
Distance to ground	d_3	500[m]
Viscous friction	f_{v1}	0.2
Viscous friction	f_{v2}	0.002

Table 5.1: Geometric parameters of the Gimbal structure

5.3 Actuators linearization

In order to derive the state-space description of the DC motor actuating the rotational motion on azimuth and elevation axis, the dynamic model explained in chapter 4 is used. The state-space description is always represented through the following equations:

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t) \end{cases}$$

In the DC motor model, the new state vector is made of three components, i.e.

$$\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T = \begin{bmatrix} i_a & \omega_m & \theta_m \end{bmatrix}^T \tag{5.17}$$

Instead the input vector consists of two terms:

$$\boldsymbol{u} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^T = \begin{bmatrix} V_s & \tau_l \end{bmatrix}^T \tag{5.18}$$

For the thesis purpose, in which a position control system is developed, the output is assumed to be the θ_m angle $\implies y(t) = \theta_m(t)$. From all the previous clarifications, the system state equations are:

$$\begin{cases} \dot{x}_1 = -\dfrac{R_a}{La}x_1 - \dfrac{K}{L_a}x_2 + \dfrac{1}{La}V_s \\ \dot{x}_2 = \dfrac{K}{J_m}x_1 - \dfrac{B_m}{J_m}x_2 - \dfrac{1}{J_m}\tau \\ \dot{x}_3 = x_2 \end{cases} \tag{5.19}$$

while the output equation is:

$$y = x_3$$

Finally, the state space matrices *A,B,C,D* are obtained as:

$$\boldsymbol{A} = \begin{bmatrix} -\dfrac{R_a}{L_a} & -\dfrac{K}{L_a} & 0 \\ \dfrac{K}{J_m} & -\dfrac{B_m}{J_m} & 0 \\ 0 & 1 & 0 \end{bmatrix} ; \boldsymbol{B} = \begin{bmatrix} \dfrac{1}{L_a} \\ 0 \\ 0 \end{bmatrix} ; \boldsymbol{C} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} ; \boldsymbol{D} = 0$$

The parameters concerning the DC motors are taken from their own datasheets and are listed in the following table:

Parameter	*Symbol*	*Value*[*SI*]
Nominal voltage	V_s	12 [V]
Terminal resistance	R_a	1.46 [Ω]
Rotor inductance	L_a	135 [μH]
Back emf constant	K_b	1.945 [mVmin^{-1}]
Torque constant	K_t	18.57 [mNm/A]
Mass	m	114 [g]
Rotor inertia	J_m	12 [gcm^2]
Speed up to	n_{max}	7000 [min^{-1}]

Table 5.2: Electromechanical parameters of the DC-motors

Chapter 6

Gimbal motion control

In the actual chapter a suitable controller for the gimbal system will be designed, essentially applying two different techniques: the loop-shaping and the Linear Quadratic Integrator (LQI). The starting point to keep in mind is to think the gimbal system as *decoupled*: the rotation around the azimuth axis is independent from the rotation around the elevation axis, due to the fact that there are two separate DC-motors. In other words, each DC-motor generates a torque τ_i which will only affects the corresponding state variables θ_i. This approach to the problem is the so-called *decentralized control.*
If the section rotating around the two axis are independent by each others, then the controller design is dealing with SISO systems (*Single-Input-Single-Output*). The controller works well if the output signal (i.e. the gimbal angular position) follows the reference angle satisfying the given requirements in time domain.
The results of the simulations will be used later, in chapter 9, to conduct a DC motor survey. In fact, regardless of the motors and reducers already present in the Digisky's laboratories, it has been asked if there could be gearmotors on the market that better match the needs of this specific application.

6.1 Premises and Requirements of the control design

As common starting point, the state matrices obtained in the previous chapter, are copied in the *Matlab* environment, through which the control design is performed. Then, two different control systems are developed, the first one to obtain a control action regulating the behavior around the azimtuh axis, while the second one to govern the elevation rotation by acting on the other DC-motor.
Defined the A, B, C, D state matrices and all the needed geometric parameters, by means of the Matlab command

```
>> sys_2axis=ss(A,B,C,D);
```

the gimbal continuous time transfer function is created.
Later, in order to work with two decoupled systems, the transfer functions of each system are extracted, via the line code:

```
>> [NUM1,DEN1]=ss2tf(A,B,C,D,1);
SYS_Z=zpk(minreal([sys11;sys12;sys13;sys14]));

>>[NUM2,DEN2]=ss2tf(A,B,C,D,2);
SYS_Y=zpk(minreal([sys21;sys22;sys23;sys24]));
```

One would expect that the typical transfer functions of a rotating system are of the form:

$$\begin{cases} \dfrac{\theta_i}{\tau_i} = \dfrac{k}{s(s+p_1)} \implies y(t) = \theta_i \\ \dfrac{\dot{\theta}_i}{\tau_i} = \dfrac{k}{s+p_1} \implies y(t) = \dot{\theta}_i \end{cases} \tag{6.1}$$

Effectively, from Matlab computations, appears that the two systems are rotating and decoupled:

```
>> SYS_Z =

           10.711
  1:   -----------
       s (s+2.142)

  2:   0

        10.711
  3:   ---------
       (s+2.142)

  4:   0

 >>SYS_Y =

  1:   0

         85.125
  2:   ------------
       s (s+0.1703)

  3:   0

        85.125
  4:   ----------
       (s+0.1703)
```

Note that *system z* and *system y* have transfer functions consistent with (6.1). Moreover, if torque τ_1 acts, it only affects the state variables x_1 and x_3; on the contrary if torque τ_2 is present, the state variables x_2 and x_4 are influenced. Remembering the definition of the state vector

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T = \begin{bmatrix} \theta_1 & \theta_2 & \dot{\theta}_1 & \dot{\theta}_2 \end{bmatrix}^T$$

means that the two systems are really decoupled, then the idea of two difference control systems makes sense.

Then is necessary to clarify that we're dealing with a position control, then the variables to control (i.e. the outputs of the control system) are the angular positions of the decoupled gimbal system:

$$\begin{cases} y_1 = \theta_1 \implies \text{control action on azimuth axis DC-motor} \\ y_2 = \theta_2 \implies \text{control action on elevation axis DC-motor} \end{cases}$$

Then the requirements concerning the dynamics and the statics of the control system, given by the client, are collected and listed below:

- *dynamics* of the control system: characterize the dynamics of the control system means define the performances during the transient
    - absence of overshoot $\implies \widehat{s} = 0\%$;
    - rise time: $t_r \leq 5s$;
- *statics* of the control systems: it deals with the performances of the control system at the end of the transient, then at steady state condition
    - of greater interest is the value of error between the reference signal and the output of the system at steady-state: $\implies |e_r^\infty| = 0$ if the reference signal is a step ($r(t) = \varepsilon(t)$).

In the next sections the Loop-Shaping and LQI designs will be explained. While in the Loop-Shaping approach the time domain requirements above will be translated into frequency domain constraints, in the LQI controller design the given requirements will be reached by conveniently tuning the control law.

6.2 Loop-Shaping Design

The basic idea of the Loop-Shaping control technique is to opportunely design the controller of a plant, so that the given requirements are satisfied.

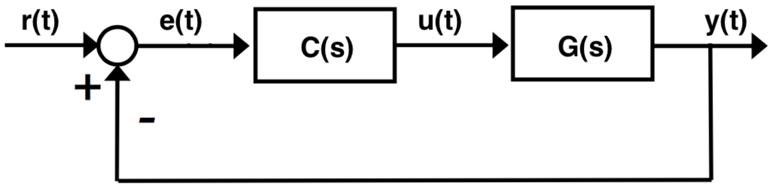


Figure 6.1: Control system reference schem

In the loop-shaping approach we're interested to design the controller, in order to obtain an open loop transfer function $G_a(s) = C(s)G(s)$ having the needed features to satisfy the given requirements. In fact, the *'loop-shaping'* expression alludes to choice of $C(s)$ such that G_a assumes the desired shape to fulfill the specifications.

6.2.1 Structure of the controller

The analysis of the well known specifics allows to find some constraints on the $C(s)$ transfer function, as said in the general introduction of the motion control problem. The LTI controller of the gimbal structure is planned starting from the general definition:

$$C(s) = \frac{k_c}{s^\nu} \prod_i \left(\frac{1 + \dfrac{s}{z_{di}}}{1 + \dfrac{s}{m_{di} z_{di}}} \right) \prod_j \left(\frac{1 + \dfrac{s}{m_{ij} p_{ij}}}{1 + \dfrac{s}{p_{ij}}} \right) \tag{6.2}$$

So, the given requirements are translated into constraints on the controller, i.e.:

- the static specifications allow to identify restrictions regarding the controller gain k_c and the ν poles in s=0. The steady-state gain of $C(s)$ is defined as

$$k_c = \lim_{s \to 0} s^\nu C(s)$$

  Also the plant has p poles in s=0 and its steady-state gain is instead

$$k_p = \lim_{s \to 0} s^p G(s)$$

  $\Longrightarrow$ $\dfrac{k_c}{s^\nu}$ is said to be the *static part* of the controller;

- the transient time domain requirements are used to design the remaining part of the controller expression

$$\Longrightarrow \prod_i \left(\frac{1+\dfrac{s}{z_{di}}}{1+\dfrac{s}{m_{di}z_{di}}} \right) \prod_j \left(\frac{1+\dfrac{s}{m_{ij}p_{ij}}}{1+\dfrac{s}{p_{ij}}} \right)$$

6.2.2 Requirements translation

In order to translate the given requirements into constraints on the $C(s)$ transfer function shape, the 2^{nd} order prototype model is considered, through which is possible to explicit the desired constraints in terms of *dumping (ζ)* and *normal frequency (ω_n)* of a system. The dynamic behavior of a 2^{nd} order prototype system can be expressed as:

$$\frac{w_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

The requirements, known in time response domain, must be translated in frequency response domain to be used in the controller design. From definitions, concerning the frequency response of the system, the following entities should be evaluated:

$$\begin{cases} \zeta = \dfrac{|\ln(\hat{s})|}{\sqrt{\pi^2 + \ln^2(\hat{s})}} \\ T_p = \dfrac{1}{2\zeta\sqrt{1-\zeta^2}} \\ S_p = \dfrac{2\zeta\sqrt{2+4\zeta^2+2\sqrt{1+8\zeta^2}}}{\sqrt{1+8\zeta^2}+4\zeta^2-1} \end{cases} \tag{6.3}$$

Due to the definitions in (6.3), the dumping coefficient is strictly dependent from the maximum overshoot requirements. Once defined ζ, the constraints on the sensitivity function peak (S_p) and on the complementary sensitivity function peak (T_p) can be reached, and it is possible to demonstrate that S_p and T_p are related to the *gain margin* and *phase margin*. Then:

$$\hat{s} = 0\% \implies \begin{cases} \zeta \geq 0.7 \\ T_p \leq 1.002 \\ S_p \leq 1.276 \end{cases}$$

In detail, the values of the peak functions allow to define the constant magnitude loci, on the Nichols plane, in which the controller is designed through the loop-shaping method. In other words, they represent the limits that cannot be exceed by the Nichols plot of the frequency response of the loop function $L(j\omega)$.
On the other side, from the rise time and settling time specifications, the bounds dealing

with the normal frequency (and consequently with the crossover frequency) are available. From definitions:

$$\begin{cases} t_r = \dfrac{\pi - \arccos\zeta}{\omega_n\sqrt{1-\zeta^2}} \\ \omega_c = \omega_n\sqrt{\sqrt{1+4\zeta^4}-2\zeta^2} \end{cases} \tag{6.4}$$

$$\begin{cases} t_{s,\alpha\%} = \dfrac{\ln\alpha}{\omega_n\zeta} \\ \omega_c = \omega_n\sqrt{\sqrt{1+4\zeta^4}-2\zeta^2} \end{cases} \tag{6.5}$$

From(6.4) and (6.5) two different constraints on the *crossover frequency* exist: actually, the constraint on ω_c is chosen as:

$$\omega_c = \max(\omega_{c,ts}, \omega_{c,tr})$$

Then:

$$\begin{cases} t_r \le 5s \\ t_{s,2\%} \le 4s \end{cases} \Longrightarrow \begin{cases} \omega_{n,tr} = 2.863rad/s \\ \omega_{c,tr} = 1.855rad/s \\ \omega_{n,ts} = 1.117rad/s \\ \omega_{c,ts} = 0.724rad/s \end{cases} \Longrightarrow \omega_{c,des} \simeq 0.724rad/s$$

6.2.3 Design procedure

The controller design is completed if three steps are followed:

1. The steady-state and the transient requirements are converted from time domain into frequency domain, applying the relations illustrated in the previous section;
2. The controller is planned by means of the loop-shaping design: it means chose a suitable number of controller poles at the origin (ν), set the controller gain value (k_c) and draw on the Nichols plane the frequency response of the loop function. Then ;
3. Make sure that all the time domain requirements are satisfied; if not, return to step 2 and optimize the controller design until the given requirements are not fulfill.

6.2.4 Azimuth rotation controller

Once translated the time domain specifications into frequency domain constraint, the controller design can be performed. The plant transfer function is:

$$G_p(s) = \frac{10.71}{s(s+2.142)} \tag{6.6}$$

As is clearly shown by (6.6), the plant has one pole in $s = 0$, then if p is the number of G_p poles in zero $\implies p = 1$.
From now on, each time domain requirement influences the controller design.

- $|e_r^\infty| = 0$ **if** $r(t) = \varepsilon(t)$
  In order to guarantee a null tracking error at steady-state when the input signal is a step type, the *final value theorem* must be applied. The provided definition of the theorem is:

$$\begin{aligned} e_r^\infty = \lim_{t\to\infty} e_r(t) = \lim_{s\to 0} s \cdot e_r(s) = \\ = \lim_{s\to 0} s \cdot e_r(s) = \frac{s^{\nu+p}K_d^2}{s^{\nu+p}K_d + K_cK_pG_a}\frac{R_0}{s^{h+1}} \end{aligned} \tag{6.7}$$

  where:

$$\begin{cases} h = 0, \quad p = 1, \quad \nu = 0 \\ K_d = 1 \\ K_c \quad \text{is the steady-state controller gain,} \quad K_p \quad \text{is the steady-state plant gain} \\ R_0 = 1 \quad \text{is the step amplitude of the reference signal} \end{cases}$$

  The time domain limit exists and is bounded when $\nu + p \geq h$ (where h is the reference order). Being the reference signal a step, $h = 0$. Thanks to the final value theorem the needed number of the controller poles at zero (ν) is evaluated, to provide the requirements fulfillment. Summarizing, until now:

$$p = 1, \quad \nu = 0, \quad h = 0$$

  Replacing all the known values, and completing the limit computation in (6.15), can be demonstrate that no poles at zero are needed in the $C(s)$ transfer function $\implies \nu = 0$, accordingly to the result concerning the system type and zero tracking error:
  The tracking error of an LTI feedback control system is guaranteed to be null, at steady-state, only if the system type $\nu + p$ is greater than the reference input signal order.
  Moreover, since the limit is null no constraint on controller gain k_c is obtained $\implies k_c = 1$.

- $\hat{s} = 0\% \implies \zeta \geq 0.7$, $T_p \leq 1.002$, $S_p \leq 1.276$

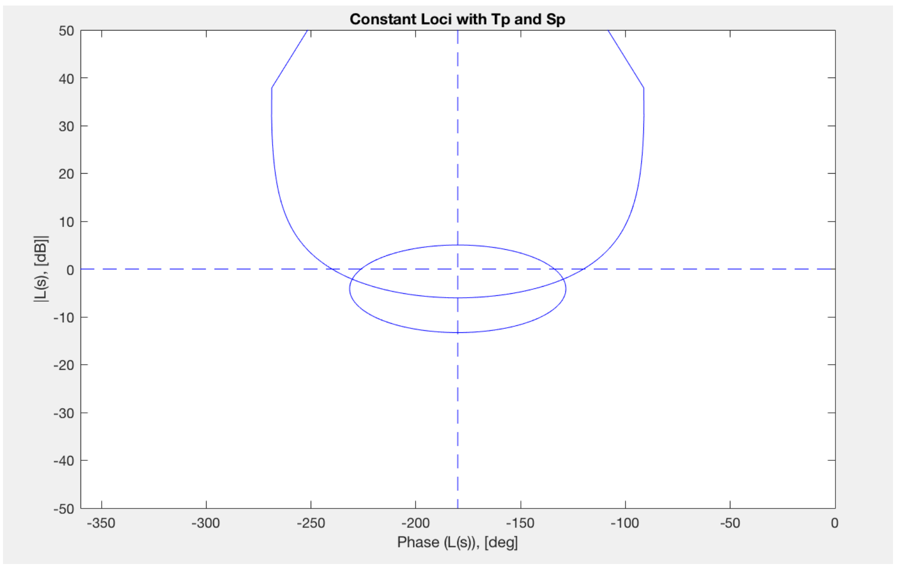


Figure 6.2: Constant loci Tp and Sp

As said before, the constant loci curves are dealing with the phase and gain margins: the gain margin indicates how much the gain can increase without causing the instability of the closed loop system; analogously the phase margin should not be exceeded to not have an unstable response. In other terms, during the controller design phase, one should avoid the intersection between the constant loci curves and the Nichols plot of the frequency response of the loop function $L(j\omega)$.

- unitl now, we're dealing with a controller having $K_c = 1$ and $\nu = 0$, then the static part of the controller is:

$$C(s) = \frac{K_c}{s^\nu} = 1 \implies L(j\omega) = G_p \quad \text{at the beginning of the design phase}$$

The dynamic part of $C(s)$ remains to be designed, by means of the **compensation networks**: they are the so-called *lead and lag networks*, already mentioned in the definition of the controller general structure (6.2). Via the following Matlab line code, the constant loci curves are overlapped to the Nichols plot of the frequency response of the loop function $L(j\omega)$, in order to evaluate if lead or lag actions are necessaries:

```
>> figure, myngridst(Tp0,Sp0), hold on, nichols(L)
```

The result is shown in the next figure:

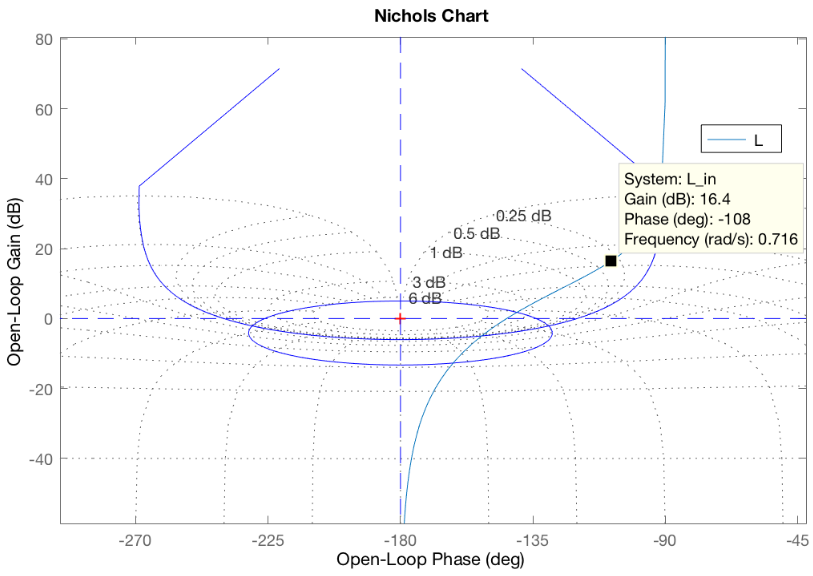


Figure 6.3: Frequency response of $L(j\omega)$ before design phase

Observing the figure above and analyzing the datas provided by Matlab, it's possible to conclude that a lead action is required. In fact, in order to fulfill the given requirements, the system needs two essential actions:

1. *lead action*: the point corresponding to the $\omega_{c,des}$ frequency must be located outside the constant loci curves and this causes a phase lead. The loop function $L(j\omega)$ should be subjected to a phase lead equal to $(-108° + 90°) \simeq 18°$;

2. *magnitude attenuation*: $L(j\omega)$ has to cut the $0dB$ axis at the $\omega_{c,des}$ frequency. The magnitude of the loop function should decrease of $\simeq 16dB$;

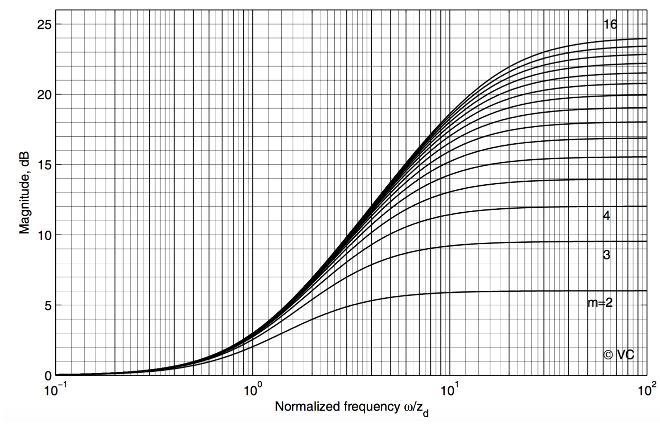
However, first the lead action is performed in the design of a controller through the loop-shaping techniques: after this step, again the Nichols plot of the new $L(j\omega)$ is observed such that the frequency response can be improved by means of a lag action (or magnitude actuation).

Lead network

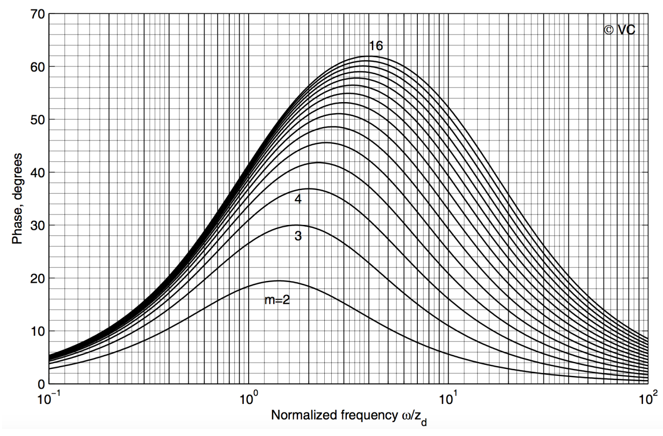
The lead network is added to the $C(s)$ controller expression. It is defined as:

$$R_d(s) = \frac{1 + \dfrac{s}{z_d}}{1 + \dfrac{s}{m_d z_d}} \tag{6.8}$$

From (6.8) is clear that the network introduces a real zero at $s = -z_d$ and a real pole at $s = -m_d z_d$. Then the lead network has to be designed, by make an appropriate choice of z_d and m_d: to perform this is useful to refer to universal diagrams of magnitude and phase of the compensation networks, shown in the following figure.



(a) Magnitude diagram



(b) Phase diagram

Figure 6.4: Magnitude and Phase universal diagrams

Referring to 7.12:

- m_d is chosen accordingly to the requested phase margin in $\omega_{c,des}$;
- z_d is obtained by computing

$$z_d = \frac{\omega_{c,des}}{\omega_n}$$

where ω_n is the chosen normalized frequency such that the phase lead happens exactly at the frequency $\omega_{c,des}$. In the gimbal controller design, are selected:

$$\begin{cases} m_d = 16 \\ \omega_n = 0.25 \end{cases} \implies z_d = 2.89$$

According to (6.8):

$$R_d(s) = \frac{49.27s + 134.4}{2.898s + 134.4} \tag{6.9}$$

While the new controller expression is:

$$C(s) = R_d(s)$$

Instead the overall $L(j\omega)$ loop function becomes:

$$L(s) = G_p(s)C(s) \tag{6.10}$$

If the Nichols plot of the new loop function is plotted on Matlab, the result is:

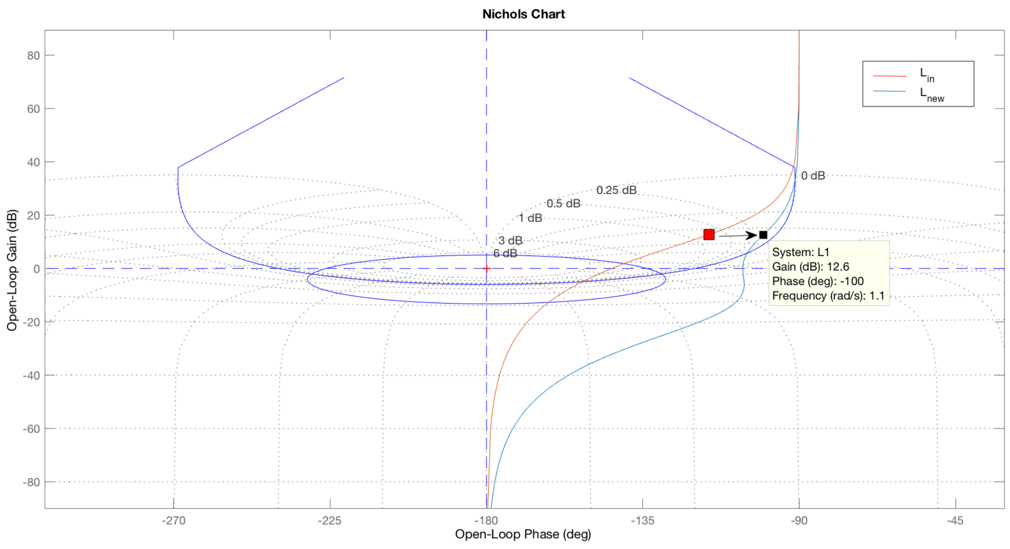


Figure 6.5: $L(j\omega)$ Nichols plot after lead action

Magnitude attenuation

From 6.5 appears the necessity of a magnitude action, due to the desire of intersect the $0dB$ axis at the frequency $\omega = \omega_{c,des}$. Then the magnitude value of the $L(j\omega)$ loop function is computed at $\omega_{c,des}$; the new controller gain is evaluated and the final loop function $L_{final}(j\omega)$ is obtained, as exploited below through the Matlab commands:

```
>> [m,p]=bode(L1,wc_d)
>> kc=(1/m)
>> L_final=kc*L1
```

With regard to the assumptions above, the new expression of the controller $C(s)$ is:

$$C_{final}(s) = k_c R_d(s) \tag{6.11}$$

Consequently, the final expression of the loop function will be:

$$L_{final}(s) = G_p(s) k_c R_d(s) \tag{6.12}$$

and its frequency domain response will be evaluated, by means of the Nichols plot, below:

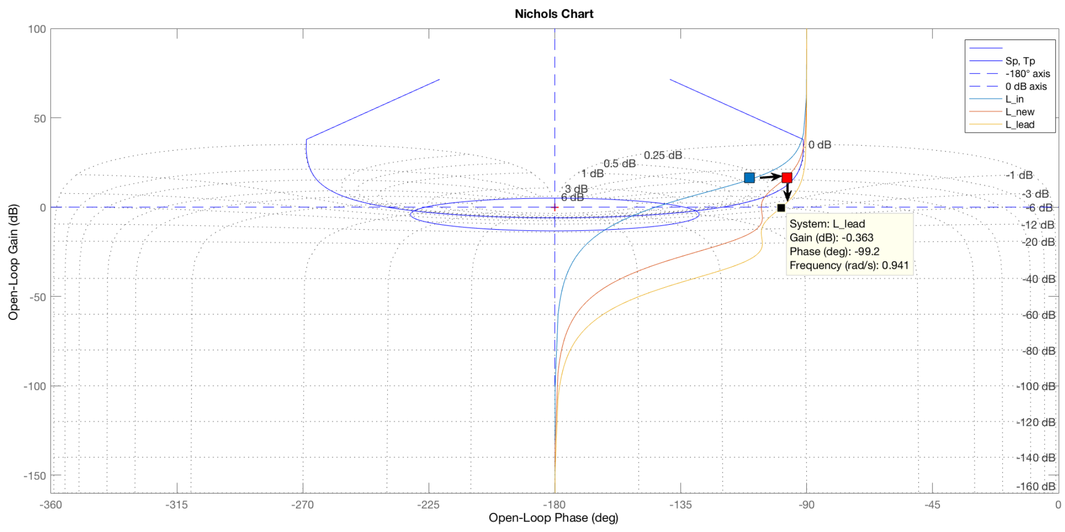


Figure 6.6: $L(j\omega)$ Nichols plot at the end of the design

Once the requirements on Nichols plot are satisfied, i.e. the loop function doesn't intercept the constant magnitude loci curves, the closed loop function behavior is simulated, when the reference command coincides with a step. The *closed loop function* of a control system is usually defined as:

$$W(s) = \frac{L(s)}{1 + L(s)} = \frac{G_p(s)C(s)}{1 + G_p(s)C(s)} \tag{6.13}$$

By means of $W(s)$ the validation of the given requirements is allowed. During the check phase some dynamics specification is not satisfied, the controller $C(s)$ or its gain k_c should be conveniently improved. Thanks to the Matlab command

```
>> figure, step(W)
```

the closed loop response of the system is obatined, in time domain, as illustrated below:

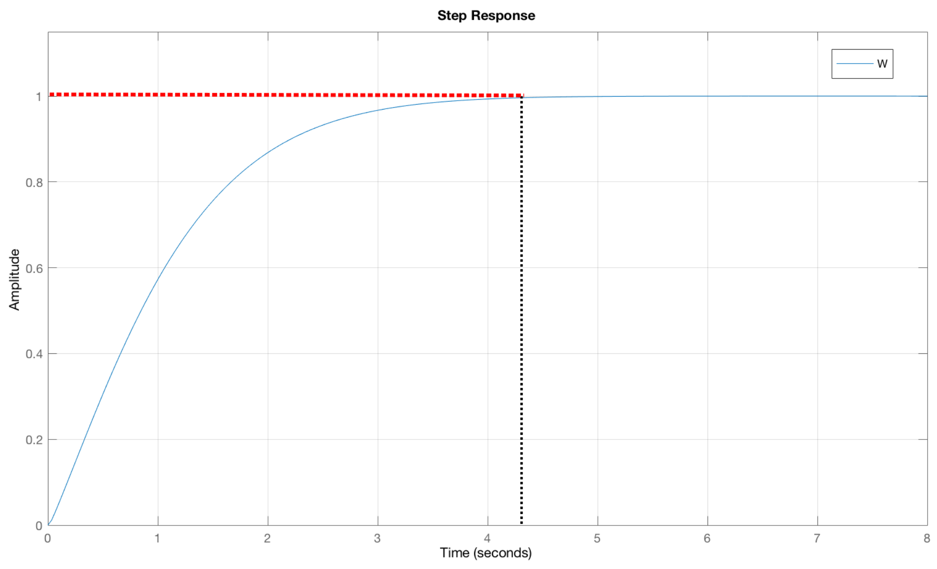


Figure 6.7: Closed loop function time response

Note from 6.7 the fulfillment of the given specifications in both time and frequency domain:

- $|e_r^\infty| = 0$ when the reference $r(t) = \varepsilon(t)$ with amplitude $R_0 = 1$;
- the time response never exceed the command reference value $\implies \hat{s} = 0\%$;
- both t_r and $t_{s,2\%}$ are sufficiently lower than the imposed maximum values.

6.2.5 Elevation rotation controller

As done for the azimuth rotation controller design also for the 2^{nd} DC-motor controller the loop-shaping method is applied. The requirements to be satisfied are the same of the previous design and also the followed logic is the same.
The plant is obviously different from the elevation case one, and it is:

$$G_p(s) = \frac{85.125}{s(s+0.1703)} \tag{6.14}$$

Again, the plant as one pole in $s = -0.1703 \implies p = 1$. As known, each requirement allows to extract a constraint in the controller design phase:

- $|e_r^\infty| = 0$ **if** $r(t) = \varepsilon(t)$: the final value theorem is applied:

$$\begin{aligned} e_r^\infty &= \lim_{t\to\infty} e_r(t) = \lim_{s\to 0} s \cdot e_r(s) = \\ &= \lim_{s\to 0} s \cdot e_r(s) = \frac{s^{\nu+p} K_d^2}{s^{\nu+p} K_d + K_c K_p G_a} \frac{R_0}{s^{h+1}} \end{aligned} \tag{6.15}$$

one should deduce that no pole at $s = 0$ is needed when the $C(s)$ expression is defined. Then:

$$p = 1, \quad \nu = 0, \quad h = 0$$

Besides no constraint on the controller gain is obtained, so $k_c = 1$;

- $\hat{s} = 0\% \implies \zeta \geq 0.7$, $T_p \leq 1.002$, $S_p \leq 1.276$.

Once clarified the starting point for the controller design, the Nichols plots of the loop function $L(j\omega)$ will be examined with reference to the constant loci curves of figure 6.2, and it is modified each time in order to achieve a response for which the shape of $L(j\omega)$ doesn't cross the T_{p0} and S_{p0} curves.
First of all the Nichols plot of $L(j\omega)$ is required:

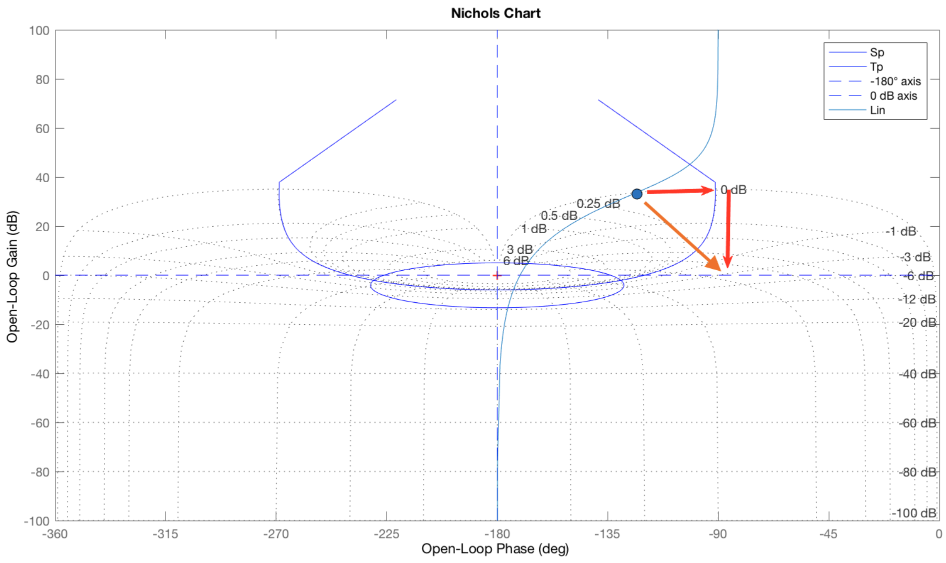


Figure 6.8: $L(s)$ Nichols plot, before design

Note that, again, two compensation actions are needed:

- add a lead network;
- adjust the loop function shape by means of magnitude attenuation.

Lead network

Adding a lead network $R_d(s)$, the highlighted point in 6.8 will shifts toward right with respect the old position. This means that the point corresponding to the frequency $\omega = \omega_{c,des}$, after the design of the $R_d(s)$ network will be located outside the constant loci curves. As always,

$$R_d(s) = \frac{1 + \frac{s}{z_d}}{1 + \frac{s}{m_d z_d}}$$

Then, chosen:

$$\begin{cases} m_d = 16 \\ \omega_n = 0.5 \end{cases} \implies z_d = 1.44$$

$$\implies R_d(s) = \frac{24.63s + 33.59}{1.449s + 33.59}$$

An additional clarification concerning the controller expression is needed. Since the plant has no unstable poles, a stable zero-pole cancellation is allowed when the product $G_p(s)C(s)$ is performed: the low frequency pole is neglected, and it is replaced with a pole having a frequency near the desired one. Then an initial expression of the controller is:

$$C_{in}(s) = \frac{s + 0.1703}{s + 1.5}$$

After the lead network design, by means of suitable choice related to m_d and z_d, the controller becomes:

$$C_2(s) = \frac{s + 0.1703}{s + 1.5} \cdot \frac{24.63s + 33.59}{1.449s + 33.59} \implies L_2(s) = G_p(s)C_2(s)$$

At the end of this first design phase, one expected the desired crossover frequency point outside the constant loci magnitude curves, but not jet crossing the $0dB$ axis (due to the magnitude attenuation not jet actuated). The result is illustrated in the following picture:

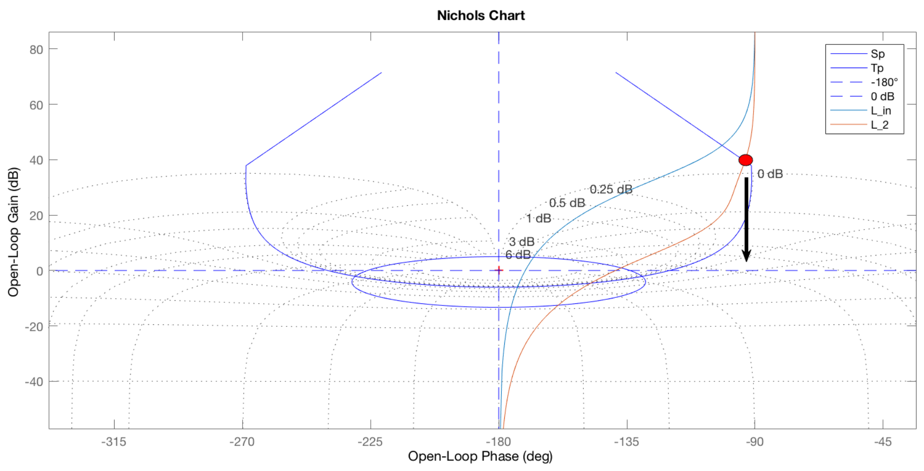


Figure 6.9: $L(s)$ Nichols plot, after lead design

Magnitude attenuation

From 6.9 appears the necessity of a magnitude attenuation action: more precisely an attenuation of $40dB$ is required. Through the Matlab commands exploited in the previous section, the magnitude of $L(j\omega)$ is evaluated in $\omega = \omega_{c,des}$; then a new controller gain is obtained and consequently a new loop function. The final result is the next one:

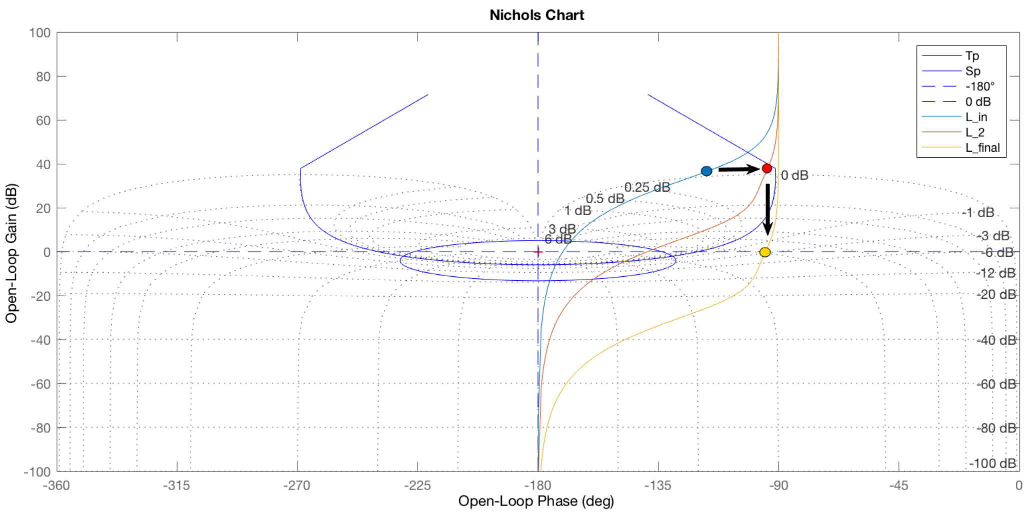


Figure 6.10: $L(s)$ Nichols plot, after controller design

Since the Nichols plot requirements are satisfied, the behavior in time domain of the closed loop function $W(s)$ can be tested. As done for the motion control on the azimuth axis, again the closed loop response is illustrated, when the reference command is the step one. Also this time, the designed controller seems to work very well:

- $\hat{s} = 0\%$ as requested;
- $t_r \leq 5s$ from specifications and $t_r \simeq 4.8s$;
- $t_{s,2\%} \leq 4s$ and it is $t_{s,2\%} \simeq 3.6s$

The listed values can be proved from the next figure.

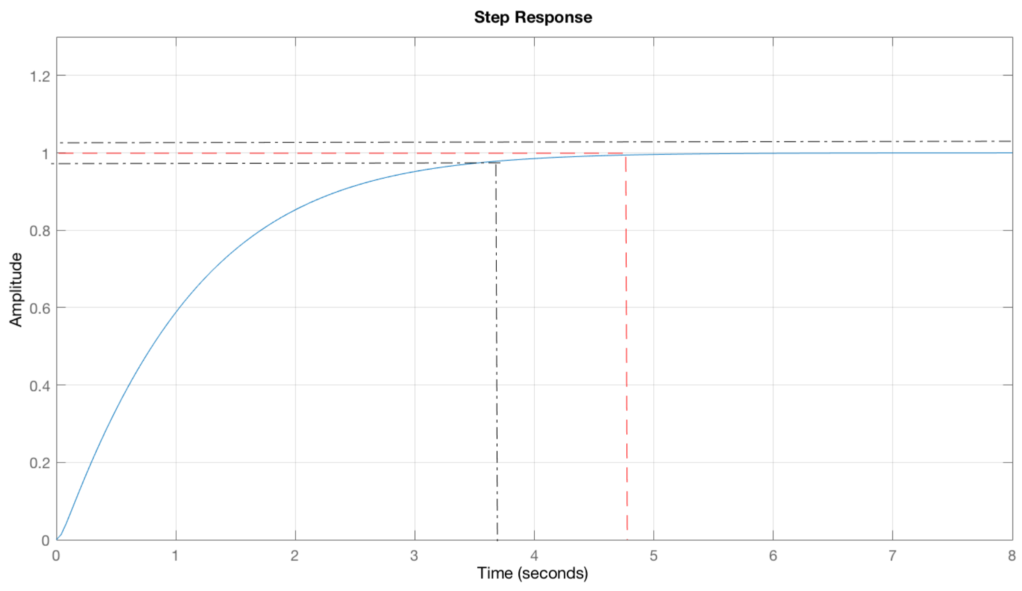


Figure 6.11: $W(s)$ step response

In *Appendix A* are supplied the Matlab scripts concerning the design of the controllers $C(s)$ of both azimuth and elevation rotations.

6.2.6 Simulation results

Once individually simulated the controllers of the motion around the elevation and azimuth axis, the unique control system scheme on *Simulink* is created and tested.

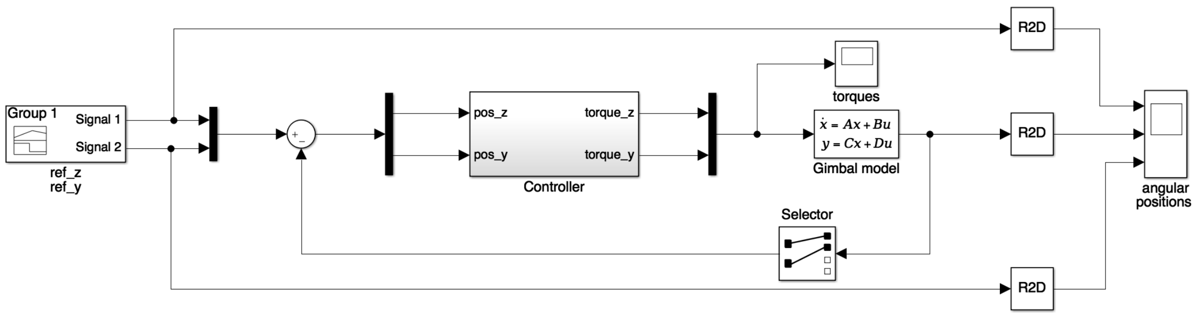
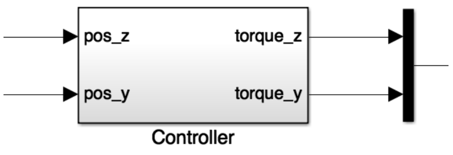


Figure 6.12: Overall control system scheme

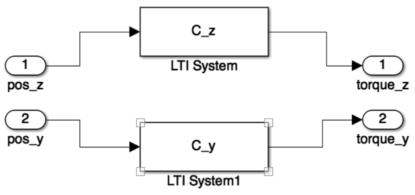
The scheme in 6.12 is briefly elucidated below:

- the *Group 1* block generates the reference angles for both the axis rotation;

- the *Controller* block gets as inputs the reference angles and provides as outputs the torques required to move the gimbal system in order to follow the command inputs, accordingly to the dynamic equations of motion. Inside the block, can be found the two controllers, singularly designed in the previous sections.



(a) Black box



(b) Inside the block

Figure 6.13: *Controller* block detail

- the *Radiants to Degrees* block performs the conversion from the angles expressed in radiant measure of unit, into the degree values for the same angles;
- as outputs of the control system are observed the two angular positions of the gimbal system, i.e. the variables to effectively control.

The control system of the gimbal is simulated, where as input are given:

$$\theta_1 = x_1 = 3.18\frac{rad}{s} \implies 180^\circ \quad , \quad \theta_2 = x_2 = 1\frac{rad}{s} \implies 57^\circ$$

and the results are shown in the next figures.

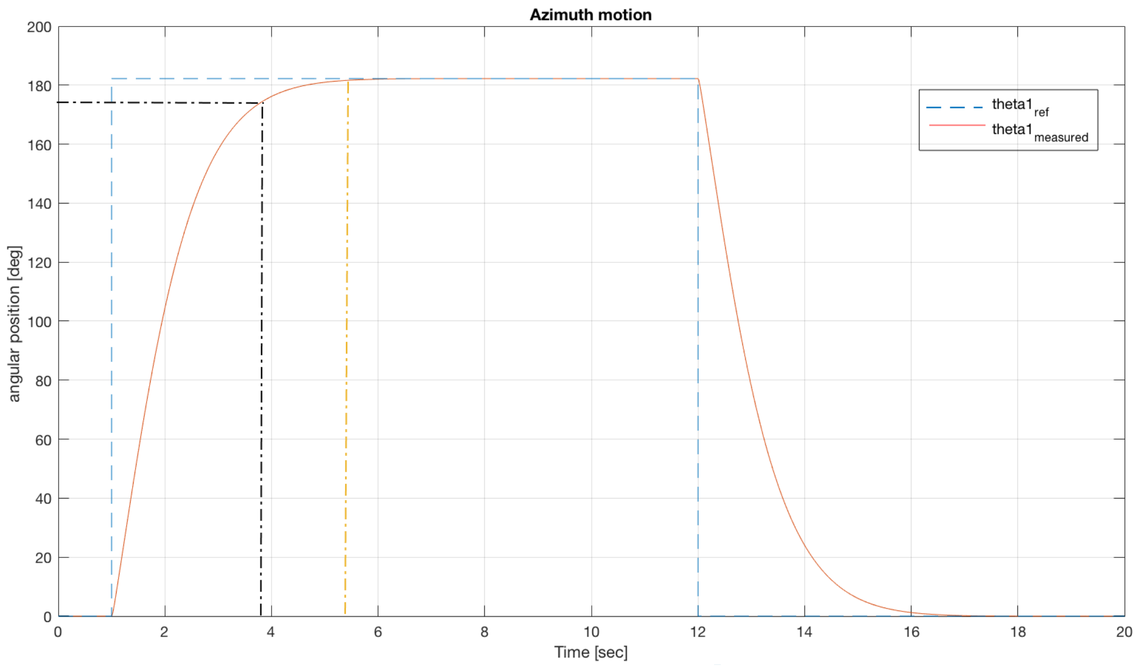


Figure 6.14: *Measured* and *reference* angular positions on azimuth axis

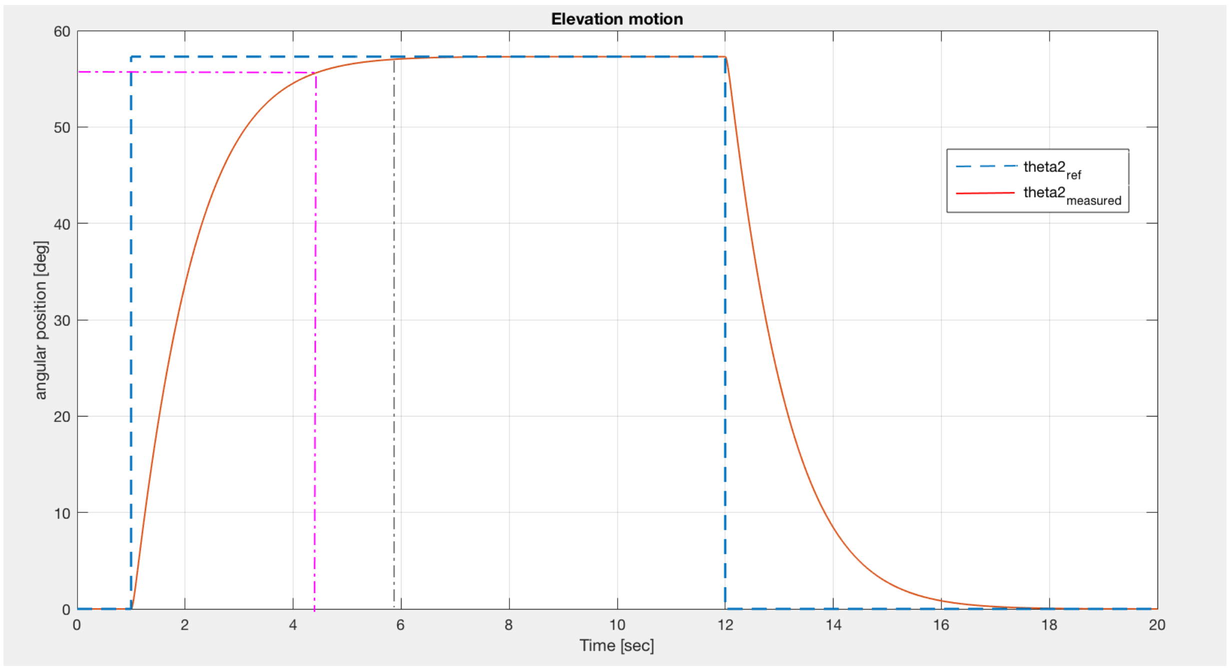


Figure 6.15: *Measured* and *reference* angular positions on elevation axis

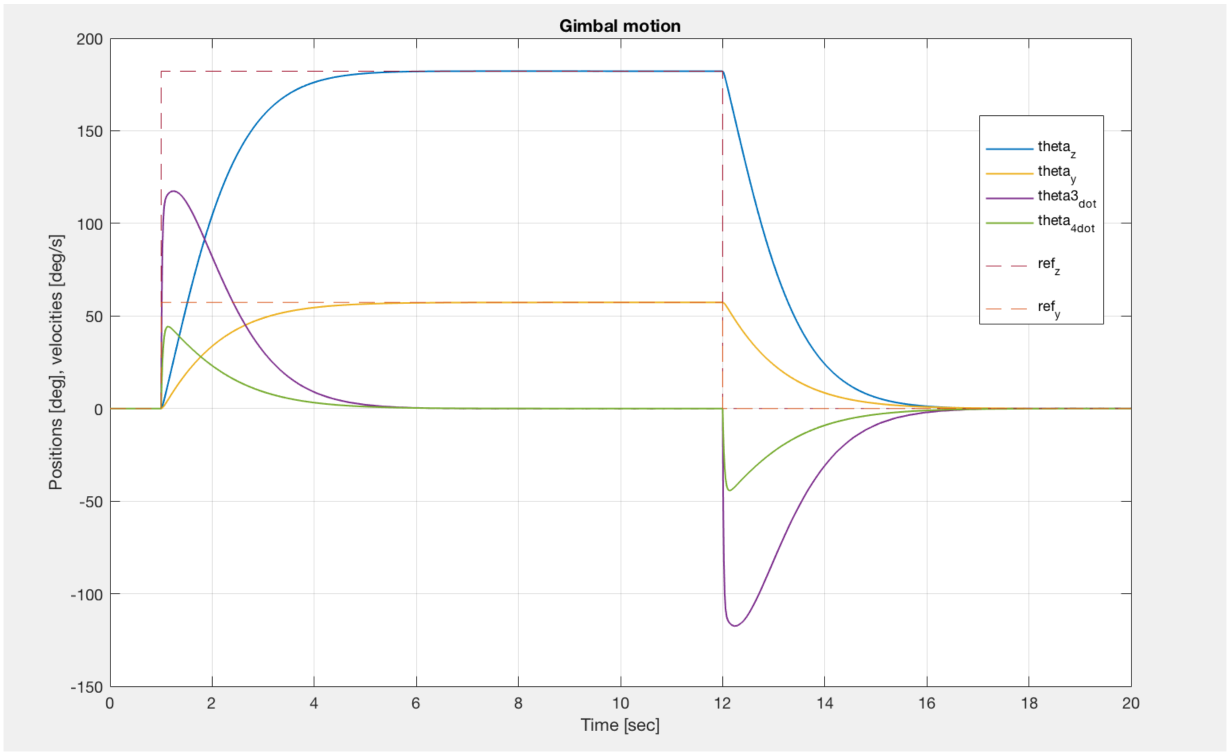


Figure 6.16: *Measured* and *reference* angular positions and velocities of the overall gimbal system

As expected, from the previous figures, all the given requirements are fulfilled:

	$\hat{s}$	$t_r[sec]$	$t_{s,2\%}[sec]$	$\lvert e_r^\infty \rvert$
required	0%	≤ 5 s	≤ 4 s	0
azimuth axis	0%	4.6 s	2.8 s	0
elevation axis	0%	4.6 s	3.1 s	0

Table 6.1: Results of the Loop-shaping design

then, the loop-shaping design has been completed.

6.3 Linear-Quadratic Regulator Desing

Differently from the Loop-shaping approach, the *Linear-Quadratic Regulator* technique is not based on the closed loop pole placement. In fact, the LQR controller is obtained by means of minimization of the cost function $J(x,u)$ identifying the suitable tradeoff between the weight of the system states and the input variables. It is one of the optimal controls and the cost function to be optimized is expressed through the quadratic form, as below:

$$J = \int_{t=0}^{t_f} \big(x^T(\tau)Qx(\tau) + u^T(\tau)Ru(\tau)\big)d\tau \tag{6.16}$$

Then the problem is to find a suitable input $u(t)$ which is the solution of the cost function (6.16):

$$\min_{u(t),t\in[0,t_f]} J(u) = \min_{u(t),t\in[0,t_f]} x^T(t_f)Su(t_f) + \int_0^{t_f} \big(x^T(\tau)Qx(\tau) + u^T(\tau)Ru(\tau)\big)d\tau$$

where

$$Q, S \in \mathbb{R}^{n,n} : Q = Q^T \geq 0, \quad S = S^T \geq 0, \quad R \in \mathbb{R}^{p,p} : R = R^T > 0$$

The Q matrix is defined as semi-positive diagonal matrix, while the R matrix is defined as positive one: for each Q and R matrices always exists a $u_{ott}(t)$ which minimizes the $J(x,u)$ cost function. Moreover the *finite horizon Linear Quadratic* control problem is of little interest, due to the fact that a *time-variant* controller is obtained.

Infinite horizon LQ problem

If $t_f \to \infty$ the cost function looses the final cost term $x^T(t_f)Su(t_f)$ and becomes:

$$\min_{u(t),t\in[0,t_f]} J(u) = \min_{u(t),t\in[0,t_f]} \int_0^{t_f} \big(x^T(\tau)Qx(\tau) + u^T(\tau)Ru(\tau)\big)d\tau$$

where Q and R are the *design parameters* of the LQ controller. As always:

$$Q = Q^T \geq 0, \quad R = R^T > 0$$

The control law of the LQ controller is defined as:

$$u(t) = -Kx(t) \tag{6.17}$$

where $u(t)$ represents the solution to the infinite horizon LQ control problem, concerning the static state feedback architecture, shown below:

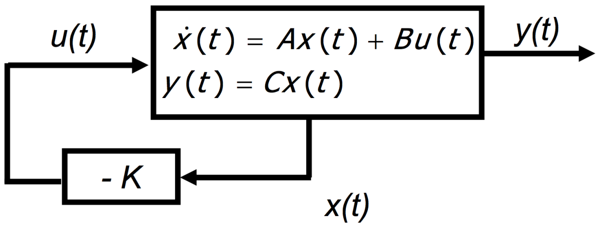


Figure 6.17: Static state feedback control architecture

In detail, K is a matrix, from definition:

$$K = R^{-1}B^TP \in \mathbb{R}^{p,n} \tag{6.18}$$

and P is provided by the solution of the *Algebraic Riccati Equation (ARE)*:

$$\begin{aligned} &Q - PBR^{-1}B^TP^T + PA + A^TP = 0 \\ &P = P^T > 0 \end{aligned} \tag{6.19}$$

System Reachability

By the evaluation of the M_r matrix, the reachability of the dynamic system can be stated. The reachability matrix is defined as:

$$M_r = \begin{bmatrix} B & AB & \dots & A^{n-1}B \end{bmatrix}$$

If $\rho(M_r) = n$, being n the number of the system states, then the system is said to be *reachable*, and the matrix couple (A, B) is reachable. The Matlab commands to verify the reachability are:

```
>> Mr=crtb(A,B);
>> rho_Mr=rank(Mr);
```

Obviously the dynamic system to refer with is:

$$\dot{x}(t) = Ax(t) + Bx(t)$$

System Observability

Considering a different matrix, M_o, defined as:

$$M_o = \begin{bmatrix} C & CA & \dots & CA^{n-1} \end{bmatrix}^T$$

if

$$\rho(M_o) = n$$

then the rank of M_o matrix is the maximum one and the dynamic system is said to be *observable* (i.e. the couple (A, C) is observable). The observability of the system can be checked through the following Matlab commands:

```
>> Mo=obsv(A,C);
>> rho_Mo=rank(Mo);
```

6.3.1 Tuning of LQ control law

In order to obtain a Linear Quadratic Regulator some mandatory steps must be followed:

1. Derive the state-space representation of the system to control, as done in *chapter 5*;
2. Tune the LQ regulator, i.e. choose the values of each term of the weight matrices Q and R. The weight matrices are diagonal, then if the system has n states and p inputs, $(n + p)$ parameters should be chosen:
   a) at the first iteration, the user usually choose the elements of Q and R (q_{ij} and r_{ij} respectively) such that the state variables and the control variables have the same order of magnitude into the cost function;
   b) from the 2nd iteration on, the single values of q_{ij} and r_{ij} are conveniently changed in order to obtain the required control system performances. Do not forget that the coefficients to chose coincide with the relative weight of a variables with respect to the others.
3. Guarantee $q_{ij} \geq 0$ and $r_{ij} > 0$, bearing in mind that their values are set according to the relative importance of each state or control variables.

A clarification about the matrices Q and R are needed: the Q matrix deals with the system states; the R matrix instead concerns the weight of the control input inside the cost function. Then:

- if $R >> Q$ the greater weight in the cost function is represented by the control effort $u(t)$: an expansive control law solution is reached:
- if $Q >> R$ the cost function J is weighted by the state errors, then the result will be a faster system response.

As said, the design of an LQ controller consists of a tradeoff between control input and state variables. The values of the q_{ij}, r_{ij} coefficients also depend on the limitations concerning the dynamic system to control. In fact, if having large control input signal is not dangerous in the analyzed system, then small value of R can be imposed; on the other side, if having large $u(t)$ signal causes sensor noises or actuator saturation, a large value of R should be forced.
Once defined the weight matrices of the LQ regulator, the K matrix is computed by using the following command on Matlab:

```
>> K=lqr(A,B,Q,R);
```

where:

- A and B are the state-space description matrices;
- Q and R are the LQ matrices, created as previously illustrated.

6.3.2 Matlab implementation

In order to control the gimbal dynamic system, the LQ regulator technique is adopted; the Matlab script can be found in *Appendix B*.
First of all the needed geometric parameters are defined in the Matlab environment in order to make possible the computation of the state-space description matrices A, B, C, D. Then, the gimbal transfer functions are generated and employed as *plant* to control with Simulink tool. The reference Matlab command is:

```
>> sys_c=ss(A,B,C,D)
```

To design the LQ regulator, the two rotating masses are not considered as independent from each one, but only one controller for the entire gimbal system is developed. Then the controller features two input variables, u_1 and u_2 and two controlled variables, i.e. θ_1 and θ_2. We're dealing with a MIMO system (*Multi-Input Multi-Output*).
In the LQ design are defined two essential components of the control system: the integral action stage and the dynamic observer system.

Integrative stage

In addition to the system states, for each variable to control a further *integrative state* is introduced: the aim of the so-called integral action is to guarantee the steady-state zero tracking error. Therefore the matrices A, B, C become $A_{tot}, B_{tot}, C_{tot}$ according to given rules of conversion, depending on the controlled system states. Also the size of the K_{lqr} vector will change, because it will be computed by means of A_{tot} and B_{tot}. The augmented gimbal system is:

$$x(t) = \begin{bmatrix} q_{\theta 1}(k) & q_{\theta 2}(k) & x_1(k) & x_2(k) & x_3(k) & x_4(k) \end{bmatrix}^T$$

where $q(k)$ is discrete time integral of the tracking error. Consequently the state-space matrices are transformed into:

$$A_{tot} = \begin{bmatrix} 1_{n\times n} & -T_s C \\ 0_{n\times 1} & A \end{bmatrix} \in \mathbb{R}^{6\times 6} \quad , \quad B_{tot} = \begin{bmatrix} 0_{2\times 2} \\ B \end{bmatrix} \in \mathbb{R}^{6\times 2}$$

As always R is a diagonal matrix, depending on the two control signal of the system, therefore its size is $[2 \times 2]$; since the system states are 6, also Q is a $[6 \times 6]$ diagonal matrix. The chosen weights of the r_{ij} and q_{ij} coefficients are readable through the Matlab commands:

```
>> R=diag([0.01 0.01])
>> Q=diag([100 100 10 10 1 1])
```

then the K_{lqr} matrix is obtained as:

```
>> k_lqr=dlqr(A_tot,B_tot,Q,R);
>> ki=[k_lqr(:,1) k_lqr(:,2)];
>> ko=[k_lqr(:,3:6)];
```

As expected, the matrix K_{lqr} is of $[2 \times 6]$ size: 2 rows as many as the control inputs, 6 columns as many as the 6 states.

$$k_{lqr} = \begin{bmatrix} -94.6311 & 0 & 52.7996 & 0 & 9.7719 & 0 \\ 0 & -65.9435 & 0 & 36.2687 & 0 & 6.6565 \end{bmatrix}$$

and the feedback control system is modified into:

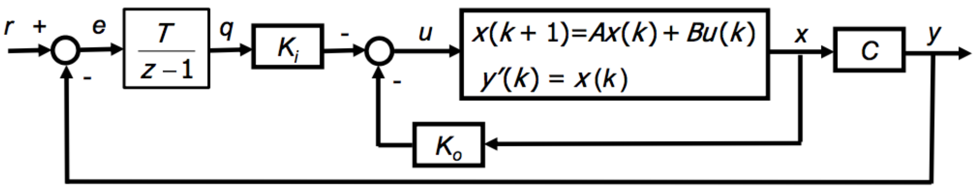


Figure 6.18: Feedback control system with integral stage

With reference to 6.18, the new control law is:

$$u(k) = -k_i q(k) - k_0 x(k)$$

Observer LTI system

The *Observer* dynamic system doesn't influence the dynamic of the plant system: it only allows the estimation of the real state vector requiring as input the control inputs of the plant and the state variables. The observer can be employed only if the system is completely observable: the reachability and the observability are checked through the following line code:

```
>> Mr=ctrb(A,B);
     rho_mr=rank(Mr);
>> Mo=obsv(A,C);
     rho_obs=rank(Mo);
```

From Matlab computations $\rho(M_r) = \rho(M_o) = n$ then the control system can be featured by the observable system. The latter modifies the control system and the control law as below:

$$u(k) = -k_i q(k) - k_0 \hat{x}(k) \quad , \quad \hat{x} \text{ is the estimated state vector}$$

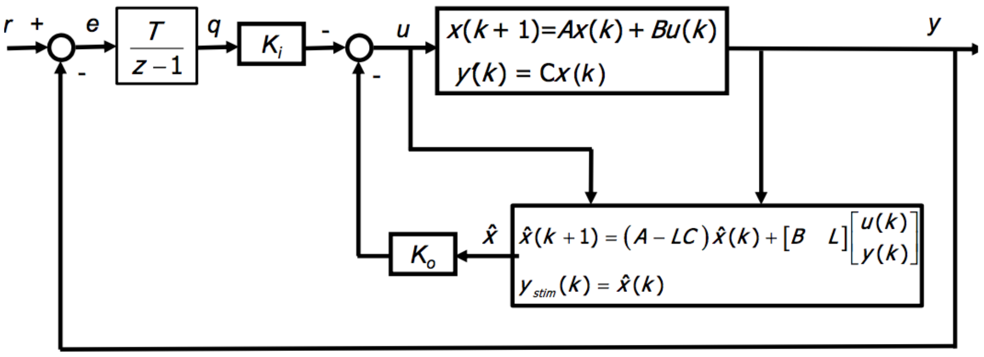


Figure 6.19: Feedback control system with integral stage and observer

The LTI observer system is created by means of the pole placement method, bearing in mind that the dynamic of the observer system is defined by the following relation:

$$\hat{x}(k+1) = (A_d - L \cdot C_d)\hat{x}(k) + \begin{bmatrix} B_d & L \end{bmatrix} \cdot \begin{bmatrix} u \\ y \end{bmatrix}$$

Then, the observer system is created by means of A_d, B_d, L matrices, as:

```
>> sys_obs=ss(A_d-L*C_obsv,[B_d L],eye(4),0,Ts)
```

where L is the *observer gain matrix*, made through the pole placement method:

```
>> L=place(A_d',C_obsv',lambda_obsv)'
```

being the $lambda_{obsv}$ the eigenvalues of $(A - L \cdot C_d)$ matrix.

$$\lambda_{obsv} = \begin{bmatrix} 0.4 & 0.41 & 0.42 & 0.43 \end{bmatrix}$$
$$C_{obsv} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

6.3.3 Simulation results

For the first time are tuned the LQI (Linear-Quadratic-Integrative) Regulator parameters and the control system is simulated in Simulink environment, in order to improve the response of the dynamic control system at each iteration. The simulated feedback control scheme is shown in the picture below:

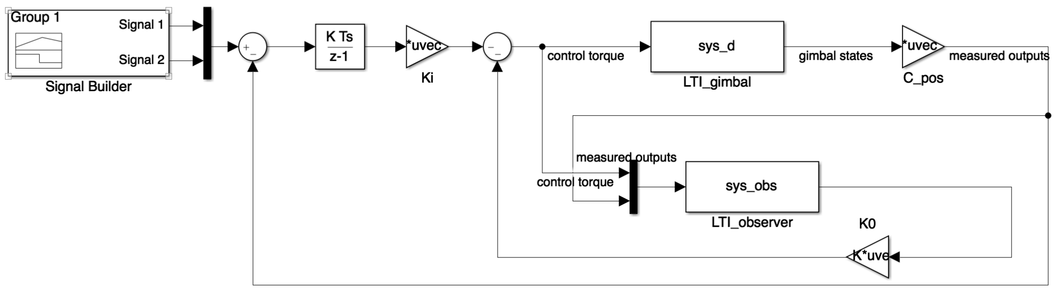


Figure 6.20: LQI Simulink control scheme

As expected, the main blocks are exploited below:

- the *Signal Builder* block which provides the reference signals;
- the *Discrete Time Integrator* block which enables the integral action;
- the K_0 *and* K_i *gain* blocks; they represent the control action of the regulator;
- the *Observer LTI system* which allows the estimation of the vector state, as jet said.

The simulation results concerning the azimuth motion control are displayed:

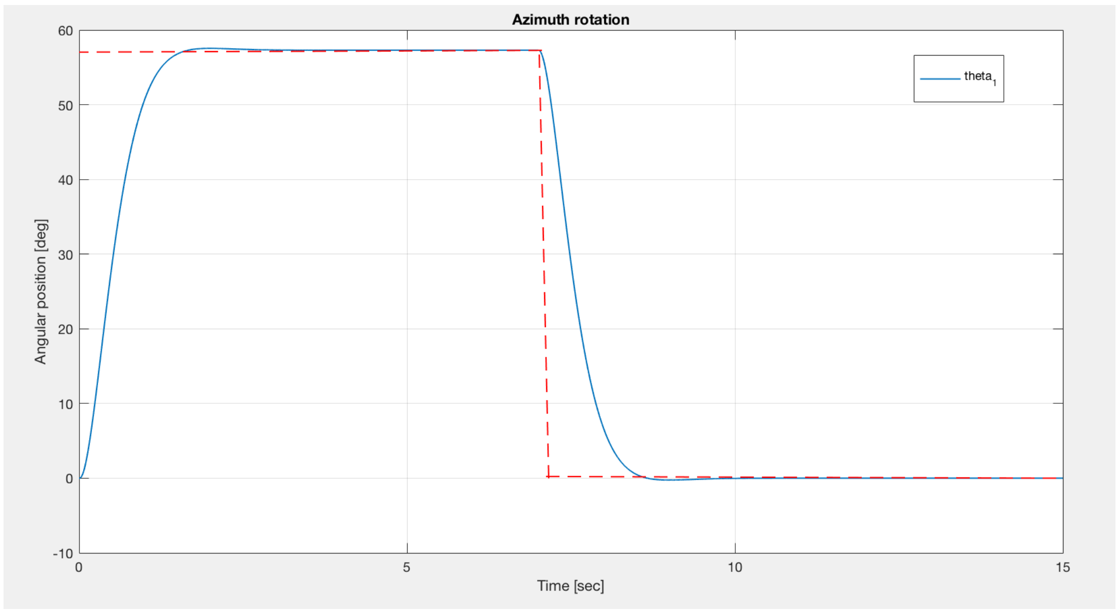


Figure 6.21: *Reference* and *azimuth* angular positions

The elevation angular position instead appear as:

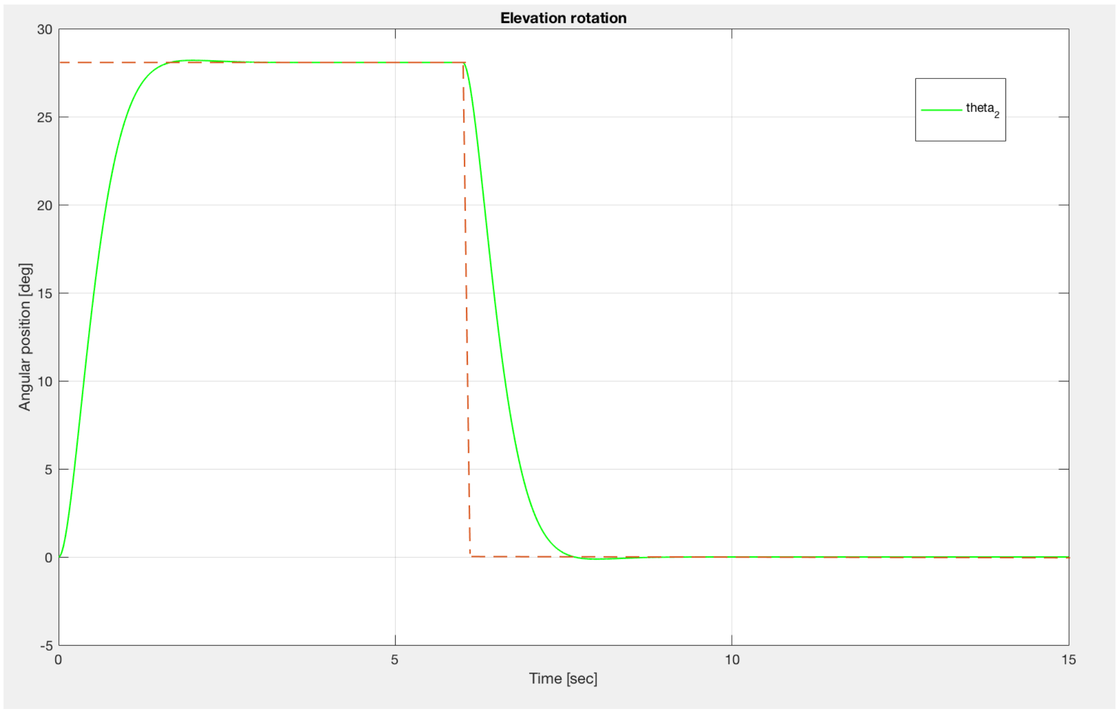


Figure 6.22: *Reference* and *elevation* angular positions

From figures 6.21 and 6.22 the following conclusions can be summarized:

	$\hat{s}$	$t_r[sec]$	$t_{s,2\%}[sec]$	$\lvert e_r^\infty \rvert$
required	0%	≤ 5 s	≤ 4 s	0
azimuth axis	0.08 %	2.9 s	1.35 s	0
elevation axis	0.08%	2.56 s	1.36 s	0

Table 6.2: Results of the LQR design

6.4 Loop-shaping and LQR results comparison

In the actual section a comparison between the results coming from the two different control techniques is developed. In detail, the required torques (on both rotation axis) due to the dynamic of the system and the required angular velocities of the system itself are compared. The control techniques should guarantee that the controlled outputs will follow the given input:

- $\theta_1 = 1$ rad $\simeq 58°$;

- $\theta_2 = 0.5$ rad $\simeq 27°$.

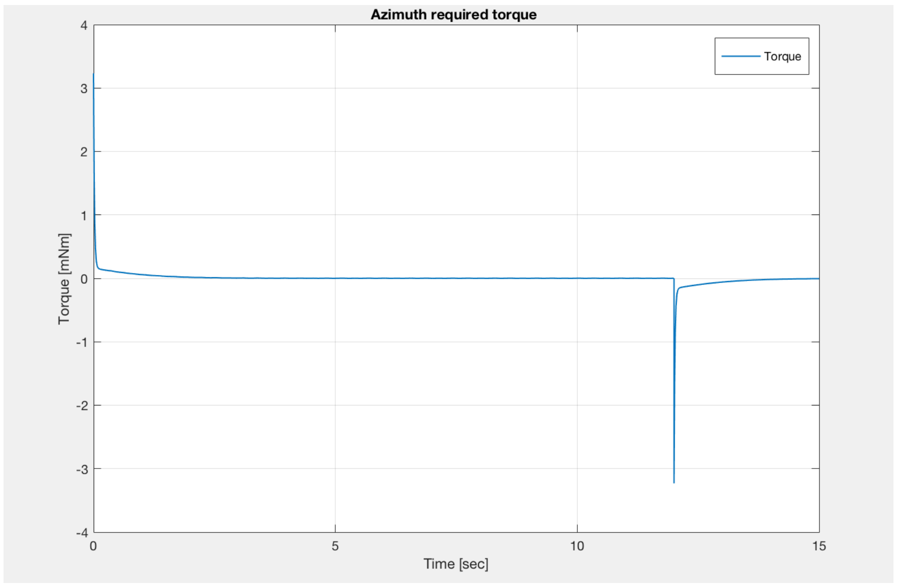
Both the torques and angular rates must be interpreted as the required ones to guarantee a satisfying gimbal system response in terms of time response, overshoot and steady-state tracking error.
Remembering the design specifications:

	required
$\hat{s}$	0%
t_r	≤ 5 s
$t_{s,2\%}$	≤ 4 s

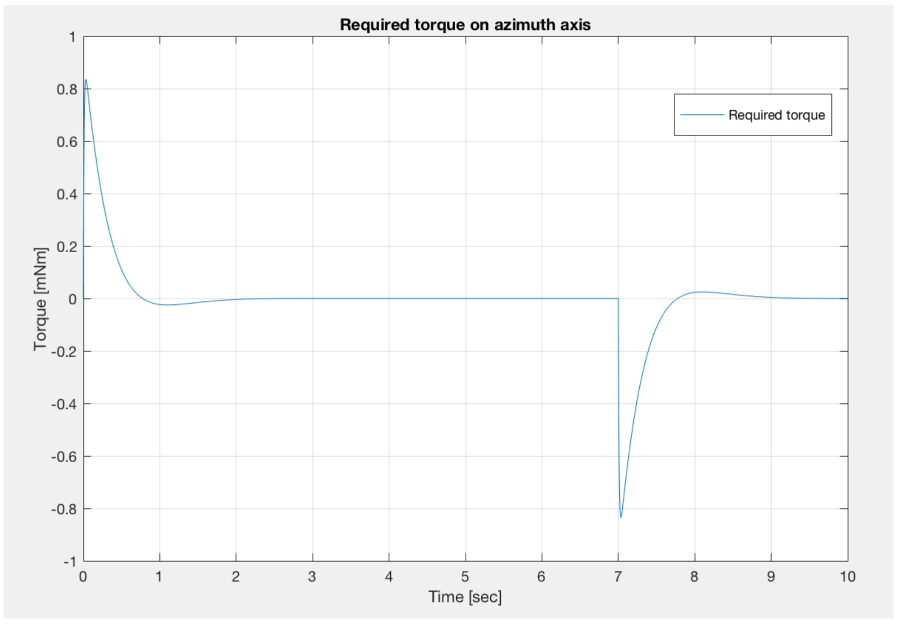
Table 6.3: Design requirements

Concerning the torques required by the control system of the gimbal:

- on the *azimuth* motion axis:



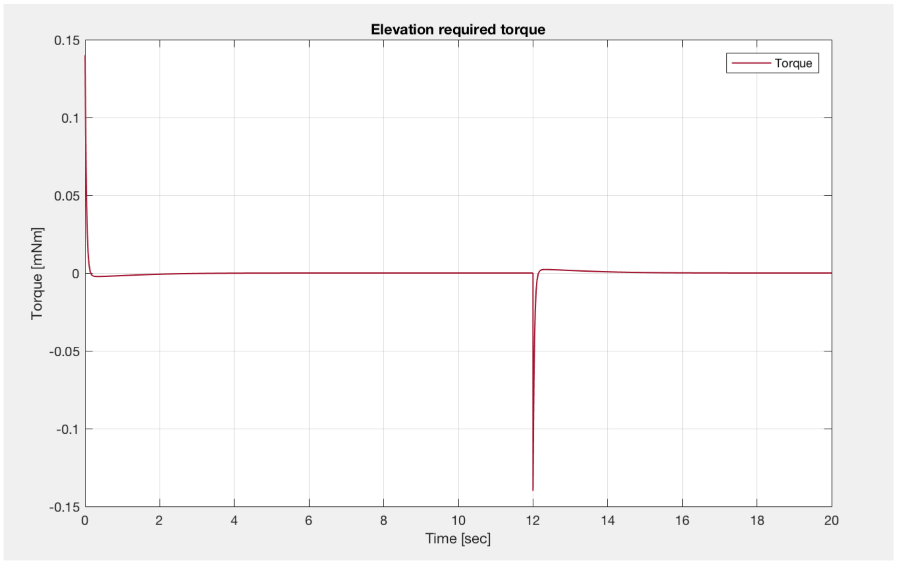
(a) Loop-shaping design



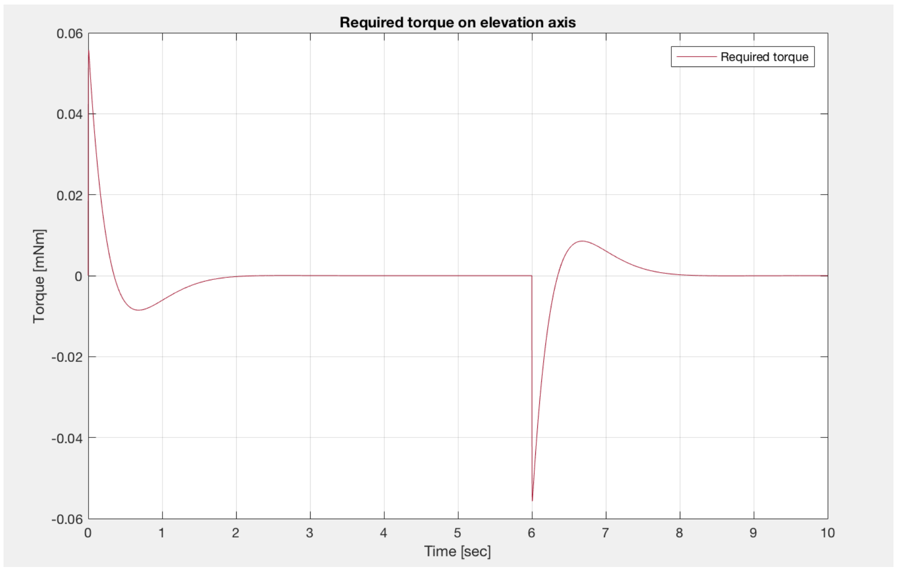
(b) LQR design

Figure 6.23: Required torques on the azimuth axis

- on the *elevation* motion axis:



(a) Loop shaping design

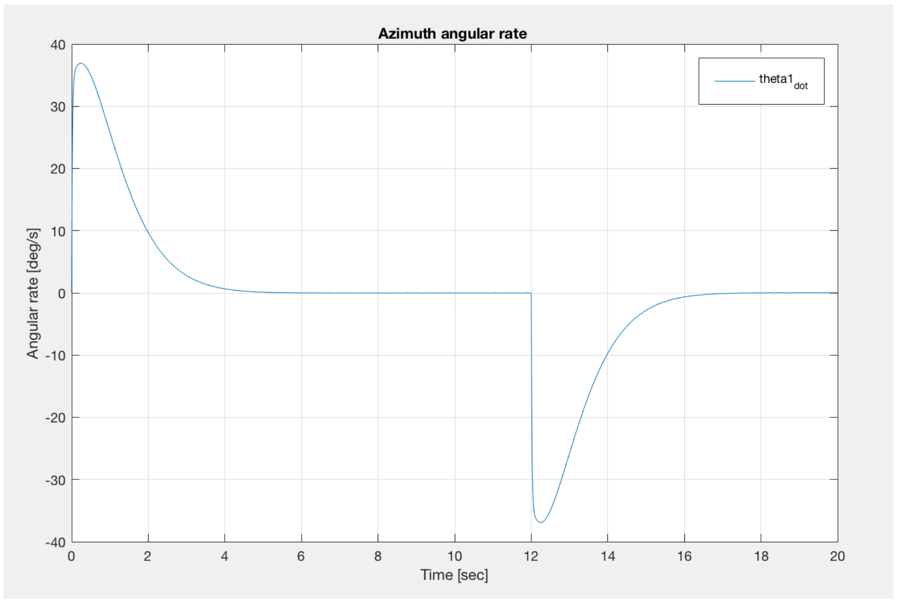


(b) LQR design

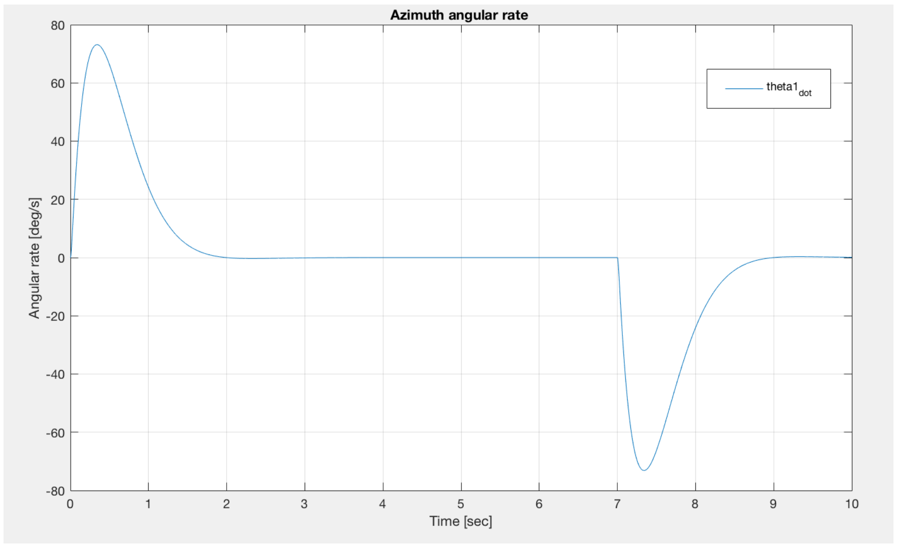
Figure 6.24: Required torque on the elevation axis

Concerning the angular velocities that the system should features to satisfy the given requirements:

- on *azimuth* rotation axis:



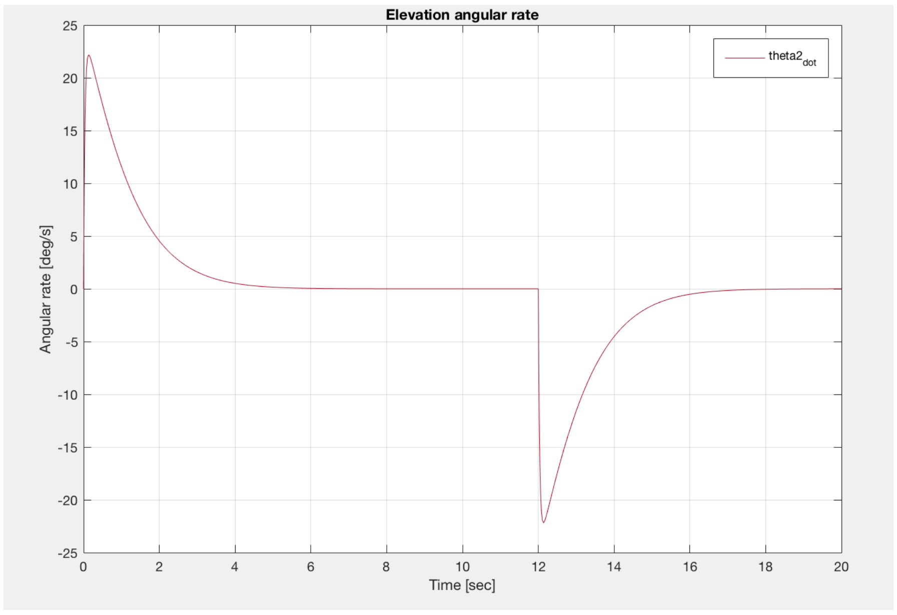
(a) Loop-shaping design



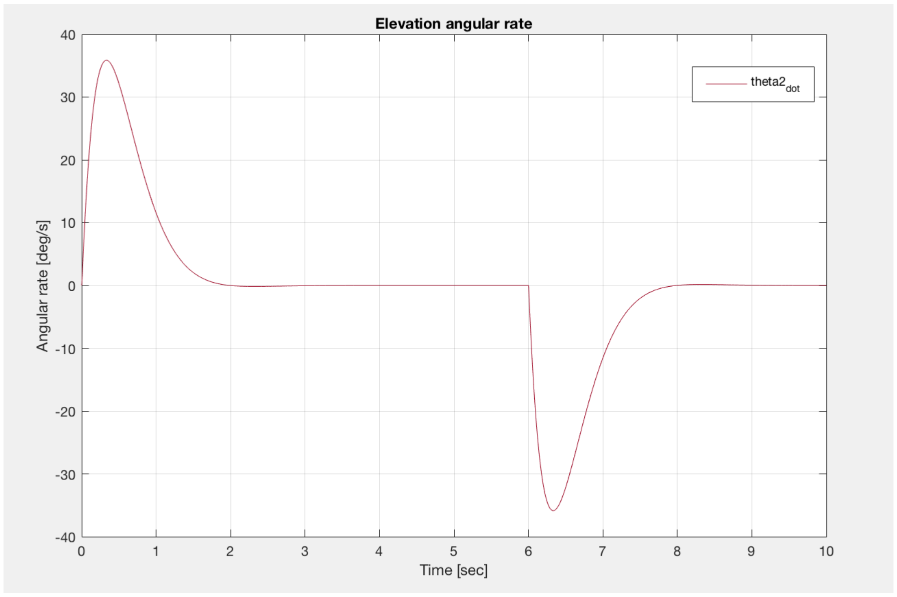
(b) LQR design

Figure 6.25: Angular rate on azimuth axis

- on *elevation* motion axis:



(a) Loop shaping design



(b) LQR design

Figure 6.26: Angular rate on elevation axis

	axis	$\hat{s}$	$t_r[sec]$	$t_{s,2\%}[sec]$	$\dot{\theta}[deg/s]$	torque $[mNm]$
required	$\times$	0%	≤ 5 s	≤ 4 s	$\times$	$\times$
Loop Shaping	azimuth	0%	4.6s	2.8s	$38°/s$	3.1[mNm]
	elevation	0%	4.6s	2.9s	$23°/s$	0.14[mNm]
LQR	azimuth	0.08%	2.9s	1.35s	$72°/s$	0.82[mNm]
	elevation	0.08%	2.5s	1.36s	$35°/s$	0.057[mNm]

Table 6.4: Results of Loop-Shaping/LQR comparison

From tab. 6.4 one can note that the time response, the rise time and the overshoot are different for the two applied control techniques. This is due to the fact that while the loop-shaping design derives the constraints of the control system from the given requirements, the LQ controller is obtained through the optimization of the cost function $J(x, u)$. The main differences are listed below:

1. $\hat{s} = 0\%$ in the loop-shaping design because the controller is developed starting from the needed to have absence of overshoot. Instead, the LQ controller is obtained by means of successive optimizations which allow to achieve a dynamic response closed to the desired one: in the best optimization scenario the user is not able to reduce the overshoot under 0.08%;

2. the *rise time* and *settling time* are higher in the loop-shaping control system, but in both cases the requirement concerning t_r and t_s are fulfilled. The reason of a faster system response in the LQ control system is due to the fact that in the choice of the design parameters $Q >> R$, as can be confirmed by referring to the script in *Appendix B*;

3. the required $\dot{\theta}_1$ and $\dot{\theta}_2$ angular velocities on the two motion axis, are higher in the LQ control system with respect to the loop-shaping design, due to the low weights assigned to the x_3 and x_3 state variables inside the cost function $J(x, u)$;

4. also concerning the required torques τ_1 and τ_2, the main difference is that the LQ control system needs of very lower torques than the loop-shaping control system. This is justified by means of the chosen R matrix values;

5. $|e_r^\infty| = 0$ specification is accommodated in both the two control systems.

Chapter 7

PID regulators

Given the needed to control the gimbal holder with the electric motors already available in the laboratory, two PID controllers are designed. In fact, is still valid the basic idea for which the two rotations around azimuth and elevation axis are totally decoupled, so the two DC-motor work in a complete independent way: one PID controller will be used to control each actuator.
The PID regulator is made of 3 main parts which are the proportional (P), the integrative (I) and the derivative (D) one. Its expression is:

$$u(t) = K_P e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt} \tag{7.1}$$

and the corresponding control scheme is:

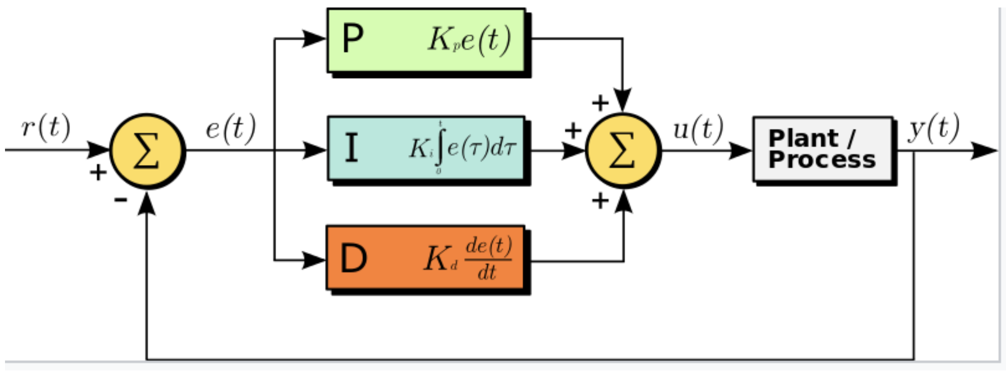


Figure 7.1: General feedback scheme of a PID regulator

Note that the gains K_P, K_I, K_D are constants and through them the control input $u(t)$ is computed, based on the actual value of the tracking error $e(t) = r(t) - y(t)$. Modify the gains value means act on static and dynamic behaviors of the controlled system, as exploited below:

Parameter	Action	Stability	Response time
K_P	$K_P \uparrow$	improve	slow down
	$K_P \downarrow$	get worse	speed up
K_I	$K_I \uparrow$	get worse	slow down
	$K_I \downarrow$	improve	speed up
K_D	$K_D \uparrow$	get worse	get worse
	$K_D \downarrow$	get worse	speed up

Table 7.1: Effects of the PID gains on the system behavior

In detail:

- K_P: the introduction of the proportional gain causes an output variation in a proportional way with respect to the actual error value, $e(t)$. The command input value $u(t)$ changes when the output $y(t)$ also changes;
- K_I: by means of the integrative gain, the offset due to the proportional action on $y(t)$, is canceled. The gain K_I is applied to guarantee a steady-state null tracking error; at the same time the integrative action should generate overshoots and oscillations on the system response. Generally, a PI controller generates a faster response without deteriorate the overall system stability;
- K_D: the purpose of the derivative action is to computing the control action also taking into account the derivative term of the error.

If the *sampling time* T_s of the system is defined, the regulator can be translated in the discrete time as:

$$u(k) = K_P e(k) + K_I T_s \sum_{k=0}^{n} e(k)dt + K_D \frac{e(k) - e(k-1)}{Ts} \tag{7.2}$$

The discrete-time regulator is used in the phase of the control system implementation on microcontroller.

7.1 Linear PI feedback control system

In order to control the motion on the two axis of the gimbal system, two DC motor must be separately controlled. In fact, what is really controllable is the supply voltage of the

DC motors: the current flowing inside the motor depends on the applied voltage and consequently the torque generated by each motor on the rotation axis is proportional to the current by means of the *constant torque* coefficient.

7.1.1 Matlab and Simulink implementation

The scripts concerning the PID controllers are added in *Appendix C*. In order to develop the motion control system of the gimbal, the state-space description of the DC motor is get from *chapter 5* and its transfer functions are extracted by means of Matlab commands, then are used as *plant* of the control system.
The Simulink feedback control system is the same in both rotation cases and it's made of the following main blocks:

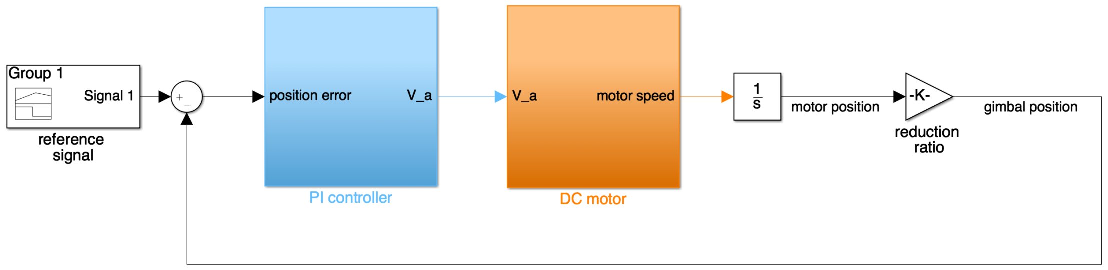


Figure 7.2: Linear feedback control system

In the previous figure, note the presence of:

- *Signal builder* block which generate the references of the control scheme;
- *PI controller* block in which the control law is computed by means of only proportional and integral actions. The PID gains are chosen as $K_P = 20$ and $K_I = 0.02$. The controller provides as output the controllable input variable called V_a, corresponding to the supply voltage of the DC-motor;
- *DC motor* block, in which the dynamics and the electromechanical properties of the used DC motor are included. In the block, thanks to an electrical and a mechanical transfer functions, the voltage is converted in current and the latter is translated into torque generated by the motor. The DC motor block receives as input the supply voltage and provides as output the motor speed ($\dot{\theta}_{mi}$);
- the *integrator* block allows to acquire the the motor position, θ_{mi};
- the *reduction ratio* aim is to convert the motor position into the gimbal position θ_i, which must be really controlled. Since the motion transmission happens by means of motor-reducer-pulley assembly, the motor speed is reduced while the

motor torque is multiplied of a factor equal to: $159 \cdot 6 \simeq 1e^3$. This quantity is called reduction ratio, identified as n in the Matlab scripts. Once computed the position gimbal, it is feed back to the control system in order to complete the control action.

With reference to fig 7.2, are shown the controller and the plant blocks. In detail:

1. the controller consist of two action, which are the proportional and the integrative ones. In the next picture the controller architecture is illustrated:

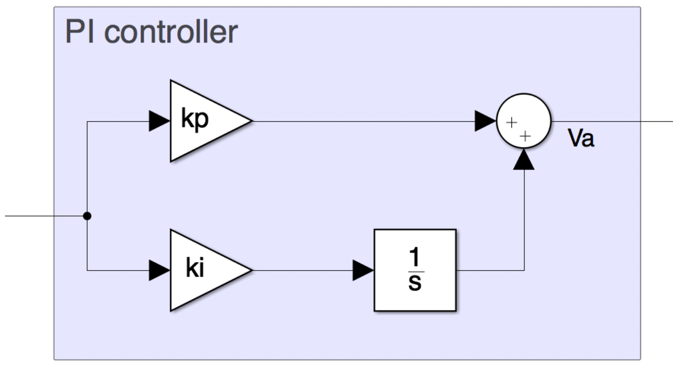


Figure 7.3: PI controller

2. the plant of the PI controller is the DC motor. It's made of two transfer functions, i.e. the electrical and the mechanical one because of, as known, it's an electromechanical system.

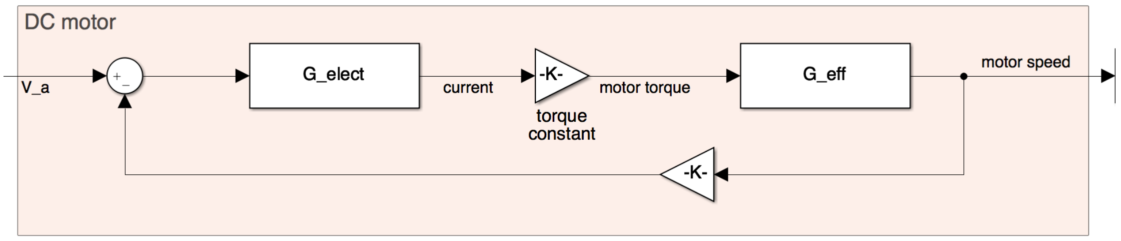


Figure 7.4: DC motor subsystem

More precisely, the electric transfer function and its pole are:

$$G_{elect}(s) = \frac{1}{0.000135s + 1.46} \implies p_{elect} = -\frac{1.46}{0.000135} = -1.08 \cdot 10^4$$

while the mechanical ones are:

$$G_{mech}(s) = \frac{1}{1.2 \cdot 10^{-6}s + 1.7 \cdot 10^{-6}} \implies p_{mech} = -\frac{1.7 \cdot 10^{-6}}{1.2 \cdot 10{-}6} = -1.41$$

It's possible to note that the electrical pole have too high frequency with respect to the mechanical one: it means that p_{elect} will never influences the DC motor dynamic behavior. Another observation concerning the mechanical transfer function: it takes into account also the gimbal inertia and viscous friction, so the plant is made of both the gimbal and the motor dynamics.

The exploited control system is then simulated and the simulation results are provided in the next section.

7.1.2 Simulation results

As always, the time domain given requirements are dealing with the rise time and the overshoot of the system dynamic response, and are the same of the jet developed control system. In this section, the control system is simulated when the input reference are:

- command input on azimuth axis: $\theta_{1,ref} = 58°$;
- command input on elevation axis: $\theta_{2,ref} = 58°$;

In terms of azimuth controlled rotation, the results shown that:

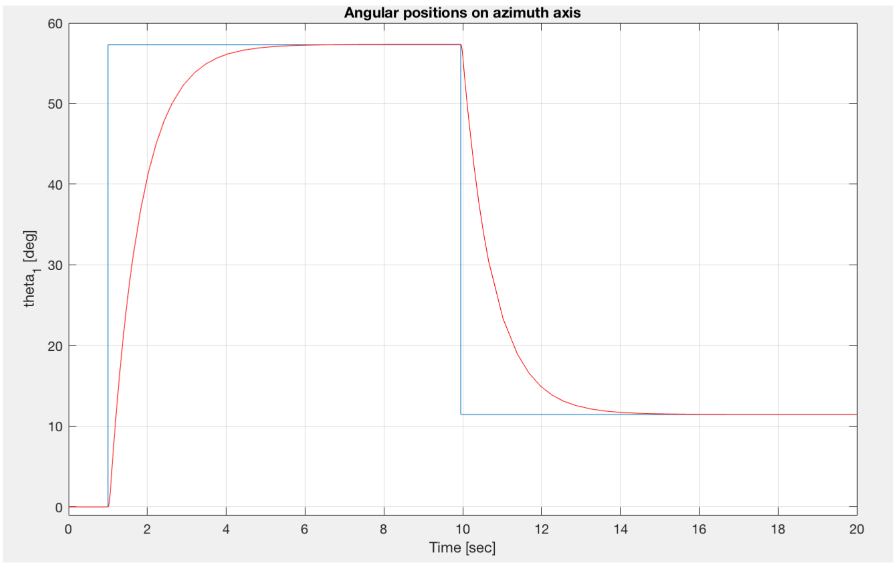


Figure 7.5: Pan angle response

Can be stated that the rise time, the overshoot and the steady-state tracking error specification are totally fulfilled.
While the motion control on the elevation axis appears as:

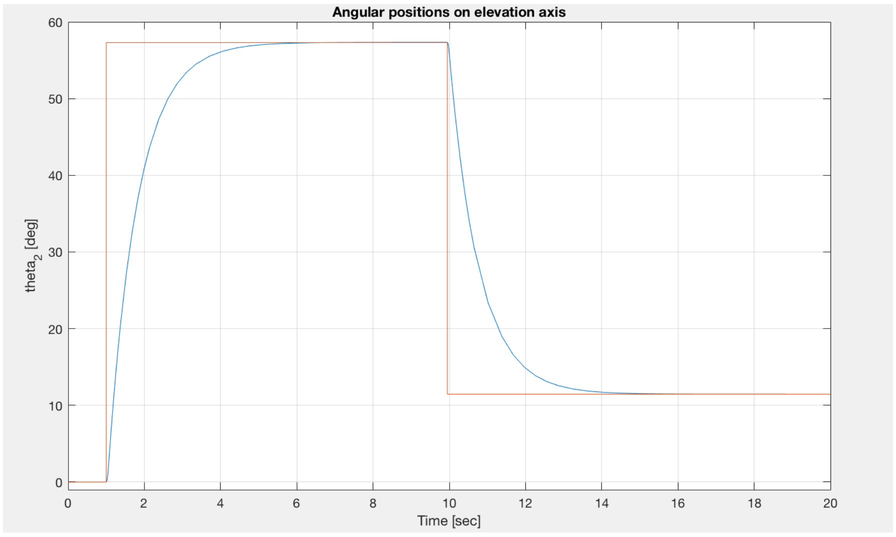


Figure 7.6: Tilt angle response

Also for this controller development all the requirements are satisfied.

7.2 A more realistic PI feedback control system

Theoretically, when one designs a PID regulator, it takes into account some typical real problems, like as the *actuator saturations*, the *anti-windup*, which is dealing with the actuator saturation and the *backlash* when the mechanics of the gearbox is analyzed. These problems are the so-called non-linearities of a DC-motor. By means of the Simulink blocks the non-linearities are integrated in the linear control system of the DC-motor in order to verify if the desired requirements are once again satisfied.

7.2.1 Non-linearities

Saturation and anti-windup scheme

The most common actuator non-linearity is represented by the physical limitations of the motors. Precisely, the torque provided by any motor is restricted due to the limit values which the current flowing into the motor circuits can acquire. Then, also the voltage supplying the motor must be in a range between a minimum and a maximum value. A possible solution to the saturation problem, is add the anti windup scheme to the control system. At the beginning the control system including the saturation phenomena is:

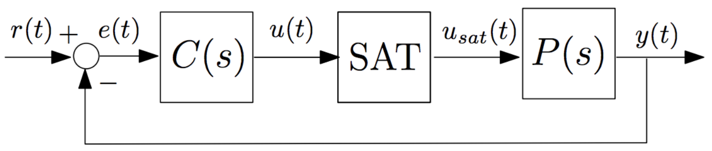


Figure 7.7: Saturation block in the feedback control system

and the saturation function is defined as:

$$u_{sat}(t) = sat(u) = \begin{cases} -U_{min} & \text{if} \quad u(t) \leq -U_{min} \\ u(t) & \text{if} \quad -U_{min} < u(t) < U_{max} \\ U_{max} & \text{if} \quad u(t) \geq U_{max} \end{cases} \tag{7.3}$$

The simultaneous presence of integral action and saturation generates the *anti windup* phenomena which deteriorates the control system performances. If the input signal is saturated and the absolute value of the error remains of the same sign, the controller continuously computes the integral of the error; if the error changes sign must wait for an interval time before to have $-U_{min} < u(t) < U_{max}$ again. The anti windup solution purpose is to guarantee that the control signal evolves according to the real variables influencing the control system (the output $y(t)$): it means that the control variable $u(t)$ must leave the saturation value as soon as the error changes its sign.
The control system, if the anti windup scheme is taking into consideration, is modified as below. The basic idea is to intercept the control input values before and after the saturation, then computing the difference

$$\Delta u = u(t) - u_{sat}(t)$$

is possible to give back the Δu value to the integrator block. There is also the *desaturation device*, represented through the $F(s)$ transfer function:

$$F(s) = \frac{1}{T_d}$$

where T_d value controls the interval time in which the desaturation happens. If T_d acquires small values, the desaturation action will be faster, and vice versa.

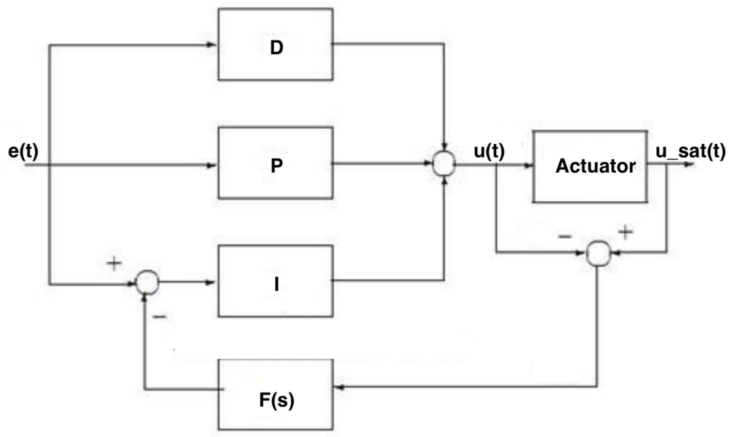


Figure 7.8: Anti windup solution embedded in the control system

Note that in order to actually adopt an anti-windup solution, is needed to have values that can be effectively measured. For this reason, the anti-windup scheme is not always applicable and its feasibility depends on the available sensors and on the control system characteristics.

Backlash of the reducer

As known, the electric motor is able to generates high angular speeds to the detriment of the machine torque: for this reason is usually placed a suitable speed reducer between motor and load. The role of the adaptor is to decrease the motor shaft speed while the generated torque is multiplied by the reduction ratio value.
Usually the mechanical assembly made of motor, reducer and load is treated as a single body, having an equivalent inertia equal to:

$$J_{eff} = J_m + \frac{J_l}{\tau^2}$$

where J_m and J_l are the motor and load inertias respectively, instead τ is the reduction ratio.
The *backlash* is a typical non-linearity of the mechanical system featuring the reducer: it exists between two moving parts and it clearly deteriorates the performances of a control system in terms of speed and positions. The effect of the backlash appears when a change in the motion direction happens: it causes the loss of the motion transmission between the motor and the load, due to the fact that the motor for a few moments is not able to control the load which can autonomously moves. Since the rotating gears aren't in perfect traction conditions when the direction of the rotation changes, a delay in the system response is present.
The Simulink tool provides a block, identified as *Backlash block*, which allows to model this particular non-linearity. The only required parameter when the block is added to the control system is the dead band (also called dead zone): the dead band, expressed

in [°] or [rad] corresponds to a range of values for which the output, i.e. the systems position, is null.

Aerodynamic moment

In order to develop a complete control system the effects of the aerodynamics are analyzed and modeled. The aerodynamics represents an external force producing moments on the gimbal system, which in standard operating condition is attached to a flying aircraft. Is considered worthwhile to model the aerodynamic interference as if it is a noise on the position measurements. In detail, it is considered as a disturbance on the bearings of the mechanical system: then, an error on the position measurements provided by the encoder, is added to the control system. The noise at issue is comparable to a sinusoidal signal, having a frequency closed to the frequency at which the encoder works and a small amplitude (in order to be considered as disturbance the signal amplitude should be quite small).

7.2.2 Simulation results

As always, the control system is developed starting from the basic idea to have two independent DC-motors. A *decentralized control* architecture is applied.

Rotation around azimuth axis

Taking into account all the non-linearities previously exploited, the gimbal control system is the following one:

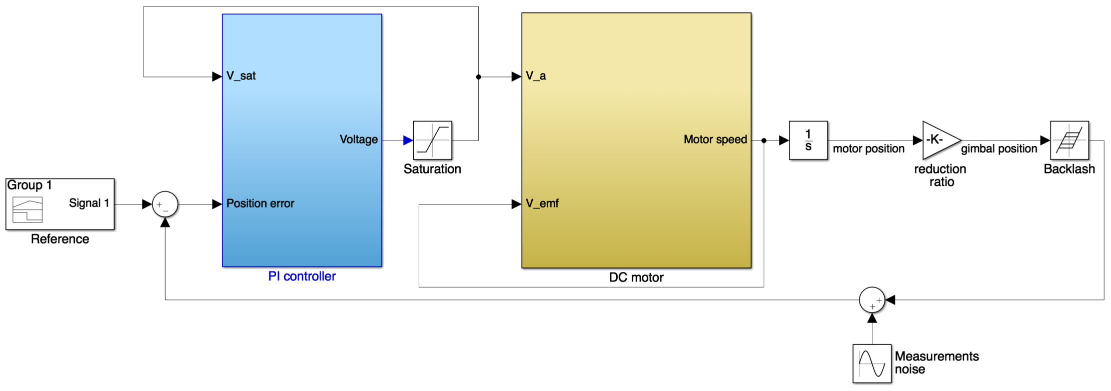


Figure 7.9: Realistic position control system

The previous one is the control system of the rotating part around azimuth axis, but it's obviously the same for the motion control on elevation axis. With reference to fig. 7.9, the *PI controller* block is shown in detail below:

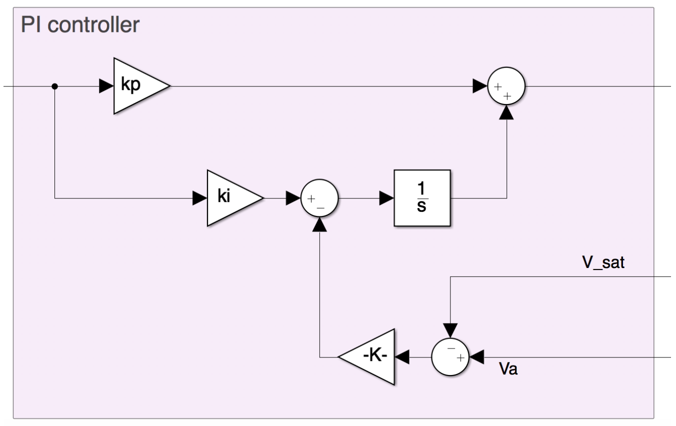


Figure 7.10: PI controller in the non linear scenario

Note that the gain K_P and K_I are the same chosen in the totally linear assumptions. Moreover in the *PI controller* block is integrated the anti windup scheme, in which

$$\Delta u(t) = v(t) - v_{sat}(t)$$

is given back to the integrator block. As usual, with reference to the general non-linear control system of figure 7.9:

- the output of the *plant* is the motor speed $\dot{\theta}_{m1}$;
- the *integrator* block $\left(\frac{1}{s}\right)$ is used to convert the motor speed into the motor position θ_{m1};
- the *reduction ratio* block allows to obtain the real gimbal position θ_1 and the latter variables is the feedback variable;
- the *backlash* block allows to model the backlash phenomena and causes a small delay in the system response;
- the *sinusoidal signal* block represents the disturbance on the encoder measurements.

Then, the control system response is observed:

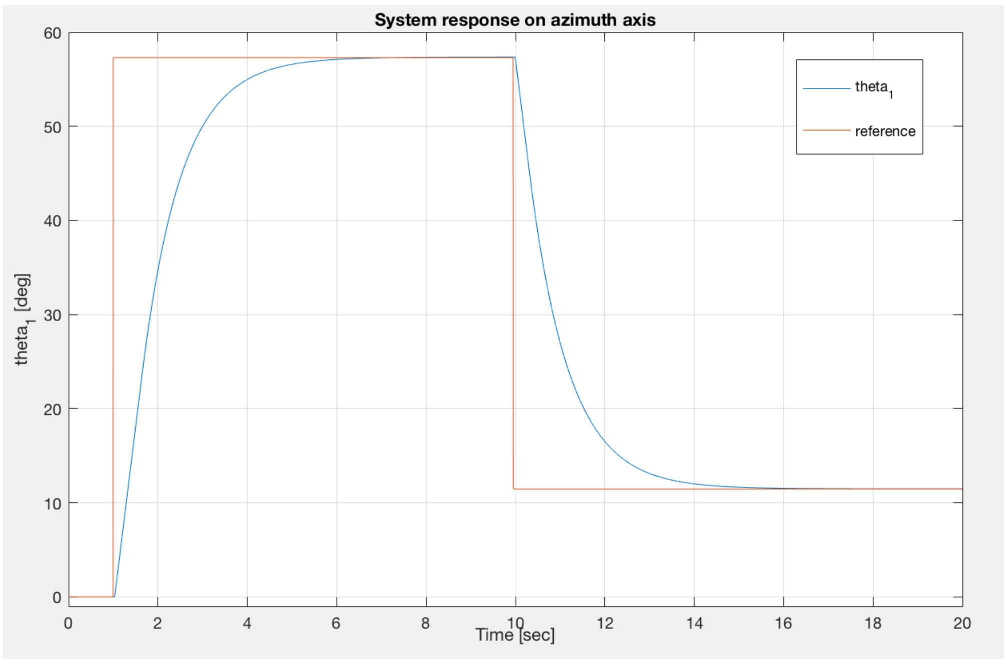
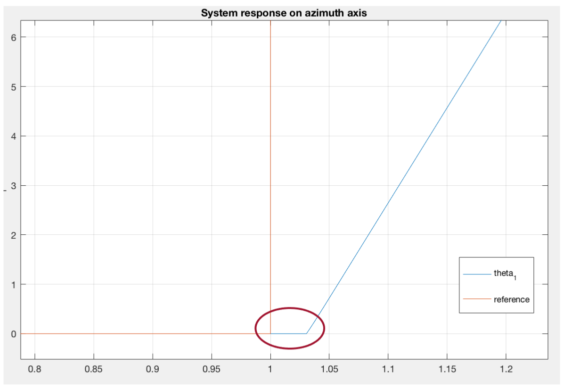


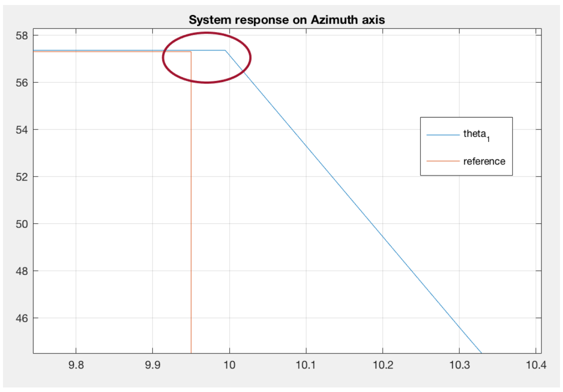
Figure 7.11: Azimuth axis angular position

Referring to fig. 7.11:

1. also if the system is make non linear by means of the Simulink blocks, the given design requirements are satisfied:
    - $\hat{s} = 0\% \implies$ absence of the overshoot;
    - $|e_r^\infty| = 0$;
    - considering that the command input is given at simulation time $t_{in} = 1s$, the steady- state condition is reached at $t_{ss} = 6.2 \implies$ the rise time and the settling time constraints are fulfilled, taking into account a small delay due to the presence of the backlash;
2. the backlash effect can be observed at every rotation direction changes, as better shown below:



(a) Backlash effect when a reference of 58° is given



(b) Backlash effect when a reference of $-45°$ is given

Figure 7.12: Backlash phenomena when a change of the motion direction happens

As said, the backlash has as effect the delay in the system response: the introduced lateness is equal to $t_{delay} \simeq 0.5s$.

Rotation around elevation axis

The control scheme of the systems rotating around the elevation axis is basically the same of figure 7.9, then are quickly shown only the simulation results below:

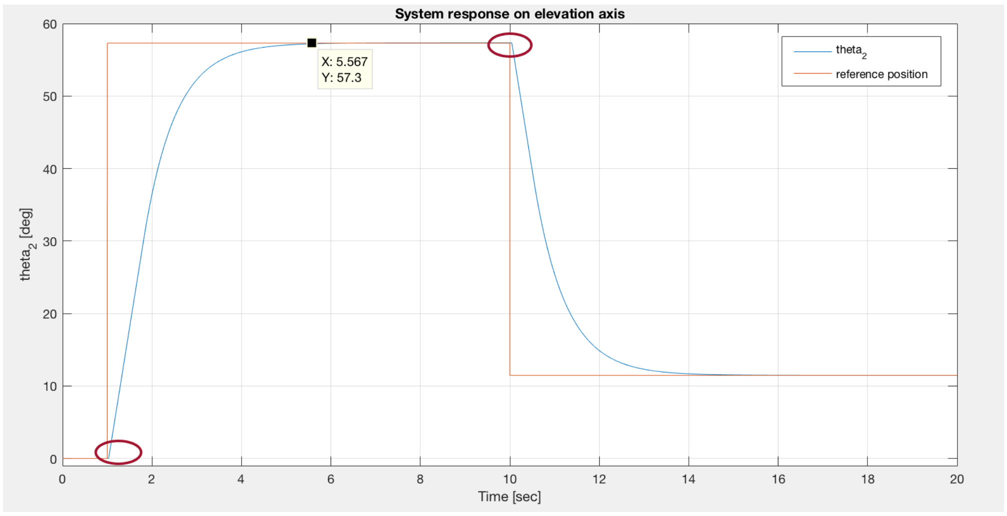


Figure 7.13: Elevation axis angular position

From fig. 7.13 the fulfillment of the design requirements is confirmed:

- $|e_r^\infty| = 0$;
- $\hat{s} = 0\%$;

- $t_r = 5.56s$, in spite of the command input is given at $t_{sim} = 1s$ and the delay introduced by the backlash: $\Longrightarrow t_r \simeq 4.3s$;

Chapter 8

Motion control implementation

The last phase of the work is committed to the implementation of the designed controller on the Arduino electronic platform, in order to test the real behavior of the existing physical prototype. In fact, the corresponding code of the controller simulink model is loaded on the Arduino shield. which is linked with the two DC-motors. In order to provide the estimated results, the SIL (Software-in-the-Loop) and PIL (Processor-in-the-Loop) phases are required. To this purposes comes in handy the Simulink tool with its own code generation tool.

8.1 Arduino electronic platform

The Arduino platform is made of different electronic boards and devices; all of them are provided of a microcontroller and they can be integrated each others thanks to the *Arduino IDE (Integrated Development Environment).*
In this gimbal motion control problem will be used: the Arduino Mega 2560 as microcontroller, the Arduino Pololu Dual VNH5019 as motor driver shield and several sensors (we will focus on the IMU - Inertial Measurements Unit).

8.1.1 Arduino Mega 2560

In the laboratory the available Arduino microcontroller is the *Arduino Mega 2650*, which is shown in the next picture.

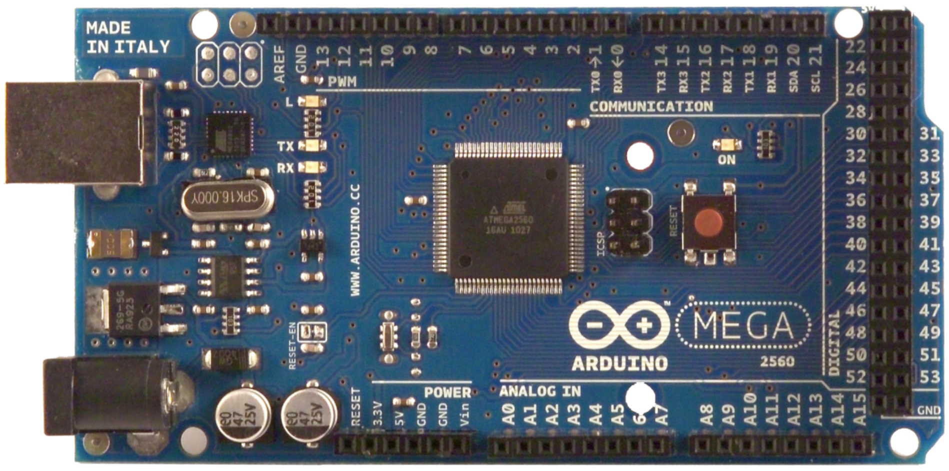


Figure 8.1: Arduino Mega 2560 Microcontroller

It has the following technical features:

Microcontroller	ATMega2560
Operating voltage	5V
Input voltage (recommended)	5-12V
Input voltage (limits)	6-20V
Digital I/O pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Table 8.1: Arduino Mega 2560 - technical specifications

8.1.2 Arduino Pololu Dual VNH5019

The *Arduino Pololu Dual VNH5019* is a motor shield, compatible with the Arduino microcontroller board, and it is used in order to make easy the control of the two gimbal DC-motors. It enables the control of two DC motor at the same time, and it is illustrated in the next figure:

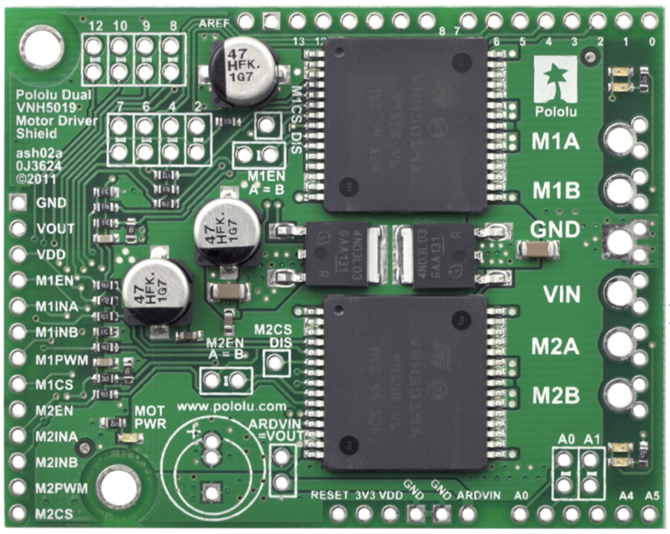


Figure 8.2: Arduino Pololu Dual VNH5019

In the following table, the motor shield technical features are listed:

Operating voltage	5.5-24 V
MOSFET on-resistance	18 mΩ
IMax PWM frequency	20 kHz
Current sense	0.14 V/A
Over-voltage shutoff	24 V min / 27 V typ
Logic input high threshold	2.1 V min
Time to overheat at 20 A	20 s
Time to overheat at 15 A	90 s
Current for infinite run time	12 A

Table 8.2: Arduino Pololu Dual VNH5019 - technical specifications

8.1.3 IMU sensor

As known, the IMU is an electronic devices made of inertial sensors like as gyroscope and accelerometers, able to monitor the dynamics of a moving body. Thanks to the IMU sensor the current position of a body (i.e. of the gimbal for our purposes) is obtained, and the IMU's measurements should be employed to perform a correction action of the motion. The available IMU is the *Xsens MTi3* one.

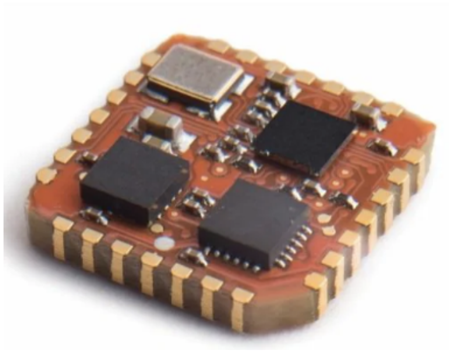


Figure 8.3: Xsens MTi3

The technical specifications are listed below:

Input voltage	2.19-3.6 V
Package	SMD, JEDEC PLCC-28
Interfaces	I2C/SPI/UART
Output data rate	0-800 Hz
Interface protocol	Xbus
Typical power consumption	$\leq$ 100 mW
Weight	<1g
Size	12.1 x 12.1 x 2.55 mm
Software interface	Xsens Device API (open source)
Gyro bias stability	10 °/h
Roll/Pitch (Static | Dynamic)	0.5°
Yaw	2°

Table 8.3: Xsens MTi3 - technical specifications

8.2 Sotware in the Loop Validation

The SIL (software in the loop) phase is subsequent to the system modeling and controller design steps. In fact, it is useful to ensure the proper operation of the designed controller: the controller model block is converted into C/C++ code and the generated code is tested in software environment before to be loaded on the target hardware.

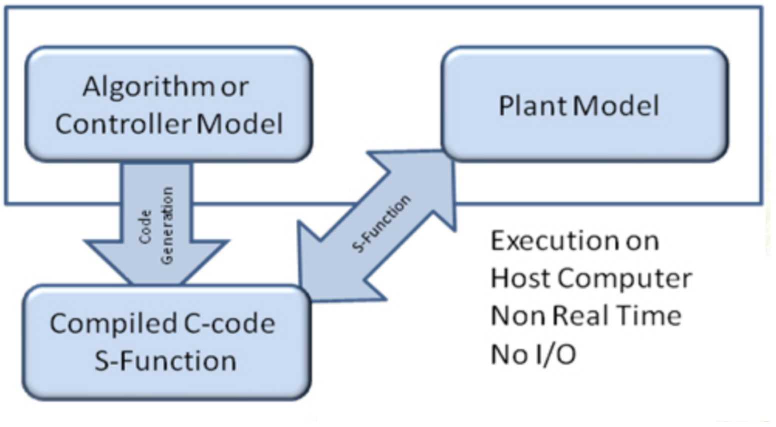


Figure 8.4: Software in the loop stage

As shown in fig.8.4 both controller and plant are running on the PC: while the controller is running but it is an executable C/C++ code, the plant exists in native simulation language (Simulink's model block). In this way the user avoids to damage the hardware, and it can test the controller operating principles also in the most critical scenarios.
To the SIL purpose 3 actions are necessaries:

1. the controller must be discretized;
2. the data type must be opportunely converted;
3. the executable in C/C++ language must be extracted by means of code generation tools;

Since the controller discretization and the data type conversions represent approximations introduced in the original control scheme, the aim of a SIL validation is to prove that the system response is still admissible, in spite of the necessaries inserted approximations. So, the generated code is restored in the Simulink environment by means of several techniques: the method that will be used in this chapter is the *S-function* generation. From now on will be exploited in detail the mandatory configurations needed to the code generation, for the rotating system around the z-axis: it will be the same for the rotation around the elevation axis.

8.2.1 Controller discretization

First of all is required a *discrete-time controller*, because of the controller becomes a task executed by the microcontroller. The discretization action happens in Matlab windows, by means of the 'c2d' command:

```
>> sys_d=c2d(sys_c, Ts, method)
```

Chosen the sampling time $T_s = 0.001$ and the discretization method as *zoh*:

```
>> C_dt=c2d(C_c, Ts, 'zoh')
```

then the dicrete-time controller is:

$$C(z) = \frac{23(z-1)}{(z-1)}$$

8.2.2 Data type conversion

A data type conversion is required so that a full executable, actually reusable on a target software, is obtained starting from the Simulink model block. By default, Simulink assigns *double* type values to the variables when not differently specified. Normally the microcontrollers don't support this data types: the conversion from double to *single* (which means integer) data types is indispensable.
Since in standard operating conditions the controller runs on the software, it works with single (integer) datas; on the contrary, the plant, which is a continuous system generates double (float) data types. The solution is to introduce the *data type conversion* blocks in the Simulink control scheme, like as:

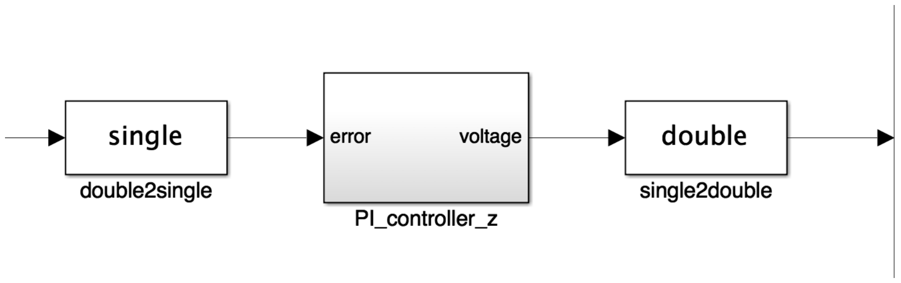


Figure 8.5: Data type conversion blocks

In this way the controller is dealing with integer variables, instead the plant which still exists in the simulation environment can accept and provide float variables.
For sure the data type conversion represents an approximation of the simulation results, then a test after the code generation is recommended.

8.2.3 Code generation

Simulink offers the opportunity to automatically generate the code in C or C++ language corresponding to the designed model block, through the so called *Code Generation* powerful tool. In this manner the Simulink model is converted into a full executable. In order to obtain the C code a small number of steps are needed.

Modifications in the Model Configuration Parameter Pane

In Simulink environment, the model configuration parameters must be changed, by click on the specific button, which will open the window concerning the model parameters:

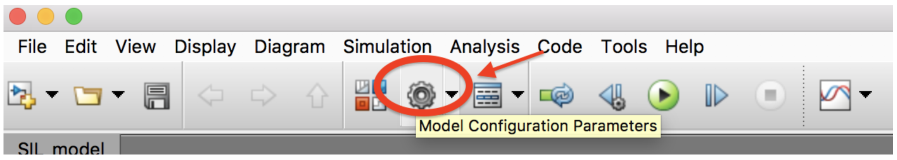
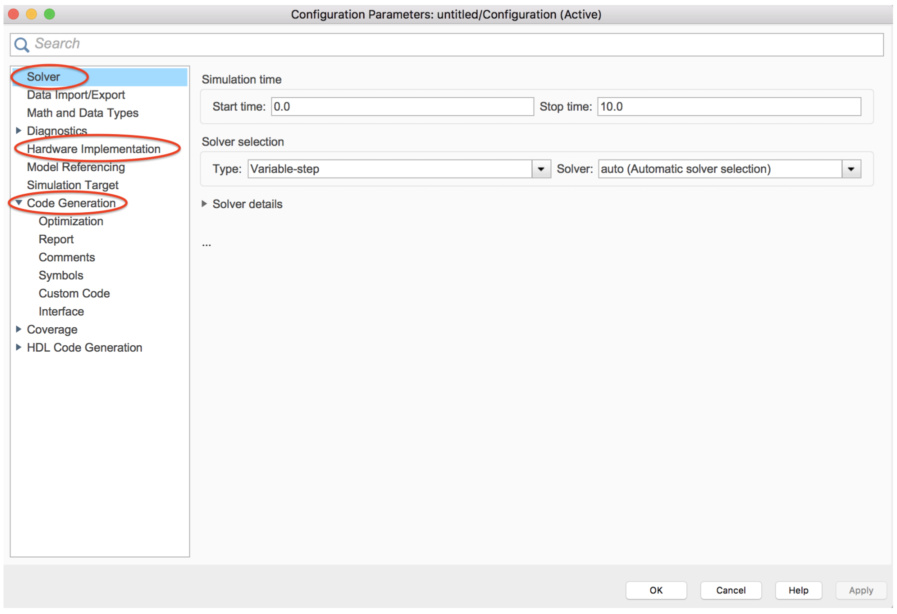


Figure 8.6: Configuration parameters button



Configuration Parameters: untitled/Configuration (Active)

Search

- Solver
- Data Import/Export
- Math and Data Types
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
  - Optimization
  - Report
  - Comments
  - Symbols
  - Custom Code
  - Interface
- Coverage
- HDL Code Generation

Simulation time

Start time: 0.0 Stop time: 10.0

Solver selection

Type: Variable-step Solver: auto (Automatic solver selection)

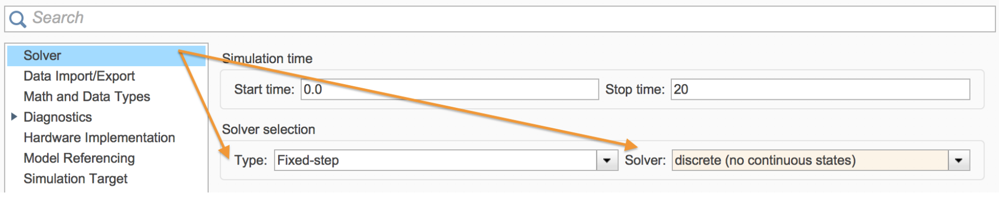
Solver details

...

Figure 8.7: Configuration parameters window

In fig. 8.7 the section requiring changes are highlighted:

1. *Solver* pane: the controller becomes a model, executed by the microcontroller when the corresponding C code is loaded on the shield. In order to make the controller task similar to a periodic task, evaluated at the beginning of each fixed interval of time, the *Solver type* must be forced to *Fixed-step, discrete time* as shown in the following figure:



Simulation time

Start time: 0.0 Stop time: 20

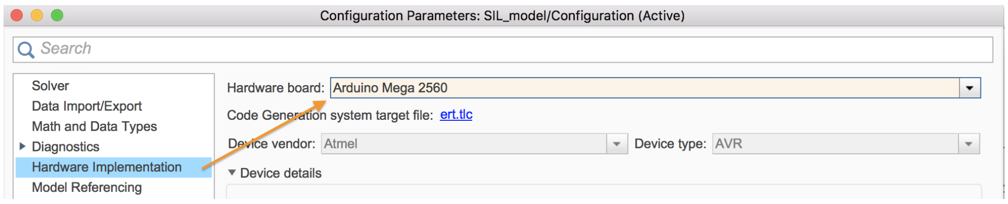
Solver selection

Type: Fixed-step Solver: discrete (no continuous states)

Figure 8.8: Solver configuration

Once imposed the fixed-step and the discrete time integration time, the solver will able to evaluate the controller model at regular time intervals (a priori chosen). When the fixed step value decreases, the accuracy of the simulation results improves while the simulation takes a longer time to give results.

2. *Hardware implementation* pane: when the hardware implementation section is configured, the target hardware must be selected. In the thesis the microcontroller will be loaded on the Arduino Mega 2560 board, as shown in the next picture:



Hardware board: Arduino Mega 2560

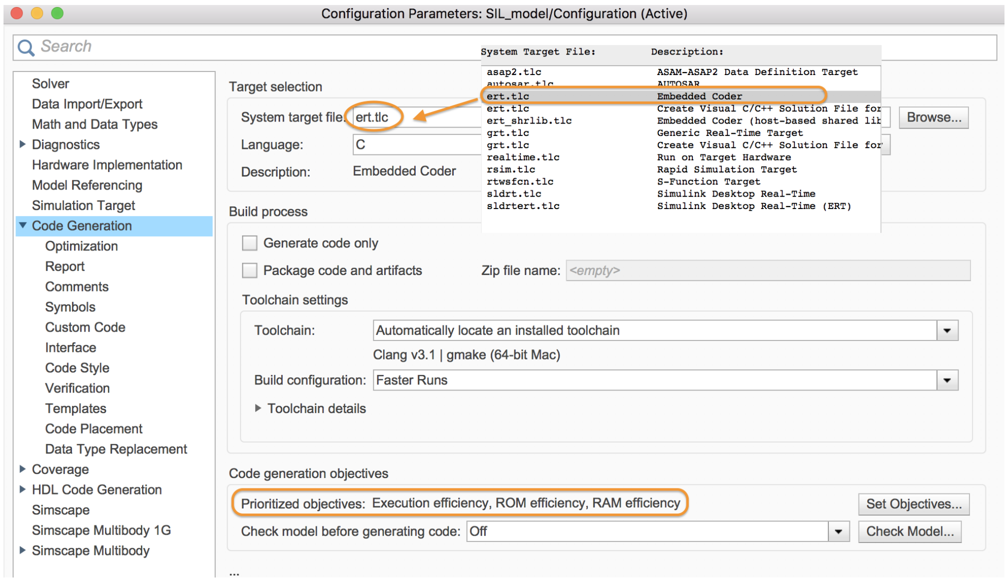
Code Generation system target file: ert.tlc

Device vendor: Atmel Device type: AVR

Device details

Figure 8.9: Hardware implementation parameters

3. *Code Generation* pane: in the present window, all the code generation parameters are set. First of all the *System target file* must be defined as *ert.tlc*, for embedded coder target. The language also should be chosen (in our case the C language), then the *code generation objectives*. Concerning the latter parameter, are selected the execution efficiency, the ROM memory and the RAM memory to be optimized.


Configuration Parameters: SIL_model/Configuration (Active)

Target selection

System target file: ert.tlc

Language: C

Description: Embedded Coder

```
System Target File:        Description:
asap2.tlc                  ASAM-ASAP2 Data Definition Target
autosar.tlc                AUTOSAR
ert.tlc                    Embedded Coder
ert.tlc                    Create Visual C/C++ Solution File for
ert_shrlib.tlc             Embedded Coder (host-based shared lib
grt.tlc                    Generic Real-Time Target
grt.tlc                    Create Visual C/C++ Solution File for
realtime.tlc               Run on Target Hardware
rsim.tlc                   Rapid Simulation Target
rtwsfcn.tlc                S-Function Target
sldrt.tlc                  Simulink Desktop Real-Time
sldrtert.tlc               Simulink Desktop Real-Time (ERT)
```

Build process

- [ ] Generate code only
- [ ] Package code and artifacts

Zip file name: <empty>

Toolchain settings

Toolchain: Automatically locate an installed toolchain

Clang v3.1 | gmake (64-bit Mac)

Build configuration: Faster Runs

Code generation objectives

Prioritized objectives: Execution efficiency, ROM efficiency, RAM efficiency

Check model before generating code: Off

Figure 8.10: Hardware implementation parameters

Configured all the necessaries model parameters, the code can be generated through the code generation tool of Simulink. The control system scheme, as usually, is the following one:

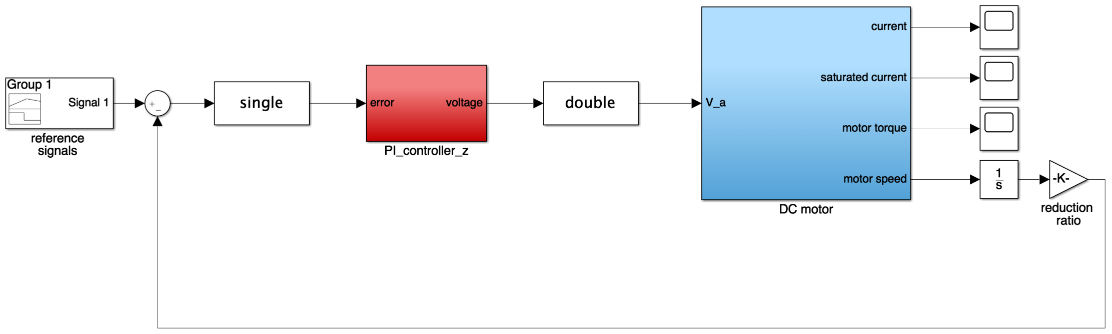
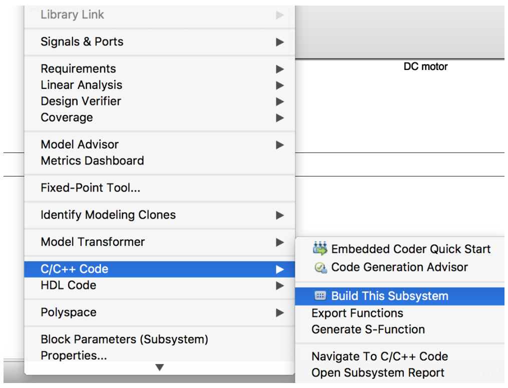
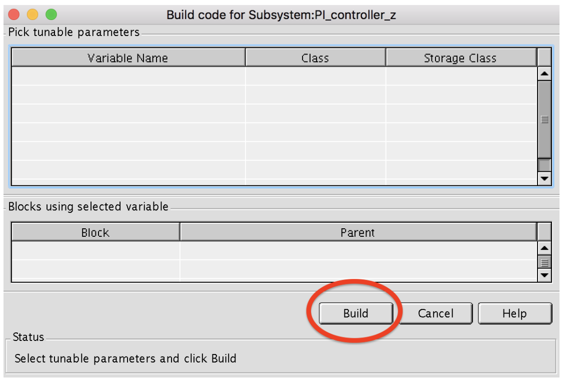


Figure 8.11: Control system scheme in normal mode simulation

With reference to fig. 8.11, the code generation is requested for the controller model block, which really must be implemented on the Arduino board. Then, by right clicking on the *PI controller* block and choosing the *C/C++ Code* option, the *Build this subsystem* button must be selected. In order to clarify what said so far, the following figures are added to the explanation:



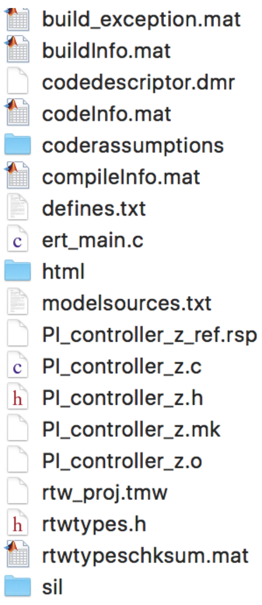
(a) Procedure to achieve the code generation


Build code for Subsystem:PI_controller_z
Pick tunable parameters
Variable Name
Class
Storage Class
Blocks using selected variable
Block
Parent
Build
Cancel
Help
Status
Select tunable parameters and click Build

(b) Build this subsystem button

Figure 8.12: Code generation steps

When the code is generated, a *Code generation report* is provided, which contains the following files:



build_exception.mat
buildInfo.mat
codedescriptor.dmr
codeInfo.mat
coderassumptions
compileInfo.mat
defines.txt
ert_main.c
html
modelsources.txt
PI_controller_z_ref.rsp
PI_controller_z.c
PI_controller_z.h
PI_controller_z.mk
PI_controller_z.o
rtw_proj.tmw
rtwtypes.h
rtwtypeschksum.mat
sil

Figure 8.13: Files provided after the code generation

Among all these files, three are the essential ones in order to guarantee a right exportation of the algorithm on the microcontroller board:

- *PI_controller_z.c*: is made of 3 main functions. The *initialize* function is the first one: it resets the model states; instead the *void* function executes a fixed integration step: here the control law is effectively implemented. Finally, through the *terminate* function, the memory is cleaned after the model evaluation;

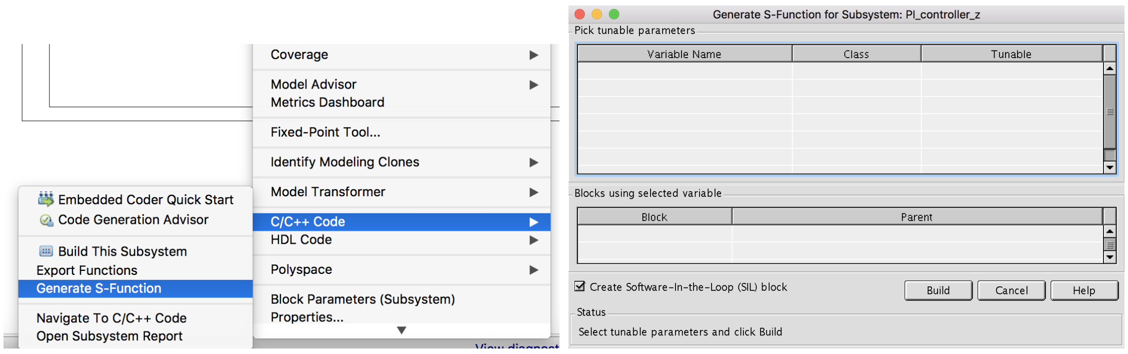
- *PI_ controller_ z.h* is made of all the *inlcude* and *define* of the code: then it contains the specifics model datas;

- *rtwtypes.h*: it translates the data type from the native simulation language into the requested by the target hardware;

- *ert_ main.c*: it represents the test branch of the model.

8.2.4 SIL validation

After the controller discretization, the data type conversion and the code generation is recommended to verify that the system and controller performances are still acceptable. To this purpose the software in the loop validation is the best technique, due to the fact that is not possible, at least in this phase, damage the target hardware. In fact, the generated code is replaced in the Simulink environment.
The setting of the model parameters doesn't change with respect to the previous *section 8.2.3*, but instead of select the option *Build this subsystem*, the *S-function* is now generated, as shown in next figure:



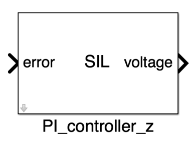
☑ Create Software-In-the-Loop (SIL) block

Select tunable parameters and click Build

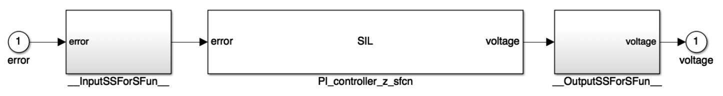
(a) (b)

Figure 8.14: S-function generation

Again the code is generated; in addition to the executable code, the controller *SIL* block is provided, within which the code is recompiled. In fact inside the block there is the S-function, i.e. a computer language representation of the controller Simulink block written in C. In this way, by means of the SIL block, the C code is replenished in simulation environment.



(a) SIL block



(b) Look under SIL block mask

Figure 8.15: Provided SIL block

Once replaced the block of fig.8.15a in the Simulink control system scheme, the resulting final scheme is:

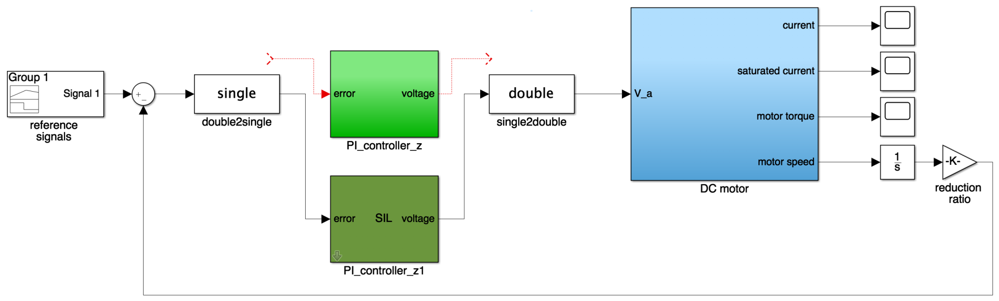
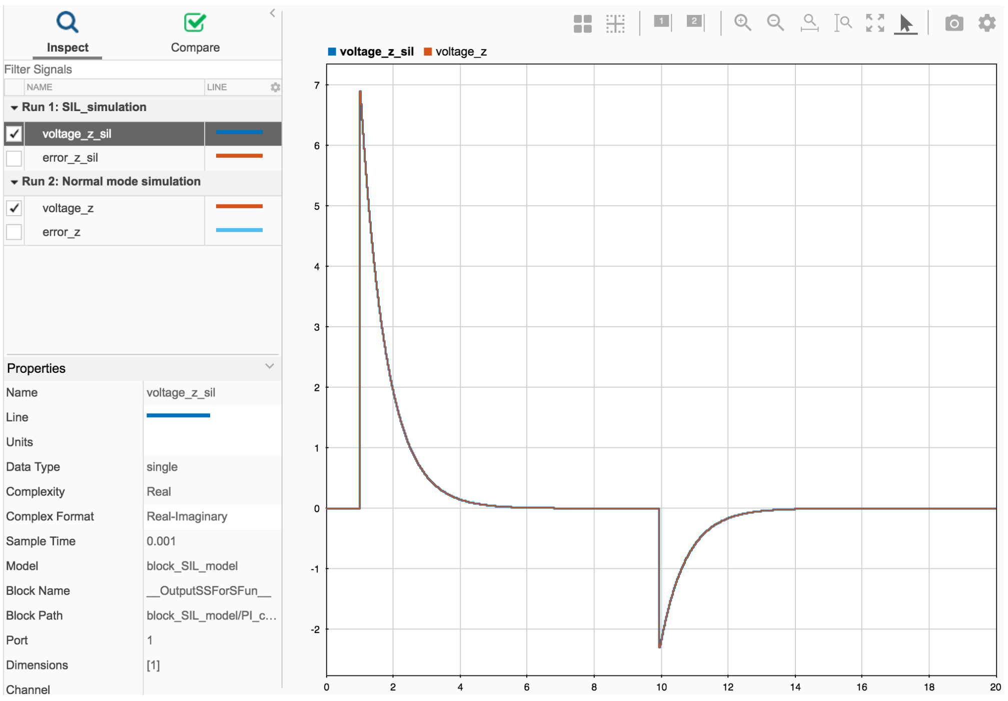


Figure 8.16: Control scheme in SIL simulation mode

The simulation can be launched, and the normal and SIL simulation results are compared thanks to the *Simulation Data Inspector* tool. Precisely the supply voltage V_a and the gimbal angular position θ_1 are of interest, because the first is the controller output while the second represents the variable that we want to control.
As known, concerning the supply voltage, it is evaluated by the controller through the proportional and integral action depending on the position error value. The simulations results are shown in the next picture:



Run 1: SIL_simulation

✓ voltage_z_sil	▬
error_z_sil	▬

Run 2: Normal mode simulation

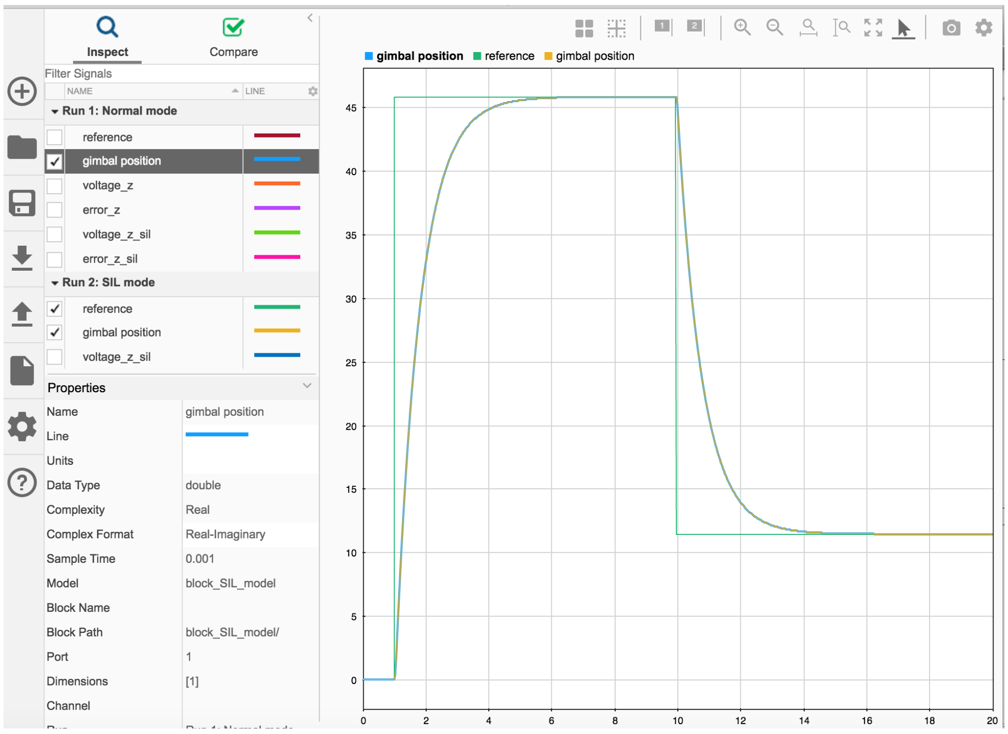
✓ voltage_z	▬
error_z	▬

Properties

Name	voltage_z_sil
Line	▬
Units	
Data Type	single
Complexity	Real
Complex Format	Real-Imaginary
Sample Time	0.001
Model	block_SIL_model
Block Name	__OutputSSForSFun__
Block Path	block_SIL_model/PI_c...
Port	1
Dimensions	[1]
Channel	

Figure 8.17: Voltage supply comparison

From 8.17 note that by linking first the controller SIL block and then the native controller block to the plant, two simulations are available in the Simulation Data Inspector tool. There is a totally overlap between the two supply voltage shapes: it means that the introduced approximations didn't deteriorate the computations.
The same is made related to the gimbal angular position and the previous considerations are still valid:



Run 1: Normal mode

☐ reference	
✓ gimbal position	
☐ voltage_z	
☐ error_z	
☐ voltage_z_sil	
☐ error_z_sil	

Run 2: SIL mode

✓ reference	
✓ gimbal position	
☐ voltage_z_sil	

Properties

Name	gimbal position
Line	
Units	
Data Type	double
Complexity	Real
Complex Format	Real-Imaginary
Sample Time	0.001
Model	block_SIL_model
Block Name	
Block Path	block_SIL_model/
Port	1
Dimensions	[1]
Channel	

Figure 8.18: Gimbal angular position comparison

8.3 Processor in the Loop Validation

In the Processor in the Loop (PIL) validation phase only the plant exists in native simulation language and still runs in the simulation environment, while the code of the controller model runs on the embedded target hardware (the microcontroller Arduino Mega 2560), as shown in the following figure.

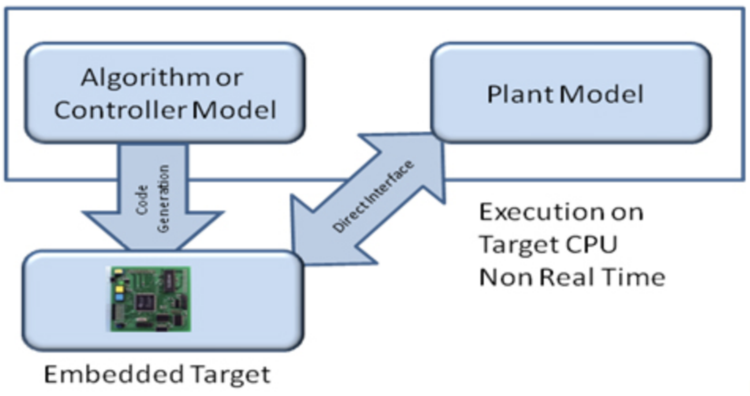
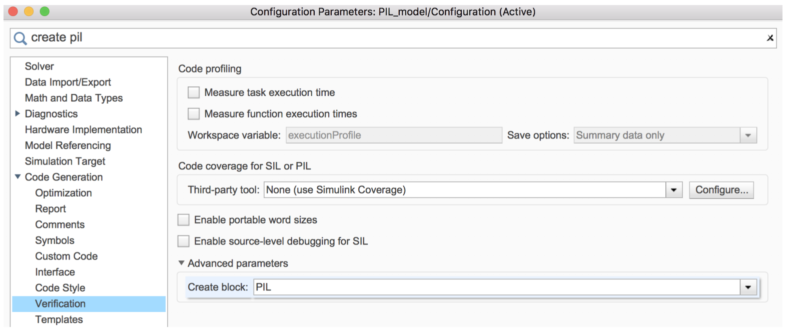

Algorithm or Controller Model
Plant Model
Code Generation
Direct Interface
Execution on Target CPU Non Real Time
Embedded Target

Figure 8.19: Processor in the loop stage

The procedure to perform a processor in the loop test is the same as done in the previous section, concerning the software in the loop validation. Also the setting of all the configuration parameters doesn't change, except the fact that is needed to explicitly require the creation of a *PIL* block with the same function of the previous SIL block. The PIL block generation happens by checking the option *Create PIL block* in the model configuration parameter pane, as:


Configuration Parameters: PIL_model/Configuration (Active)

create pil

Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Optimization
Report
Comments
Symbols
Custom Code
Interface
Code Style
Verification
Templates

Code profiling
- [ ] Measure task execution time
- [ ] Measure function execution times

Workspace variable: executionProfile Save options: Summary data only

Code coverage for SIL or PIL
Third-party tool: None (use Simulink Coverage) Configure...

- [ ] Enable portable word sizes
- [ ] Enable source-level debugging for SIL

Advanced parameters
Create block: PIL

Figure 8.20: Processor in the loop block generation

For sure, the generated PIL block is the corresponding one to the controller subsystem; in model configuration parameters the choosen *hardware board* is the Arduino Mega 2560. Then, as in the SIL validation, the option *C/C++ code* must be selected. As shown in the following figure, instead of requiring the build of the controller subsystem, the *Deploy this subsystem to hardware* possibility is choosen:

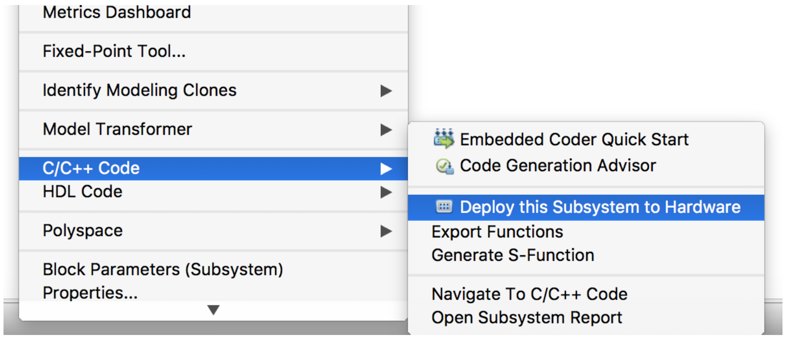


Figure 8.21: Before to run the PIL simulation

As always the *build* button appears, as in fig.8.14. After the building step, a PIL block is again provided by Simulink Embedded Coder tool, like as the SIL block in the previous section.
By means of the *Simulation Data Inspector tool* again the simulation results can be compared in order to complete the PIL validation. In fact, the first simulation starts with the original controller model linked to the plant, then it is a normal mode simulation. Instead in a second moment is the PIl block to be connected to the plant, while it is

running on the hardware. A necessary condition is to have a stable USB connection between the Arduino board and the PC before launching the simulation, because the controller runs on hardware and the datas will be transmitted from Arduino board as input to the SImulink plant model. The hardware setup is shown below:

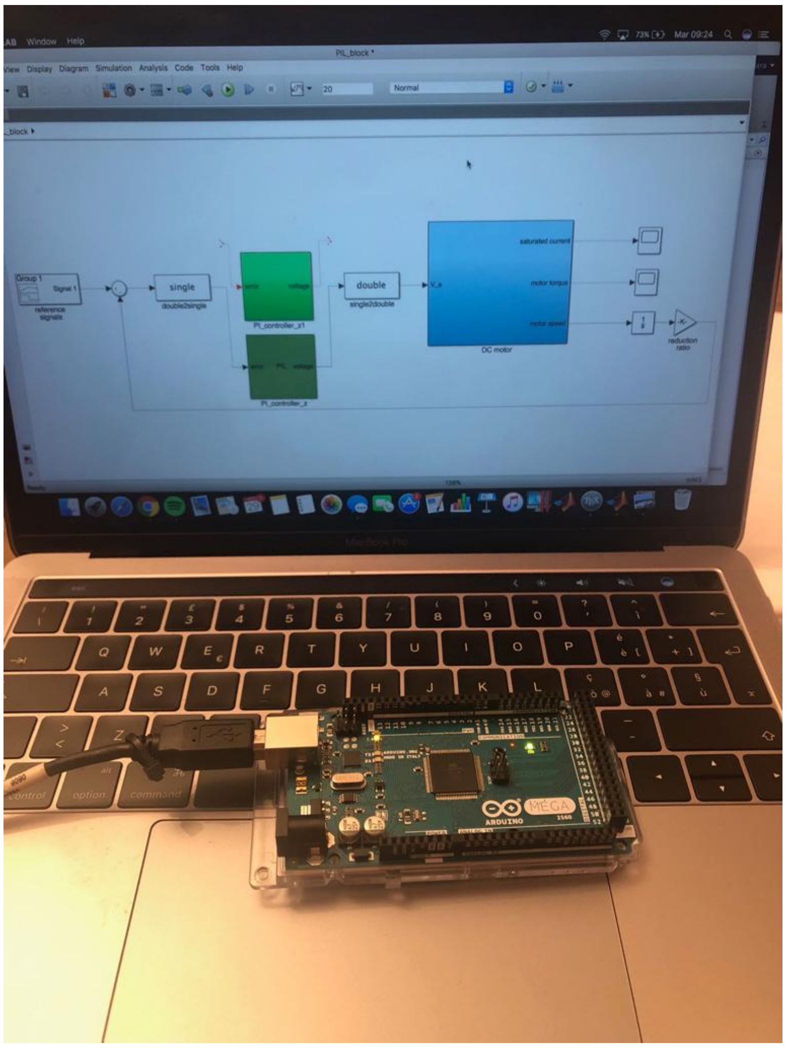


Figure 8.22: USB connection between Arduino's board and the PC

The simulated control scheme, similarly to the one illustrated in fig.8.16, is:

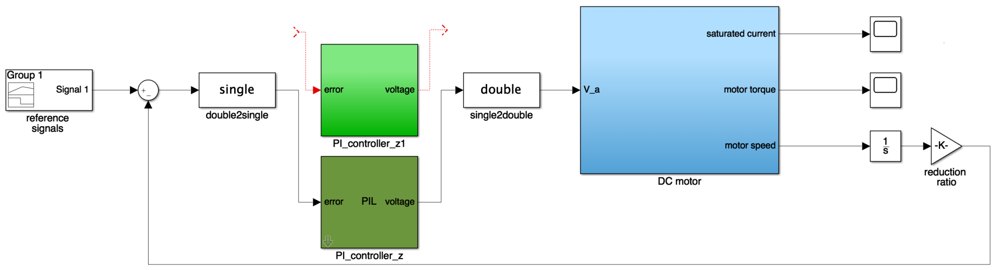


Figure 8.23: Control scheme in PIL Mode simulation

Note that, during the simulation interval time, the RX, TX leds on the Arduino's board blinked: it means that an effective serial transmission and reception of byte data between Simulink and Arduino happens.

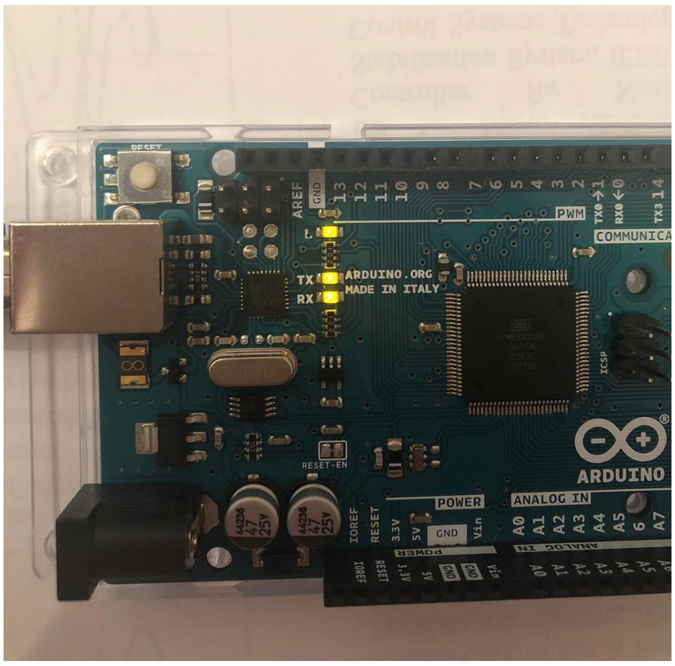


Figure 8.24: RX and TX blinking led

The simulation results are reported below:

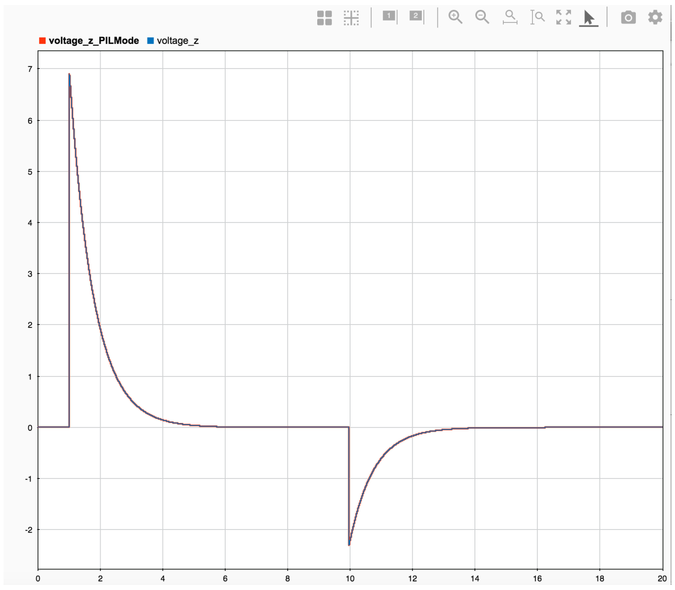


Figure 8.25: Controller output: *voltage*

As expected, the voltage values effectively evaluated by the microcontroller correspond to the ones computed in a purely simulation environment. The same can be stated concerning the gimbal angular position θ_1, which we really should control, reported in the next figure:

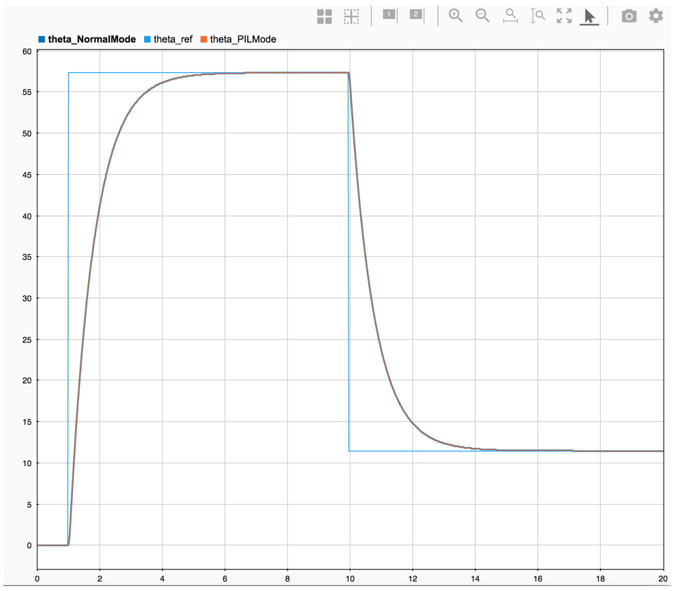


Figure 8.26: Control scheme output: θ_1

8.4 Test and Validation on the physical system

To complete the study, the final tests on the existing physical system must take place. It means understand how sensors, microcontroller and motors are connected each others; then, is needed to identify the used communication protocol to guarantee a right interpretation and evaluation of the exchanged datas.
The integration architecture is shown in the following figure:

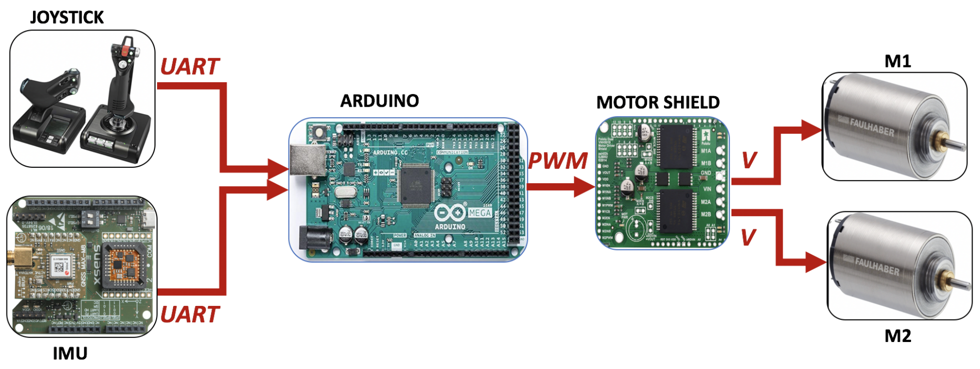


Figure 8.27: Interconnection of the peripherals architecture

1. The reference positions ($\theta_{1,ref}$ and $\theta_{2,ref}$), i.e. the angles that the gimbal should reach, are given to the microcontroller throughout the *joystick*. Instead the effective positions of the gimbal (θ_1 and θ_2) are provided by means of the *IMU*: they correspond to the feedback measurements in the designed control system
   - Both joystick and IMU establish a UART (Universal Asynchronous Receiver Transmitter) communication with Arduino's board in order to transmit or receive the datas.

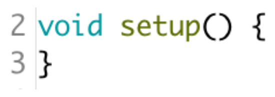
2. The controller is physically located on the *Arduino's board* and it is responsible for computing the value of the voltage supply needed to the motors actuating the gimbal system. It receives the reference and the effective angles from the jet mentioned external peripheral device, then it computes the error in terms of difference between the desired angle values and the real ones.
   - The computed voltage value will be opportunely converted into a PWM signal, then it will be transmitted to the motor shield.

3. The motor shield is a board designed to enable Arduino's board to control the bidirectional motors. It is powered through the Arduino and receives PWM signals. In this manner the PWM signals is converted into effective voltage value which supply the motors.

8.4.1 Arduino's IDE

A completely new code in C language is written in the IDE of Arduino to implement the previously designed controllers in the Simulink environment. The choice of not integrate the code automatically generated by Simulink is in order to avoid complications during the implementation on microcontroller, because some internal variables are introduced by the code generation process.
The Arduino's IDE consist of two main *void* functions:

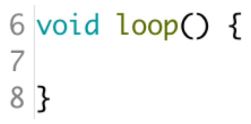
1. the *setup* function which appear as:



```
void setup() {
}
```

   Here are passed to Arduino all the necessary informations before the program execution. For example, the input or output ports are configured, or the serial communications are established. In the setup function also some command can be written but all the tasks inside the actual function are executed only once, i.e. when the Arduino is powered up.

2. the *loop* function,



```
void loop() {

}
```

   It contains all the routines performing the desired task. As the function name suggests, these commands are repeatedly executed, at defined intervals of time given via the *delay* command. Inside the loop function, the code is sequentially run.

8.4.2 Controllers implementation on the microcontroller

In order to manage in a proper way the received datas and to perform a correct motion control, the Arduino must be setup. To this scope, a sketch in the Arduino's IDE has been developed, and step by step exploited below.

Arduino set up: the setup function

As anticipated, the inputs to the board come from the joystick and IMU sensor. They are transmitted based on serial communication protocols and are certainly of the *float* data types. In the next picture are highlighted the chosen pins through which the UART (or serial) transmissions should happen. In details, the TX1/RX1 couple is set as the

communication port between Arduino and the IMU; instead the TX2/RX2 couple will receive angle values provided by the joystick.
The Arduino's outputs are the voltage values which are computed by the microcontroller and are transferred to the motor shield. Since the latter only accepts PWM signals, is mandatory to set, among all the pins, two output pins configurable as PWM ones.

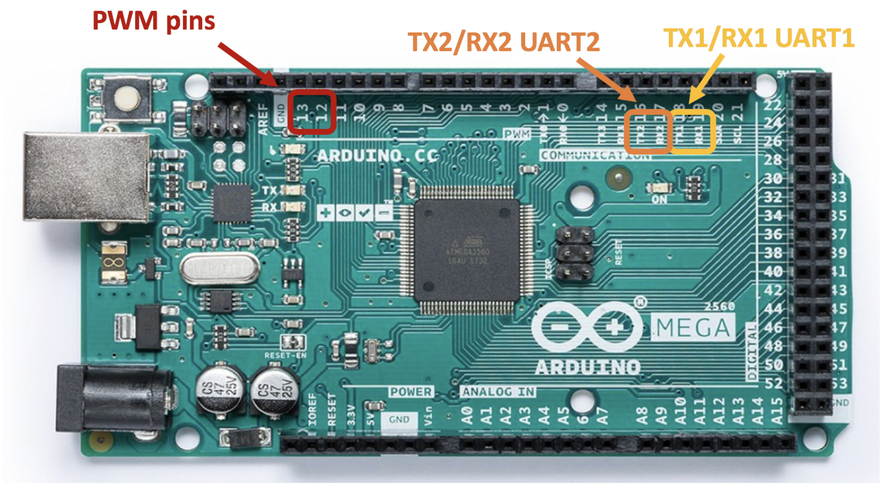


Figure 8.28: Configured I/O pins on Arduino's board

Below is shown the C code corresponding to the previous explanation.

```

// Configure PWM pins
int V_azimuth = 12;
int V_elevation = 13;
```

Outside from the setup function, a name is associated to each PWM pin: the pin 12 is named *V_ azimuth* to indicate that on it will be written the PWM voltage value to control the rotation around the azimuth axis. For the same reason, pin 13 is associated to the name *V_ elevation*.

```
28 void setup() {
29   Serial1.begin(115200);            //Start the communication with the IMU sensor
30   Serial2.begin(115200);            //Start the communication with the joystick
31   pinMode (V_azimuth, OUTPUT);
32   pinMode (V_elevation, OUTPUT);
33 }
```

- line 29-30: the command *Serial1.begin* enable the UART serial data transmission between Arduino and the IMU. In the brackets the baud rate, also known as bps, is specified: *115200* indicates that the serial port is able to transfer maximum 115200 bits per second. Through *Serial2.begin* the UART communication with the joystick is opened;

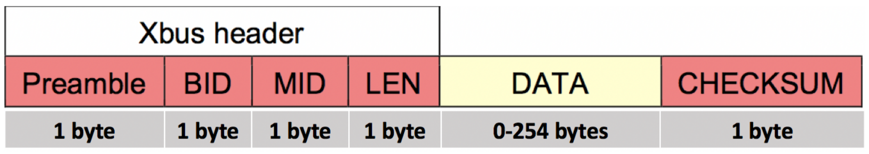
- line 30-31: the *pinMode(pin, mode)* allows to configure the chosen pin as output.

The loop function

Inside the loop function the incoming datas are opportunely converted in data types usable by the microcontroller to perform the required computations. Then the control algorithm is developed and the output values are written on the specified output pins.

- Serial transmission Arduino/IMU

Concerning the angle values provided by the IMU, with a frequency of 100 Hz a message is received by the Arduino. In particular, the IMU sends a string, called **MTData2**, containing several informations including the Euler angles. The standard MTData2 message sent by the IMU Xsens, is structured as below:



Xbus header					
Preamble	BID	MID	LEN	DATA	CHECKSUM
1 byte	1 byte	1 byte	1 byte	0-254 bytes	1 byte

Figure 8.29: MTData2 message structure

By the previous figure, appears that the Euler angles are the datas we're interested to and they're contained in the *Data* field. Then, every 10 ms, Arduino receives a message: the board must read the incoming string and has to skip the first 4 bytes in order to extract the informations dealing with the angles. Outside from the loop function some declarations are made in order to read the message coming from the IMU peripheral:

```
 6 // To read the IMU string
 7 String IMU_raw, roll_raw, pitch_raw, yaw_raw;
 8 char c;
 9 float roll, pitch, yaw;
10 bool IMU_rx;
```

- line 7: the string *IMU_ raw* is defined in order to allow Arduino to read the incoming string from the board buffer and to save it into the declared string. In addition, three substrings are defined as *roll_ raw, pitch_ raw, yaw_ raw* to associate the single Euler angle to the corresponding substring;

- line 9: since the Euler angles exist as *char* data types into the substrings is needed to convert them into the *float* data types. For this reason three float variables (*roll, pitch, yaw*) must be declared;
- line 10: a boolean variable *IMU_rx* is defined, to indicate that new bytes has been received.

Inside the loop function, the code lines performing the reading action, are:

```
33 void loop() {
34   IMU_raw = "";
35   IMU_rx = false;
36
37   while (Serial1.available()) {
38     c = Serial1.read();
39     IMU_raw += c;
40     IMU_rx = true;
41   }
```

- line 34-35: the string *IMU_raw* is initialized as empty so that it's filled by reading the incoming bytes; the boolean variable *IMU_rx* is set as *false*, so if there are available datas on the serial port it becomes true;
- line 37-40: by means of the *while* the serial buffer is read. The condition *(Serial1.available)* guarantees that new datas are sent by the IMU sensor, and consequentially the string *IMU_raw* is filled. At the same time, the boolean variable changes its state.

The *IMU_raw* string, with reference to fig.8.29, contains the preamble, the bus identifier (BID), the message identifier (MID), the length of the message itself (LEN) fields. For sure these are informations of not relevance for the motion control goal. Then, these fields can be skipped, to directly extract the Euler angles: this is done through the *.substring* function.

```
43
44   roll_raw = IMU_raw.substring(4,8);
45   roll = roll_raw.toFloat();
46   IMU[0] = roll;
47
48   pitch_raw = IMU_raw.substring(8,12);
49   pitch = pitch_raw.toFloat();
50   IMU[1] = pitch;
51
52   yaw_raw = IMU_raw.substring(12,16);
53   yaw = yaw_raw.toFloat();
54   IMU[2] = yaw;
```

From line 44 to line 54 the parsing of the *IMU_ raw* string takes place. In detail, is known that:

- the first 4 bytes of the *IMU_ string* can be skipped because they correspond to the preamble, BID, MID, LEN fields;
- the message coming from the IMU sensors is made of 3 Euler angles, which are float data types. Since 1 float occupies 4 bytes, then the *Data* field is made of 12 bytes: the roll angle value is represented through the bytes of the MTData2 from 4 to 8; the pitch angle is expressed by the bytes from 8 to 12; the yaw angle corresponds to the bytes from 12 to 16 of the IMU's message;
- by using the substring function, according to what said at the previous point:

  $$roll\_raw = IMU\_raw.substring(4, 8);$$

  The same is done for the pitch and yaw angles.
- from *char* data type each angle is converted in a *float* data via the *.toFloat* function, as in lines 45,49,53;
- to simplify the error computation (the input to the controller) a float vector *IMU* is generated and filled after the conversion process.

- Serial transmission Arduino/joystick

The angle values taken as references, are transmitted from the joystick to Arduino as *float* datas. Then, the microcontroller only has to read 3 float value which should be saved in a vector of three components. Outside the loop function:

```
//To read the joystick angle values
bool ref_rx;
float ref[3];
```

Also the boolean variable *ref_ rx* is introduced, in order to be set as *true* when new data available are sent to the Serial2 port. Then, inside the loop function:

```
72   if (Serial2.available()) {
73     for (int i = 0; i < 3; i++) {
74       ref[i] = Serial2.parseFloat();
75       ref_rx = true;
76     }
77   }
```

- line 72: the *Serial2.available()* function guarantees that new datas are sent to the serial communication port;

- line 73-74: through the *for* cicle the float values are read on the RX2 and saved into the vector *ref*;
- line 75: the boolean variable is set to *true* due to the reception of the datas;

- Prepare the inputs of the controllers

Once the board has received and converted opportunely the angle values coming from the external peripherals, is necessary to compute the angle error between the desired and the effective ones. In fact, each implemented controller expects the error value to estimate the supply voltage for the DC motor. Then, throughout a *for* loop, the error vector is computed, as shown below:

```
76
77  if (IMU_rx == true || ref_rx == true) {
78    for (int i = 0; i < 3; i ++) {
79      err[i] = ref[i] - IMU[i];
80    }
81  }
82
83  err_azimuth = err[2];
84  err_elevation = err[1];
85
86  Vz_al = Controller_azimuth(err_azimuth);
87  Vy_al = Controller_elevation(err_elevation);
```

- note the presence, in line 77, of an *if condition* depending on the value assumed from the boolean variables. It means that the new error vector is recomputed every time a reference value is received, or whenever the balance of the gimbal changes and the IMU notifies the swing;
- the error vector is made of 3 components, corresponding to the roll, pitch and yaw error respectively. The errors of interest for us concern the pitch and yaw angles (to actuate the tilt and pan correction);
- the error value on azimuth axis and on the elevation axis coincide with the inputs of the controllers. Then, two functions *Controller_ azimuth* and *Controller_ elevation* are created, outside the loop function;
- the lines 86-87 represent the call to the controller functions, having as inputs the errors and providing as output the two supply voltages *Vz_ al* and *Vy_ al*.

- Controller functions

When the controller functions are invoked, the control laws are computed. In each controller function the algorithm of a PI controller is implemented: the explanation is provided for the motion control law on azimuth axis, but is also valid for the tilt control.

```
113 float Controller_azimuth (float err_azimuth) {
114   float int_err_z = 0;
115   currentTime_z = millis();
116   elapsedTime_z = currentTime_z - lastTime_z;
117   int_err_z = err_z * elapsedTime_z;
118 
119   Vz_al = ((kp_z * err_z) + (ki_z * int_err_z)) / 1000.0;
120 
121   lastTime_z = currentTime_z;
122   return Vz_al;
123 }
124 
125 float Controller_elevation (float err_elevation) {
126   float int_err_y = 0;
127   currentTime_y = millis();
128   elapsedTime_y = currentTime_y - lastTime_y;
129   int_err_y = err_y * elapsedTime_y;
130 
131   Vy_al = ((kp_y * err_y) + (ki_y * int_err_y)) / 1000.0;
132 
133   lastTime_y = currentTime_y;
134   return Vy_al;
135 }
```

The *Controller_ azimuth* function expects as input the value of the error on the pan axis (which is a float), and returns the float variable corresponding to the supply voltage. As known, the expression of a PI controller is given by:

$$C(s) = K_P \cdot e(t) + K_I \cdot \int_0^t e(t)dt$$

Then, is necessary to know the error value (*error_ azimuth*) and to compute the integral error (*int_ err*) in order to rightly compute the proportional and integral actions. Being the integral of the error the an accumulated error signals since the start, the line through which the integral error is evaluated is the 117^{th}, clarified below:

- the integral error is computed as the error multiplied by the elapsed interval of time;
- the function *millis()* returns the number of the millisecond passed since board runs the actual program. The time value is saved into the *currentTime* variable. When the task is performed the unsigned long *lastTime* is update with the currentTime value;

- known the two previous time instant the integral error can be estimated (line 117);
- the output of the controller can be also compute (line 119) and scaled of a factor of 1000, because the controller works with [ms] and we need the voltage value in [V].

- **Write on the output pins**

Obtained the voltages as float data type, is mandatory to convert them so that they're in the range of values expected by the motor shield. As mentioned, the Arduino's outputs coincide with the Motor shield's inputs, which only accepts PWM signals in the range between -255 and +255.
With the awareness that the maximum error concerning the yaw angle is of 360°, and the maximum error relative to the pitch angle is of 180°, the maximum voltage values are computed and normalized with respect to 255. For this reason the values to write on the PWM configured pins are defined as in lines 104-105.

```
104  Vz_PWM = Vz_al * 2.67;
105  Vy_PWM = Vy_al * 5.24;
106
107  analogWrite(Vz, Vz_PWM);
108  analogWrite(Vy, Vy_PWM) ;
```

Thanks to the Arduino's function *analogWrite(pin, value)* the voltage values are passed to pin 12 and 13, being called Vz and Vy respectively.
In this way the written C code for the implementation phase is completely illustrated, so all the peripherals can be integrated with the Arduino's boards, which is ready to use in the proper manner all the incoming datas.

Chapter 9

DC motor survey

The development of the controller in *chapter 6* had the aim to conduct a survey concerning the DC motors. In fact, the final goal is to identify some DC motors ables to provide to the modeled gimbal system the required torques, powers and angular velocities in the most critical operating conditions. The control system chosen to be simulated is the one containing the controller obtained by means of the loop-shaping techniques, because of the latter design seems to be more accurate in the results with respect to the LQR method.

9.1 Operating conditions and motor requirements

The control scheme of fig. 6.12 is then simulated by means of Simulink tool, under the following reference inputs:

- $\theta_{1,ref} = 180°$, due to the fact that from experimental considerations the rotation around the azimuth axis should not exceed this given range;
- $\theta_{2,ref} = 90°$, due to the same considerations explained in the previous row.

Once defined the most critical operating conditions, the given specifications in terms of time response and overshoot are always the same: in fact if they are satisfied when 180° and 90° are required as references, they will be always fulfilled during a fly, for whatever commanded rotation. However the static and dynamic specifications are listed below:

- $t_{s,2\%} \leq 4s$;
- $t_r \leq 5s$;
- $\hat{s} = 0\%$;
- $|e_r^\infty| = 0$.

In order to start the motor survey, some needed technical parameters should be identified, i.e. required torque, power and the maximum angular velocities that the chosen motor can achieve. For this aim must pay attention to the presence, in the real gimbal physical system, of a pulley between the motor and the load: it introduces a reduction ratio $n_{pul} = 6$. Since n_{pul} increases the required angular velocities of 6 times and decreases the required torque value of 6 times, the collected values from simulations are converted of a factor n_{pul}.
The feedback control scheme is simulated, and the following are the results related to the desired torques and angular velocities at the output of the reducer:

1. dealing with the motion around azimuth axis:
   - required torque: $\tau_1 = 10$ [N $\cdot$ m] $\implies \dfrac{\tau_1}{n_{pul}} \simeq 116$ [mNm];
   - required angular velocity: $\dot{\theta}_{1,max} = 116$ [°/s] $\implies \dot{\theta}_{1,max} \cdot n_{pul} \simeq 116$ [RPM];
2. dealing with the motion around elevation axis:
   - required torque: $\tau_2 = 1$ [N $\cdot$ m] $\implies \dfrac{\tau_2}{n_{pul}} \simeq 0.116$ [mNm];
   - required angular velocity: $\dot{\theta}_{2,max} = 71$ [°/s] $\implies \dot{\theta}_{2,max} \cdot n_{pul} \simeq 71$ [RPM];

As clear, the most critical conditions are represented by the motion around the azimuth axis, then those are chosen as technical parameters of the DC motor. Since the physical system jet exists and has certain dimensions, all the features of the DC motor to look for are listed in the following table:

Required torque	τ	115 [mNm]
Required speed	n	7000 [min^{-1}]
Diameter	d	26 [mm]
Length	L	63,2 [mm]

Table 9.1: DC motor needed technical parameters

9.2 Faulhaber motors

The first step of the suitable DC motor research concerns the *Faulhaber* motors. Faulhaber company is committed in the production of motion control system offering advanced micro driver technologies. The Faulhaber's drive solution are optimally designed in order to achieve performances as possible closed to the desired needs, taking into consideration also the reduced spaces in which the drivers will be probably placed.

Starting from the requirements reported in tab. 9.1, the next recommended step is to evaluate the expected power which should be provided by the chosen DC motor:

$$\begin{aligned} P_{output} = \tau \cdot 2\pi n &= 115[\text{mNm}] \cdot 2\pi \cdot 7000[\text{min}^{-1}] \\ &= 0.115[\text{Nm}] \cdot 733.03[\text{rad/s}] = 85\text{W} \end{aligned}$$

Once evaluated the necessary power, bearing in mind that the gimbal should be actuated through motors able to work in *continuous operation* mode, below are listed the suitable Faulhaber's series DC motor:

- DC motors:
  - 3272...CR;
  - 3863...CR;
- Brushless DC motors(4 poles technology):
  - 4490...B;
  - 3268...BX4;
  - 3274...BP4;

9.2.1 DC motors CR Graphite Communtation Series

3272024CR DC motor:

$$\begin{cases} P = 85W \\ \tau = 120mNm \end{cases}$$

Technical features:

Nominal voltage	V	24 [V]
Terminal resistance	R	0.82 [Ω]
Rotor Inductante	L	185 [μH]
Rotor inertia	J	63 [g·cm^2]
Torque constant	K$_m$	41.6 [mNm/A]
Back-EMF constant	K$_E$	4.35 [mV/min^{-1}]
Speed up to	n$_{max}$	6000 [min^{-1}]
Nominal speed	n	5150 [min^{-1}]
Diameter	d	32 [mm]
Length	L	94 [mm]

Table 9.2: 3272024CR DC motor technical parameters

3863024CR DC motor:

$$\begin{cases} P = 110W \\ \tau = 131mNm \end{cases}$$

Technical features:

Nominal voltage	V	24 [V]
Terminal resistance	R	0.64 [Ω]
Rotor Inductante	L	180 [μH]
Rotor inertia	J	120 [g·cm^2]
Torque constant	K$_m$	39.8 [mNm/A]
Back-EMF constant	K$_E$	4.17 [mV/min^{-1}]
Speed up to	n$_{max}$	7000 [min^{-1}]
Nominal speed	n	5510 [min^{-1}]
Diameter	d	38 [mm]
Length	L	106 [mm]

Table 9.3: 3863024CR DC motor technical parameters

9.2.2 Brushless DC motors Series

3274024BP4 Brushless DC motor

$$\begin{cases} P = 150W \\ \tau = 165mNm \end{cases}$$

Technical features:

Nominal voltage	V	24 [V]
Terminal resistance	R	0.25 [Ω]
Rotor Inductante	L	60 [μH]
Rotor inertia	J	48 [g·cm^2]
Torque constant	K$_m$	28.04 [mNm/A]
Back-EMF constant	K$_E$	2.97 [mV/min^{-1}]
Speed up to	n$_{max}$	16000 [min^{-1}]
Nominal speed	n	8700 [min^{-1}]
Diameter	d	32 [mm]
Length	L	94.7 [mm]

Table 9.4: 3274024BP4 Brushless DC motor technical parameters

4490024B Brushless DC motor

$$\begin{cases} P = 232W \\ \tau = 190mNm \end{cases}$$

Technical features:

Nominal voltage	V	24 [V]
Terminal resistance	R	0.22 [Ω]
Rotor Inductante	L	73 [μH]
Rotor inertia	J	130 [g·cm^2]
Torque constant	K$_m$	24.2 [mNm/A]
Back-EMF constant	K$_E$	2.53 [mV/min^{-1}]
Speed up to	n$_{max}$	18000 [min^{-1}]
Nominal speed	n	9700 [min^{-1}]
Diameter	d	44 [mm]
Length	L	115 [mm]

Table 9.5: 4490024B Brushless DC motor technical parameters

9.2.3 Compatible Gearheads and Encoders

The compatible gearheads and encoders are listed below. The symbol × affirms that one specific component is compatible with the selected motor.

Gearheads	Compatibility	
	3272024CR	3863024CR
32A	×	
32ALN	×	
32/3	×	
32/3 S	×	
38A	×	×
38/1	×	×
38/1 S	×	×
38/2	×	×
38/2 S	×	×
44/1	×	×

Table 9.6: Gearheads compatibility tabel

Encoders	Compatibility	
	3272024CR	3863024CR
IE3-1024	×	×
IE3-1024L	×	×
IERS3-500	×	×
IERS3-500L	×	×
IER3-10000	×	×
IER3-10000L	×	×

Table 9.7: Encoders compatibility tabel

For the brushless DC motors there is a total compatibility also.

Chapter 10

Conclusions and further works

The gimbal electromechanical system will be mainly employed for a target tracking application. In order to achieve this functionality, a lot of work is still left to do.
First of all, the generated C code must be loaded on the Arduino board in order to be tested on the real HW and consequently optimized, if needed. The electrical wires supplying the microcontroller and the external peripherals are already joined properly; the cables enabling the data transmission between all sensors, the Arduino and the motors are also successfully interfaced. After this step, the physical prototype is ready to be validated.
If the controller works as expected, the next phase entails on integrating the control law with an image acquisition algorithm, in charge of analyzing the camera recordings. Therefore, given a sequence of images, the goal is to track a single target (which can be a car or a truck) by centering the target in the view. Finally, the image acquisition process should provide with a variable identifying an error as output. The latter is the controller input variable, i.e. the distance between the target position into the picture of the camera and the central point in the image view. In this way, the reference coordinates are automatically generated throughout the image acquisition algorithm, which is a totally pilot non-dependent and automated process. As a further work, the possibility to extend both detection and tracking for the several classes of objects could be explored.
The tracking system idea introduces another issue, the so-called trajectory planning. In fact, the trajectory planner is responsible of converting the signals coming from external devices into reference signals that the gimbal controller can use. The gimbal motion control, or in general the control in the robotics field, is strictly dependent on the trajectory planner part of the system, since a coordinates conversion between local and global frames is necessary.

Appendix A

Loop-shaping scripts

Main of the loop shaping design script

```
clear all
close all
clc

s=tf('s');

%geometric parameters
J1z=0.0817408;           %[kg*m^2]
J2x=0.01174741;          %[kg*m^2]
J2y=0.008261759;         %[kg*m^2]
J2z=0.01162105;          %[kg*m^2]
m1=4.87;                 %[kg]
m2=1.957;                %[kg]
g=9.81;                  %[m/s^2]
d3=500;                  %[m]
fv1=0.2;                 %dimensionless
fv2=0.002;               %dimensionless

A33=-fv1/(J1z+J2z);
A44=-fv2/(J2x);

A=[0 0 1 0;
   0 0 0 1;
   0 0 A33 0;
   0 0 0 A44];
B=[0 0 1/(J1z+J2z) 0;
   0 0 0 1/(J2x)]';
C=eye(4);
D=zeros(4,2);

sys_2axis=ss(A,B,C,D);
```

```
%extract the plant t.o.f.
[NUM1,DEN1]=ss2tf(A,B,C,D,1);
sys11=tf(NUM1(1,1:5),DEN1);
sys12=tf(NUM1(2,1:5),DEN1);
sys13=tf(NUM1(3,1:5),DEN1);
sys14=tf(NUM1(4,1:5),DEN1);
SYS_Z=zpk(minreal([sys11;sys12;sys13;sys14]));

[NUM2,DEN2]=ss2tf(A,B,C,D,2);
sys21=tf(NUM2(1,1:5),DEN2);
sys22=tf(NUM2(2,1:5),DEN2);
sys23=tf(NUM2(3,1:5),DEN2);
sys24=tf(NUM2(4,1:5),DEN2);
SYS_Y=zpk(minreal([sys21;sys22;sys23;sys24]));

%Observability and Reachability
Mo=obsv(A,C);
rho_obs=rank(Mo);

Mr=ctrb(A,B);
rho_mr=rank(Mr);

%requirements
ts=5;
tr=4;
zed=0.7;

%requirements translation
wn_tr=pi-(acos(zed))/(tr*sqrt(1-zed^2));
wn_ts=-log(0.02)/(ts*zed);
wc_tr=wn_tr*sqrt(sqrt(1+4*zed^4)-2*zed^2);
wc_ts=wn_ts*sqrt(sqrt(1+4*zed^4)-2*zed^2);
wc_d=max(wc_tr,wc_ts);

%chosen parameters md and wn for both the controllers
mdz=16;
wndz=0.25;

mdy=16;
wndy=0.3;

%function calls
C_z= control_z (tr, ts, zed, mdz, wndz);
C_y= control_y (tr, ts, zed, mdy, wndy);
```

Function to compute motion controller on the azimuth axis

```
function C_z = control_z (ts, tr, zed, mdz, wndz)

%initial loop function and its Nichols plot
L_in=sys;
figure(1), myngridst(Tp0,Sp0), hold on, nichols(L_in)

%Lead network design and the new Nichols plot
zdz=wc_d/wndz;
Rdz=1+(s/zdz)/(1+(s/(mdz*zdz)));
L1=L_in*Rdz;
hold on, nichols(L1)

%Magnitude attenuation and final Nichols plot
[m,p]=bode(L1,wc_d);
kc_new=(1/m);
L_lead=kc_new*L1;
hold on, nichols(L_lead);

%Azimuth rotation controller
C_z=kc_new*Rdz;
L=C_z*sys;
W=L/(1+L);
figure, step(W)
end
```

Function to compute motion controller on the elevation axis

```
function C_y = control_y(ts, tr, zed, mdy, wndy)

%initial loop function and its Nichols plot
p=pole(sys);
C_in=(s-p(2))/(s+1.5);
L_in=sys*C_in;
figure(1), myngridst(Tp0,Sp0), hold on, nichols(L_in)

%Lead network design and the new Nichols plot
md=16;
wnd=0.5;
zd=wc_d/wnd;
Rd=1+(s/zd)/(1+(s/(md*zd)));
L1=L_in*Rd;
hold on, nichols(L1)

%Magnitude attenuation and final Nichols plot
[m,p]=bode(L1,wc_d);
kc_new=(1/m);
L_lead=kc_new*L1;
hold on, nichols(L_lead)

%Azimuth rotation controller
C_y=C_in*Rd*kc_new;
L=C_y*sys;
W=L/(1+L);
figure, step(W)
end
```

Appendix B

LQ Regulator script

```
clear all
close all
clc

s=tf('s');
Ts=0.001;

%geometric parameters
J1z=0.0817408;              %[kg*m^2]
J2x=0.01174741;             %[kg*m^2]
J2y=0.008261759;            %[kg*m^2]
J2z=0.01162105;             %[kg*m^2]
m1=4.87;                    %[kg]
m2=1.957;                   %[kg]
g=9.81;                     %[m/s^2]
d3=500;                     %[m]
fv1=0.2;                    %dimensionless
fv2=0.002;                  %dimensionless

A33=-fv1/(J1z+J2z);
A44=-fv2/(J2x);

%state-space matrices
A=[0 0 1 0;
   0 0 0 1;
   0 0 A33 0;
   0 0 0 A44];
B=[0 0 1/(J1z+J2z) 0;
   0 0 0 1/(J2x)]';
C=eye(4);
D=0;

sys_c=ss(A,B,C,D);
```

```
%observability and controllability
Mo=obsv(A,C);
rho_obs=rank(Mo);

Mr=ctrb(A,B);
rho_mr=rank(Mr);

%discretization
sys_d=c2d(sys_c,Ts,'zoh');
A_d=sys_d.a;
B_d=sys_d.b;
C_d=sys_d.c;
D_d=sys_d.d;

C_pos=[1 0 0 0;
       0 1 0 0];

C_vel=[0 0 1 0;
       0 0 0 1];

%QR design parameters
R=diag([0.01 0.01]);
Q=diag([100 100 10 10 1 1]);

%augmented state system matrices
A_tot=[eye(2) -Ts*C_pos;
       zeros(4,2) A_d];
B_tot=[zeros(2,2);
        B_d];

k_lqr=dlqr(A_tot,B_tot,Q,R);
ki=[k_lqr(:,1) k_lqr(:,2)];
ko=[k_lqr(:,3:6)];
x0=[pi/2;pi/4;0;0];

%observer LTI system
C_obsv=C_pos;
lambda_obsv=[0.4 0.41 0.42 0.43];
L=place(A_d',C_obsv',lambda_obsv)';
sys_obs=ss(A_d-L*C_obsv,[B_d L],eye(4),0,Ts);
```

Appendix C

PI scripts

PI azimuth rotation controller

```
clear all
close all
clc

s=tf('s');
Ts=0.001;                        %sampling time

%Electromechanical parameters of the DC motor
n=159*6;                         %reduction ratio
Ra=1.46;                         %[ohm]
La=135*1e-6;                     %[microH]
fv=1.7*1e-6;                     %dimensionless
J=0.0000012;                     %[kgcm^2]
Jl_z=0.0817408;
Jeff_z=J+(Jl_z/(n^2));
fric_z=fv+(0.2/(n^2));
kt=0.0185;                        %[mNm/A]
kem=1.945*(pi/30)*1e-3;

%PI parameters choice
kp=20;
ki=0.02;

G_pos_z=zpk(kt/(s*((J*s+fv)*(La*s+Ra)+kt^2)));
G_elect_z=1/(Ra);
G_mech_z=1/(J*s+fv);
G_vel_z=zpk(kt/(((J*s+fv)*(La*s+Ra)+kt^2)));
p=pole(G_vel_z);

G_eff_z=zpk(1/(Jeff_z*s+fric_z));
G_pos_eff_z=zpk(kt/(s*((Jeff_z*s+fric_z)*(La*s+Ra)+kt^2)));
Gp=zpk(kt/(s*((Jeff_z*s+fric_z)*(Ra)+kt^2)));
```

PI elevation rotation controller

```
clear all
close all
clc

s=tf('s');
Ts=0.001;                     %sampling time

%Electromechanical parameters of the DC motor
n=159*6;                      %reduction ratio
Ra=1.46;                      %[ohm]
La=135*1e-6;                  %[microH]
fv=1.7*1e-6;                  %dimensionless
J=0.0000012;                  %[kgcm^2]
Jl_y=0.008261759+0.01162105;
Jeff_y=J+(Jl_y/(n^2));
fric_y=fv+(0.02/(n^2));
kt=0.0185;                     %[mNm/A]
kem=0.0185;

%PI parameter choice
kp=25;
ki=0.02;

G_pos_y=zpk(kt/(s*((J*s+fv)*(La*s+Ra)+kt^2)));
G_elect_y=1/(La*s+Ra);
G_mech_y=1/(J*s+fv);
G_vel_y=zpk(kt/(((J*s+fv)*(La*s+Ra)+kt^2)));
p=pole(G_vel_y);

G_eff_y=zpk(1/(Jeff_y*s+fric_y));
G_pos_eff_y=zpk(kt/(s*((Jeff_y*s+fric_y)*(La*s+Ra)+kt^2)));
```

Bibliography

[1] Erhan Poyrazoglu. Detailed modeling and control of a 2-dof gimbal system, 2017.

[2] L. Villani G. Oriolo B. Siciliano, L. Sciavicco. *Robotics Modeling,Planning and Control.* Springer, 2010. ISBN 978-1-84628-641-4.

[3] M. Vidyasagar Mark W. Spong, Seth Hutchinson. *Robot Dynamics and Control.* John Wiley and Sons, second edition, 2004. ISBN 978-0471649908.

[4] Viboon Sangveraphunsiri Prasatporn Wongkamchang. Control of inertial stabilization systems using robust inverse dynamics control and adaptive control. *Journal of Low Power Electronics*, 13, April-June 2008.

[5] Y Guo F Xu Z G Li Z M Zhao, X Y Yuan. Modelling and simulation of a two-axis tracking system. *Journal of System and Control Engineering*, 224, 2009.

[6] Per Skoglar. Modelling and control of ir/eo-gimbal for uav surveillance applications, 2002.

[7] Roy Featherstone. *Rigid Body Dynamics Algorithms.* Springer, 2008. ISBN 978-0-387-74314-1.

[8] Dr. Kravitha P. Rajesh R. J. Camera gimbal stabilization using conventional pid controller and evolutionary algorithms. *IEEE International Conference on Computer, Communication and Control*, 2015.

[9] Fahad Farooq Nourallah Ghaeminezhad, Wang Daobo. Stabilizing a gimbal platform using self-tuning fuzzy pid controller. 93, 2014.