

UNIVERSITÀ DEGLI STUDI DI PADOVA

SCUOLA DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Tesi di Laurea Magistrale

**ASSEMBLY, INTEGRATION AND TESTING OF A ROBOTIC
FACILITY FOR THE SIMULATION OF SPACECRAFT ATTITUDE
AND ORBITAL MANEUVERS**

Candidato
Alex Caon
1111055

Relatore
Prof. Alessandro Francesconi

Correlatore
Dr. Andrea Antonello

ANNO ACCADEMICO 2017/2018

Alla mia famiglia

Abstract

Recent developments in Active Debris Removal and On-Orbit Servicing are setting the need for new autonomous systems, capable of a different cohort of operations, such as maintenance, upgrades and re-fueling. The success of this type of missions is dependent on the way satellites interact with each other in a micro-gravity environment. Nowadays, there are limited ways for simulating micro-gravity in a laboratory setting, and they mainly consist in water pools, low friction tables, drop towers, parabolic flights and robotic arms. Among these techniques, robotic arms constitute the only option that allows to easily and repeatedly simulate the full pose (position and attitude) of a rigid body; through dedicated algorithms, robotics facilities permit to reproduce the physics of micro-gravity. Furthermore, the software can implement relative motion, attitude reaction-control systems and docking impacts between target and chaser.

In this thesis, we illustrate the development and testing of a low-cost robotic arm for the simulation of rendezvous and docking maneuvers. While the target is fixed, the chaser is represented by the robotic arm's end effector. In this work, we first give an overview on the robotic arm structure and on the control hardware, after we illustrate in deep the control algorithm to test the interactions between vehicles in micro-gravity environment in a laboratory settings with the ground gravity, and the strategies used for the hardware and software architecture and how to implement them into the electronic hardware used for the control.

Sommario

I recenti sviluppi in Active Debris Removal e On-Orbit Servicing stanno stabilendo la necessità di nuovi sistemi autonomi, in grado di compiere diverse operazioni, come la manutenzione, gli aggiornamenti e il rifornimento. Il successo di questo tipo di missioni dipende dal modo in cui i satelliti interagiscono tra loro in un ambiente a microgravità. Al giorno d'oggi, ci sono modi limitati di simulare la microgravità in un laboratorio, e consistono principalmente in piscine, tavoli a basso attrito, torri di lancio, voli parabolici e bracci robotici. Tra queste tecniche, i bracci robotici costituiscono l'unica opzione che consente di descrivere facilmente e ripetutamente i sei gradi di libertà (posizione e assetto) di un corpo rigido; attraverso algoritmi dedicati, le facility robotiche consentono di simulare la fisica della microgravità. Inoltre, il software può implementare il moto relativo, i sistemi di controllo alla reazione di assetto e gli impatti di docking tra target e chaser.

In questa tesi, illustriamo lo sviluppo e il test di un braccio robotico a basso costo per la simulazione di rendezvous e manovre di docking. Mentre il target è fisso, il chaser è rappresentato dall'end effector del braccio robotico. In questo lavoro, forniamo prima una panoramica sulla struttura del braccio robotico e sull'hardware di controllo, dopo aver illustrato in profondità l'algoritmo di controllo per riprogettare virtualmente le interazioni tra veicoli che avvengono in microgravità in un laboratorio a terra e le strategie utilizzate per l'architettura hardware e software e come implementarli nel computer industriale utilizzato per il controllo .

Ringraziamenti

Ecco il capitolo tanto temuto, i ringraziamenti. Temuto sia perché mi provocano imbarazzo sia perché ci sono tante persone da ringraziare e sicuramente me ne dimenticherò qualcuna. Dunque iniziamo.

Inizio con il ringraziare il mio migliore amico Spartaco, per tutto il supporto che mi ha dato in questi anni. È sempre stato leale e sincero, senza il suo aiuto non ce l'avrei fatta a superare alcuni momenti. Ringrazio anche zio Marco per tutte le serate passate insieme a grigliare e a bere birra; insieme a lui ringrazio anche sua moglie Marica, Cristian, Livio e Rita per i bei momenti passati insieme, per il supporto datomi in questi ultimi anni e perché mi hanno fatto sentire parte della famiglia del Kustom Store.

Tra gli amici da ringraziare ci sono anche i compagni di classe dell' ITIS E. Barsanti: Riccardo Berti, Mattia Cimolin, Francesco Parisotto con i quali ho studiato per i primi esami, grazie a loro l'impatto con l'università è stato meno duro. Gli amici di "Operazione Marmellata" (Matteo Meneghel, Alberto Compagnin, Alberto Cenzato, Enrico Lungavia, Matteo Duzzi, Giulia Sarego, Lorenzo Olivieri, Gilberto Grassi, Arthur), che mi hanno dato supporto fisico e morale durante la tesi e gli ultimi esami, un divano per dormire (grazie Lorenzo e Giulia) e tanti bei momenti. Poi ci sono altri amici di esami da ringraziare, tra i quali Lorenzo Berto e Gabriele Fonti per le interminabili chiamate-studio in Skype. Vorrei aggiungere alla fine, poco prima di stampare la tesi, un ulteriore ringraziamento a Matteo Meneghel, che mi ha aiutato a fare i video finali del braccio robotico in movimento.

Colgo l'occasione per ringraziare il Prof. Alessandro Francesconi che mi ha fatto da relatore anche per questa tesi. Ringrazio anche il Dr. Andrea Antonello per avermi seguito durante il mio lavoro e avermi corretto gli infiniti errori di inglese per le risate fatte in laboratorio e per avermi fatto provare uno degli amplificatori per chitarra da lui costruiti. E lasciatemelo dire, quell'amplificatore ha uno dei migliori suoni che io abbia mai sentito, anche se il chitarrista lasciava a desiderare. Vorrei anche ringraziare Antonio Valsecchi, che durante il corso da lui tenuto ha insegnato in modo chiaro ed esauriente le basi del funzionamento dei PLC e ha sempre risposto alle mie domande (a volte anche assurde) sempre in modo preciso.

Passiamo ora a ringraziare la mia famiglia a partire dai nonni paterni che mi hanno fatto capire che la mia strada era lo studio e i nonni materni con i quali si è

creato un forte legame. Vorrei fare un ringraziamento speciale a mio nonno materno per avermi portato in cerca di funghi e sua moglie per avermi sempre procurato dei dolcetti da portarmi via durante le scampagnate. Ringrazio anche gli zii tutti e le rispettive famiglie. Un ringraziamento speciale va al mio gatto Balù che con il suo affetto mi ha rallegrato in molti momenti. Alla fine i ringraziamenti più speciali, cioè quelli alle cui persone è dedicata questa tesi. Parto con il ringraziare mio fratello Michael, che nonostante in questi anni ci siamo visti poco, io continuerò a volergli bene e a ricordare che alcuni tra bei momenti della mia vita li ho vissuti con lui. Ringrazio mio papà che non ha potuto vedere tutto ciò, ma che mi ha insegnato i veri valori della vita, oltre ad avermi insegnato a pescare. Infine ringrazio la persona più importante di tutti, quella senza la quale io non sarei arrivato fin qui, quella che più di ogni altro ha creduto in me, quella che si è sacrificata più di tutti per farmi arrivare sin qui, quella che mi ha supportato e sopportato, quella che mi ha insegnato che la vita può essere dura, ma domani è comunque un altro giorno e va sempre affrontato al meglio. Grazie mamma.

Contents

1	Introduction	1
1.1	The need for a robotic arm facility	1
1.2	Thesis motivation	2
2	Robotic arm structure and electronics configuration	3
2.1	Mechanical structure	3
2.2	Electronic configuration	3
3	Robotic theory	9
3.1	Frames rotation and translation	9
3.2	Direct and inverse kinematics	11
3.3	Differential kinematics	12
3.4	Dynamics	14
3.4.1	Newton-Euler approach	14
3.4.2	Euler-Lagrange approach	15
3.4.3	CRBA: Composite-Rigid-Body Algorithm	16
3.5	Joint control architecture	17
3.6	Computed-Torque Controller	19
3.6.1	Tuning of PD controller	21
3.7	Trajectory generator	23
4	CANopen Communication	25
4.1	Introduction	25
4.2	Physical structure of the CANopen network	25
4.3	Data transfer	26
4.4	Object Dictionary	29
4.5	Device errors	40
4.6	Device monitoring via Heartbeat messages	43
5	Electronic hardware Configuration and settings	45
5.1	MAXON®EPOS2 Controller	45
5.1.1	EPOS2 control architecture	45
5.1.2	EPOS Studio	52
5.1.3	EDS File and DCF file	61
5.1.4	LED in EPOS2 Controllers	62
5.1.5	Digital Input on EPOS2	62

5.2	B&R Automation [®] PLC	64
5.2.1	Digital Input module	65
5.2.2	Hardware configuration with Automation Studio	68
6	Software configuration	73
6.1	Automation Studio environment for software configuration	73
6.1.1	Software configuration	73
6.1.2	Assignment of the variables	77
6.2	CANopen CiA 402 protocol	81
6.2.1	Controlword and Statusword: Finite State Machine	81
6.3	Position-based modes of operation	83
6.3.1	Homing Mode	83
6.3.2	Profile Position Mode	88
6.4	Interpolated Position Mode	94
6.4.1	Axis Synchronization	102
6.4.2	PVT Algorithm	103
6.5	The transition control method	104
6.5.1	React to the dangers	106
6.6	B&R Automation [®] package for Simulink [®]	107
7	Final Results	115
7.1	Discrete Motion	115
7.2	Test results	116
8	Conclusions	121
8.1	Comments	121
8.2	Future works	121

List of symbols and abbreviations

List of Symbols

Robotic theory

O_f	generic reference frame origin
O_w	world reference frame origin
${}^A P$	distance vector between two reference frames expressed in frame A
v_A	vector expressed in reference frame A
${}^A_B R$	rotation matrix from frame A to frame B
${}^A_B T$	roto-translation matrix from frame A to frame B
\tilde{v}	normalized vector
ω	angular rate
$\dot{\omega}$	angular acceleration
v	linear velocity
\dot{v}	linear acceleration
v_c	center of mass linear velocity
\dot{v}_c	center of mass linear acceleration
q	joint angular position
\dot{q}	joint angular rate
\ddot{q}	joint angular acceleration
I	inertia matrix
F	inertia force
N	inertia couple
f	force
n	couple
t_i	torque generated by the $i - th$ joint
τ	external torque (CRBA)
M	mass matrix (CRBA)
C	acceleration vector (CRBA)

List of abbreviations

Robotic theory

CRBA	Composite-Rigid-Body Algorithm
CTC	Computed Torque Control

Communication

CAN	Controller Area Network
node-ID	node identification
OD	Object Data
RW	Read and Write
RO	Read Only
DC	Direct Current (referred to a motor)
PM	Permanent Magnets (referred to a motor)
BL	Brush Less (referred to a motor)
SDO	Service Data Object
PDO	Process Data Object
COB	Communication Object (CAN message)
TxPDO	Transmit PDO
RxPDO	Receive PDO

Software configuration

AS	Automation Studio
CiA	CAN in Automation
FSM	Finite State Machine

Chapter 1

Introduction

1.1 The need for a robotic arm facility

The fact that the number of human objects in space is increasing leads to new missions for the Active Debris Removal and for the On Orbit Servicing. These missions set a new challenge for autonomous systems capable of different operations, such as maintenance, upgrades and re-fueling. The success of such missions, is strictly connect on how the vehicles interact with each other in micro-gravity environment. The facilities which are able to simulate relative motions between orbiting objects includes water pools, parabolic flights, drop tower, low friction tables and robotic arms.

Swimming pools have the benefits of the zero buoyancy, once it have been met. But the drag force the water acting on the system plays an important role both on the description of the problem and on the system maintenance, in fact, the zero buoyancy could be disturbed by the drag force.

Parabolic flights allows to reproduce orbital conditions, but for a short time lapse and with many constraints. Moreover it is an expensive solution and it is not suitable for long testing campaigns. Drop towers have the same application limits than the parabolic flights.

Low friction tables, instead, have the advantages to replicate the same conditions of a micro-gravity environment if the setup is adequate (platform balanced and surface flat and smooth), the only lack is that them guarantees to test only 2 of the 3 translation degree of freedom.

Ultimately, the robotics arm constitute the only option that allows to simulate the full pose (position and attitude) of a rigid body. But the micro-gravity scenario can be only simulated through special algorithms by imposing the motion characterized by the desired dynamics. Software is also able to perform orbital operations in which contact is present.

Nowadays there are very few robotic facilities. One of the most important is the European Proximity Operations Simulator (EPOS) experiment conducted by the Deutches Zentrum für Luft und Raumfahrt (DLR). Where an industrial PC feeds in synchronous trajectory via a Simulink[®] interface. This, with the control measuring system allow an high position and angular accuracy. All the trajectories are carried out via an implementation of Clohessy-Wiltshire expressions.

1.2 Thesis motivation

This thesis is the natural continuation of the Andrea Antonello's PhD thesis. In which he developed a low cost robotic arm facility which, through dedicated algorithms, is able to simulate the motion of vehicles in micro-gravity environment in a laboratory setting. My thesis work starts where Dr. Andrea Antonello ended his work [2]. So my activities were about the assembly of the robotic arm and about the programming of the joint positioning controllers through the programming of the B&R Automation[®]PLC. Then I made some other works, such as the supporting structure design and some wiring. Finally a simple interface between the B&R Automation[®]PLC and Simulink[®] was done.

Take this project into the operating state would be very important for the activities under study at CISAS research center for Active Debris Removal as well as for on-orbit servicing: the robotic arm could serve as the main testing facility for the verification and simulation of theoretical and numerical analysis. Moreover it could be used as well as for testing realistic orbital operation by instrumenting the end effector with innovative sensors.

Chapter 2

Robotic arm structure and electronics configuration

A robotic arm is the result of the love story between mechanic and electronic : the mechanic part provides the links structure while the electronic provides the motion for the links. For this reason, in a robotic arm, one cannot exist without the other.

2.1 Mechanical structure

The mechanical structure is that part which connects the motors to the links. Hence it is important that it is enough stiff for avoiding misalignments during the manipulator operations. Misalignments could be deleterious for the precision of the tasks the robotic arm is called to make. For more information about the choice of motor configuration see [2]. From figure 2.1 to figure 2.4 all the joint blocks (with the respective renders) are shown. Finally the figure 2.5 pictures the render of the entire robotic arm.

In figure 2.6 is shown the complete mechanic plant of the manipulator, including both the robotic arm and the support structure. This is used to support the links 2 and 3 weight when the motors aren't fed with the power supply.

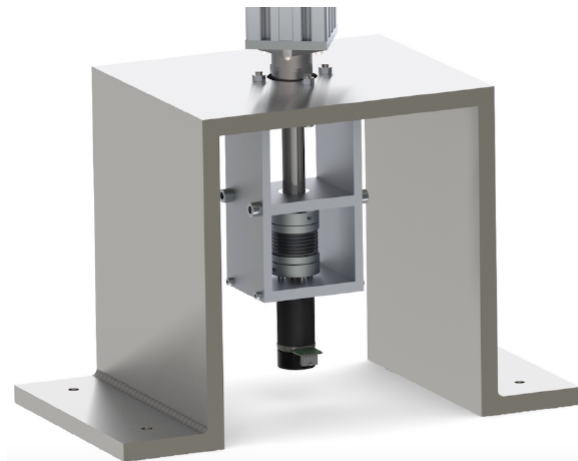
2.2 Electronic configuration

The electronic configuration is that part which makes the links move, and controls their movement in order to satisfy the request precision. The chosen motors are manufactured by MAXON[®] motors and their main characteristics are listed in tables 5.1. The joint positions are controlled by Electronic POSition controllers (EPOS2 by MAXON[®]). Due to the robotic arm specific tasks, a position control was chosen. In fact, in order to simulate relative orbit position it is easier to make a position control instead of a torque control (which also requires a great knowledge of the robotic arm geometry). However, a torque control is used in the virtual environment of Simulink[®] for getting the joint positions vectors.

The choice of the motors is based on the search of a trade off between their performance and the costs. To improve the torque and to reduce the velocity a



(a) Joint block 1.

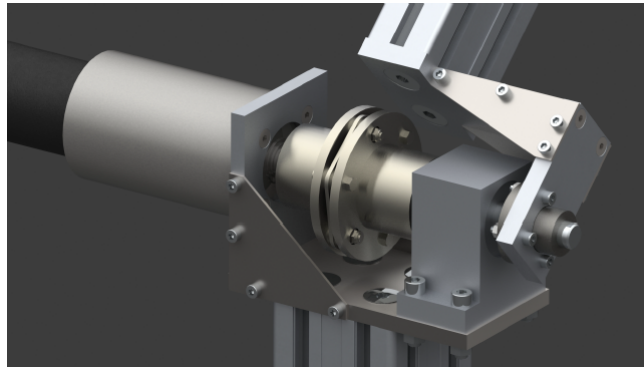


(b) Render Joint 1.

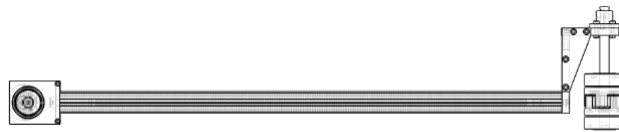
Figure 2.1: Block and render of the joint 1 [2].



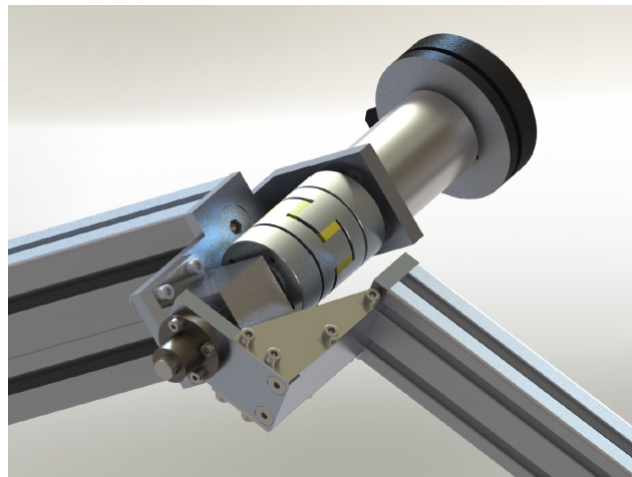
(a) Joint block 2.



(b) Render Joint 2.

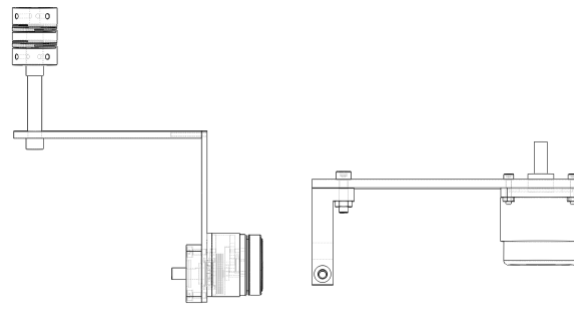
Figure 2.2: Block and render of the joint 2 [2].

(a) Joint block 3.

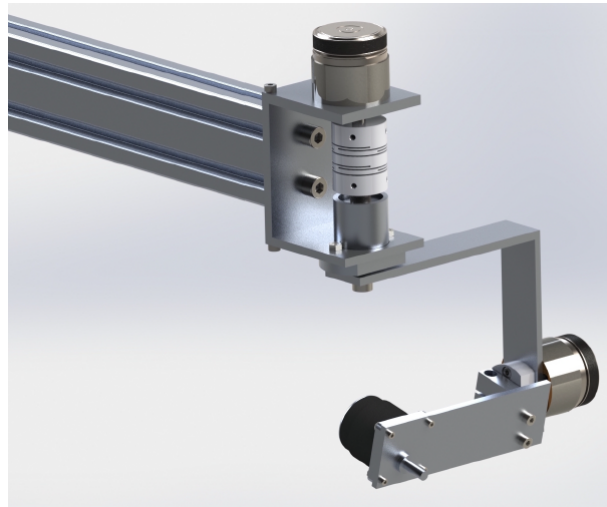


(b) Render Joint 3.

Figure 2.3: Block and render of the joint 3 [2].



(a) Joint blocks 4 and 5.



(b) Render end effector.

Figure 2.4: Block of joints 4 and 5 and render of the end effector [2].

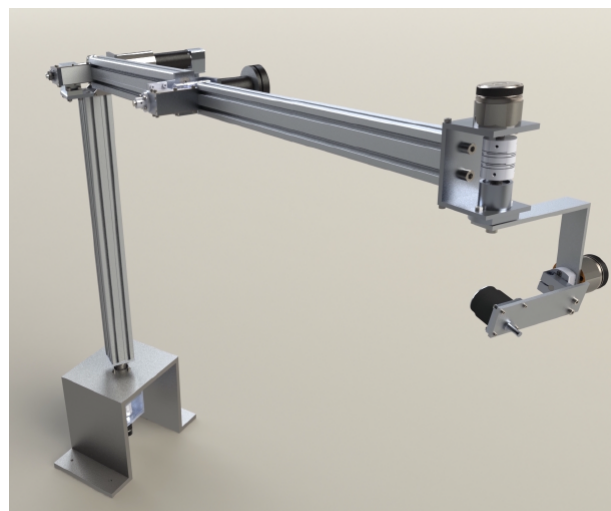


Figure 2.5: Render of the complete robotic arm [2].

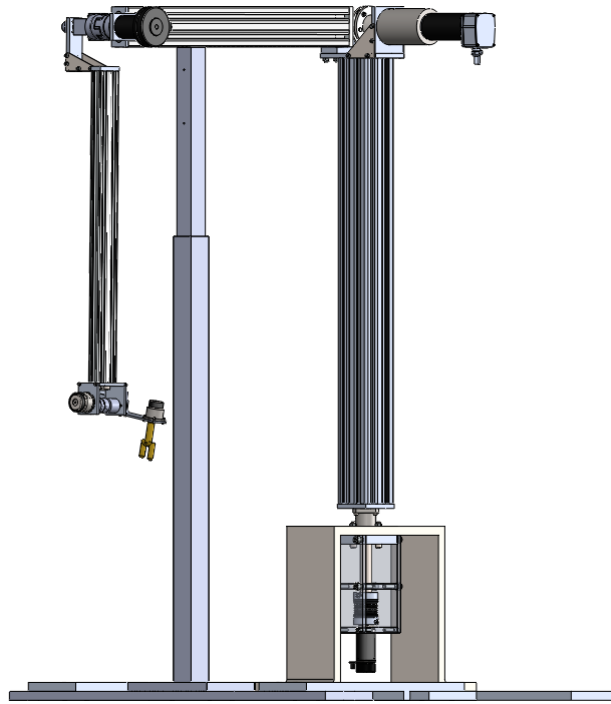


Figure 2.6: Complete plant of the robotic arm with the support structure.

reduction gear for each motors was chosen. Finally to improve the precision of the motion, one incremental encoder for each motor was chosen as well. Except for the motor number 6, which has not the encoder, but has only the hall sensors. Figure 2.7 shows the set of the six motors with gear and encoders.

The electronic configuration and setting is explained in chapter 5.



Figure 2.7: The six joints. One joint is composed by the motor, the gear and the encoder. The sixth joint hasn't got the encoder.

Chapter 3

Robotic theory

In the last sixty years many robotic theories have been developed. In this chapter we will analyze some robotics theories and how to implement them to find solution of some problems like the inverse kinematic and the inverse dynamic [4].

3.1 Frames rotation and translation

In figure 3.1 there are two frames, which are rotating and translating one with respect to the other. Let the frame 1 be the reference frame (or *world frame*), and the frame 2 be an arbitrary frame.

Translation represents a 3D vector that describes the distance between two reference frame origins O_f and O_w along the three physical dimensions x, y, z , so that:

$$O_f = O_w + {}^w P \quad (3.1)$$

We must pay attention that translation vector P must be written respect the world frame, i.e. ${}^w P$, otherwise we must take the opposite of it¹

$${}^w P = -{}^f P$$

The rotation is given by the rotational matrix. In this context we use only the Euler angles to describe rotation between two frame, but many other are possible (quaternions, directors cosines, ecc). We indicate rotation matrix with symbol ${}^f_w R$, which means the rotation that aligns the frame f axis into the frame w axis. In general ${}^w_f R$ is a full 3×3 matrix:

$${}^w_f R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (3.2)$$

Where r_{ij} are the rotation matrix elements and they are composed by composition of trigonometric functions of the three angles that create rotation. These three angles

¹In general the left apex tells us in which frame a vector is referred. While for vector elements a right subscript is used.

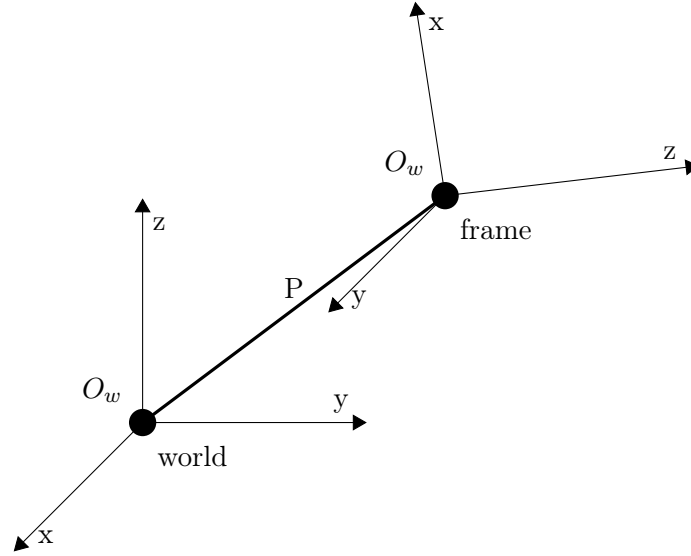


Figure 3.1: Example of two frames which are rotated and translated one with respect to the other.

are three rotations about the three axis. So a vector v_f in the generic frame, will be rotated in the world frame:

$$v_w = {}^w_f R \cdot v_f \quad (3.3)$$

So, we can transform a vector in the frame f into a vector in the world frame w in the following way:

$$\begin{bmatrix} v_{w_x} \\ v_{w_y} \\ v_{w_z} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \cdot \begin{bmatrix} v_{f_x} \\ v_{f_y} \\ v_{f_z} \end{bmatrix} + \begin{bmatrix} p_{f_1} \\ p_{f_2} \\ p_{f_3} \end{bmatrix} \quad (3.4)$$

A simpler way to transform v_f into v_w is to use a roto-translation matrix. This is the composition of rotation and translation ${}^w_f T$, so in only one 4×4 matrix we have both the translation and the rotation:

$$\begin{aligned} {}^w_f T &= \left[\begin{array}{c|c} {}^w_f R & {}^w_f P \\ \hline 0 & 1 \end{array} \right] \\ &= \left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & p_{w_x} \\ r_{21} & r_{22} & r_{23} & p_{w_y} \\ r_{31} & r_{32} & r_{33} & p_{w_z} \\ \hline 0 & 0 & 0 & 1 \end{array} \right] \end{aligned} \quad (3.5)$$

The latter is only a brief way to write the ensemble of translation and rotation into a single matrix, in which the last row has the meaning to normalize the matrix and makes it a square 4×4 matrix; this implies that also the vectors v_w and v_f must be normalized into a 4×1 vectors:

$$\tilde{v}_w = \begin{bmatrix} v_{w_x} \\ v_{w_y} \\ v_{w_z} \\ 1 \end{bmatrix} \quad \tilde{v}_f = \begin{bmatrix} v_{f_x} \\ v_{f_y} \\ v_{f_z} \\ 1 \end{bmatrix}$$

So the translation and rotation of a vector v_f in the f frame into a vector v_w into a vector in the world frame w can be written like:

$$\begin{aligned} \tilde{v}_w &= {}^w_f T \cdot \tilde{v}_f \\ \begin{bmatrix} v_{w_x} \\ v_{w_y} \\ v_{w_z} \\ 1 \end{bmatrix} &= \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{f_x} \\ v_{f_y} \\ v_{f_z} \\ 1 \end{bmatrix} \end{aligned} \quad (3.6)$$

Finally, if we have n frames (such as those attached to the n joints), the roto-translation matrix between reference frame $n - th$ and the world frame is given by:

$$\begin{aligned} {}^w_n T &= {}^w_1 T \cdot {}^1_2 T \cdot \dots \cdot {}^{i-1}_i T \cdot {}^i_{i+1} T \cdot \dots \cdot {}^{n-1}_n T \\ &= \prod_{i=0}^n ({}^i_{i+1} T) \end{aligned} \quad (3.7)$$

where, $i = 0$ refers the world frame.

3.2 Direct and inverse kinematics

The kinematics is that branch of the robotic theory which studies the position of a generic joint into the Cartesian space, giving the upstream joints rotation. The space of the joint rotation is called *joint space*. For sake of simplicity, suppose the first three joint (without the three end-effector joint) are rotated of $q_1 = \pi$, $q_2 = \pi/6$ rad and $q_3 = -\pi/3$ rad, so the position of the end-effector (ee) in the 3D Cartesian frame fixed at the robotic arm base is given by:

$$\begin{cases} X_{ee} = \cos(q_1)[L_2 \cos(q_2) + L_3 \cos(q_2 + q_3)] \\ Y_{ee} = \sin(q_1)[L_2 \cos(q_2) + L_3 \cos(q_2 + q_3)] \\ Z_{ee} = L_1 + L_2 \cos(q_2) + L_3 \cos(q_2 + q_3) \end{cases}$$

where L_1 , L_2 and L_3 are the three links length respectively. Figure 3.2 shows the robotic arm in this configuration.

Inverse kinematics allows the retrieval of the joint angles once a predefined cartesian displacement is given. Often we can obtain more than one solution for the joint rotation, this is due to the fact that the inverse kinematics is based on the inverse trigonometric functions.

Let us suppose the end-effector has to reach a desired position in the Cartesian space described by X_{dee} , Y_{dee} and Z_{dee} and the end-effector must have a specified rotation with respect to the world frame given by ${}^w_{ee} R_d$, these give the roto-translation matrix ${}^w_{ee} T_d$ (subscript d stands for desired). From equation 3.7 we obtain the end-effector roto-translation matrix (in which we place $n = ee$). To find out the equations that describe the inverse kinematics, we will write the equality

$${}^w_{ee} T = {}^w_{ee} T_d$$

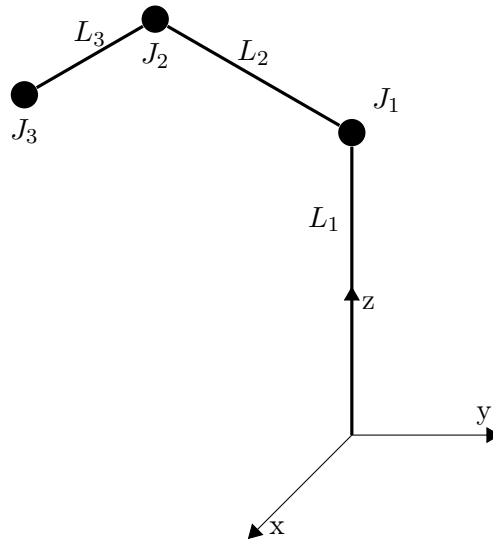


Figure 3.2: Robotic arm in the configuration described above

or, in extended way:

$$\begin{bmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R_{d11} & R_{d12} & R_{d13} & X_{dee} \\ R_{d21} & R_{d22} & R_{d23} & Y_{dee} \\ R_{d31} & R_{d32} & R_{d33} & Z_{dee} \\ \hline 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.8)$$

The equality above gives us all the trigonometric equations to solve the inverse kinematic.

There are several way to find out the solution, the easier is the geometrical approach, which consist to compare the solution with the robotic arm configuration and find the most appropriate for the goal. Other solutions are those that solve the system of equation with some numerical routine (see [4] for more details).

3.3 Differential kinematics

From a mathematical perspective, the Jacobian matrix is a multidimensional form of derivative. We consider six functions y_i (like the absolute liner or angular velocity v or ω) of six independent variables x_i (like the joints velocity \dot{q})

$$\begin{aligned} y_1 &= f_1(x_1, x_2, x_3, x_4, x_5, x_6) \\ y_2 &= f_2(x_1, x_2, x_3, x_4, x_5, x_6) \\ &\vdots \\ y_6 &= f_6(x_1, x_2, x_3, x_4, x_5, x_6) \end{aligned}$$

in vector notation:

$$Y = F(X)$$

Let us calculate the differential of y_i , then:

$$\begin{aligned}\delta y_1 &= \frac{\partial f_1}{\partial x_1} \cdot \delta x_1 + \frac{\partial f_1}{\partial x_2} \cdot \delta x_2 + \dots + \frac{\partial f_1}{\partial x_6} \cdot \delta x_6 \\ \delta y_2 &= \frac{\partial f_2}{\partial x_1} \cdot \delta x_1 + \frac{\partial f_2}{\partial x_2} \cdot \delta x_2 + \dots + \frac{\partial f_2}{\partial x_6} \cdot \delta x_6 \\ &\vdots \\ \delta y_6 &= \frac{\partial f_6}{\partial x_1} \cdot \delta x_1 + \frac{\partial f_6}{\partial x_2} \cdot \delta x_2 + \dots + \frac{\partial f_6}{\partial x_6} \cdot \delta x_6\end{aligned}$$

in vector notation:

$$\begin{aligned}\delta Y &= \frac{\partial F}{\partial X} \cdot \delta X \\ \delta Y &= J(X)\delta X\end{aligned}\tag{3.9}$$

where $J(X)$ is the Jacobian matrix, which has in the place of J_{ij} the partial derivatives of f_i taken with respect to x_j .

It is important to remark that this anthropomorphic robotic arm is used to simulate orbital maneuvers, so the orbital velocity v is known from the trajectory planing. This will be helpful for us because we can use the Jacobian to obtain the joints velocity and, after an integration, the joints variable position. We can define the Jacobian as follow: Jacobian J is that operator which maps the absolute velocity (both linear and angular) through the joints velocity, then:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} J_p \\ J_o \end{bmatrix} \cdot \begin{bmatrix} \dot{q} \end{bmatrix}$$

Hence, we can compute joints velocity from the absolute orbital velocity v , or ω , but in this case the Jacobian must consider also the transformation between \dot{q} and ω . Since the velocity v is known from the trajectory planning, we can obtain the joint velocities²:

$$\dot{q} = J^{-1}(q) \cdot v\tag{3.10}$$

and the joint position by integrating \dot{q}

$$\int_0^T \dot{q}(t) dt + q(0)\tag{3.11}$$

the initial position $q(t = 0)$ needs to be known in order to start integration and this could be obtain from the inverse kinematics (see section 3.2). In the code, the equation 3.11 is implemented using a numerical integration, which yields:

$$q(t_{i+1}) = q(t_i) + \dot{q}(t_i)\Delta t$$

we can resume the integration of the inverse differential kinematics with the block diagram in figure 3.3.

²There are several way to invert the Jacobian, we refer it to a specific text.

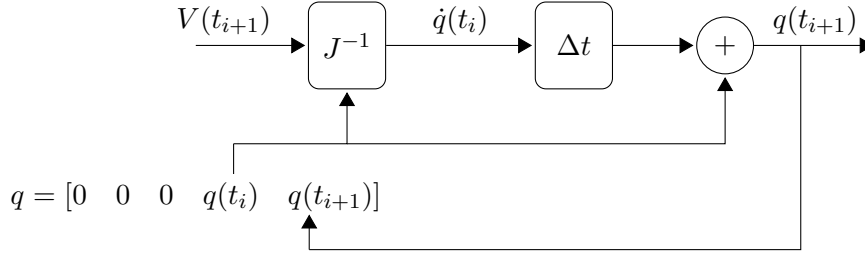


Figure 3.3: Inverse kinematics with integration method

3.4 Dynamics

The dynamics studies the forces acting on the robotic arm, and establishes the relation between forces and motion. There are two main ways to study the dynamics: the Newton-Euler approach and the Euler-Lagrange method.

3.4.1 Newton-Euler approach

The Newton-Euler approach is based on the balance of all forces³ acting along the generic robotic arm link. This method is well suited for a recursive approach. With this method we have to do two recursive blocks of computation. The first is from the base $i = 0$ to the end-effector $i = n$ to find out all the velocities (algorithm 1).

```

1 for  $i = 0$  to  $n$  do
2    ${}^{i+1}\omega_{i+1} = {}^{i+1}R_i \cdot {}^i\omega_i + \dot{q}_{i+1} \cdot {}^{i+1}\hat{z}_{i+1}$ ;
3    ${}^{i+1}\dot{\omega}_{i+1} = {}^{i+1}R_i \cdot {}^i\dot{\omega}_i + {}^{i+1}R_i \cdot {}^i\omega_i \times \dot{q}_{i+1} \cdot {}^{i+1}\hat{z}_{i+1}$ ;
4    ${}^{i+1}\dot{v}_{i+1} = {}^{i+1}R_i \cdot [{}^i\omega_i \times {}^iP_i + {}^i\omega_i \times ({}^i\omega_i \times {}^iP_{i+1}) + {}^i\dot{v}_i]$ ;
5    ${}^{i+1}\dot{v}_{c,i+1} = {}^i\dot{\omega}_i \times {}^iP_{c,i} + {}^i\omega_i \times ({}^i\omega_i \times {}^iP_{c,i}) + {}^i\dot{v}_i$ ;
6 end

```

Algorithm 1: Newton-Euler forward routine

and the second is from the end-effector $i = n$ to the base $i = 0$, to find out the all the forces (algorithm 2).

```

1 for  $i = n$  to 0 do
2    ${}^iF_i = m_i \cdot {}^i\dot{v}_{c,i}$ ;
3    ${}^iN_i = {}^cI_i \cdot {}^i\dot{\omega}_i + {}^i\omega_i \times {}^cI_i \cdot {}^i\dot{\omega}_i$ ;
4    ${}^if_i = {}^iF_i + {}^{i+1}R_i \cdot {}^{i+1}f_{i+1}$ ;
5    ${}^in_i = {}^iN_i + {}^{i+1}R_i \cdot {}^{i+1}n_{i+1} + {}^iP_{c,i} \times {}^iF_i + {}^iP_{i+1} \times {}^{i+1}R_i \cdot {}^{i+1}n_{i+1}$ ;
6    $t_i = {}^in_i \cdot {}^{i+1}\hat{z}_i$ ;
7 end

```

Algorithm 2: Newton-Euler backward routine

in algorithm 2, t_i is the acting torque that the joint has to generate, in order to perform the motion. It is given in the joint reference frame.

³Name "forces" means generalized forces, so it includes both forces both torques

3.4.2 Euler-Lagrange approach

The Euler-Lagrange approach, instead, is an energy based method, so the equations of motion are independent from the reference frame. The Lagrange equation is:

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}} \right) - \left(\frac{\partial L}{\partial q} \right) = \tau$$

$$L = E_K - U$$

where L is the Lagrange function, E_K and U are the kinetic and potential energy respectively and τ is the generalized forces vector acting on the links, it include also the joints torque and the external forces. Although the formulation is intuitive, its implementation doesn't. The Lagrange equation could be re-written as:

$$M(q)\ddot{q} + v(q, \dot{q})\dot{q} + F_v\dot{q} + F_d(\dot{q}) + G(q) + \tau_d = \tau \quad (3.12)$$

This is the dynamic equation of a robotic arm; where $M(q) \in \mathbb{R}^{n \times n}$ is the function that maps the accelerations into inertial forces, hence it is the inertial matrix (or mass matrix) of the arm and it depends on the configuration of the arm and on the inertial properties of the hardware, and it is explicitly given by the following formula:

$$M(q) = \sum_{i=1}^n (m_i J_{p,i}^T J_{p,i} + J_{o,i}^T R_i I_i R_i^T J_{o,i})$$

The term $V(q, \dot{q}) \in \mathbb{R}^{n \times n}$, accounts for Coriolis F_v and centrifugal F_d terms:

$$V(q) = v(q, \dot{q})\dot{q} + F_v\dot{q} + F_d(\dot{q})$$

and it is obtained by:

$$V(q) = \dot{M}(q)\dot{q} - \frac{1}{2} \frac{\partial}{\partial q_i} (\dot{q}^T M(q) \dot{q})$$

And the last term $G(q) \in \mathbb{R}^{n \times 1}$ is the gravity term compensation, it is obtained from the derivation of the potential gravitational energy:

$$G(q) = \sum_{i=1}^n \frac{\partial U_g}{\partial q_i}$$

where

$$\frac{\partial U_g}{\partial q_i} = - \sum_{j=1}^n m_i g_0 J_{o_{ij}}(q)$$

Using these expressions to compute equations 3.12 is not computationally friendly. In the next section (3.4.3) we will see how to simplify this computation.

3.4.3 CRBA: Composite-Rigid-Body Algorithm

In this section we will see the Composite-Rigid-Body Algorithm [13] to compute the mass matrix $M(q)$ and the acceleration matrix $C(q, \dot{q})$. CRBA computes the inertial parameters of composite set of rigid bodies from the last link to the first link of the robotic arm. The columns of the mass matrix are computed very efficiently through successive application of inverse dynamic, setting the joint velocity to zero and setting the joint acceleration to zero or one. This means that only one joint is in motion at time, hence the inverse dynamic analysis becomes the much more simple study of a base set of links in static equilibrium and a composite rigid body in motion at the rest of the robotic arm.

In other words, let us suppose that only the i -th joint acceleration is set to one, and all the other accelerations are set to zero. This yields that the inverse dynamic has only to study static equilibrium of the first $i - 1$ joints and the motion of the last i -th to n -th joints.

The computational burden of the CRBA is reduced from a value proportional to N^4 , with the classical Lagrange method, to a value $\sim N$, where N is the robotic arm Degrees of Freedom (DoF). The Newton-Euler method has the same number of computation, but it is computationally heavy for the number of sums and multiplications.

Let us write the dynamic equation 3.13 without the disturbance term τ_d

$$M\ddot{q} + C(q, \dot{q}) = \tau$$

where (like above) \ddot{q} is the $N \times 1$ vector of the generalized accelerations, τ is the vector of the generalized forces, M is the mass matrix and C is the vector of the accelerations and gravity react. Then.

$$\begin{aligned} M\ddot{q} &= D(q, \dot{q}, \ddot{q}) - D(q, \dot{q}, 0) \\ &= D(q, 0, \ddot{q}) - D(q, 0, 0) \end{aligned}$$

$D(q, \dot{q}, \ddot{q})$ is a function which compute the inverse dynamic. The velocity is set to zero, so that the velocity terms cancel. The gravity term get canceled as well. Previous equation gives us a simply way to compute the mass matrix $M(q)$:

$$M\delta_i = D(q, 0, \delta_i)$$

where δ_i is a $n \times 1$ vector with a 1 in the i -th row and zeros elsewhere. So the expression $M\delta_i$ represents the i -th column of the mass matrix M .

Since, in our case, joints have only one DoF, then there is 1:1 correspondence between the joint number and the mass matrix column⁴; therefore we may interpret δ_i as a unit acceleration vector for the i -th joint, so that every mass matrix element $M_{j,i}$ is the force required at i -th joint to produce the acceleration δ_i . Or even, equivalently, giving a unit acceleration at i -th joint a force will be created and the latter is the action required at j -th joint to react the force create by the i -th acceleration, indeed is just the mass matrix element $M_{j,i}$ (algorithm 3).

⁴Here the algorithm is described for one DoF joints, but it works also with multi DoF joints considering the mass matrix composed by a block of matrices. For more explanation see [13].

```

1  for  $i = 1$  to  $n$  do
2      for  $j = 0$  to  $n$  do
3           ${}^{j+1}\omega_{j+1} = {}^{j+1}_j R \cdot {}^j\omega_i + \dot{q}_{j+1} \cdot {}^{j+1}\hat{z}_{j+1};$ 
4           ${}^{j+1}\dot{\omega}_{j+1} = {}^{j+1}_j R \cdot {}^j\dot{\omega}_i + {}^{j+1}_j R \cdot {}^j\omega_i \times \dot{q}_{j+1} \cdot {}^{j+1}\hat{z}_{j+1};$ 
5           ${}^{j+1}\dot{v}_{j+1} = {}^{j+1}_j R \cdot [{}^j\omega_i \times {}^j P_i + {}^j\omega_i \times ({}^j\omega_i \times {}^j P_{j+1}) + {}^j\dot{v}_i];$ 
6           ${}^{j+1}\dot{v}_{c,j+1} = {}^j\dot{\omega}_i \times {}^j P_{c,j} + {}^j\omega_i \times ({}^j\omega_i \times {}^j P_{c,j}) + {}^j\dot{v}_i;$ 
7      end
8      for  $k = n$  to  $0$  do
9           ${}^k F_i = m_i \cdot {}^k \dot{v}_{c,k};$ 
10          ${}^k N_i = {}^{c,k} k_i \cdot {}^k \dot{\omega}_i + {}^k \omega_i \times {}^{c,k} k_i \cdot {}^k \dot{\omega}_i;$ 
11          ${}^k f_i = {}^k F_i + {}^{k+1}_k R \cdot {}^{k+1} f_{k+1};$ 
12          ${}^k n_i = {}^k N_i + {}^{k+1}_k R \cdot {}^{k+1} n_{k+1} + {}^k P_{c,k} \times {}^k F_i + {}^k P_{k+1} \times {}^{k+1}_k R \cdot {}^{k+1} n_{k+1};$ 
13          $\tau_i = {}^k n_i \cdot {}^{k+1}\hat{z}_i;$ 
14     end
15      $M_i = \tau_i;$ 
16 end

```

Algorithm 3: CRBA routine to find out the mass matrix $i - th$ column. All the velocities are equal to zero.

The previous interpretation is correct because we give a unit acceleration δ_i to the robotic arm so the remaining downstream links of the $i - th$ joint will behave like a single rigid body which reacts to the unit acceleration, while the upstream links of the $i - th$ joint will remain in static equilibrium.

The computation of the $n \times 1$ vector $C(q, \dot{q})$ is very similar to the mass matrix algorithm (algorithm 4). This one is computed by setting all the joint accelerations to zeros, so during the robot arm operations, we obtain only the terms that depend by the velocities, both the absolute velocities (\dot{v} and $\dot{\omega}$) and the relative joint velocity (\dot{q}). In fact, during the robotic arm operations, every joint has its own velocity \dot{q}_i and this will be interacting with the other upstream links velocities. Ultimately, this yields the Coriolis and centrifugal accelerations, so they are just the actions the joint must make to react to Coriolis and centrifugal accelerations. We can compute the gravity force for each link, or, more simply, we can insert the gravity acceleration vector $g = [0 \ 0 \ g]^T$ at the acceleration initial condition⁵.

3.5 Joint control architecture

The problem of controlling a robot is to determine the torque to be developed by the joints, in order to guarantee the following of the trajectory. Several techniques are available, but the main distinction is due to the way they operate: joint space or operational space. We will focus only on the first technique, but the second could be used as a future implementation, thus we give a brief description of

⁵Note that the joint have to react to gravity, so it must insert in the opposite direction respect to the real one

```

1  $\dot{v}_0 = [0 \ 0 \ g]^T ;$ 
2 for  $j = 0$  to  $n$  do
3    ${}^{j+1}\omega_{j+1} = {}^{j+1}_j R \cdot {}^j\omega_i + \dot{q}_{j+1} \cdot {}^{j+1}\hat{z}_{j+1};$ 
4    ${}^{j+1}\dot{\omega}_{j+1} = {}^{j+1}_j R \cdot {}^j\dot{\omega}_i + {}^{j+1}_j R \cdot {}^j\omega_i \times \dot{q}_{j+1} \cdot {}^{j+1}\hat{z}_{j+1};$ 
5    ${}^{j+1}\dot{v}_{j+1} = {}^{j+1}_j R \cdot [{}^j\omega_i \times {}^j P_i + {}^j\omega_i \times ({}^j\omega_i \times {}^j P_{j+1}) + {}^j\dot{v}_i];$ 
6    ${}^{j+1}\dot{v}_{c,j+1} = {}^j\dot{\omega}_i \times {}^j P_{c,j} + {}^j\omega_i \times ({}^j\omega_i \times {}^j P_{c,j}) + {}^j\dot{v}_i;$ 
7 end
8 for  $k = n$  to  $0$  do
9    ${}^k F_i = m_i \cdot {}^k \dot{v}_{c,k};$ 
10   ${}^k N_i = {}^{c,k} k_i \cdot {}^k \dot{\omega}_i + {}^k \omega_i \times {}^{c,k} k_i \cdot {}^k \dot{\omega}_i;$ 
11   ${}^k f_i = {}^k F_i + {}^{k+1}_k R \cdot {}^{k+1} f_{k+1};$ 
12   ${}^k n_i = {}^k N_i + {}^{k+1}_k R \cdot {}^{k+1} n_{k+1} + {}^k P_{c,k} \times {}^k F_i + {}^k P_{k+1} \times {}^{k+1}_k R \cdot {}^{k+1} n_{k+1};$ 
13   $\tau_i = {}^k n_i \cdot {}^{k+1} \hat{z}_i;$ 
14 end
15  $C_k = \tau_k;$ 

```

Algorithm 4: CRBA routine to find out the force vector $C(q, \dot{q})$. All the velocities and accelerations are equal to zero except for the initial linear acceleration.

it.

Operational space control enable the robotic arm to reach a greater precision in the cartesian space, since the end effector position is actively controlled and it is no longer dependent on the accuracy with which the geometry of the robotic arm is known [2]. This approach requires great complexity, in fact, now the inverse kinematics algorithm is embedded into the feedback control loop. This slow down the algorithm and requires high computational performances. Furthermore the end effector cartesian position is not performed directly, but via the applications of the direct kinematics to the encoders' readings. For the active control of the end effector position we can use a computer vision system, or a stroboscopic system.

In *joint space* techniques, the control is focused on the $q(t)$ values to track the reference points, calculated with the inverse kinematics procedure from the desired trajectory. The drawback of this solution is that the end effector cartesian position is affected by any difference between the known geometric data and real ones. Thus it is mandatory to well know the mechanical design of the structure. Furthermore the way the motion is transferred through the joints has its effects. If the motors are coupled with high-ration reduction gears, the problem becomes linear, but all the non-linear effects (such as friction) might produce others uncertainties.

We use high-ration reduction gears, then a linear approximation can take place. This leads to analyze each link as a SISO independent system (this type of control are often referred to as *decentralized control* [14]). The differential equation describing the motion of a n degree of freedom robot 3.12 can be rewritten as :

$$M(q)\ddot{q} + \tilde{C}(q, \dot{q})\dot{q} + G(q) = \tau$$

where $\tilde{C}(q, \dot{q})$ is the Coriolis and centrifugal acceleration term and $G(q)$ considers

the gravity effects. This last equation describes the dynamics of a multi-body system when some generalized forces τ are acting [15]. In the next section we will see how to control such dynamic equation.

3.6 Computed-Torque Controller

In this years, many robot control scheme were presented. Among them, we will focus in this chapter on the *Computed-Torque Controllers*. These are special application of feedback linearization of non-linear systems [2].

The equation 3.12 could be written in the following way:

$$M(q)\ddot{q} + C(q, \dot{q}) + \tau_d = \tau \quad (3.13)$$

where τ_d is the disturbance external torque and $C(q, \dot{q}) \in \mathbb{R}^{n \times 1}$ is the compensation of centrifugal, Coriolis and gravity terms:

$$C(q, \dot{q}) = V(q, \dot{q}) + G(q)$$

All the quantities are time-based, since them depend on the joints variables $q_i(t)$. Let's suppose that a trajectory $q_d(t)$ has been selected. We define a *tracking error* $e(t)$, and its first and second order differential:

$$e(t) = q_d(t) - q(t) \quad (3.14)$$

$$\dot{e} = \dot{q}_d - \dot{q}$$

$$\ddot{e} = \ddot{q}_d - \ddot{q}$$

$$(3.15)$$

Solving the dynamic equation 3.13 for \ddot{q} and replacing in it the value of \ddot{e} , we get

$$\ddot{e} = \ddot{q}_d + M^{-1}(C + \tau_d - \tau)$$

Defining the input control equation and the disturbance function as

$$u = \ddot{q}_d + M^{-1}(C - \tau) \quad (3.16)$$

$$w = M^{-1}\tau_d \quad (3.17)$$

we can rewrite the *tracking error dynamic* as

$$\frac{d}{dt} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} u + \begin{bmatrix} 0 \\ I \end{bmatrix} w \quad (3.18)$$

This system represents a linear error consisting of n pairs of double integrators $1/s^2$, one per joint. This error system is driven by the control input $u(t)$ and the disturbance $w(t)$.

The input control equation 3.16 may be inverted to yield the *computed-torque control law*

$$\tau = M(\ddot{q}_d - u) + C \quad (3.19)$$

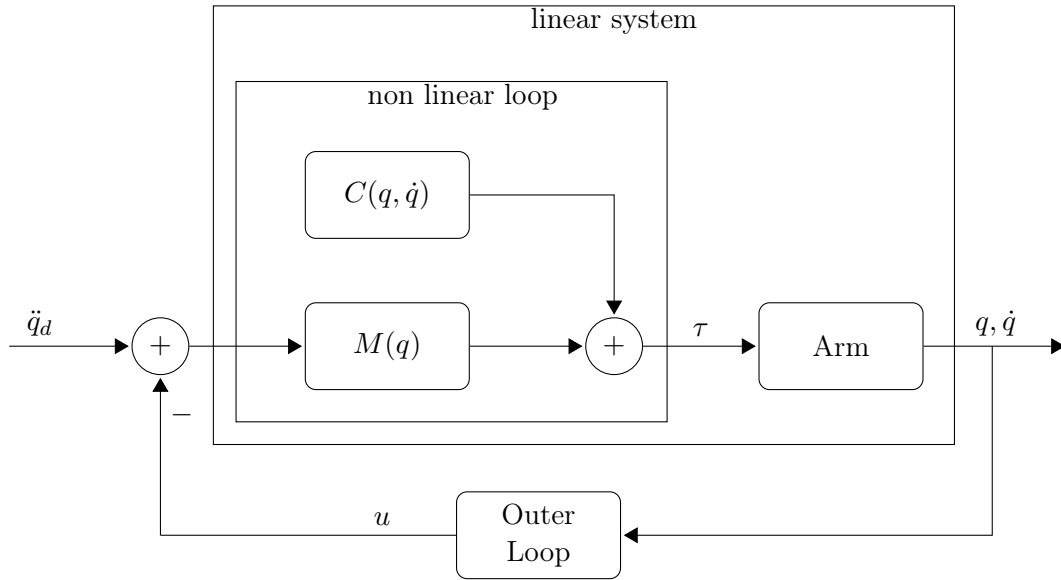


Figure 3.4: Block diagram of the Computed Torque Control algorithm.

If we choose a control input function $u(t)$ which stabilizes the tracking error 3.18, so that $e(t)$ goes to zero, then the nonlinear control input $\tau(t)$ will induce trajectory following in the robot arm: this is the usefulness of the Computed-Torque Controller equation 3.19. In fact, substituting equation 3.19 into equation 3.13 yields:

$$\begin{aligned} M(q)\ddot{q} + C(q, \dot{q}) + \tau_d &= M(\ddot{q}_d - u) + C \\ \ddot{e} &= u + M^{-1}\tau_d \end{aligned}$$

which is just the 3.18.

The nonlinear transformation 3.16 has converted a complicated nonlinear controls design problem into a simple design problem for a linear system, formed by n decoupled subsystem, each obeying Newton's law. The resulting system is shown in Figure 3.4.

We can observe that the diagram in the figure is composed by three parts: one nonlinear loop, one linear loop and an external feedback loop which creates the *auxiliary* outer signal $u(t)$. Since $u(t)$ will depend only on $q(t)$ $\dot{q}(t)$, the external loop will be a feedback loop. For these reasons, a dynamic controller $C(s)$ is selected to control the tracking error signal:

$$U(s) = H(s)E(s)$$

in this case, the transfer function between tracking error and auxiliary signal is

$$T(s) = s^2I - H(s)$$

We could note that the function Computed-Torque Controller 3.19 computes $\tau(t)$ replacing $(\ddot{q}_d - u)$ for $\ddot{q}(t)$ in 3.13, in this way the inverse arm dynamics must be

computed: for this reason it is important to have a Newton-Euler routine for the inverse dynamic formulation (see section 3.4.1).

Furthermore an error control by the compensator $C(s)$, could create troubles with non-minimum-phase system, but in this case, the rigid arm dynamics are minimum phase system.

3.6.1 Tuning of PD controller

One way to compute the external signal $u(t)$ is to use a proportional plus derivative (PD) feedback

$$u = -K_d \dot{e} - K_p e$$

Then the robot arm input, or else the computed-torque controller equation 3.19, becomes

$$\tau = M(\ddot{q}_d + K_d \dot{e} + K_p e) + C \quad (3.20)$$

and the closed-loop error dynamic is

$$\ddot{e} + K_d \dot{e} + K_p e = w \quad (3.21)$$

or, in state form

$$\frac{d}{dt} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -K_d & -K_p \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} u + \begin{bmatrix} 0 \\ I \end{bmatrix} w$$

The closed-loop characteristic polynomial is

$$p(s) = \| s^2 I + K_d s + K_p \|$$

which is stable if all the coefficients are positive. According to Routh-Hurwitz criterion the polynomial has only roots with negative real part only if all the coefficients have the same sign (this is true just for the second order polynomials). Since the inertia I is positive, also the other coefficients must be positive in order to obtain the stability.

Usually, the $n \times n$ gains matrices is chosen diagonal, so that

$$K_d = \text{diag}(k_{d_i}) \quad K_p = \text{diag}(k_{p_i})$$

then, the characteristic polynomial becomes

$$p(s) = \prod_{i=1}^n (s^2 + k_{d_i} s + k_{p_i})$$

so, for the Routh-Hurwitz criterion, the error system is asymptotically stable only if the k_{d_i} and k_{p_i} are all positive. Furthermore, we can say that as long as the disturbance $w(t)$ is bounded, so also the error $e(t)$ is bounded⁶. Note that, being the mass matrix $M(q)$ bounded (and so also its inverse is bounded), stating that $w(t)$ is bounded is equivalent to stating that τ_d is bounded too.

⁶The subsystem with input $w(t)$ and output $e(t)$ results Bounded-Input Bounded-Output (BIBO) stable.

However, choosing the PD gains matrices diagonal, we obtain a decoupling only in the outer loop, while the joint-controller remains strictly coupled. In fact, the multiplication by the mass matrix $M(q)$, which is not diagonal in general, and the adding of the nonlinear term $C(q, \dot{q})$ in feed-forward inner loop generates the coupling of the signal $u(t)$ among all the joint. Thus, the position $q(t)$ and the velocity $\dot{q}(t)$ information of all the n joints are necessary to compute the torque $\tau(t)$ for every single joint. In other words, every joint position and velocity information is necessary to compute the torque for a single joint.

The classic form of a second order characteristic polynomial is:

$$p(s) = s^2 + 2\zeta\omega_n + \omega_n^2$$

where ζ is the rate of critical damping and ω_n is the natural frequency of the system. By comparing this last one with the equation of single joint tracking error, yields:

$$k_{p_i} = \omega_n^2 \quad k_{d_i} = 2\zeta\omega_n$$

where, in this case, ζ and ω_n are the desired critical damping ratio and the natural frequency for the i -th joint controller. It is useful to select the response at the end of the arm (where the moving masses are lighter), faster than the response near the base, where the moving masses are heavier. Furthermore, it is crucial for a robotic arm not have any overshoot. All this gives the idea that a robotic arm must be *critically damping* or *over damping*: $\zeta \geq 1$. In the critically damping case, we obtain:

$$k_{p_i} = \omega_n \quad k_{d_i} = 2\sqrt{k_{p_i}}$$

Natural frequency ω_n governs the speed of response, in fact the raise time of the step response is proportional to the bandwidth, which in turn is proportional to the natural frequency. In light of this, one could choose the greatest possible value of ω_n , but is not like this. In fact, there are at least two reasons that make ω_n upper bounded:

1. real links, although are very stiff, have several vibration modes, whose first resonance frequency ω_r could be written as:

$$\omega_r = \sqrt{k_r/J}$$

with J and k_r the link inertia and stiffness (respectively). To avoid exciting the resonant mode that could be deleterious or destructive for some weak components, we should choose $\omega_n < \alpha\omega_r$, with $\alpha < 1$ factor of safety. The link inertia J can change during the robotic arm operation (it depends on the joints rotation $q(t)$), then its maximum value is used to calculate ω_r .

2. another reason for having n upper bounded ω_n , is given by consideration on the joints saturation. If the PD gains are too high, the torques $\tau(t)$ could reach their maximum limits.

Other considerations on the choice of k_{p_i} and k_{d_i} are based on the error limits. In fact, in case of critically damping systems, the position error $e_i(t)$ decreases as k_{p_i} grows, and the velocity error $\dot{e}_i(t)$ decreases as k_{d_i} grows.

3.7 Trajectory generator

In close approach maneuvers, one object is passive (the target) and the other object is active (the chaser) and trying to approach the target. According to [5], the equations which describe the relative orbital motion between chaser and target are called Clohessy-Wiltshire (CW) expressions. These equations describe the motion of the chaser with respect to the frame centered in the target center of mass. In particular in this frame the x axis is along the Earth-target radius, the y axis points in the local horizon of the target's orbit and the z axis is chosen to complete the right handed frame. The CW equations in hypothesis of circular orbit are now given⁷:

$$\begin{cases} \delta\ddot{x} &= 3\frac{\mu}{r_0^3}\delta x + 2\sqrt{\frac{\mu}{r_0^3}}\delta\dot{y} \\ \delta\ddot{y} &= -2\sqrt{\frac{\mu}{r_0^3}}\delta\dot{x} \\ \delta\ddot{z} &= -\frac{\mu}{r_0^3}\delta z \end{cases} \quad (3.22)$$

From equations 3.22 we can obtain position and velocity by some integration steps. We first define:

$$\delta r(t) = \begin{Bmatrix} \delta x(t) \\ \delta y(t) \\ \delta z(t) \end{Bmatrix} \quad \delta v(t) = \begin{Bmatrix} \delta\dot{x}(t) \\ \delta\dot{y}(t) \\ \delta\dot{z}(t) \end{Bmatrix} \quad (3.23)$$

whose corresponding initial values are:

$$\delta r_0 = \begin{Bmatrix} \delta x_0 \\ \delta y_0 \\ \delta z_0 \end{Bmatrix} \quad \delta v_0 = \begin{Bmatrix} \delta\dot{x}_0 \\ \delta\dot{y}_0 \\ \delta\dot{z}_0 \end{Bmatrix} \quad (3.24)$$

finally we obtain:

$$\begin{Bmatrix} \delta r(t) \\ \delta v(t) \end{Bmatrix} = \begin{bmatrix} \Psi_{rr}(t) & \Psi_{rv}(t) \\ \Psi_{vr}(t) & \Psi_{vv}(t) \end{bmatrix} \cdot \begin{Bmatrix} \delta r_0 \\ \delta v_0 \end{Bmatrix} \quad (3.25)$$

where

$$\Psi_{rr}(t) = \begin{bmatrix} 4 - 3\cos(nt) & 0 & 0 \\ 6[\sin(nt) - 1] & 1 & 0 \\ 0 & 0 & \cos(nt) \end{bmatrix} \quad (3.26)$$

$$\Psi_{rv}(t) = \frac{1}{n} \begin{bmatrix} \sin(nt) & 2[1 - \cos(nt)] & 0 \\ [\cos(nt) - 1] & [4\sin(nt) - 3nt] & 0 \\ 0 & 0 & \sin(nt) \end{bmatrix} \quad (3.27)$$

⁷We do not make all the algebraic steps to get the equations, but they are well explained in [2] or in [5].

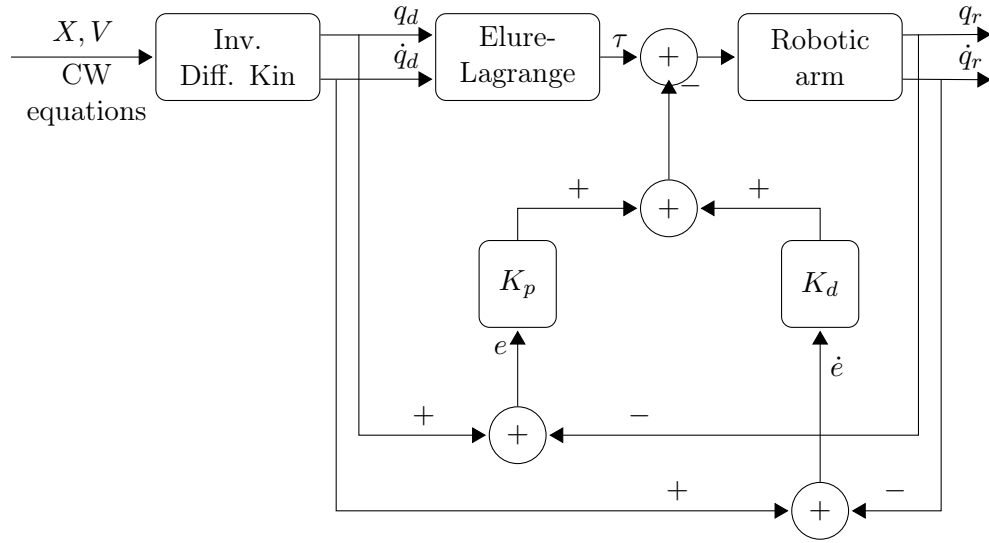


Figure 3.5: Control system based on Euler-Lagrange method. The block "Inv. Diff. Kin" is the block that resolve the inverse differential kinematics, whose block diagram is pictured in figure 3.3.

$$\Psi_{vr}(t) = \begin{bmatrix} 3n \sin(nt) & 0 & 0 \\ 6n [\cos(nt) - 1] & 0 & 0 \\ 0 & 0 & -n \sin(nt) \end{bmatrix} \quad (3.28)$$

$$\Psi_{vv}(t) = \begin{bmatrix} \cos(nt) & 2 \sin(nt) & 0 \\ -2 \sin(nt) & 4 \cos(nt) - 3 & 0 \\ 0 & 0 & \cos(nt) \end{bmatrix} \quad (3.29)$$

When a robotic arm facility is used for the simulation of orbital maneuvers, the relative motion is simulated correctly with the aid of CW expressions [1]. From the equations above we obtain the position vector \mathbf{X} and the velocity vector \mathbf{V} , and with the inverse differential kinematics we get the joint positions values at every time step. The complete block diagram of how we obtain the joints positions and velocities are pictured in figure 3.5, where we use the Euler-Lagrange based method to obtain the joints torques τ .

To simulate an impact we can calculate the Δv which the impact cause:

$$\Delta v = \int_{t_{\text{impact}}} \frac{|F|}{m} dt$$

now the compute of the modify trajectory is trivial [2].

Chapter 4

CANopen Communication

4.1 Introduction

Communication between PLC and EPOS2[®] controller is a crucial point of this thesis. In fact, two types of communications were available: the classical PLC and each EPOS2, and a more clean CANopen communication. The first needs of a large amount of cables and wires, since every controller has to be connected to a PLC output. Thus a CANopen communication was chosen. CANopen is a standardized application for distributed autonomous systems based on CANopen (Controller Area Network), which offers the following performance features:

- transmission of time-critical process data;
- standardized device (node) description (data, parameters, functions, programs) in the form of the so-called "Object Dictionary" (OD)
- access to all devices with standardized transmission protocol based on the producer-consumer principle;
- standardized services for node monitoring (node guarding, heartbeat), error signalization and network coordination;
- standardized system services for synchronous operations
- standardized help function for configuring and device identification number (node-ID), via the bus.

4.2 Physical structure of the CANopen network

In figure 4.1 it is shown the architecture of CANopen network. The physical medium is a differently driven 2-wire (CAN high and CAN low) bus line with common return. To avoid reflection of signal, both network ends must be terminated (using two resistors of 120Ω). The devices are identified by the node-ID, that is a code number from 1 to 127. For CANopen it is important to underline that no node-ID may exist twice, and all the devices must be configured with the same bit rate.

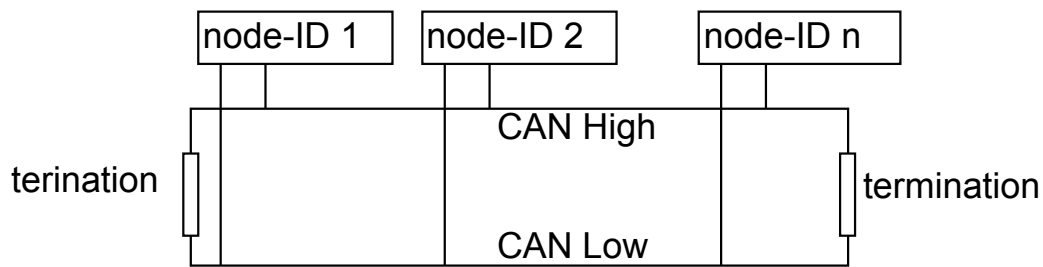


Figure 4.1: The physical structure of a CANopen network. The devices are connected to the bus line, where two terminations are plugged

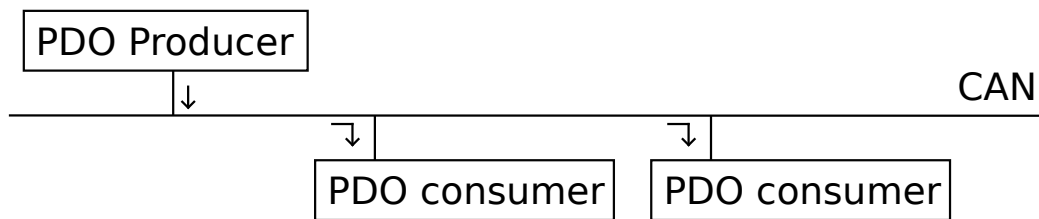


Figure 4.2: The Producer-Consumers principle on which the PDO transmission is based

4.3 Data transfer

CANopen represents a standardized application layer and communication profile [11]. A Data Frame is produced by a CAN node when it hears to transmit data or when another node requests a data. Within one frame up to 8 byte data can be transported.

CANopen provides some Communication Objects (COB). They are described by protocols and services. The predefined Communication Objects are: PDO Object, SDO Object, SYNC Object, EMERGENCY Object and NMT Services. First three communication objects are identify also by a proper address, the COB-ID, on EPOS2 all the COB-ID are immutable.

PDO Object

Process Data Object (PDO) communication follows the producer-consumer principle, as shown in figure 4.2. This means that a message sent by the producer node (the PLC) is received by all other nodes (the consumers, the EPOS2s), but only the node that has the specific node-ID is able to read the message, so in the message it is only necessary to identify the node-ID and write the operational instructions. In fact, the producer sends a Transmit PDO (TxPDO) with a specific identifier (node-ID) that corresponds to the identifier of the Receive PDO (RxPDO) of one or more consumers.

PDOs can be either Write or Read, depending on the nature of the device entry they describes. On EPOS2 the number of supported PDO is 4 TxPDO and 4 RxPDO, each of them can mapped up to 8 process variables, hence we are able to read and write up to 64 device entries. For our purpose we map at most 5 variables into RxPDO and 5 variables into RxPDO, because only these variables we need to

Table 4.1: COB-IDs for the Transmit PDOs and the Receive PDOs.

PDO	COB-ID
TxPDO 1	node-ID+180 _h
TxPDO 2	node-ID+200 _h
TxPDO 3	node-ID+280 _h
TxPDO 4	node-ID+300 _h
RxPDO 1	node-ID+380 _h
RxPDO 2	node-ID+400 _h
RxPDO 3	node-ID+480 _h
RxPDO 4	node-ID+500 _h

manipulate with the robotic arm programs.

Note that a PDO cannot be mapped (the voice PDOMapping is false), but we can mapped variables via PDO. We can mapped only variables which have the voice PDOMapping = true (or yes, or 1). Furthermore, only SDO communication is able to read and write in Object Dictionary, in fact with the PDO mapping we don't read or write the variable in the Object Dictionary directly, but we map them and read or write a copy of them.

CANopen communication distinguished three message triggered modes:

1. Message transmission is triggered by an internal event of the device
2. Transmission of Asynchronous PDOs are triggered by external request. Often we shall use this type of transmission because the external source is the PLC.
3. Synchronous PDOs are triggered only when a SYNC object appears, then the PDO is triggered within a specified time period. We will use this type of transmission to move all the robotic arm axis at once with the Interpolated Profile Mode (see section 6.4).

RxPDOs are identified with index from index 1400_h to index 1403_h, after from index 1600_h to index 1603_h (and its sub-indexes) the TxPDO mapping object are identified. While TxPDOs are identified with index from index 1800_h to index 1803_h, after from index 1A00_h to index 1A03_h (and its sub-indexes) the TxPDO mapping object are identified.

The PDO COB-ID, COB-ID is the node-ID number plus an hexadecimal number different for every PDO, as shown in table 4.1.

SDO Object

Service Data Object (SDO) provides the access to the Object Dictionary entries. Entries are different from the variables, because the entries are the device configuration parameters. The SDO communication messages are used for the device configuration during the Pre-Operational state, and for accessing the Object Dictionary during the "Operational" state (see below).

SDOs are identified with index 1200_h and its 3 sub-indexes. The SDO COB-ID is node-ID+580_h for the Transmit SDO and node-ID+600_h for the Receive SDO.

SYNC Object

The SYNC producer provides the synchronization of the SYNC consumer. But if the producer is not able to make a time stamp accurate enough, an EPOS2 can be used as the time stamp master maker.

As the SYNC consumer receives the signal, they start carrying out their synchronous tasks. In general, fixing of the transmission time of synchronous PDO message, with the periodicity of the SYNC Object's transmission guarantees the synchronization of the moving axis within microseconds. SYNC Object is identified with index 1005_h and the COB-ID is 80_h .

Synchronous transmission of a PDO means that the transmission is fixed in time with respect to the transmission of the SYNC Object. The synchronous PDO is transmitted within a given time window with respect to the SYNC transmission and one for every period of the SYNC.

EMERGENCY Object

Emergency message is triggered when a device internal fatal error occurs. It is transmitted from the device which has the fatal error to the other devices with the highest priority, thus making them suitable for "interrupt" type error. The Emergency message COB-ID is $\text{node-ID}+80_h$ and it is described in the object number 1014_h COB-ID EMCY.

NMT Services

The CANopen Network Management is node oriented and follows the master-slave principle. It requires one master device (the PLC), while the others devices are the slaves. The slaves are uniquely identified by the node-ID. NMT services provide the following functionality:

- Module Control Service for the slaves initialization.
- Error Control Service for supervision the communication state of the network nodes.
- Configuration Control Services for upload or download of configuration data from or to a network node.

The CANopen NMT Slave devices implement a finite state machine (figure 4.3) that brings every device into "Pre-Operational" state, as soon as the power is apply to them and once they are initialized.

Power on: the system is connected to a power source.

Initialization: in order to make a partial reset of nodes, this state is subdivide into three sub-state:

- **Reset Application:** here the parameters of the manufacturer-specific and of the standard device profile are reset to their power-on values, which are the latest saved values. Then the node changes to the next sub-state;

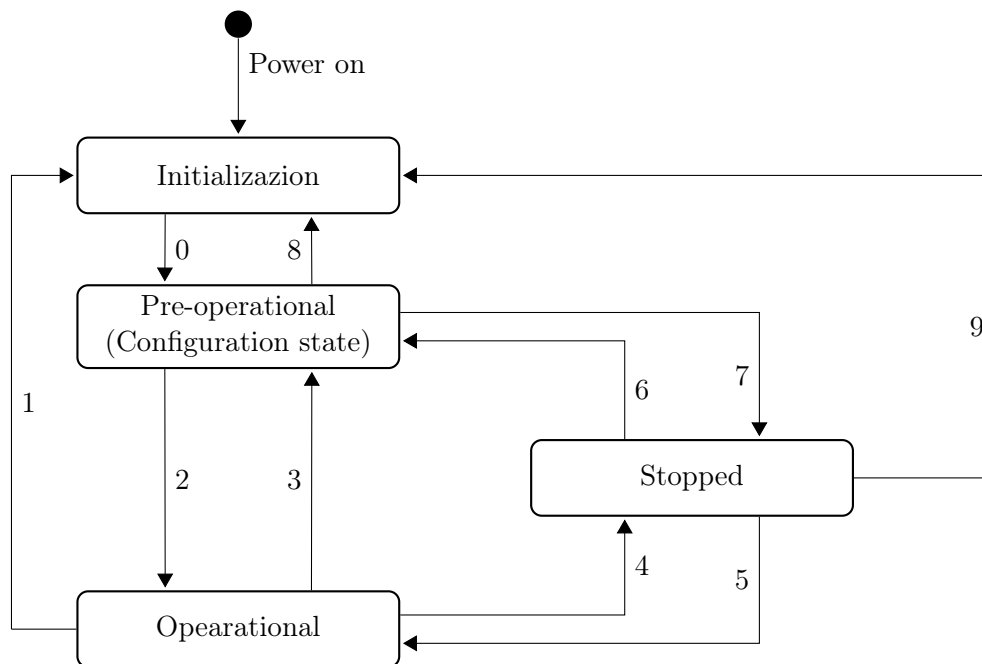


Figure 4.3: FSM of the CANopen NMT network slaves states

- **Reset Communication:** the parameters of the communication profile are reset to the power-on values. Then the node state changes to Initializing;
- **Initializing:** in this sub-state the node makes the basic initialization, e.g. host controller, CANopen controller, software and firmware verify. After basic initialization, the node sends the boot-up message, and changes itself into the Pre-Operational state.

Pre-Operational: the main use of this state is the configuration of CANopen devices via SDO (using a configuration tool). Therefore, PDO communication is not permitted. The NMT master may switch from "Pre-Operational" to "Operational" and vice versa.

Operational: this is the state in which the devices work, meaning that the PDO communication is possible. "Operational" can be used to achieve certain application behavior defined by the device profile's scope. Here all the communication objects are active. Object Dictionary access via SDO is possible.

Stopped: force to stop every SDO and PDO communication.

In table 4.2 the transitions shown in figure 4.3 are explained.

The COB-Id for the NMT services is 00_h .

4.4 Object Dictionary

One of the most important properties of CANopen is a standardized protocol called Object Dictionary (OD), it fully describes the device. Every device in the network is

Table 4.2: Explanation of the transitions between the states in FSM in figure 4.3. *Transition 0 is automatically executes. **Indicates the remote bit, the bit 9 of the Statusword.

Transition	Service	Remote**	Functionality
0*, 3, 6	Enter Pre-Operational	0	Communication: SDO protocol; Emergency Objects; NMT protocol
1, 8, 9	Reset Communication	0	Calculates SDO COB-ID. Setup dynamic PDO mapping and calculates PDO COB-ID. Communication: during initialization, no communication is active; upon completion a boot-up message is sent to the CAN bus.
1, 8, 9	Reset Node	0	General reset of EPOS2 software (as the same effect as turning off and on the supply voltage). No saved parameters will be overwritten with the values saved in EEPROM.
2, 5	Start Remote Node	1	Communication: SDO protocol; PDO protocol; Emergency Objects; NMT protocol.
4, 7	Stop Remote Node	0	Communication: NMT protocol; Heartbeat protocol.

described with OD. The OD is a table with the entries of all devices, thus it is easy access to all data functions and parameters of a single device using is a 16-bit index, and a 8-bit sub-index. A 16-bit index is used to address all entries within the OD. In the case of a simple variable, it references the value of this variable directly (this happens, for example, when we write or read the values of process variables during a program) and the sub-index is always zero. In case of a complex variable (such as arrays, or structures), the index refers to the entire data structure. The elements of the data structure are addressed by the sub-index.

The number of index helps us to understand what area the variable refers to:

- 0000_h: reserved.
- 0001_h ÷ 0099_h: data types (not supported on EPOS2).
- AAA0_h ÷ 0FFF_h: reserved.
- 1000_h ÷ 1FFF_h: Communication Profile area (CiA 301).
- 2000_h ÷ 5FFF_h: Manufacturer specific Profile area (Maxon Motor).
- 6000_h ÷ 9FFF_h: Standardized device Motion Control area (CiA 402).
- A000_h ÷ FFFF_h: reserved.

The Electronic Data Sheet (EDS) is a file containing all the object dictionary entries, so all data types and functions of each device are listed [7]. Depending on the the object, it can be read and write (RW), or only write (RO).

Object dictionary has 149 entries (without considering sub indexes of the indexes), only three of them are mandatory and automatically compiled with the EPOS2 Studio configuration tool. These are the entries number:

- 1000_h Device type : specify the device type. The value 402 (0192_h) means that the device follows the CiA 402 Device Profile Drive and Motion Control.
- 1001_h Error Register: an error register for the device. The device maps internal errors in this byte (see section 4.5).
- 1018_h Identity Object: the CANopen vendor identification of "maxon motor ag" defined by CiA is 000000FB_h

Others objects require to be completed manually. They are discussed in the following, in table 4.3 there are some brief information of them.

These entries are shown and described below, and is a good thing change all the following value during EPOS2 aren't in operational state, and upload the eds file with Automation Studio in order to reboot the device.

Many entries are physical quantities for whom a measure unit is required:

- **Position.** Position unit is the "Step", defined as:

$$1\text{Step} = 4 \times \text{Encoder Counts per Revolution}$$

- **Velocity.** Velocity unit is the "rpm" (Revolution per Minute).
- **Acceleration.** Acceleration unit is the "rpm/s" (Revolution per Minute per second).

Table 4.3: Brief overview of the objects discussed in the following. The symbol * means that the value depends on the motor. The abbreviations "int" and "uint" means "integer" and "unsigned integer" respectively, the number next to them indicate them length in term of number of bits. All the object are of type RW. Symbols – indicates that the value is not available

Name	Index [Hex]	Type	Value	
			Default	Range
Consumer 1 Heartbeat Time	1016 _{sub1}	uint 32	0	–
Producer Heartbeat Time	1017 _{sub0}	uint 16	0	–
CAN Bitrates	2001 _{sub0}	uint 16	9	0 ÷ 9
Pulse Number Incremental Encoder 1	2210 _{sub1}	uint 32	500	16 ÷ 2500000
Position Sensor Type	2210 _{sub1}	uint 32	01	0 ÷ 8
Gear Ratio Numerator	2230 _{sub1}	uint 32	0	1 ÷ 4294967295
Gear Ratio Denominator	2230 _{sub2}	uint 32	0	1 ÷ 65535
Abort Connection Option Code	6007 _{sub0}	int 16	3	1 ÷ 3
Shutdown Option Code	605B _{sub0}	int 16	0	0 ÷ 1
Disable Operation Option Code	605C _{sub0}	int 16	1	0 ÷ 1
Fault Reaction Option Code	605E _{sub0}	int 16	2	–1 ÷ 2
Max. Following Error	6065 _h	uint 32	2000	0 ÷ 4294967295
Home Offset	607C _{sub0}	int 32	0	–
Min. Software Position Limit	607D _{sub1}	int 32	–2147483648	±2147483648
Max. Software Position Limit	607D _{sub1}	int 32	2147483648	±2147483648
Maximal Profile Velocity	607F _{sub0}	uint 32	25000	see table 4.10
Profile Velocity	6081 _{sub0}	uint 32	1000	see table 4.10
Profile Acceleration	6083 _{sub0}	uint 32	10000	*
Profile Deceleration	6084 _{sub0}	uint 32	10000	*
Homing Method	6098 _{sub0}	uint 8	7	–4 ÷ 35
Speed for Switch Search	6099 _{sub1}	uint 32	100	*
Speed for Zero Search	6099 _{sub2}	uint 32	10	*
Homing Acceleration	609A _{sub0}	uint 32	1000	*
Max Acceleration	60C5 _{sub0}	uint 32	4294967295	*
Motor Type	6402 _{sub0}	uint 16	10	see table 4.11
Continuous Current Limit	6410 _{sub1}	uint 16	*	*
Output Current Limit	6410 _{sub2}	uint 16	*	*
Pole Pair Number	6410 _{sub3}	uint 8	1	2 ÷ 255
Maximal Motor Speed	6410 _{sub4}	uint 32	2500	see table 4.12
Thermal Time Constant Winding	6410 _{sub5}	uint 16	40	1 ÷ 5400

Table 4.4: Available Bit rates and meaning of the index in the 0x00 entries of the OB

Table index	Bit rate [<i>Kbit/sec</i>]
0	1000
1	800
2	500
3	250
4	125
5	reserved
6	50
7	20
8	10 (not supported)
9	automatic bit rate detection

Consumer Heartbeat Time 1016_h

Object number 1016_h sets the heartbeat time in milliseconds for the Consumers, the devices (see section 4.6). All the devices must have the same heartbeat time. For every device there are two consumer heart beat time, we use only the first with the sub index 1:

- Consumer 1 Heartbeat Time 1016_{sub1}: sets the value of the consumer heart-beat time.

Producer Heartbeat Time 1017_h

Object number 1017_h sets the heartbeat time in milliseconds for the Producer or the PLC (see section 4.6).

CAN Bitrate 2001_h

Object number 2001_h sets the bit rate of the CANopen network. Note that all the devices must have the same Bit rate (see section 4.3). There are several values of the bit rates depending on the device and the amount of the data to be transferred. It varies from 0 to 9 depending on the value of the bit rate; the number 5 is reserved and the number 8 is not supported by the EPOS2. The choice depends on the tasks the EPOS2 will fulfill. In table 4.4 are listed the available bit rates supported by EPOS2

Sensor Configuration 2210_h

Many sensor features are described with the objects 2210_h and its sub indexes. In EPOS2 we are using, it is possible to use only one sensor (called Incremental Encoder 1), in figure 4.4 is shown the architecture of regulation, sensors and gear.

The pulse number must be set to the connected incremental encoder's number of pulse per revolution. The pulse number of incremental encoder 1 (2210_{sub1}) must

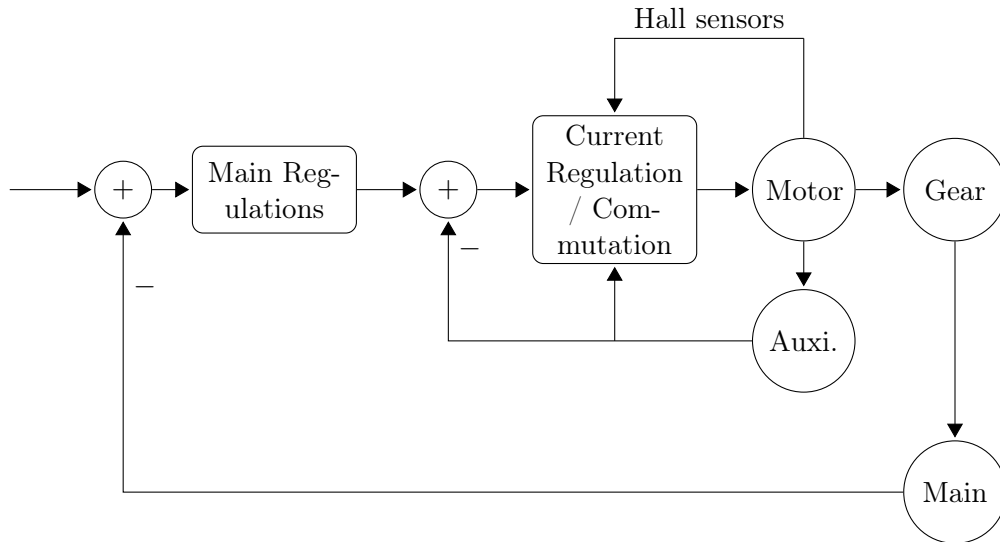


Figure 4.4: Overview of the regulation, gear and sensor architecture. Blocks "Auxi" and "Main" stand for for Auxiliary and Main sensor respectively.

be greater or equal to $16 \cdot (\text{pole pair number})$, if this condition is not respected, a Position Sensor Error 7320_h will be set at "Enable Operation" command.

We use two types of encoder. Motor 2 is connected to an incremental encoder with index (3-channel), while motors 1, 3, 4, 5 and 6 are connected to to an incremental encoder without index (2-channel). The type of encoder is set in the object 2210_{sub2}.

Object 2210_h has four entries, but in our case, only entries number 1 and 2 are useful:

- Pulse Number Incremental Encoder 1 2210_{sub1}: the value depends on the encoder, so the value is equal to the value showed in chapter Robotic Arm Structure.
- Position Sensor Type 2210_{sub2}: as shown above, we use two type of encoder, so the value is 1 for motor 2, and it is 2 for motors 1, 3, 4, 5 and 6. All the other values is non-supported by EPOS2 24/5 or they aren't in our case.

Gear Configuration 2230_h

Object 2230_h and its entries define the gear ratio given in terms of numerator and denominator:

$$\text{gear ratio} = \frac{\text{numerator}}{\text{denominator}}$$

these are set with the entries:

- Gear Ratio Numerator 2230_{sub1}: defines the value of the numerator.
- Gear Ratio Denominator 2230_{sub2}: defines the value of the denominator.

Object 2260 defines also the maximal gear velocity with sub index number 3, but this is strictly connected with the Motor Maximal Speed described in object 6410_h, so we don't set this entry.

Table 4.5: Abort connection option code

Value	Description
1	Fault signal only
2	Disable voltage command
3 (default)	Quick stop command

Table 4.6: Shutdown option code

Value	Description
0 (default)	Disable drive function (switch off power stage)
1	Decelerate with slowdown ramp; disabling of the drive function

Abort Connection Option Code 6007_h

This object specifies what action is performed when one of the errors labeled with an "a" in table 4.14 will be detected (see table 4.5). It contains all communication errors (CANopen errors included).

If the value is set to 1 the Emergency Message Frame is sent out and the Bit 7 of the "Statusword" is set to 1 if an error occurs.

"Disable voltage" (transitions number 7, 9, 10, 12, 13 and 17) and "Quick stop" (transitions number 9, 10, 13, and 17) commands are command that change the state of the EPOS2, they are strictly connected with the transitions number 12

Shutdown Option Code $605B_h$

CiA 402 protocol, defines a finite state machine for the controller. The controller is driven by commands called transitions which change its state. The object $605B_h$ indicates the action that will be performed by transitions number 8 and 9, or when the EPOS2 is driven from state "Operation enabled" to state "Ready to switch on" or "Switch on disable". Two choices are available as shown in table 4.6

Disable Operation Option Code $605C_h$

Like the previous one, object $605C_h$ describes the actions that will be performed during transition number 5, or from the state from "Operation enabled" to state "Switched on". The values are shown in table 4.7 Disable drive function makes the power stage disable, the motor will continue to move due to its inertia and gravity. Slow down on quick stop ramp causes the fast stop of the motor, this could be harmful for links and joints because of their inertia. Finally for our purpose, only

Table 4.7: Disable operation option code

Value	Description
0	Disable drive function (switch off power stage)
1 (default)	Decelerate with slowdown ramp; disabling of the drive function

Table 4.8: Fault reaction option code. The four behavior for EPOS2

Value	Description
-1	Fault signal only
0	Disable drive function
1 (default)	Slow down on slow down ramp
2	Slow down on Quick stop ramp

Slow down on slow down ramp is the right choice.

Value of the "slow down ramp" is the same as the profile Deceleration object.

Fault Reaction Option Code $605E_h$

When an error labeled with a "f" in tables 4.14 occurs, we can choose between four behavior for EPOS2 (table 4.8). A fault signal only is not recommended, because the fault becomes only an "Emergency Message" which is visible only with the EPOS Studio if the device is connected with computer via USB cable; when the error occurs the Bit 7 of the "Statusword" is set to 1. Disable drive function makes the power to the motor disable, which continues to move due to its inertia and gravity. Slow down on quick stop ramp causes the fast stop of the motor, this could be harmful for links and joints because of them inertia. Finally for our purpose, only Slow down on slow down ramp is the right choice (note that is also the default value).

Maximal Following Error 6065_h

Maximal Following error sets the maximum difference between Position Actual Value (6064_h) and Position Demand Value (6062_h). This object is used in all the Profile mode: Profile Position Mode (subsection 6.3.2) and Interpolated Profile Mode (section 6.4).

When the difference between Position Actual Value and Position Demand Value reaches the value set in Maximal Following Error, the error number 8611_h "Following error" raises and an Emergency message is sent over the CANopen network. The EPOS2 that sent the message turns itself into Fault state.

Home Offset $607C_h$

This object describes the distance (in term of joint rotation) between the home and the zero of the robotic arm. The homing is defined by the limit switches for all the links, and it is the position in which the robotic arm stays when the power stage is disconnected, so in this position there is a mechanical structure that supports the robotic arm¹. The zero, instead, is the position in which the robotic arm starts its operation, so this position must be far away from the mechanical structure to avoid collisions.

During the homing operation, the robotic arm first searches the limit switches, and then it reaches the zero position. The value of the offset is given in terms of

¹We use a mechanical structure because in our simulator motor brakes have not been installed.

Table 4.9: Value of the Homing Offset for the six joints

Joint number	Home Offset value
1	val1
2	val2
3	val3
4	val4
5	val5
6	val6

Table 4.10: Upper limit of maximum profile velocity

Maximal profile velocity		Motor max speed > Gear max speed	Motor max speed < Gear max speed
Gear	no	Gear maximal speed	Motor maximal speed
	yes	$(\text{Gear max speed}) * (\text{Gear ratio Denominator}) / (\text{Gear ratio Numerator})$	$(\text{Motor max speed}) * (\text{Gear ratio Denominator}) / (\text{Gear ratio Numerator})$

encoder pulses. We can also set the zero position to be the absolute position for the next (see section 6.3.1) motions. In table 4.9 all the six homing offset are given.

Software Position limit $607D_h$

This object allows to set via software the minimal and maximal position limit. Position limit is given in position units. When a move will runs out these limits, the error number $FF09_h$ "Software position limit error" raises.

Minimal and maximal position limit are set with the objects:

- Minimal Position Limit $607D_h$ sub01_h: sets the absolute negative position for the Position Demand value.
- Maximal Position Limit $607D_h$ sub02_h: sets the absolute positive position for the Position Demand value.

Maximal Profile Velocity $607F_h$

With this object we set the velocity limit in the position move used in all the position modes (Profile Position, Position and Interpolated Position). It is also used in all the velocity modes, but we don't use these modes of operation.

The value depends if we use or not the gear. The value is given in rpm (see table 4.10). In our case the gear speed is lower than the motor speed, so we use second column formula, the values are listed in the DCF file for each motor (see section 5.1.2).

Profile Velocity 6081_h

The profile velocity is the velocity reached at the end of the acceleration ramp during a profiled move.

After we have made the simulation with Simulink Simscape we found the vectors of Positions and Velocities for the six joints. But we didn't care about the gears (for the sake of simplicity), so the trajectory we get from simulation assumes that all the joints have the same velocity. Hence we have to modify the value of the Profile Velocity in the DCF file so as to obtain the same velocities for all the joints shaft. Where the shaft is that at the downstream of the gear. Another reason to modify this value, is due to the fact that we made the tuning without the links, so it could be non-optimal. The optimal value will be found only with tests when the entire robotic arm mounted (see section 7.2). In the DCF file (see section 5.1.2) there are the value for each motor.

Profile Acceleration 6083_h

This values is used as acceleration in position (or velocity modes) modes. The value for each motor is in the DCF file (see section 5.1.2). For the Profile Acceleration (and Deceleration), we can make the same observation we made for the Profile Velocity. Also the Profile Acceleration (and Deceleration) the value was found without the link, so its value could be non-optimal. In section 7.2 the optimal values are shown.

Profile Deceleration 6084_h

This values is used as deceleration in position (or velocity modes) modes. The value for each motor is in the DCF file (see section 5.1.2).

Homing Method 6098_h

Object 6098_h is used to select the desired homing method. 16 methods are available, we choose the method number 1 and number 2. This choice is due to the structure and hardware used for the robotic arm (see section 2.1). The explanation of Homing methods number 1 and 2 is in subsection 6.3.1.

Homing Speeds 6099_h

This is used to set the velocity for searching the limit switch and the velocity to reach the zero position. There are two sub indexes:

- Speed for Switch Search 6099_{sub1}: set the velocity to search the limit switch during the homing operation.
- Speed for Zero Search 6099_{sub2}: set the velocity to reach the zero position. After the homing operations are completed, robotic arm reach the zero position with this velocity.

The upper limits for both velocities are equal to the Maximal Profile Velocity.

Homing Acceleration 609A_h

Used to define the acceleration and deceleration ramps in the homing profile. We may choose (without any problem) this value as the same value of Profile Acceleration.

Table 4.11: Motor types according to CiA 402

Value	CiA 402 Motor Type	Description
1	Phase-modulated DC motor	brushed DC motor
10	Sinusoidal PM Bl motor	EC motor sinus commuted with Hall sensors and Incremental Encoder 1
11	Trapezoidal PM Bl motor	EC motor block commuted only with Hall sensors
65535	Manufacturer-specific	EC motor sinus commuted with Hall sensors and Incremental Encoder 2 (only supported with EPOS2 70/10 and EPOS2 50/5)

Max Acceleration 60C5_h

This object permits to limit the acceleration and deceleration to prevent mechanical damages. This value is the limit of the other acceleration and deceleration objects. The value is given in [*rpm/sec*].

Motor Type 6402_h

With object 6402_h we select the type of the motor driven by the controller. The types are shown in table 4.11.

All the robotic arm motors are of type 10.

Motor Data 6410_h

After the control objects (such as Controlword, Statusword, Mode of operation etc), object 6410_h is one of the most important objects because it sets many upper limits, first of all current and velocity limits. The object's entries are:

- Continuous Current Limit 6410_{sub1}: represents the maximal permissible continuous current limit of the motor [mA]. Operating the motor at this current and at 25 °C will cause the winding reach the maximal winding temperature².
- Output Current Limit 6410_{sub2}: maxon motor recommends to set the output current limit at a value double of continuous current limit [mA]
- Pole Pair Number 6410_{sub3}: number of magnetic pole pairs (number of poles divided by 2) of the rotor of a brushless DC motor.
- Maximal Motor Speed 6410_{sub4}: to avoid mechanical destroys and make the robotic arm more safe it is possible to limit the velocity. See table 4.12.

²If a heat sink is used, this value can increased.

Table 4.12: Compute of the maximum motor speed in base of the CiA 402 motor type. All the robotic arm motors are of type 10

Motor Type	Description	Maximum Velocity [rpm]
1	brushed DC motor	25000
10	EC motor sinus commuted	25000 / pole pairs number
11	EC motor block commuted	100000 / pole pairs number
65535	EC motor sinus Inc2	25000 / pole pairs number

- **Thermal Time Constant Winding:** is used to calculate the time how long the maximum output current is allowed for the connected motor. It is also a constant that identify physically the motor.

If the pole pairs number is set to a value that is in conflict with the actual resolution of the encoder used for commutation, a Position Sensor Error 7320_h will be set on "Enable Operation" command.

Encoder Resolution [mm/rev] > 64[mm/rev] · pole pairs number

Note that the values of continuous current limit, pole pairs number, maximal motor speed and thermal time constant winding are proper of the single motor so they are all listed in chapter Robotic Arm structure. It may be that the maximal motor velocity is less than that listed, this is due to security.

4.5 Device errors

CANopen devices are able to send to the producer the Error message, which is described with the error code³ in 8 byte. The error code is an hexadecimal number that specifies the error categories, as described in table 4.13(a).

The OD index of error is 1003_h, where the error history is stored. The error register is content of the OD entry 1001_h, with bit-wise coding of the error cause. The meaning of the triggered bit that specifies the error cause is shown in table 4.13(b).

The EPOS2 device can detect internal errors caused by EPOS2 malfunctions. When one of these errors occurs, EPOS2 will transmit an Emergency message over the CANopen network using the COB-ID EMCY (1014_h).

The most frequent EPOS2 internal errors are listed in table 4.2, for all the other errors we remand to the EPOS2 manuals[12]. The errors are self-explanatory through the name of the specific error, and with the Error Code and the Error Register we can identify the source of the error. In most cases the effect is to turn the EPOS2 into the Fault state (Statusword bit 3 high), so the only reaction is to give the command Fault Reset by raising the Controlword bit 7. For more information we remand the reader to the EPOS2 manuals[12], or in a much more practical way we suggest to connect the EPOS2s to the pc via USB cable and watch what the error is about with the EPOS Studio.

³The error codes are specified in CiA (CANopen in Automation) 301 protocol DS [6]

Table 4.13: Error message explanation

(a) Description of the category of the error by error code

Error code (hex)	Error category description
00xx	Error Reset / No Error
10xx	Generic Error
2xxx	Current
3xxx	Voltage
4xxx	Temperature
50xx	Device Hardware
6xxx	Device Software
70xx	Additional Modules
8xxx	Monitoring
90xx	External Error
F0xx	Additional Function
FFxx	Device Specific

(b) Description of the case of the error by the triggered bit

Bit on (bin)	Error cause
0	Generic error
1	Current
2	Voltage
3	Temperature
4	Communication Error
5	Device Profile Specific
6	Reserved (always 0)
7	Manufacturer Specific

Table 4.14: EPOS2 internal errors. In fourth column there is how the EPOS2 reacts to the error.

Error name	Error code	Bit triggered	Error reaction
No error	0000 _h	—	—
Generic error	1000 _h	0	d
Over current error	2310 _h	1	d
Short circuit/Earth leakage error	2320 _h	1	d
Over voltage error	3210 _h	2	d
Under voltage error	3220 _h	2	d
Over temperature error	4210 _h	3	d
Internal software error	6100 _h	5	d
Software parameters error	6320 _h	5	f
Position sensor error	7320 _h	5	d
CAN overrun error (object lost)	8110 _h	4	a
CAN overrun error	8111 _h	4	a
CAN passive mode error	8120 _h	4	a
CAN heartbeat error	8130 _h	4	a
CAN Rx queue overflow error	81FE _h	4	a
CAN Tx queue overflow error	81FF _h	4	a
Following error	8611 _h	5	f
Hall sensor error	FF01 _h	7	d
Index processing error	FF02 _h	7	d
Encoder resolution error	FF03 _h	7	d
Hall sensor not found error	FF04 _h	7	d
Negative limit switch error	FF06 _h	7	f
Positive limit switch error	FF07 _h	7	f
Hall angle detection error	FF08 _h	7	f
Software position limit error	0000 _h	7	f
Interpolated Position Mode error	FF0C _h	5	f
Auto tuning identification error	FF0D _h	5	d
Gear scaling factor error	FF0E _h	5	d
Main sensor direction error	FF11 _h	5	d

About table 4.14 we have to say that in column "Error reaction", errors with a "d" will turn EPOS2 into disable (red LED on), errors with an "f" have the effect specified with the object "Fault Reaction Option Code (605E_h)", and errors with an "a" have the affect specified with the object "Abort Connection Reaction Option Code (60507_h)".

By default, EPOS2 starts into fault state, with the errors "CAN passive mode error" and "CAN Tx queue overflow error". However by triggering the Controlword Fault Reaction bit, these errors disappear.

All the errors with Error Register bits 5 and 7 triggered are due to wrong parameters value with respect to the same parameters value on the DCF file.

The "Software position limit error" occurs when the actual position runs out of the position imposed by the "Software Position Limit (607D_h)" object.

The "Following error" is the consequence of the "Maximal Following Error (6565_h)" object . Error "Interpolate Profile Mode error" happens only in Interpolated Profile mode, and its description is given by reading the Interpolation Buffer Status explained in 6.4.

The "Negative limit switch error" and "Positive limit switch error" indicates that the limit switches for the Homing are ON, while they have to be Low, it is a wiring error, because both the switches must be Normally Open.

4.6 Device monitoring via Heartbeat messages

With node monitoring based on the Heartbeat principle, a node transmits its communication ability (heartbeat message) automatically, at regular time intervals. The time intervals between two heartbeat messages is called "Heartbeat producer time", and it is configured via the object dictionary entry 0x1017⁴. The "Heartbeat consumer time" (object number 0x1016 in the OD) describes the maximum time in which an heartbeat message is expected from a particular node.

In order to avoid false NMT errors due to delay of consumer messages caused by generation and identification time, it is better that the Producer Heartbeat Time is longer than the Consumer one:

$$\text{Consumer heart beat time} \geq \text{Producer heart beat time} + 5 \text{ milliseconds}$$

We have chosen that Consumer Heartbeat Time is longer than 500ms the Consumer Heartbeat Time.

⁴A value of zero disables the node guarding via heartbeat. A node monitoring is always necessary, so if we don't use the heartbeat guarding, we must use a node monitoring via Node-Guarding. We use only the heartbeat method.

Chapter 5

Electronic hardware Configuration and settings

The Electronic hardware is composed of the PLC which is the manager of the network; the six EPOS2 boards which are the motors controllers; and the six brushless motors. Configuration consists in some operations which set the motors parameters and the EPOS2s features. These operations are made with the aid of two software: EPOS Studio (by MAXON[®]) which makes the tuning of the EPOS2 controllers, and Automation Studio (by B&R Automation[®]) which generates all the necessary settings for the communication between the PLC and EPOS2s controller. (figure 5.1)

5.1 MAXON[®]EPOS2 Controller

The EPOS2 is an electronic board which is connected both with the motor and with the PLC. The EPOS2s are connected with the motor via the three phase winding cables, the encoder and Hall sensor wiring. The EPOS2s are also connected among each other via the CANopen cable and only the first EPOS2 is connected to the PLC via CANopen cable (figure 5.2).

After all the motors and CANopen connections are made, the only thing we have to do on the EPOS2s board is setting the EPOS2s node-ID by acting in their first seven DIP switches shown in figure 5.3 (it is a binary code with 7 bit). A CANopen network must be terminated, so we set to ON the 8th DIP switch of the last EPOS2 in the network (the last EPOS2 is the one whom node-ID is the highest).

5.1.1 EPOS2 control architecture

The EPOS2s provides an accurate position control. This is done with a set of control structures which are now illustrated.

In figure 5.4 there is the overview of the EPOS2 control architecture. EPOS2 controller is made of three main control structures: the Current Control (figure 5.5), the Velocity Control (figure 5.7) and the position Control (figure 5.8). In the figure we refer to a Velocity and position plant which are showed in figure 5.6. In this

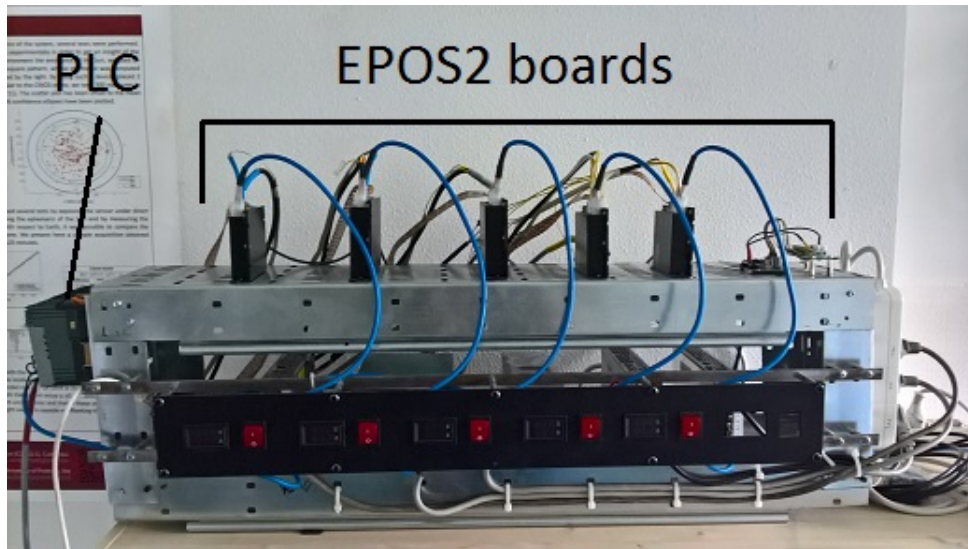


Figure 5.1: The electronic rack. On the left there is the PLC. Above there are the six EPOS2s with the motors and the CANopen wirings. Below there is the power supply.

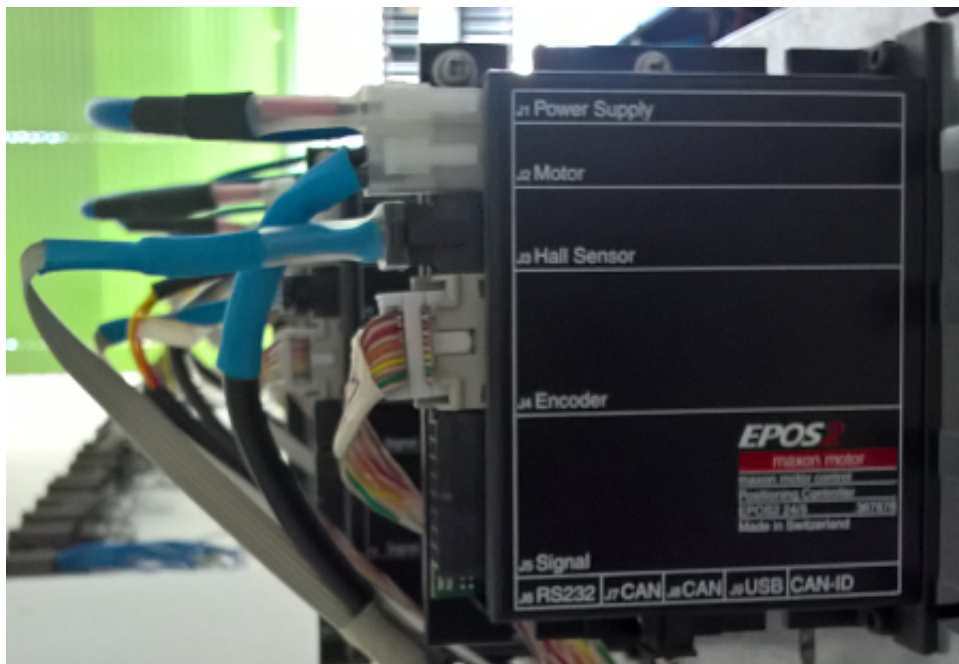


Figure 5.2: The EPOS2 board. We see the two CANopen connectors (one for the previous and one for the following board), the motors connections modules, the power supply connectors, the input modules and the USB connector. It also indicates the place where the DIP switches are for the node-ID configuration.

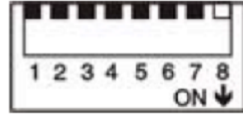


Figure 5.3: EPOS2 DIP switches. The first seven are used to set the EPOS2 node-ID, while the 8th is used to enable the second termination of the CANopen network

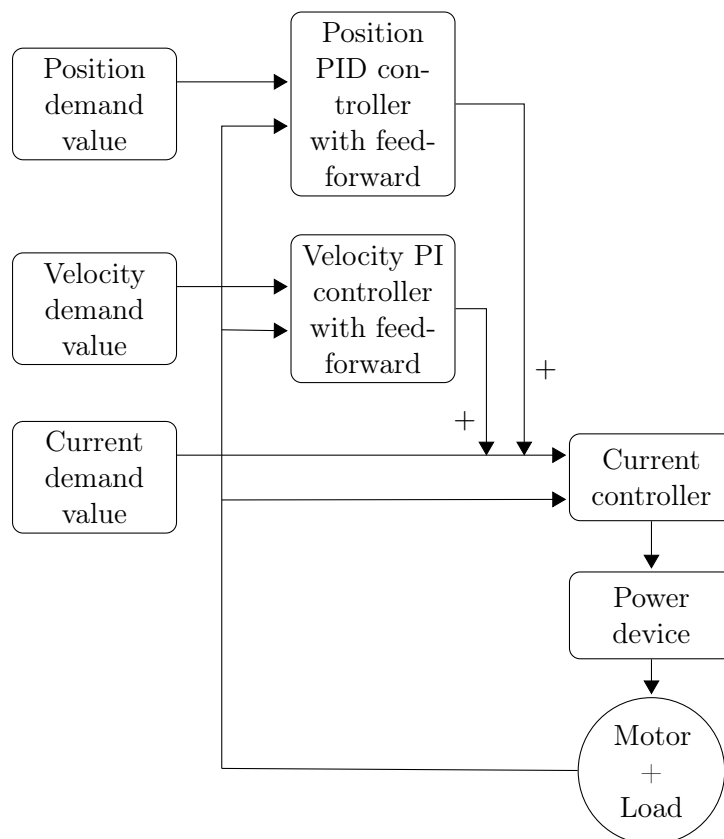


Figure 5.4: Control Overview

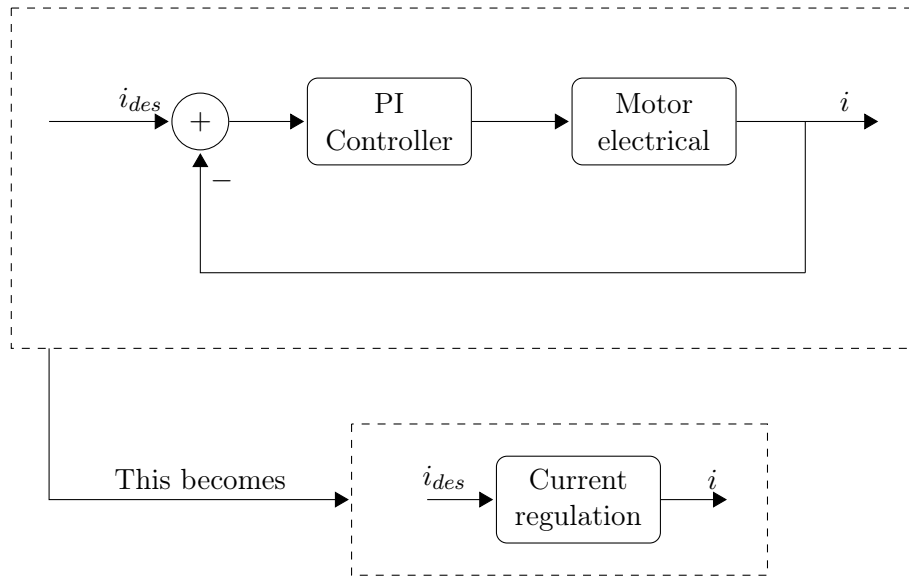


Figure 5.5: Current control block diagram

last figure the difference between "Velocity plant" and "Position plant" is just an integration, in fact: $j = \frac{1}{s}\omega$.

After all those controllers are connected together in one single controller called Position Regulation (figure 5.9). Where the block "Motion Trajectory Planning" is a function provided by MAXON[®], which makes the position and acceleration control, from the "Position Demand value". The Function creates a polynomial function for the position, starting from a trapezoidal shaped velocity function (see also 6.3).

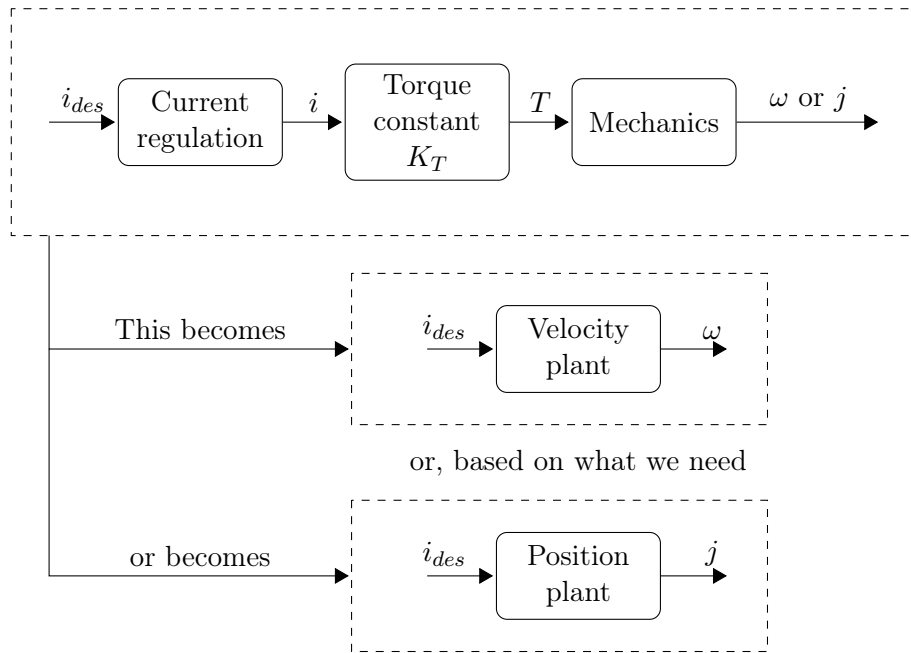


Figure 5.6: Velocity and position plant

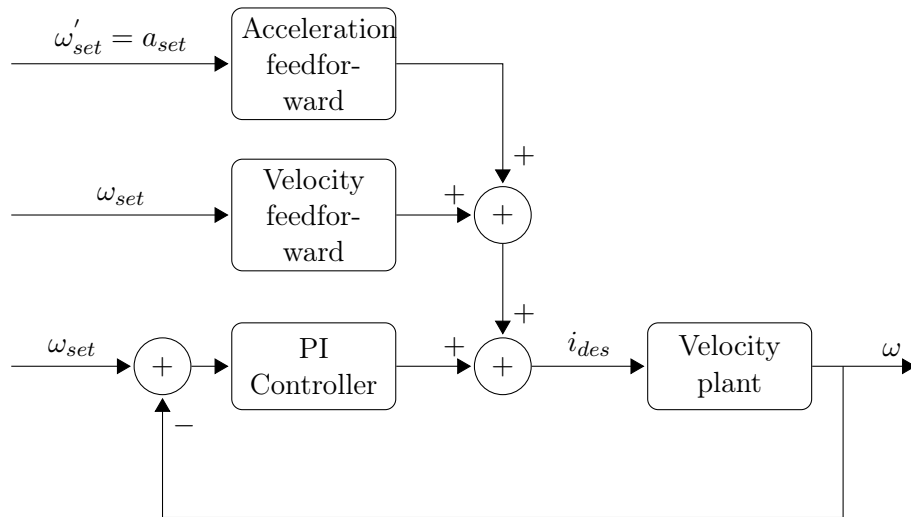


Figure 5.7: Velocity control

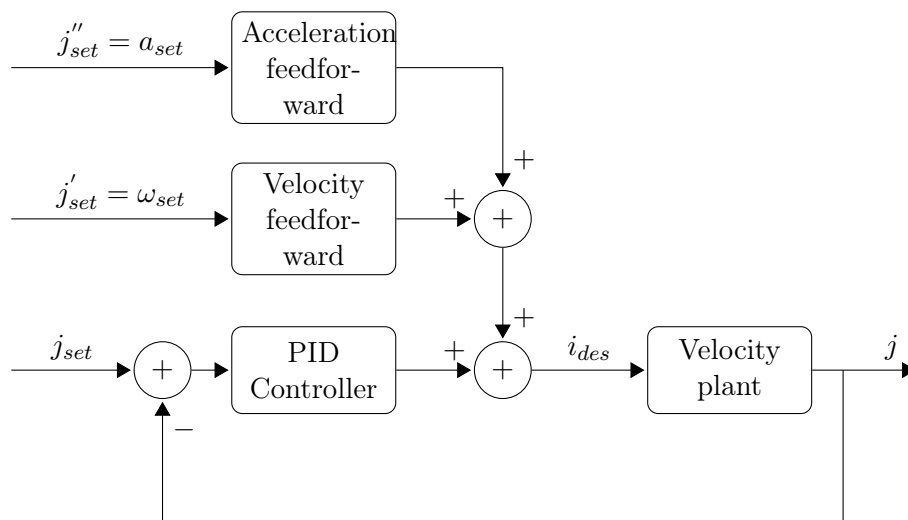


Figure 5.8: Position control

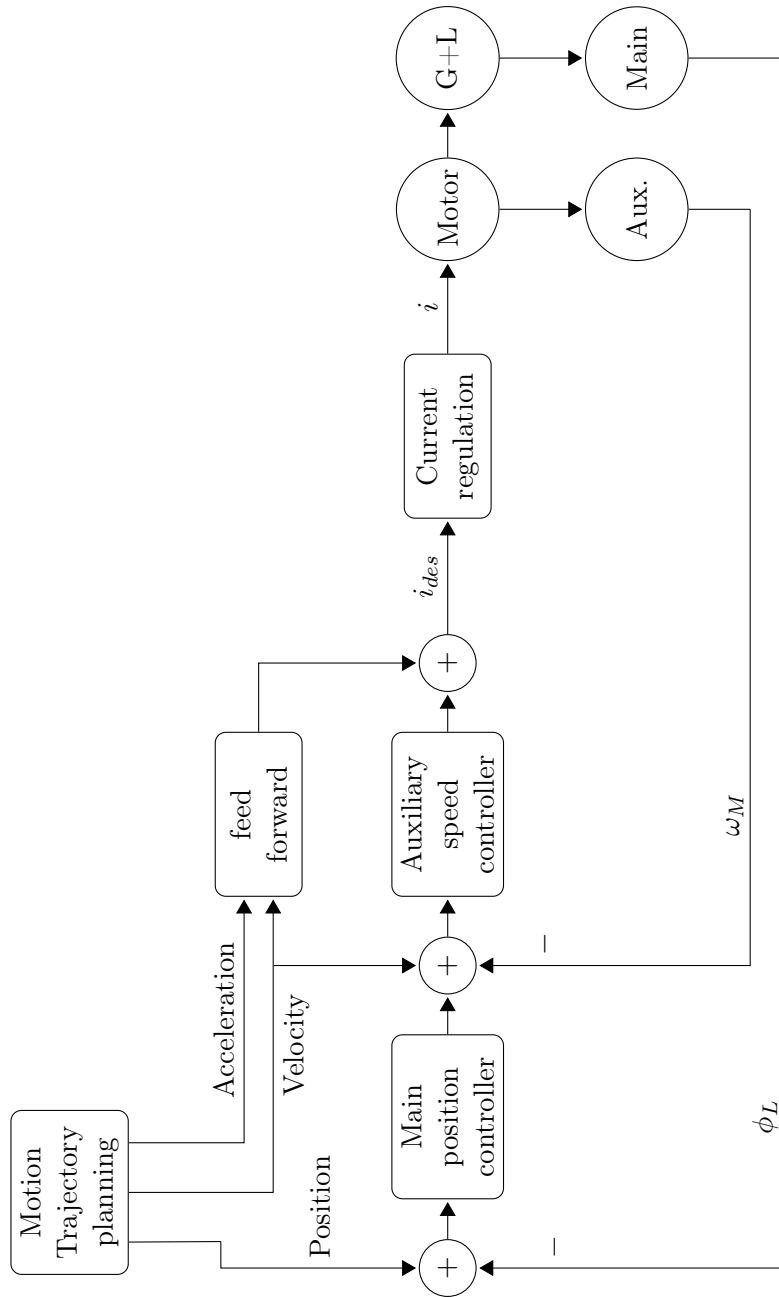


Figure 5.9: Position regulation. The block "G + L" indicates the set composed by Gear and Load. "Aux" and "Main" are the auxiliary encoder and the main encoder. ω_m and ϕ_L are the motor angular velocity and the load angular position respectively.

The auxiliary control is designed just to stabilize the loop, while it is the main controller that provides the correct position feedback. The dual loop (main controller and auxiliary controller together) is realized as a PID controller. The position PID controller is showed in figure 5.10. When we make the EPOS2 tuning we find all the gains for the position PID controller and for the velocity PI controller (figure 5.11). In our case the velocity PI controller is not used because we don't use any velocity mode of operation (neither the Profile Velocity Mode nor the Velocity Mode), it is used by the control architecture to make the auxiliary control loop. And Current Regulation has a proper PI control (figure 5.12).

The controls gains are: $K_{p,c}$ and $K_{i,c}$ for the Current; $K_{p,v}$ and $K_{i,v}$ for the Velocity; $K_{p,p}$, $K_{i,p}$ and $K_{d,p}$ for the Position; and K_a and K_ω for the angular acceleration and angular velocity feed forward. All these control gains are found by the EPOS2 tuning with EPOS Studio.

5.1.2 EPOS Studio

EPOS Studio is the software environment made by MAXON[®] in which we configure the motor and the CANopen communication and where we tune the EPOS2 controllers. It is necessary that all the wiring is connected and the motor must be connected with the EPOS2 before opening EPOS Studio. Granted that, one EPOS2 at a time will be connected to the PC via USB cable. It is important that the CANopen cables are disconnected from the EPOS2 we are regulating, otherwise an error raises into the EPOS Studio environment. Another important thing is that after we have done the motor configuration and the EPOS2 tuning, we don't connect the EPOS2s with the PLC (via CANopen cable) until we haven't exported the DCS file (see below).

Before proceeding, we have to open EPOS Studio, and click on "Connect All" icon on the status bar at the top of the EPOS Studio window (figure 5.14) to connect the EPOS2 and the motor to the PC and let EPOS Studio reads the entries of the Object Dictionary that describes both the EPOS2 and the Motor.

EPOS2 tuning

The EPOS2 tuning is quite easy to explain. We click on the icon "Regulation Tuning" in the Wizard (figure 5.15) and choose "Auto Tuning" after we follow the steps (note that the motor will move). The auto tuning will tune the Current Controller, the Position Controller and the Velocity Controller in automatic (figure 5.16). When the tuning has been done, we have to save the results into the EPOS2.

Finally, we set the CANopen communication. We click on the icon "CANopen Wizard" and we set only the Heartbeat times for the producer and for the consumer (only the consumer number 1). The RxPDO and TxPDO settings are used to map some variables and to the PDOs be asynchronous or synchronous, but we will make this with the BR Automation Studio.

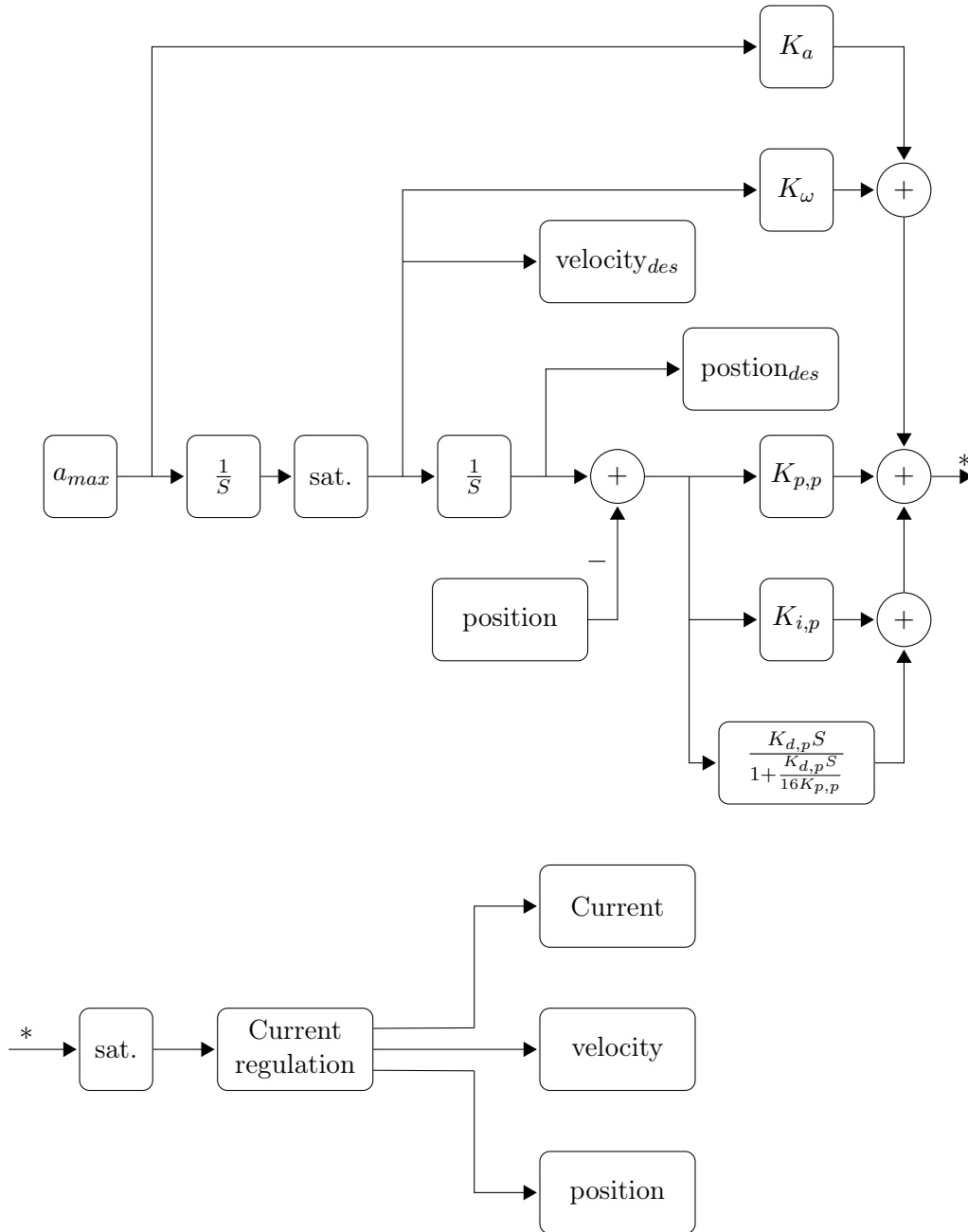


Figure 5.10: Block diagram for the position PID controller with feed forward. The block "Sat." stands for Saturation, this block controls that the velocity first and the current after don't exceed the upper or lower bounds.

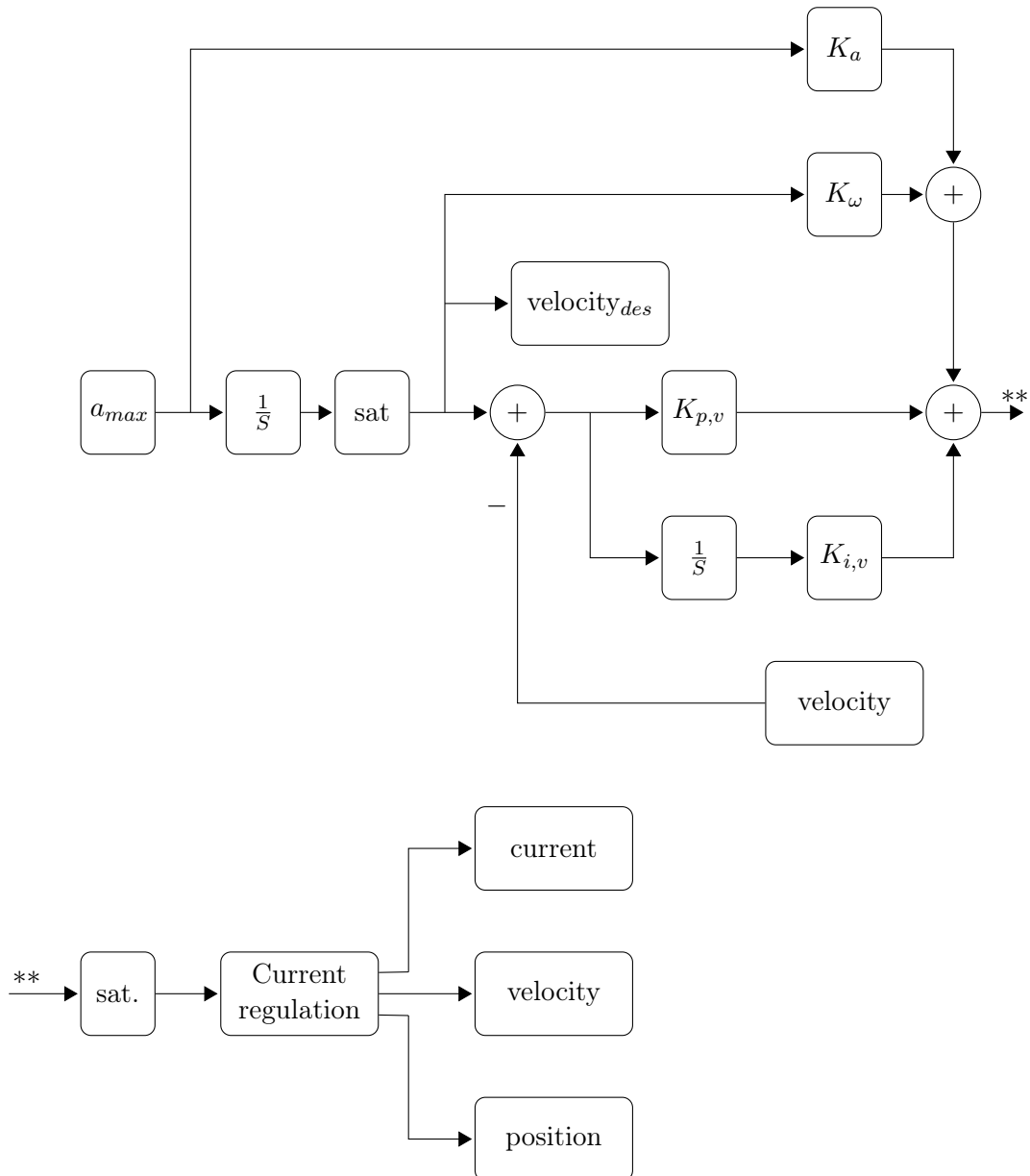


Figure 5.11: Block diagram for the Velocity PI controller with feed forward. The block "Sat." stands for Saturation, this block controls that the velocity first and the current after don't exceed the upper or lower bounds.

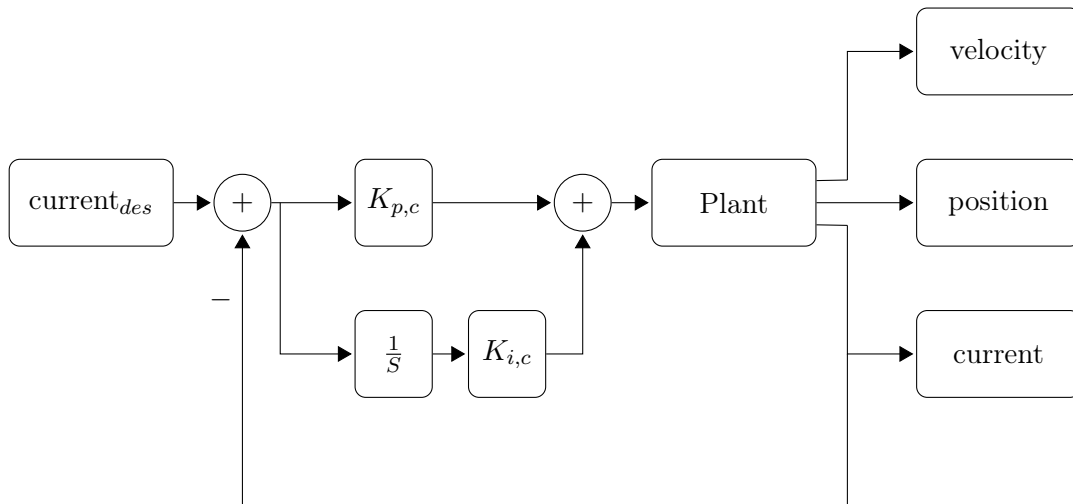


Figure 5.12: Current PI regulation

Another thing we could do with EPOS Studio is to monitor all the Object Dictionary while the EPOS2 is running. In fact, if we connect the EPOS2s together with the CANopen cable, and connect one of them to the PC via USB cable, we can watch the values of the object dictionary entries when the EPOS2 is running (is the same as using the "Watch" command in Automation Studio). Also in this case, the change of variables is not real time, but it is delayed. To do this we click on the icon "Object Dictionary" in the Tool menu (figure 5.17).

In the Tool menu we can make the EPOS2 perform some motions (in one of the operating modes defined by the CiA 401 protocol [8]), by clicking the respective icon. It is the same as we make with the programs in Automation Studio, but with EPOS Studio we just give the command and the EPOS2 will move without write any program. This is useful only if we want to verify the controller.

Motor configuration

The Motor configuration is very simple. We open the "Startup Wizard" in the "Wizard" menu (figure 5.15) and we start with the motor configuration. Motor Configuration is quite simple because we have just to insert the values of the motor characteristics that we find in the motor data sheet. The main joint features are listed in table 5.1. After we save the results into the EPOS2.

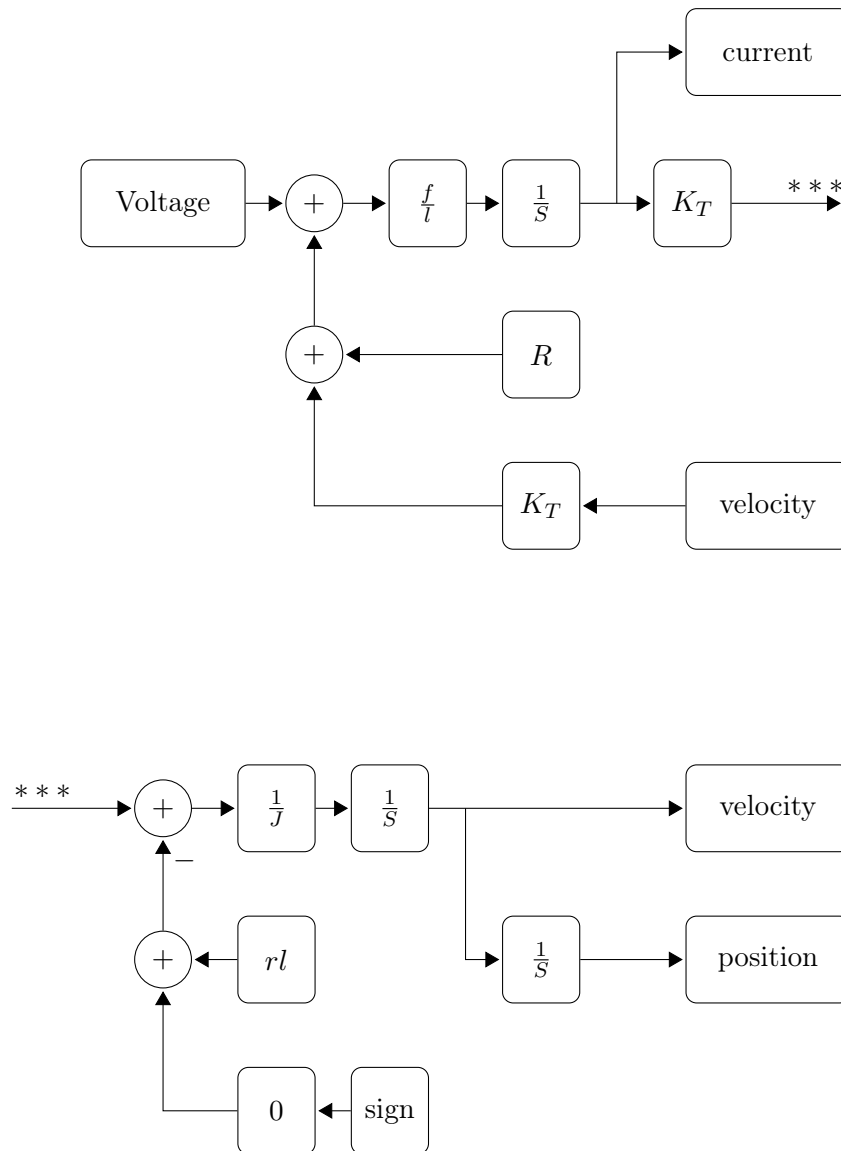


Figure 5.13: Model of the Plant showed in figure 5.12.



Figure 5.14: EPOS Studio status bar.

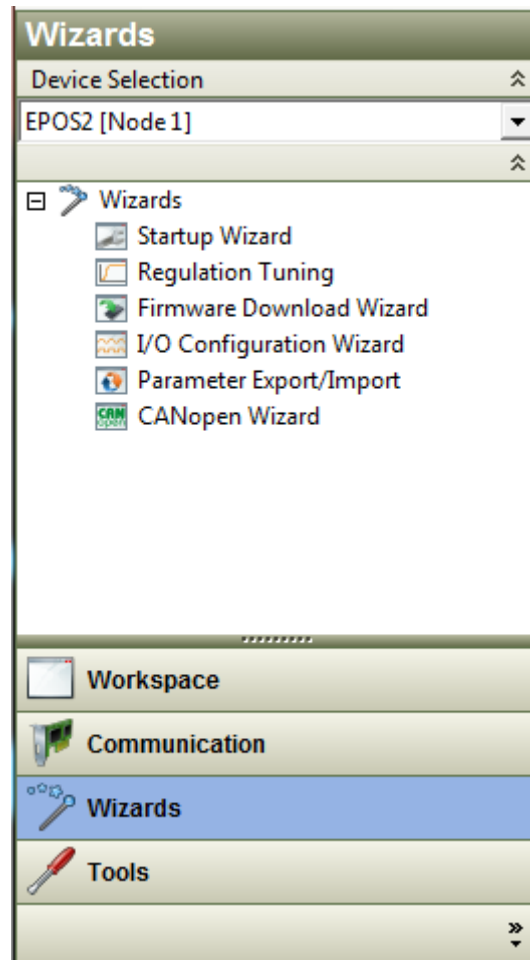


Figure 5.15: EPOS Studio Wizard menu.

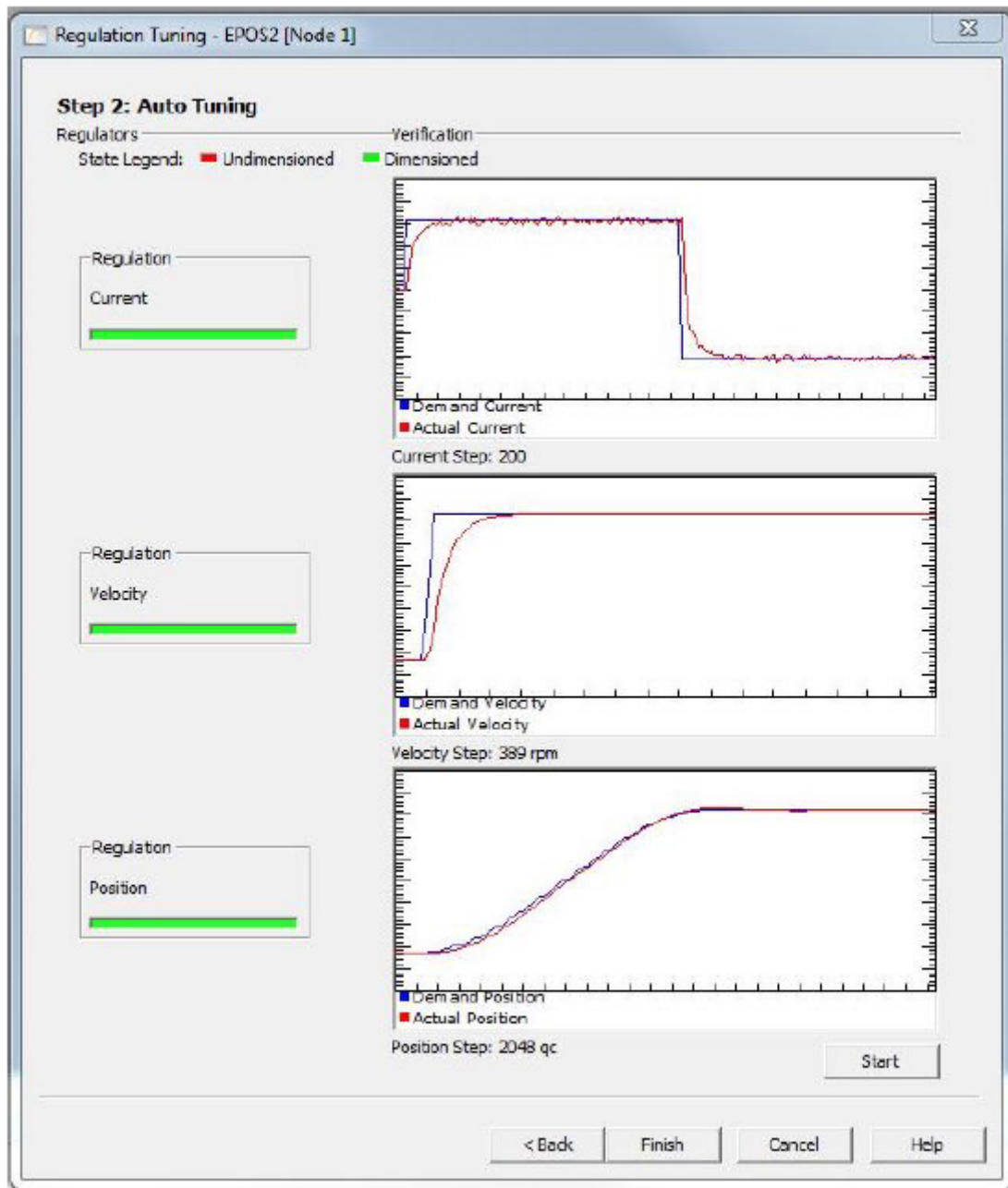


Figure 5.16: EPOS Studio Auto Tuning view.

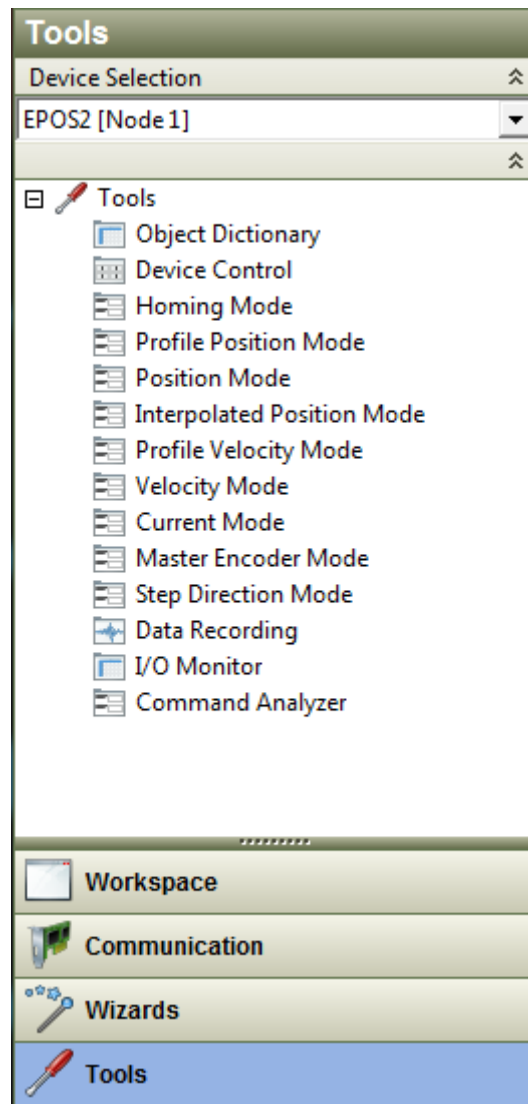


Figure 5.17: EPOS Studio Tools menu.

Table 5.1: Joint electric characteristics. All of them are discussed as entries of the Object Dictionary in section 4.4. Joint number 6 hasn't got the encoder.

	Joint number	1	2	3	4	5	6
MOTOR	Continuous current limit [A]	3.21	5.94	2.27	2.02	2.02	1
	Pole pairs number	8	1	12	8	8	4
	Max motor speed [rpm]	4860	2670	1610	2940	2940	2790
	Thermal time constant winding [s]	29.6	33.9	46	11.4	11.4	8.78
GEAR	Reduction factor	126 : 1	308 : 1	113 : 1	47 : 1	47 : 1	30 : 1
ENCODER	Count per second	2048	500	6400	2048	2048	—
	Channels	2	3	2	2	2	—

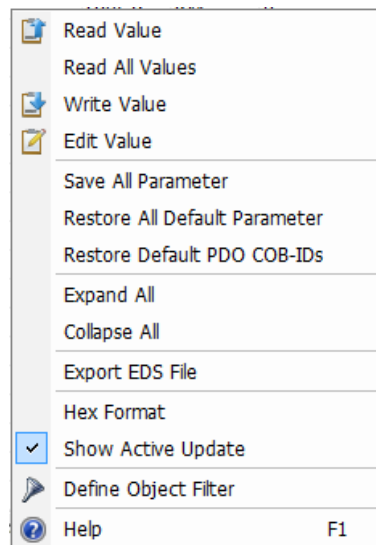


Figure 5.18: The window which opens when we right click on the window "Object Dictionary". Here we can save the EDS file.

5.1.3 EDS File and DCF file

When the EPOS2 is still connected via USB (only via USB, all the CANopen connections can't exist), we can export two type of files. The EDS file contains only the parameters which describe the device, but it doesn't contain the value of the parameters (it contains only the default value of them) [7]. This choice is due to the fact that EDS file is only a description of the device firmware. If the firmware will be updated, the EDS file has to be modified (by the manufacturer) and a new version of it will be released. To export the EDS file we open the Object Dictionary by clicking its icon in the Tools menu (figure 5.17), on the right side the window with all the entries will open, after we right click and select "Export EDS file" (figure 5.18) and we decide where the file will be saved.

The DCF file, instead, has the value of the parameters that we set with the EPOS2 Tuning and with the Motor Configuration. DCF file structure is very similar to that of the EDS file, the differences are only in the value of the parameters (there are values of the motor-depended objects described in section 4.4 as well). To export the DCF file we click on the icon "Parameters Export/Import" in the Wizard menu (figure 5.15), after a window opens (figure 5.19) and we have to choose the file extension (.dcf) and where the file will be saved. With the same procedure we can also import a DCF file and save it into an EPOS2, which takes the parameters values described in the DCF file we have just transferred.

When we import the "Fieldbus device" into Automation Studio, we import the EDS files of the EPOS2s. Hence, when we upload the program into the PLC, all the device parameters will turn into the default values. Not only in the PLC but also in the EPOS2s the parameters are going to be overwritten by the default values. So we have to modify the Object Dictionary entries with the correct values listed in DCF file. To make this we right click on the EPOS2 icon in the Physical View and click on "Configuration" and a window appears. We have to modify the value of all

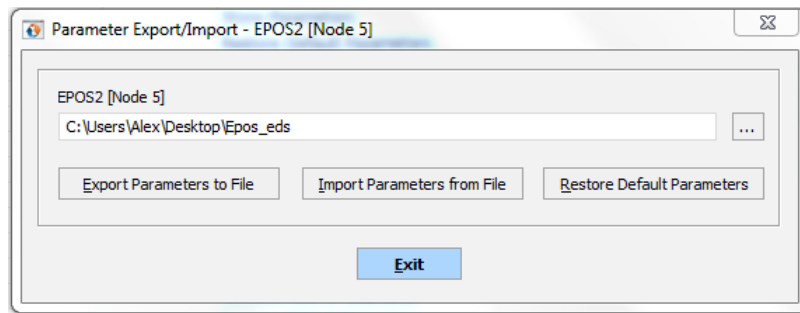


Figure 5.19: The window which opens when we click on "Parameters Export/Import" in the Wizard menu. Here we can save the DCF file.

the entries described in section Object Dictionary (section 4.4), and we have also to set the value of the gains we have found while performing the EPOS2 Tuning. The current PI gains are saved in the object Current Control Parameters Set ($60F6_h$) and its sub-indexes; the velocity PI gains and its feed forward gains are saved in the object Velocity Control Parameters Set ($60F9_h$) and its sub-indexes; and the position PID gains and its feed forward gains are saved in the object Position Control Parameters Set ($60FB_h$) and its sub-indexes. We set only these entries because they are the most significant, the others entries are for the experts.

Note that the EPOS2 channels (which are the PLC Input and Output), haven't a value (the default value is always 0000_h) because they assume different value, depending on what the EPOS2 is performing. For example the Controlword and Statusword have the value that depends on which state the EPOS2s is. The same goes for the "Target Position", the "Position Actual Value" etc.

5.1.4 LED in EPOS2 Controllers

EPOS2 has two LEDs, one green and one red in the board upper left corner, if seen from the front. The two LEDs tell us something about the EPOS2 status and errors, in table 5.2 the LED combinations and their explanation are shown [9] [10]. The states are the same as those explained in CiA 402 protocol (see section 6.2), in fact in the first row the possible states are those within Power Disabled area, the states in the second row are those in Power Enabled area, and the state in the third row are those in Fault area.

Flash stands for Flashing, which means that during 1 second, the LED is off for 0.9 seconds and on for 0.1 seconds.

Slow stands for Slow Blinking, which means that the LED blink with a frequency of $\sim 1Hz$.

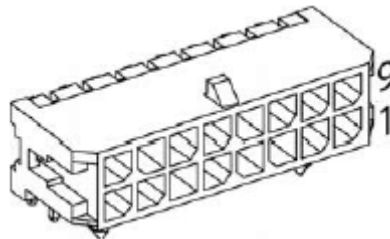
5.1.5 Digital Input on EPOS2

On the upper side of each EPOS2 there are the input and output module (called J5 signal, figure). We are only interested on the input module, because here we connect the limit switches for the Home position (see also subsection 6.3.1). The EPOS2 input and output connector is illustrated in figure 5.20(a), while in table 5.3 the

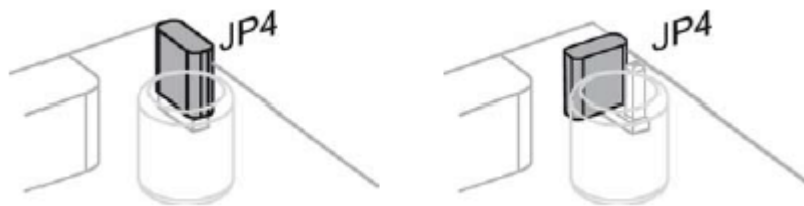
Table 5.2: Explanation of red LED and green LED combination.

Red LED	Green LED	States and errors
Off	Slow	Power stage is disabled. The possible EPOS2 states are: "Switch On Disabled"; "Ready to Switch On"; "Switched On".
Off	On	Power stage enabled. The possible EPOS2 states are: "Operation Enable"; "Quick Stop Active".
On	Off	EPOS2 is in "Fault" State.
On	On	Power stage enabled. The EPOS2 is in the temporary "Fault Reaction Active" state.
On	Flash	No valid firmware, or firmware in download.

connectors pinout is explained.



(a) Input and output module. Input are both digital and analog.



(b) Jumper JP4 closed (left) and open (right). Opening the housing we can find the Jumper JP4 in the bottom right corner. It is connect to the pin 9 in the input and output module.

The Analog Inputs could be used as reference voltage for the analog position set point when we are using Position Mode (see [12]).

We use only the pins 3, 4 and 9 because they are necessary for the Homing Mode. In fact, here we connect the limit switches for the Homing position. The second and third EPOS2s, whom motors require the Homing method number 1, have the limit switches signal on the pin 3. While the first EPOS2, whom motor requires the Homing method number 2, has the limit switches signal on the pin 4. Note that the existing switches have both the normally Open and the normally Closed connection,

Pin	Description	Signal
1 and 2	Digital signal ground	D_Gnd
3	Digital input 6 "Negative Limit Switch"	DigIN6
4	Digital input 5 "Positive Limit Switch"	DigIN5
5	Digital input 4 "Home Limit Switch"	DigIN4
6 ÷ 8	Digital input 3, 2, 1 "General purpose"	DigIN3, 2, 1
9	Auxiliary supply voltage output	+V _{out}
	Logical supply voltage input	+V _c
10	Digital output 4 "Brake"	DigOUT4
11 ÷ 13	Digital output 3, 2, 1 "General purpose"	DigOUT3, 2, 1
14	Analog signal ground	A_Gnd
15 ÷ 16	Analog input 2, 1	AnIn2, 1

Table 5.3: The explanation of the J5 Signal connector pins. Pin 9 is a voltage output if jumper JP4 is closed, or a supply voltage can be connect if jumper JP4 is open.

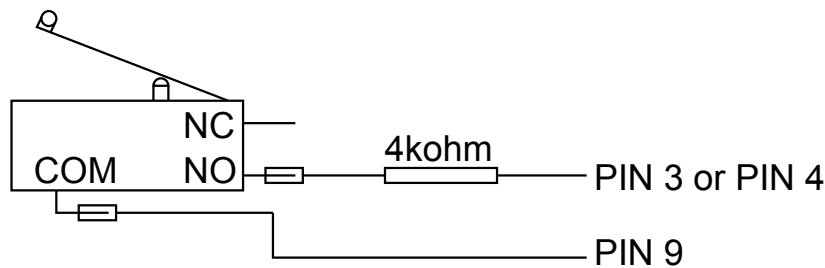


Figure 5.20: Wiring example for the connection of the Homing Limit Switch to the EPOS2. Every EPOS2 has a limit switch. The limit switch Normally Open connector is connected to the EPOS2 J5 Signal pin number 3 or 4 based on the Homing method.

so we have to connect to the EPOS2 input pin only the normally Open connection. The pin 9 is a Voltage output (hence the jumper JP9 is closed) and it is connect to the limit switch "COM" connection. In order to reduce the input current, a resistor is present between the limit switch "Normally Open (NO)" connector and the EPOS2 input pin. The maximal current is 6 mA at 24 VDC, so the resistor has to be a value of 4 k Ω (figure 5.20).

5.2 B&R Automation[®]PLC

The Programmable Logic Controller (PLC) is a digital computer which which is adopted for the control of industrial process, such as assembly lines or robotic device. Our PLC is made by B&R Automation[®].

Explaining how PLC works is quite simple. It is made in the same way as a computer, but without monitor, mouse and keyboard. PLCs have got a processor, a timer and a power supply. Our PLC has also an input module and the module for the CANopen interface, which it is necessary for making the PLC play the role of the Producer in CANopen network and the role of the Master in CANopen network management (see section 4.3).

Every module can be placed on the right side of the power supply module. Virtually we can place an infinite number of modules, but really, the number of them is limited by the maximum power that PLC delivers, in fact every module needs some amount of power, so the sum of all the modules power gives the maximum number of modules that we can add. In our case, only one input module was added. The hardware architecture is shown in figure 5.21, from left to right there are:

- PLC (X20 CP 1584): this is the real place where programs run, here are located the CPU, the timer, cooler and all the other necessary hardware that makes a computer works. Here there are also the Flash memory in which the program is saved and the Ethernet port for the communication between computer and PLC. Ethernet communication has two main goals: the first is to write program and configuration on the flash memory, the second goal is to read (or write) the variables while the PLC is working (see below).
- CAN interface (X20 IF 1072): this is used for the communication based on CANopen network. With this module the PLC becomes the producer in CANopen network and the master in the CANopen network management.
- Power supply module: this is simply the module which feeds the PLC with the 24V from the power line.
- Input module (X20 DIF 371): this is the module that provides digital input to PLC. Currently, only the emergency red button is connected in the Input module.
- EPOS2: the other devices are the motor controller EPOS2[®]. They are connected to each other with a single cable.

5.2.1 Digital Input module

The B&R Automation[®]PLC is able to manage some external digital inputs by adding the proper module. We use the X20 DIF 371 as a digital input module (figure 5.22).

We use the input module just to connect the Emergency push button (figure 5.23(a)) and three limit switches for the link 1, 2 and 3 in order to limit their moves. In fact, due to the robotic arm structure links 2 and 3 are not able to make a 360 deg movement, and due to the laboratory environment also link 1 is not able to make a 360 deg movement without catastrophic consequences. We place the limit switches directly on the base for link 1, on link 1 for link 2 and on link 2 for link 3, in this way, if a link hit its limit switch all the EPOS2s receive the command to stop the motion (see section 6.5.1 React to the dangers). Also in this case we use the limit switches Normally Open connector as the signal (figure 5.23(b)). A resistor is required, but only for prevention because the digital input and the internal electronics are opto-isolated.

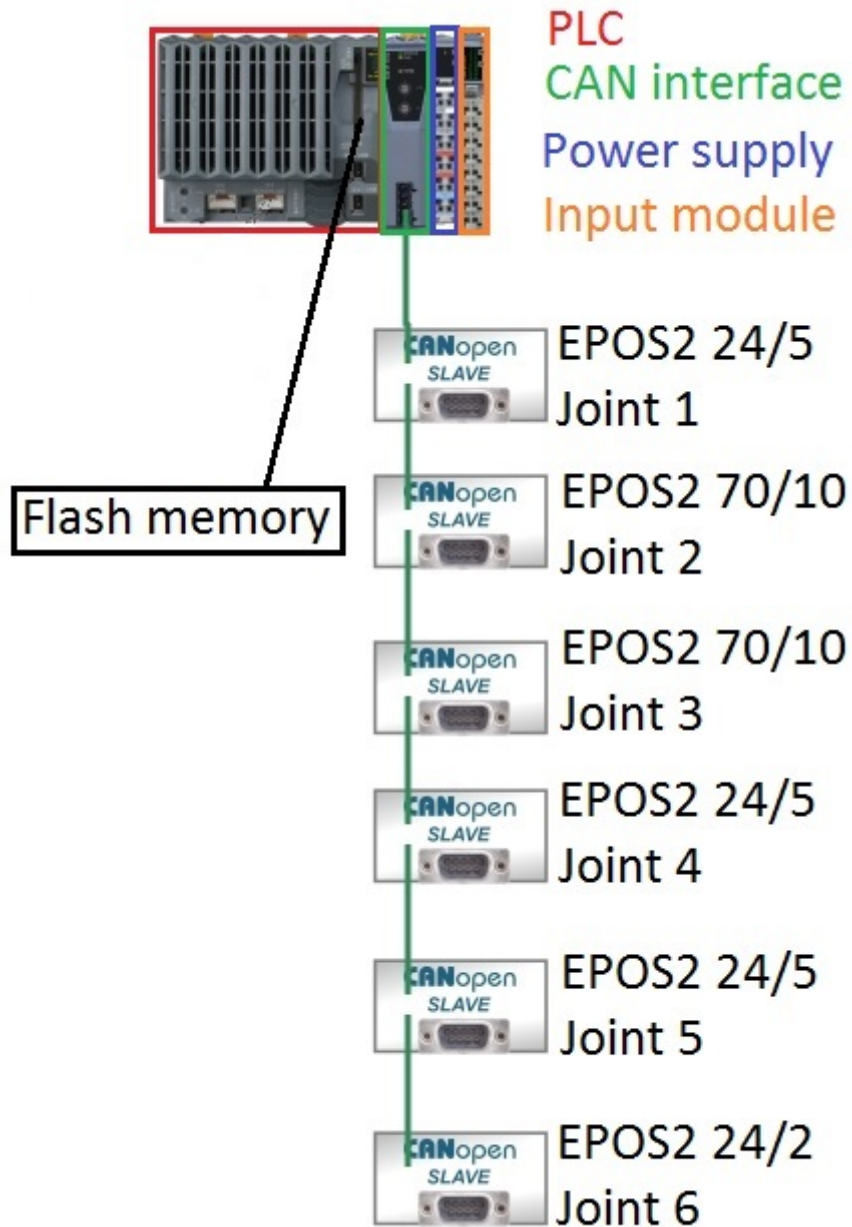


Figure 5.21: PLC hardware architecture. From left to right: PLC with the CPU and the timer; CAN interface; power supply module; input module. Below, there are the six EPOS2s

Connection example

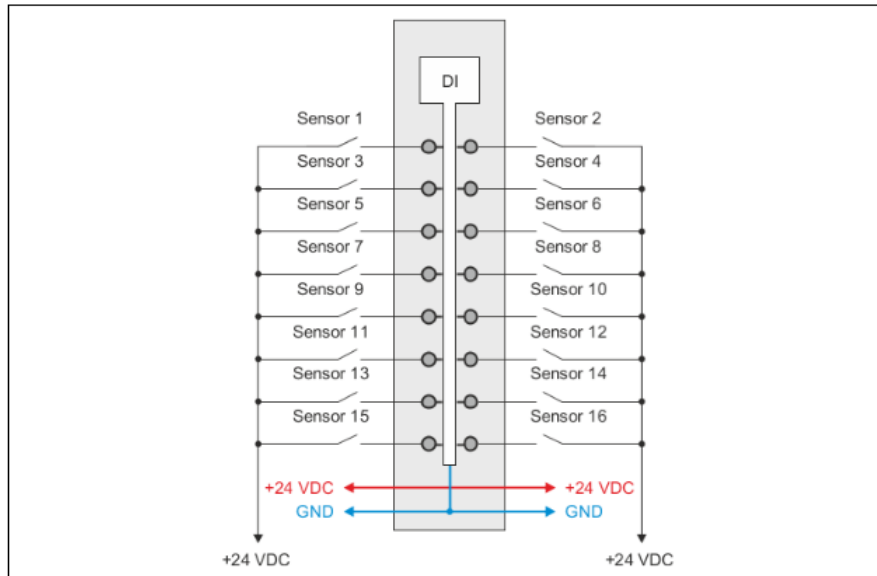


Figure 5.22: X20 DIF 371 digital input module overview.

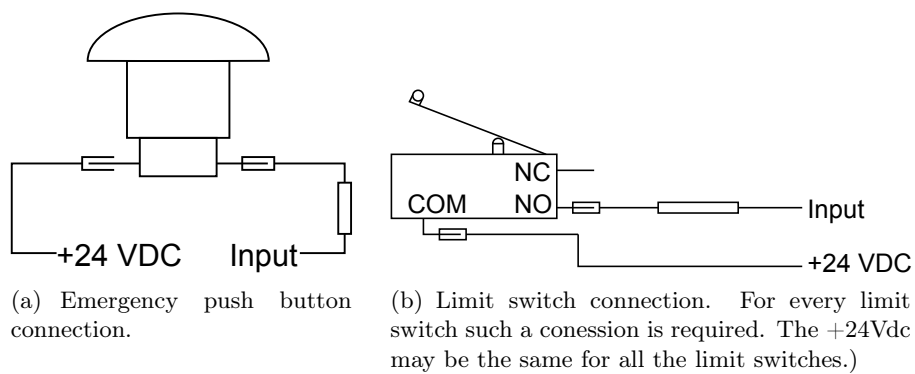


Figure 5.23: Example of the Emergency push button and limit switch connections on the X20 DIF 371 digital input module.

5.2.2 Hardware configuration with Automation Studio

Automation Studio is a software environment where we can build and configure the control hardware (and software) architecture [3].

The hardware part is stored in the Physical View. Every BR device is listed in the physical tool box (figure 5.24), so here we can find the PLC, the I/O module and the CAN interface, but not the EPOS2 drives, because these are made by MAXON[®]. The EPOS2 drive (like all the others devices which are not provided by B&R Automation[®]) must be called by the command "Import Fieldbus Device" (figure 5.25), which allows to import in AS the electronic data sheet of the device (file .eds). Now the EPOS2 EDS file are listed under the third party device in the tool box (figure 5.26). After we import the EPOS2s in the Physical View (by dragging them with the mouse from the third party device menu to the Physical View) and we connect them to the CAN interface and with each other in the same order than the real case (figure 5.27). We must also set the node number, this must be equal to the node number set up by the DIP switch on the EPOS2 controller. To set the node number we right click on the EPOS2 icon and we select "Change node number", on the right column two arrows appear for setting the node number.

Finally we have to set the value of some device parameters. When we import the EDS file, we import just the description of all the device parameters, but not their correct values (the values we import are the default ones), so now with the aid of the DCF file we set some fundamentals parameters. We right click on the EPOS2 icons and chose "Configuration", here we find all the editable objects of the OD under the nodes "Channels" and "Devices parameters". Descriptions and values of parameters that we have to modify are in section 4.4. The motor, gear and encoder main features are shown in table 5.1.

From the Online menu Configuration (figure 5.28) we can connect the PLC with Ethernet port and download the programs to the PLC directly with Ethernet port. If the Ethernet port hasn't been yet configured, we have to download the program on the flash memory by selecting "Offline Install" in the Tools menu (figure 5.25). To configure the Ethernet communication we have to set the IP Address of the PLC. To do this we click with the mouse right button on the icon "ETH IF2" in the Physical View (5.27); we then choose "Enter IP Address manually" in the voice "IP Address", and here we enter the IP Address of the PLC. Then in the window that opens when we click on "Online settings" we set the INA node number of the PC, while the INA node number for the PLC is set by rotating the two rotational switches at the bottom of the PLC (see figure 5.29). It isn't important what the numbers are, but it is mandatory that they are different, otherwise the communication via Ethernet does not take place because in the plant there are two devices with the same INA node number. Finally we click on the icon for searching the Online PLC and we start the communication by right clicking on the PLC icon and choosing "Communication".

In figure 5.29 we can see also the Ethernet port and the PLC battery which is used only for the EEPROM memory.

Now we have to set some feature of the CANopen network. First we set the

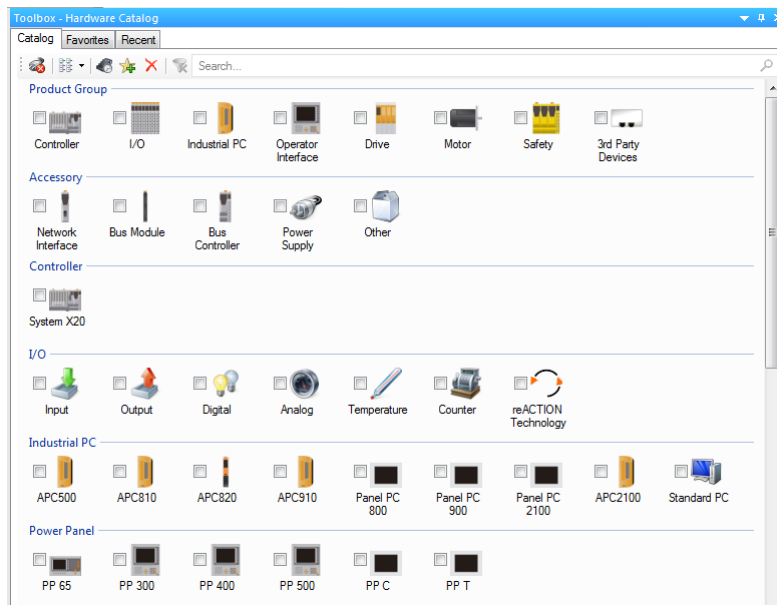


Figure 5.24: The tool box for the devices. Here are listed all the B&R Automation[®] devices. To open this window we have to click on the Physical View and on the right the toolbox appears. Note that when we choose a device, in the tool box will be listed only devices that can be connected with chosen device.

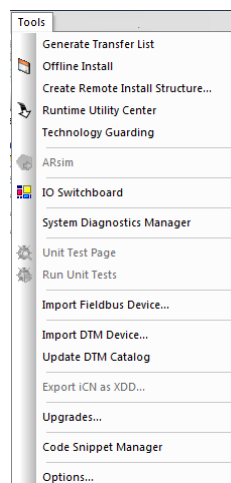


Figure 5.25: Tools menu. Here we import the Fieldbus device and we can install the configuration on the flash memory in case of the Ethernet port hasn't been yet configured.

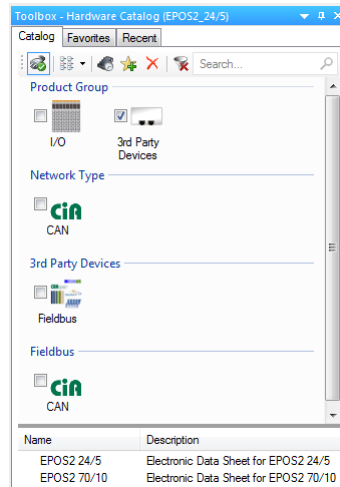


Figure 5.26: Third party device under the physical toolbox. Note that here are listed only the two type of EPOS2[®] (the 24/5 and the 70/10), there aren't all the six EPOS2s, this because of the .eds file is the same for the same EPOS2. To create the six motor hardware we will choose three EPOS2 24/5, two EPOS2 70/10 and one EPOS2 24/2.

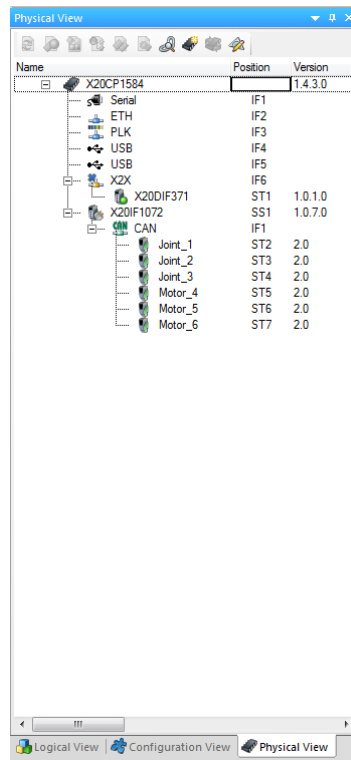


Figure 5.27: The Physical View. Here we can see how the hardware is organized and the connections between all the devices.

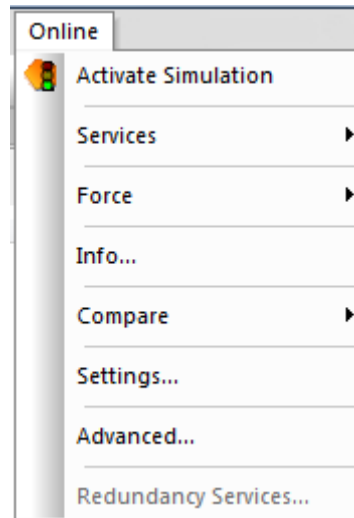


Figure 5.28: The Online menu. Here we can activate the communication between the PC and the PLC via Ethernet



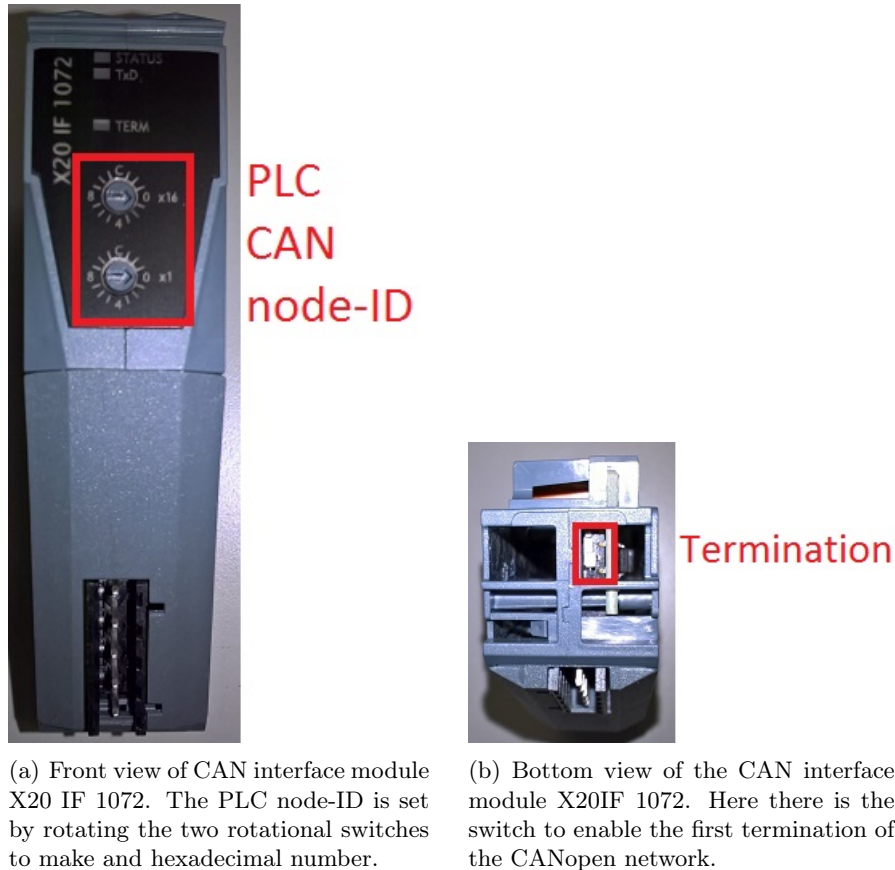
PLC INA node number

Ethernet cable port

PLC battery

Figure 5.29: With these two rotational switches we set the INA node number of the PLC. They are two because they make an hexadecimal number.

PLC node-ID by rotating the rotational switches on the CAN interface X20 IF 1072 (figure 5.30(a)). CANopen network must be terminated both at the initial and at the end. In order to terminate the network initial part, we set to ON the switch which is in bottom of the CAN interface module X20 IF 1072, (figure 5.30(b)).



(a) Front view of CAN interface module X20 IF 1072. The PLC node-ID is set by rotating the two rotational switches to make and hexadecimal number.

(b) Bottom view of the CAN interface module X20IF 1072. Here there is the switch to enable the first termination of the CANopen network.

Figure 5.30: Front view and bottom view of the CAN interface X20 IF 1072.

Now the control hardware architecture settings are finalized. In the next chapter, we will address the software architecture analysis.

Chapter 6

Software configuration

6.1 Automation Studio environment for software configuration

B&R Automation[®] is able to accept programs in several programming languages. Some of which are IEC (International Electrotechnical Commission) languages (such as Structured Text and Ladder Diagram), but B&R Automation[®] accepts also other languages such as C and C++. I chose C++. B&R Automation[®] offers a programming environment called Automation Studio (AS). In AS we built all the project part, from the hardware connection, to the software writing [3].

6.1.1 Software configuration

To create a program we open the "Logical View" (figure 6.2) and the toolbox for the programs appears on the right (figure 6.3). In this toolbox we can choose between some icons. Icon "Library" allows us to import library or create a new library, for now we just import existing libraries that we can find under the node "Import BR Library". Libraries we import are those that allow us to write the program in C++ using some parts of program in Structured Text. Icon "Program" is used to create the folder with the program and all the other files that are connected with it. Under the window the list with all the possible programs appears and we choose "C++ program". Now AS provides to create three sub-programs:

1. *Init* program. It will be executed only one time at the beginning of the cyclic where the program is located. This is useful when we need to give the initial values at some variables, for example the index for the arrays scanning should start from 0 (zero), but if we are modifying the program we could forget to set it to zero, so we write the line that sets it to zero in the Init program and we are sure that every time the program starts for the first time¹, the index starts to zero.
2. *Exit* program. It will be executed only one time when we decide to shut down

¹The program starts for the first time in when we switch on the PLC and the program has been already downloaded to the PLC, or every time after we download the entire project to the PLC.

the PLC. It is useful, in our specific case, to drive the robotic arm to the home position before shutting down the PLC.

3. *Cyclic* program. It is the real program with all the instruction and tasks. It will be executing at every lapse of time corresponding to the Task Class time in which the program is saved.

We have two choices, have all the three sub-programs in a single C++ file, or have three different C++ files, one for the single sub-program. I chose the second.

Due to the fact that the first programming languages of the PLC are the languages defined by the International Electrotechnical Commission (IEC), which are Structured Text and Ladder Diagram, all variables (both the global variables and the local variables) are easier to declare with Structured Text. In fact, Automation Studio provides a simple table² in which we can declare the variables. The using of this table is very intuitive. We could also not declare the variables with Structure Text, but we have to declare the variables in a C header file, after this file will be include in C++ programs. So it is easier to declare the variables and types with the table provided by the Automation Studio (figure 6.1(a) and figure 6.1(b)). Referring to the figures, "Reference" means that the variable is referred to its memory address (this is not possible with the Process Variables); "Constant" means that this is a constant whom value is specified in column "Value"; "Retain" means that the variable will be save into a memory with a battery (EEPROM memory) and not only in the RAM, so after a build procedure the variable and its value don't disappear; "Value" allows us to give the value for a constant or the initial value for a variable.

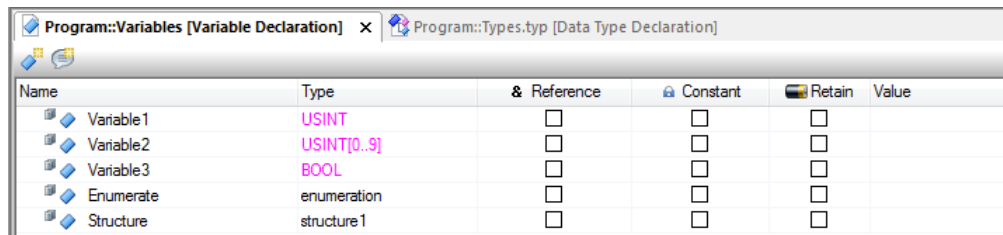
Also the declaration of the types (such as Enumerations and Structure) is easier with the Automation Studio method, but the used language is still the Structure Text. The only thing we have to do for telling to the PLC that the program is written in C++, but the variables and the types are declaring in Structured Text is to include in all the programs the header files "bur/plc.h" and "bur/plctypes.h" and the library "cstdlib"³. For the rest the programming can be in other languages different from those defined by IEC.

The difference between Global variables and Local variables is that the first ones can be read and written by all the programs, while the second ones can be read and written only by the program in which they are declared. Variables connected to the PLC channels are also called Process Variables (PV), while the other variables declared in .var file are called Variables. We are also able to declare some support variables inside the Cyclic program, but we cannot know them values during the PLC is running because they cannot be added in the Watch window (see below).

Icon "File" allows us to create file, for example, when we need to create an header file (.h), we choose File and under the window a list with all the possible files appears, we choose "Header file" that will be added in the selected program folder.

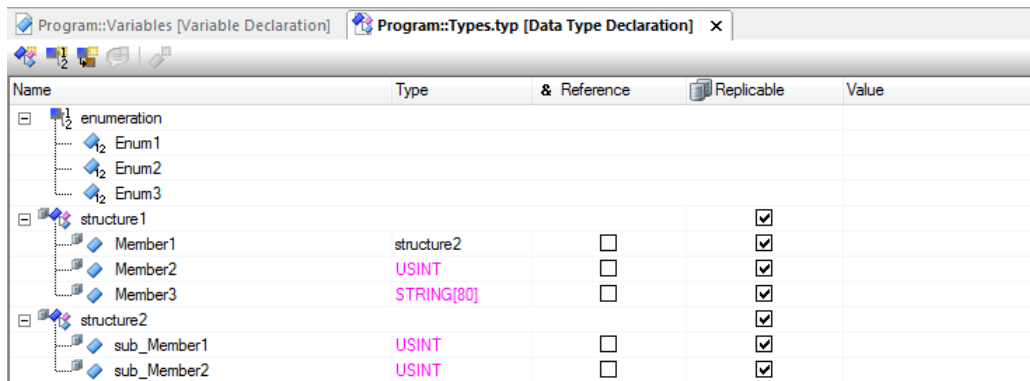
²This table is just a way to simplify the declaration of the variables, because the variables file (.var) is written in Structured Text. The same happens with the declaration of the Types in the .typ file.

³These header file and library allow also to use some data type which are not present in C++, for example the data type int16 in C++, becomes UINT in the PLC and so on.



Name	Type	& Reference	Constant	Retain	Value
Variable1	USINT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Variable2	USINT[0..9]	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Variable3	BOOL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Enumerate	enumeration	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Structure	structure1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

(a) Table for Variables declaration.



Name	Type	& Reference	Replicable	Value
enumeration				
Enum1				
Enum2				
Enum3				
structure1			<input checked="" type="checkbox"/>	
Member1	structure2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Member2	USINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Member3	STRING(80)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
structure2			<input checked="" type="checkbox"/>	
sub_Member1	USINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
sub_Member2	USINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

(b) Table for Types declaration.

Figure 6.1: Tables for Variables and Type declaration. Some Variables examples are shown, among which the Enumeration and the Data Structures are shown. Note that if a variable is defined as a type, in the column "Type" there is the name of the declared type.

Software architecture was a crucial point, because many tasks have been implemented, first of all security and smoothness of motion. In order to satisfy the tasks, B&R Automation[®] AS offers a simply way to architect software, in fact AS is based on *Task Classes*. Task Classes are virtual places where we can save one or more (at most four per cyclic) program, these programs are executed at every time lapse. The lapse of time varies based on the number of Task Class. There are eight Task Classes in hierarchical order, the number of the cyclic indicate the importance and the length⁴ of the Task Class. For example, the number one is the most important and it executes the program in it every 10ms, no other Task Class can be executed while the CPU is running the Task Class number one, and every other Task Class is stopped to run the number one (when the execution of the Task Class overlap). Instead Task Class number four is repeated every 100ms, it could be stopped by the Task Classes number three, two and one, but the number four can stop Task Classes from number five to eight.

The Task Classes based method is very useful to structure the software and respect the different tasks that the CPU performs. In fact, for example, the program that manages the emergency will be assigned to the Task Class number one, while the program which manages the robotic arm motion is assigned to the fourth Task

⁴The lengths have some default values, but they can be modify by the programmer. If the program saved in the cyclic requires more time than the length of the cyclic plus the tolerance, an error occurs and the PLC turns itself into the Service mode.

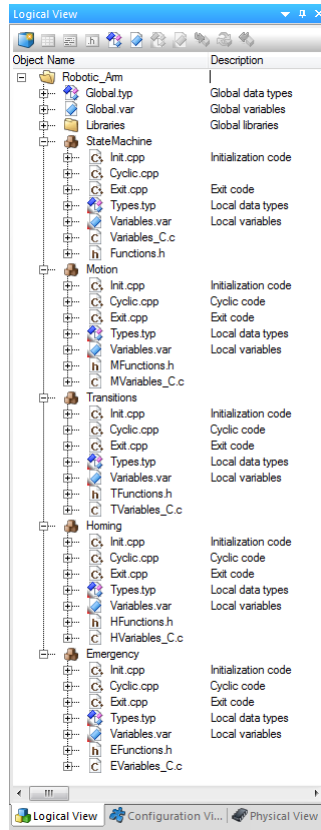


Figure 6.2: The Logical View. Here all the programs are listed. Every program in a cyclic is composed by some files, all these files are included in a folder named like the program. We can also note the variables and types declarations, which are in .var and .typ file respectively.

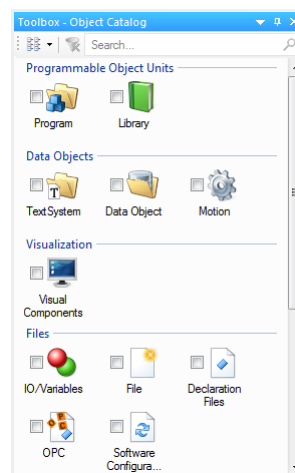


Figure 6.3: The toolbox for Logical part. The mains icon are "Program", "Library" and "File".

Object Name	Version	Transfer To	Size (bytes)	Source	Source File
Cyclic #1 - [10 ms]					
Emergency	1.00.0	UserROM	0	Emergency	Config1\X20CP1584\Cpu.sw
Cyclic #2 - [20 ms]					
Transition	1.00.0	UserROM	0	Transitions	Config1\X20CP1584\Cpu.sw
Cyclic #3 - [50 ms]					
Cyclic #4 - [100 ms]					
StateMachi	1.00.0	UserROM	386108	StateMachine	Config1\X20CP1584\Cpu.sw
Motion	1.00.0	UserROM	386256	Motion	Config1\X20CP1584\Cpu.sw
Homing	1.00.0	UserROM	0	Homing	Config1\X20CP1584\Cpu.sw
Cyclic #5 - [200 ms]					
Cyclic #6 - [500 ms]					
Cyclic #7 - [1000 ms]					
Cyclic #8 - [10 ms]					
Data Objects					
No Data Objects					
Visualization					
Binary Objects					
TCData	1.00.0	SystemROM	1259192		Config1\X20CP1584\Cpu.sw
Library Objects					
AsCANopen	4.25.0	UserROM	17592	Libraries.AsCANopen	Config1\X20CP1584\Cpu.sw
CAN_Lib	4.25.0	UserROM	46080	Libraries.CAN_Lib	Config1\X20CP1584\Cpu.sw
canio		UserROM	0	Libraries.canio	Config1\X20CP1584\Cpu.sw
brsystem	4.25.0	UserROM	15496	Libraries.brssystem	Config1\X20CP1584\Cpu.sw
sys_lib	4.25.0	UserROM	11448	Libraries.sys_lib	Config1\X20CP1584\Cpu.sw
AsArLog	4.25.0	UserROM	5764	Libraries.AsArLog	Config1\X20CP1584\Cpu.sw
AsTCP	4.25.0	UserROM	14744	Libraries.AsTCP	Config1\X20CP1584\Cpu.sw
runtime	4.25.0	UserROM	33728	Libraries.runtime	Config1\X20CP1584\Cpu.sw
ArEventLog	4.25.0	UserROM	11732	Libraries.ArEventLog	Config1\X20CP1584\Cpu.sw
Source Objects					
reACTION Technology Objects					
Configuration Objects					
sysconf	4.25.0	SystemROM	67648		Config1\X20CP1584\Cpu.sw
Role	1.00.0	UserROM	760		Config1\X20CP1584\Cpu.sw
User	1.00.0	UserROM	656		Config1\X20CP1584\Cpu.sw
iomap	1.00.0	UserROM	12852		Config1\X20CP1584\Cpu.sw
ashwd	1.00.0	SystemROM	2760		Config1\X20CP1584\Cpu.sw
asfw	1.00.0	SystemROM	300760		Config1\X20CP1584\Cpu.sw
arcconfig	1.00.0	SystemROM	206208		Config1\X20CP1584\Cpu.sw

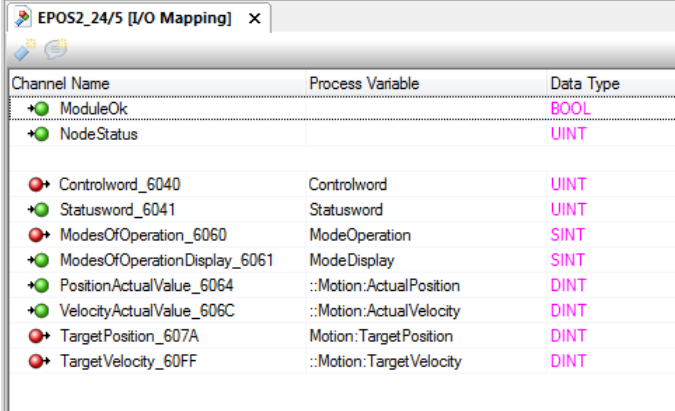
Figure 6.4: Everything that is assigned to the CPU. Here we can see the software architecture: the programs are saved in different Task Class (here called Cyclic) because of their importance in the architecture.

Class. In figure 6.4 is shown the software architecture and the assignment of programs to the Task Classes, and all that is saved into the memory and it is necessary to the CPU to run the programs (such as the libraries).

6.1.2 Assignment of the variables

Automation Studio doesn't provide the using of the PLC inputs and outputs in the programs. For this reason we have to assign to PLC inputs and outputs the variables which have been already declared. To do this, we right click on EPOS2 icon and select "I/O Mapping", after the window opens (figure 6.5). On left column (Channel Name) are listed all the PLC inputs and outputs. On next column (Process Variable) we select the variable to connect to the channel, when we click on this column a window opens with only the variable that we can connect to the channel. In fact, we are able to connect only variables declared with the same data type shown on the third column. This must to be done for all the EPOS2 devices, every process variable can be connected with only one Channel.

As we have already said, the subject is the PLC, so every input and output is referred to it. This means that the EPOS2 inputs (such as the Controlword and



Channel Name	Process Variable	Data Type
ModuleOk		BOOL
NodeStatus		UINT
Controlword_6040	Controlword	UINT
Statusword_6041	Statusword	UINT
ModesOfOperation_6060	ModeOperation	SINT
ModesOfOperationDisplay_6061	ModeDisplay	SINT
PositionActualValue_6064	::Motion:ActualPosition	DINT
VelocityActualValue_606C	::Motion:ActualVelocity	DINT
TargetPosition_607A	Motion:TargetPosition	DINT
TargetVelocity_60FF	::Motion:TargetVelocity	DINT

Figure 6.5: IO Mapping. In the first column on the left there are the name of the PLC channels. In the second column there are the Process Variables connected with the channels. In the third column there are the data type of the channels.

the Target Position) become the PLC outputs, and vice versa, the EPOS2 outputs (such as the Statusword and the Actual Position) become the PLC inputs. We can write the EPOS2 inputs (or PLC output), while the EPOS2 outputs are used only in read mode to verify the status of the EPOS2. Graphically, in Automation Studio PLC inputs are indicated with a green dot, while PLC outputs are indicated with a red dot, this convention remains in all the windows where the Process Variables are listed.

In figure 6.5 there are the standard EPOS2 channels, but we can add input and outputs. To do this we right click on EPOS2 icon in Physical View and select Configuration. After in Channel list, or in Device Parameters list we find the channel we want to manage. We open the description of this parameter (for example the Current) and under the node "PDO mapping" we select in which of the four PDO we want to map it. Finally this channel appears into the I/O Mapping window and we connect it to a Process Variable. Obviously it becomes a PLC input if it is of type Read Only (RO), or becomes a PLC output if it is of type Read and Write (RW) or Write Only (WO). The action just described is called "PDO Mapping" and it is allowed only for some channels (most of them), some other channel can't be mapped because it doesn't make sense. To see if a channel can be PDO mapped we have to see its description in the EDS file (or in the EPOS2 Firmware Specification manual) at the node "PDO Mapping" there is "Yes" if the channel can be PDO mapped or "No" if it isn't. Some Channels can be PDO mapped, but other can't, because of they describes the EPOS2 settings or the motor features, and after they are changed the target (the EPOS2) have to be reboot.

When we have done all the settings, we are able to transfer the project to the PLC. If the Ethernet port has been configured we click on the icon "Transfer" in the status bar (figure 6.6).

Other actions from the Status bar are:

- Built: builds the project keeping all the variables saved. The project could be

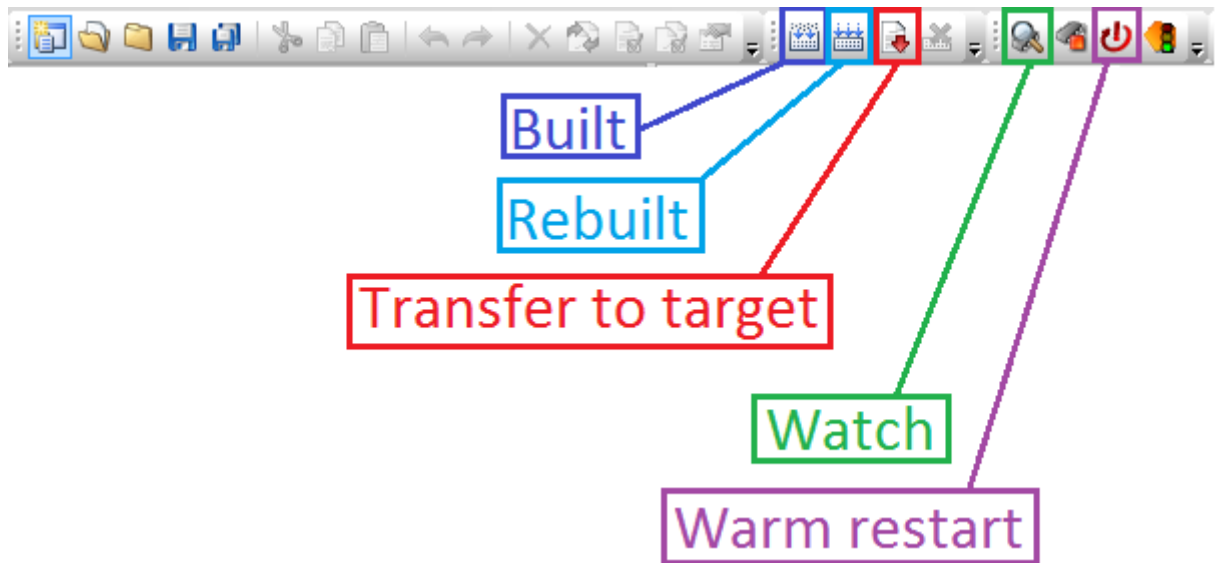


Figure 6.6: The Status bar. Here there is the Transfer icon to upload the project to PLC. There are also some other important icons: Built, Rebuilt, Watch and Warm restart.

or not be transfer after is ha been built.

- Rebuilt: builds the project after eliminating all the variable, the project will be built like the first time. The project could be or not be transfer after it has been rebuilt.
- Watch: while the PLC is running we can observe the value of the variables with this option.
- Warm Restart: when PLC turns from Service mode into RUN mode, the command Warm Restart is required before the execution of the programs. The variables are saved into the EEPROM memory, after the project will be built, and transfer to the PLC.

By clicking on Watch icon we can observe the value the Process Variables assume while the PLC is running in the same window of the I/O Mapping (figure 6.7). Here we can see the value of both the Channels (in "Physical Value") and the Process Variables in ("Process Variables Value"), we are able to force the Channel to assume the value of the PV that we manually modify in the column "Force Activated Vale" by flagging the "Force Activate" of that Channel.

In figure (6.8) is shown a watch window for a specific program. To open this window we right click on the program folder in Logical View and select "Watch". The difference respect the below watch is that here we are able to observe all the process Variables declared in this program and not only the PVs connected to channels. For adding Variables we right click on the window, select "Add Variables" and choose the PVs we want to watch. Also here we can modify manually the values of the Variables. Note that, only the Variables declared in the .var files (Global and Local) can be watched, the other variables defined within the Cyclic program cannot be

Channel Name	Physical Value	Force Activated	Force Activated Value	Process Variable	Process Variable Value	Data Type	Description [1]
ModuleOk	TRUE	<input type="checkbox"/>	FALSE			BOOL	Module state (1 = module operational)
NodeStatus	261	<input type="checkbox"/>	0			UINT	Node status
Controlword_6040	2#0000_0000_0000_1111	<input type="checkbox"/>	2#0000_0000_0000_0000	Controlword	2#0000_0000_0000_1111	UINT	Controlword
Statusword_6041	2#0000_0111_0011_0111	<input type="checkbox"/>	2#0000_0000_0000_0000	Statusword	2#0000_0111_0011_0111	UINT	Statusword
ModesOfOperation_6060	1	<input type="checkbox"/>	0	ModeOperation	1	SINT	Modes of Operation
ModesOfOperationDisplay_6061	1	<input type="checkbox"/>	0	ModeDisplay	1	SINT	Modes of Operation Display
PositionActualValue_6064	0	<input type="checkbox"/>	0	Motion:ActualPosition	0	DINT	Position Actual Value
VelocityActualValue_606C	0	<input type="checkbox"/>	0	Motion:ActualVelocity	0	DINT	Velocity Actual Value
TargetPosition_607A	0	<input type="checkbox"/>	0	Motion:TargetPosition	0	DINT	Target Position
TargetVelocity_60FF	0	<input type="checkbox"/>	0	Motion:TargetVelocity	0	DINT	Target Velocity

Figure 6.7: I/O Mapping Watch window. Here we are able to observe the value of the PLC inputs and outputs, the values of the Process Variables connected to the PLC channel and also force the value of the PLC Channels.

Name	Type	Scope	Force	Value
Command	struct_command	local		
ShutDown	USINT			1
SwitchOn	USINT			2
DisableVoltage	USINT			3
QuickStop	USINT			4
DisableOperation	USINT			5
EnableOperation	USINT			6
FaultReset	USINT			7
Controlword	UINT	global	<input checked="" type="checkbox"/>	2#0000_0000_0001_1111
StartMotion	BOOL	global		TRUE
State	struct_State	global		
Start	USINT			0
NotReadySwitchOn	USINT			1
SwitchOnDisabled	USINT			2
ReadySwitchOn	USINT			3
SwitchedOn	USINT			4
OperationEnabled	USINT			5
QuickStopActive	USINT			6
FaultReactionActive	USINT			7
Fault	USINT			8
Statusword	UINT	global	<input checked="" type="checkbox"/>	2#0000_0111_0011_0111
Step	enum_Step	local		step 12
command	USINT	local		6
state	USINT	local		5

Figure 6.8: Watch window for a program. All the program Process Variables can be added and watch here. We are able also to modify the value of the PVs, for example this program is a state machine that follow a number of step, by modifying the PV "Step", we make the State Machine return to that step and repeat the tasks.

watched. It is important to remark that the watch option is not in real time due to the delay caused by the Ethernet port and by the PC processes, then also the Force operation is not in real time.

To make the PDO be Synchronous we right click on the EPOS2 icon in the, we chose "Configuration" and under the node "Communication parameters", we can modify the type of the PDOs making them synchronous. We have also to declare the number of SYNC messages between two transmission (figure 6.9).

In Online menu Services, we find the command Cold Restart that delete all the project and variables from the PLC and re-install the entire project again. The variables will be delayed both from the RAM and from the EEPROM.

Communication parameters	
RPDO1	
COB-ID	\$NODEID+0x200
Implicit activation	On
Transmission type	synchronous - cyclic
Number of sync messages between two transmissions	1

Figure 6.9: Here is shown how to make a PDO be Synchronous. We shown only the RxPDO 1, but is the same for the others RxPDOs, and for the TxPDOs.

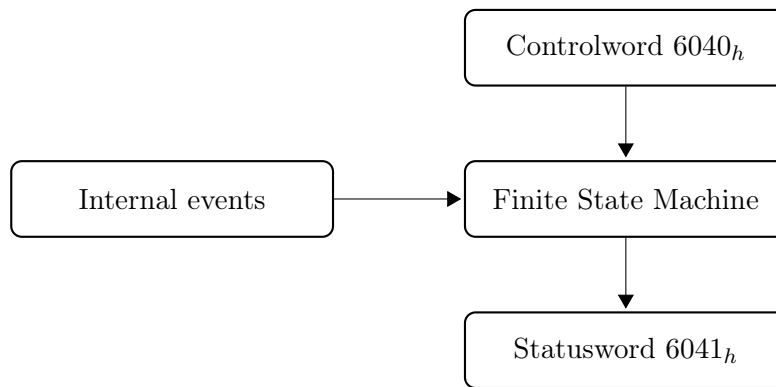


Figure 6.10: Inputs and output of the Finite state machine described in CiA 402 protocol.

6.2 CANopen CiA 402 protocol

EPOS2[®] controller are based on the CiA 402 protocol [12]. This protocol [8] provides a strict sequence of actions for the initialization, the control and the exit for the motion control. All the sequences are based on the finite-state machine (FSM). Finite state machine is a machine that can be only in one of a finite number of states at any given time. The FSM can change from a state to another state in consequence of some external input, the change from a state to another state is called *transition*. FSM can be deterministic or non-deterministic, the state machine described in CiA 402 protocol is deterministic; this means that the transitions are unique and there isn't any sort of statistical or stochastic process during transitions. Transitions are caused only by internal event (such as error messages) or by the *Controlword*, while the actual state in which the FSM is, is given by the *Statusword*. Controlword and Statusword are the two main aspects of the FSM, they are 16-bit data type (see subsection 6.2.1). In figure 6.10 are shown inputs and outputs of the FSM.

6.2.1 Controlword and Statusword: Finite State Machine

Controlword and *Statusword* are 16-bit numbers, in which every bit, or a combination of bits, give an instruction to EPOS2 (*Controlword*) or give us an information from the EPOS2 (*Statusword*). So the *Controlword* is used to give a command, while the *Statusword* tells us if command happens, and the actual state of EPOS2. In tables 6.1(a) and 6.1(b) there are the name of the bits of *Statusword* and *Controlword*.

We must note since now that we are programming the PLC, that is the subject of the system, so everything is related to it: in fact the *Controlword*, which is an input for the EPOS2, becomes an output for the PLC, and the *Statusword*, which is an output for the EPOS2, becomes an input for the PLC.

Table 6.1: Bits of Statusword and Controlword for the state machine in figure 6.11.

(a) Statusword bits.

Bit	Description	M/O
0	Ready to switch on	M
1	Switched on	M
2	Operation enabled	M
3	Fault	M
4	Voltage enabled	M
5	Quick stop	M
6	Switch on disabled	M
7	Warning	O
8	Manufacturer specific	O
9	Remote	M
10	Target Reached	M
11	Internal limit active	M
12 ÷ 13	Operation mode specific	O
14 ÷ 15	Manufacturer specific	O

(b) Controlword bits.

Bit	Description	M/O
0	Switch on	M
1	Enable voltage	M
2	Quick stop	M
3	Enable operation	M
4 ÷ 6	Operation mode specific	O
7	Fault reset	M
8	Halt	O
9 ÷ 10	Reserved	O
11 ÷ 15	Manufacturer specific	O

The State machine defined by CiA 402 protocol is a sequence of states that the drive takes. Only these states are allowed.

In figure 6.11 is shown the State machine. Every state is described by a value of the Statusword and every transition is executed when the Controlword assumes a well defined. The state machine described in figure is the one that describes the sequences of transitions to make the EPOS2 enable to run and that describe what happens in case of fault. The EPOS2, when the power is apply, starts always in Fault state, which is adjusted with the Controlword.

Transition is a sequence of steps that take the FSM from state "A" to state "B". The command of the change of state is given by the Controlword, and the information of the change in state is given by the Statusword. In figure 6.12 is shown the flow diagram to get a transition. As shown in figure, Controlword and Statusword are the two main information to complete the transition and understanding if the transition was happened. Every bit of them have a particular meaning. The status in figure 6.11 are described by the bits 0, 1, 2, 3, 5 and 6 of the Statusword⁵ (see table 6.2(a)), while the transition are described by the bits 0, 1, 2, 3 and 7 of the Controlword (see table 6.2(b)). The other bits have specific meaning based on the Mode of operation (see subsections 6.3.1, 6.3.2 and section 6.4).

6.3 Position-based modes of operation

EPOS2 controllers are needed to control of the joints motion. MAXON® have created a function which starts from position, velocity and acceleration demand values and give the amount of current to feed the motor. These functions are called Position Control Function (figure 6.13) and Current Control Function (figure 6.14) respectively.

Note that in the Position Control Function, the value of "Position demand value", "Velocity demand value" and "Acceleration demand value" are calculated by the specific function Trajectory Generator which is in the EPOS2 (see chapter 5.1).

The task of the Current Control Function is to turn the value of position, velocity and acceleration into current to feed the motor.

6.3.1 Homing Mode

Homing position is the robotic arm configuration when it isn't working. Homing position is also the first operation the robotic arm do, this because all the configurations are based on the homing position. In fact, for example, robotic arm needs to reach the operational configuration, that is done from the homing position. Home Position is also important because it becomes the reference position for all the further joints motion.

When the robotic arm is not feed with high voltage, hence the motors don't provide any torque, near to the home position there is a structure to support the robotic arm weight (see section 2.1). But the importance of this structure is also to host the limit switches. Limit switches are mounted on the mechanical structure

⁵all the other bits are irrelevant, that means that they could be 0 or 1, but the status doesn't change.

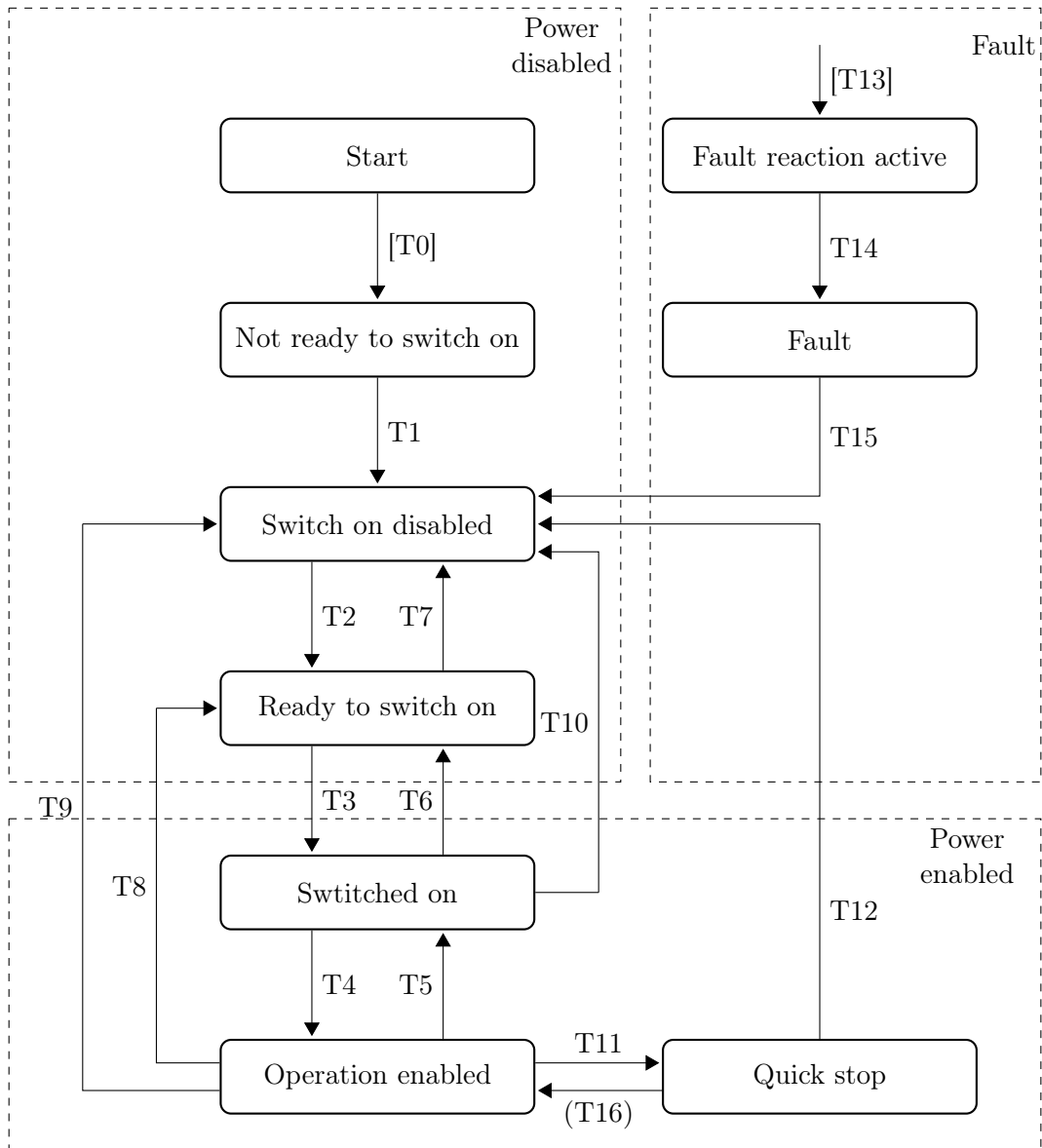


Figure 6.11: State machine defined by CiA 402 protocol. In the rectangles there are the name of the states, the transitions are named with the letter "T". The transitions in square brackets are executed automatically, and the one in brackets is optionally (we don't use it).

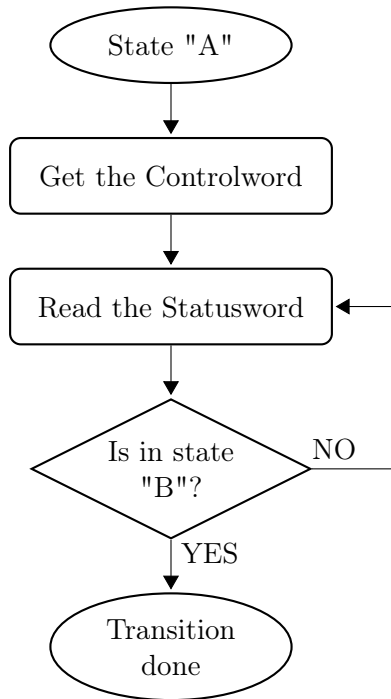


Figure 6.12: Flux diagram for a transition.

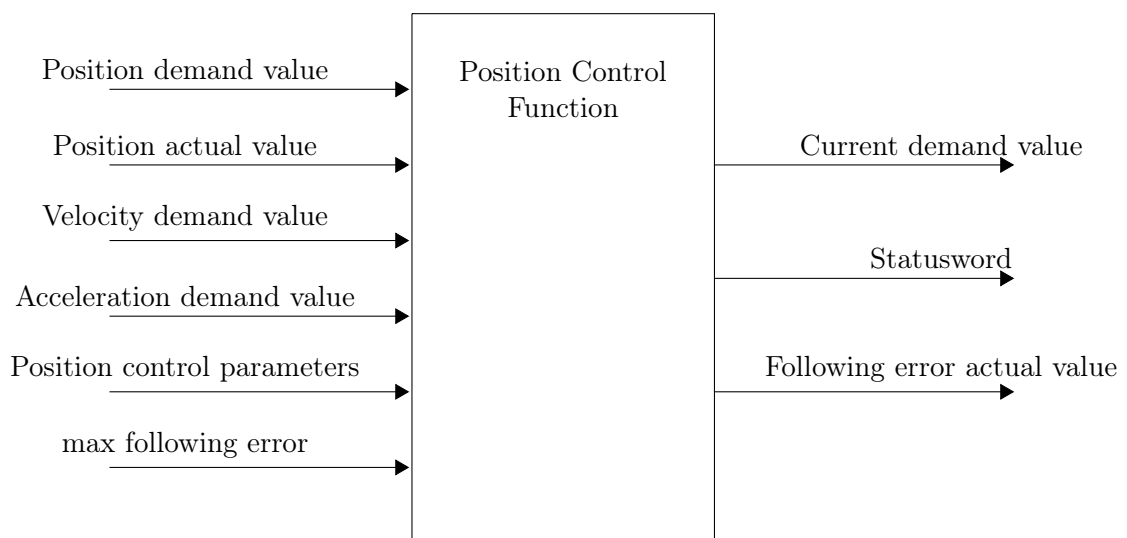


Figure 6.13: Position Control Function.

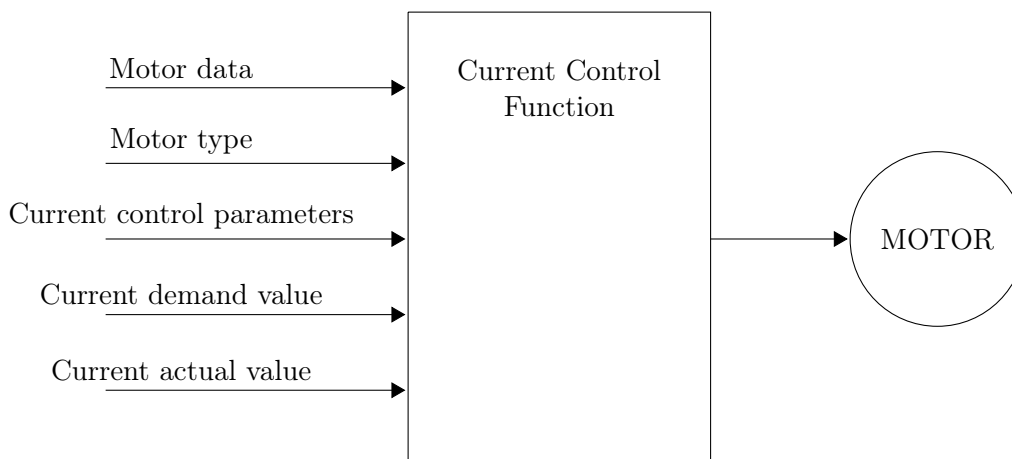
Table 6.2: Bits of Statusword and Controlword for the state machine in figure 6.11.

(a) Device state bits (bit marked with "x" is irrelevant for that state).

Bit number						State
6	5	3	2	1	0	
0	0	0	0	0	0	Not ready to switch on
1	0	0	0	0	0	Switch on disabled
0	1	0	0	0	1	Ready to switch on
0	1	0	0	1	1	Switched on
0	1	0	1	1	1	Operation enabled
0	0	0	1	1	1	Quick stop active
0	x	1	1	1	1	Fault reaction active
0	x	1	0	0	0	Fault

(b) Device control bits (bit marked with "x" is irrelevant for that command).

Command	Bit number					Transitions
	7	3	2	1	0	
Shutdown	0	x	1	1	0	2, 6, 8
Switch on	0	0	1	1	1	3
Disable voltage	0	x	x	0	x	7, 9, 10, 12
Quick stop	0	x	0	1	x	7, 10, 11
Disable operation	0	0	1	1	1	5
Enable operation	0	1	1	1	1	4, 16
Fault reset	↑	x	x	x	x	15

**Figure 6.14:** Current Control Function

to give to the controller the information on where the home position is. In fact, according to CiA 402 protocol, there are several way to define the home position and we choose the method number 1 for the second and the third joint and method number 2 for the first joint, this have to be set in object Homing Method 6098_h (see section 4.4) for every motor. In general for these two methods, the home position is the position when the first pulse of the encoder index after the limit switch is set high, plus the Home offset (607C_h, see section 4.4). Homing method number 1 follow the next step to define the home position (see also figure 6.15(a)):

1. the initial direction of movement is negative (counterclockwise) if the negative switch is inactive. The motor axle moves with Speed for Switch Search until the edge of negative switch;
2. the axle moves with Speed for Zero Search until the first encoder index pulse;
3. finally the axle moves with Speed for zero search to the Home offset.

As the same the Homing method number 2 follow the next steps (see also figure 6.15(b)):

1. the initial direction of movement is positive (clockwise) if the positive switch is inactive. The motor axle moves with Speed for Switch Search until the edge of positive switch;
2. the axle moves with Speed for Zero Search until the first encoder index pulse;
3. finally the axle moves with Speed for zero search to the Home offset.

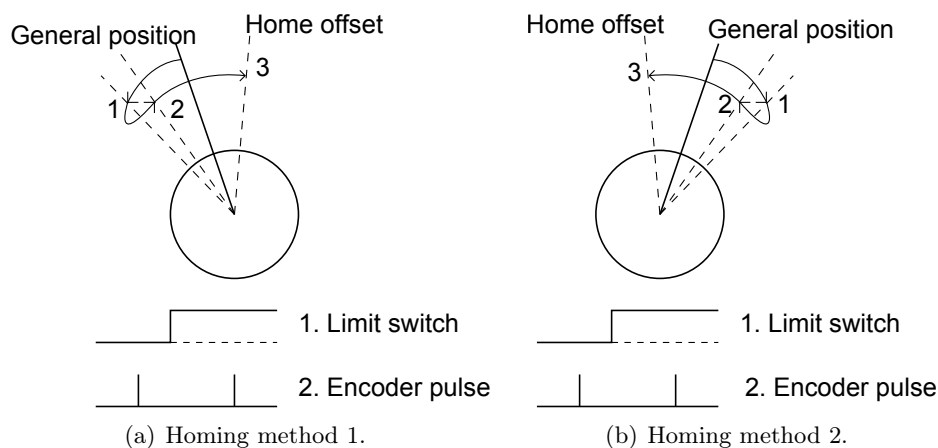


Figure 6.15: Homing methods.

We must underline some aspects. First, the direction of the rotation is the rotation of the link (because this one is able to active the limit switch) and the fact that the rotation is positive if the axle moves clockwise is due to the fact that between the motor and the link there is the reduction gear that reverses the rotation of the motor axle. Second, we must pay attention when we make the cable for the switches, in fact, for different Homing methods, existing different input pin on EPOS2 in which

Table 6.3: Bits of Statusword and Controlword for Homing Mode.

(a) Controword bits in Homing mode.

Bit	Name	Value	Description
4	Operation Start	0	Homing mode inactive
		1	Homing mode active
8	Halt	0	Move
		1	Stop the axle with Profile Deceleration (6084 _h)

(b) Statusword bits in Homing mode.

Bit	Name	Value	Description
10	Target Reached	0	Home Position not reached
		1	Home Position reached
12	Home attending	0	Homing mode has not yet completed
		1	Homing mode has completed successfully
13	Following error	0	Not Homing error
		1	Homing error. The Homing mode has been terminated not successfully. For error cause, read the error code

the switch is connected. Finally, all the switches have to be of type "normally open" (NO), this attention is due to the fact that the switches on the market have both the normally closed connection and the normally open connection (NO).

Flow diagram of the Homing Mode is in figure 6.16.

When motor reached the home position, this will be used as reference for all further moves.

In table 6.3 the meaning of Controlword bits and Statusword bits during the Homing mode are explained.

Statusword bit 10 has different meanings based on the value of Statusword bit 8, see table 6.5 for explanation.

6.3.2 Profile Position Mode

Profile Position Mode (PPM) creates acceleration and position trajectories (of the joint) starting from trapezoidal velocity trajectory. Referring to figure 6.17:

- A_{max} and A_{min} are the Profile Acceleration and Profile Deceleration defined in objects 6083_h and 6084_h respectively.
- V_{max} is the Profile Velocity defined in object 6081_h.
- P_{end} is the final position. If the movement is absolute $P_{end} = \text{Target Position}$, else if the movement is relative $P_{end} = \text{Target Position} + \text{Position Demand Value}$.

In order to implement the Profile Position Mode, we have to set the parameters described in Section 4.4 and we have to write on the object Target Position 607A_h,

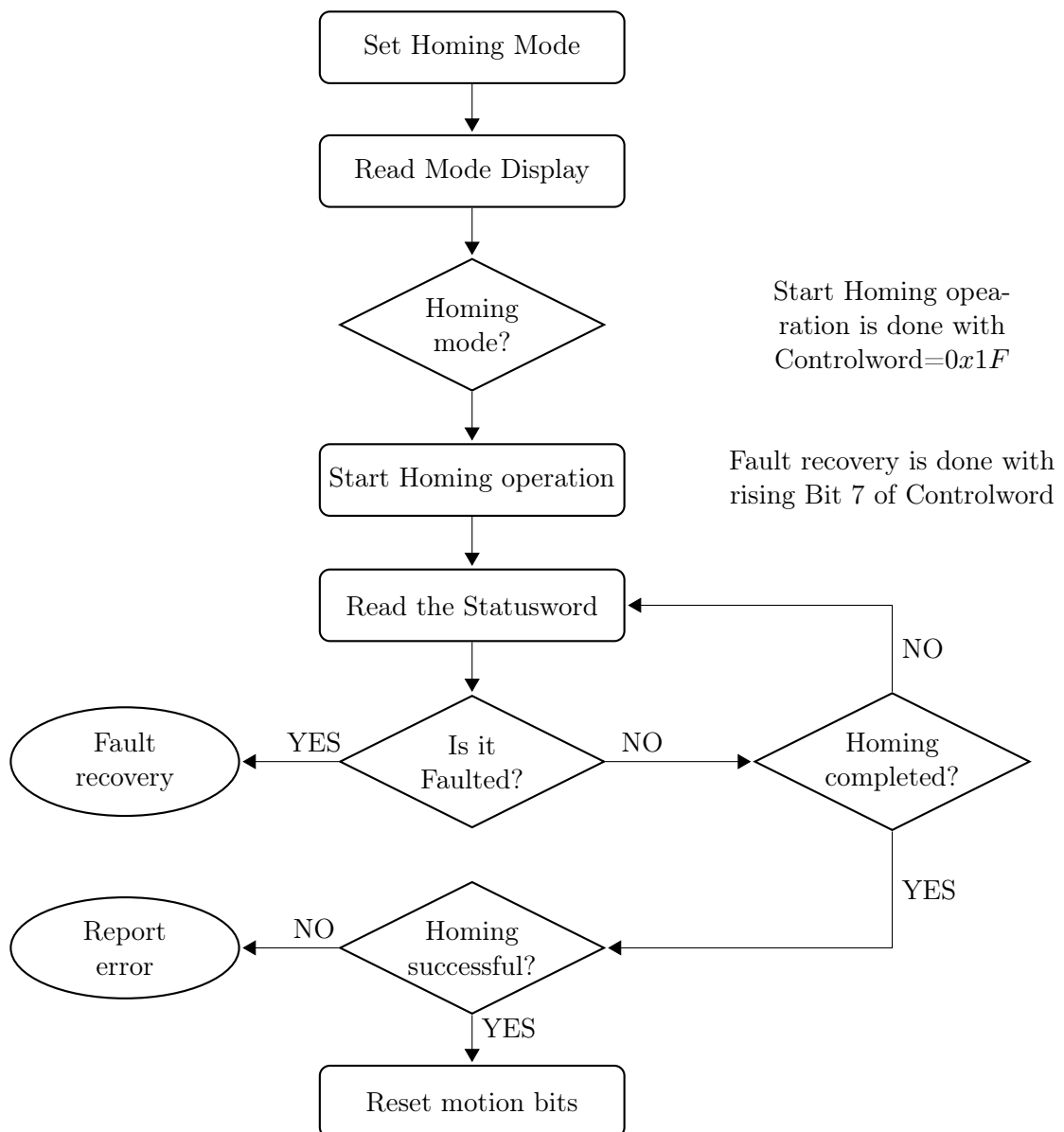


Figure 6.16: Flow diagram with the sequences of instructions to implement the Homing Mode.

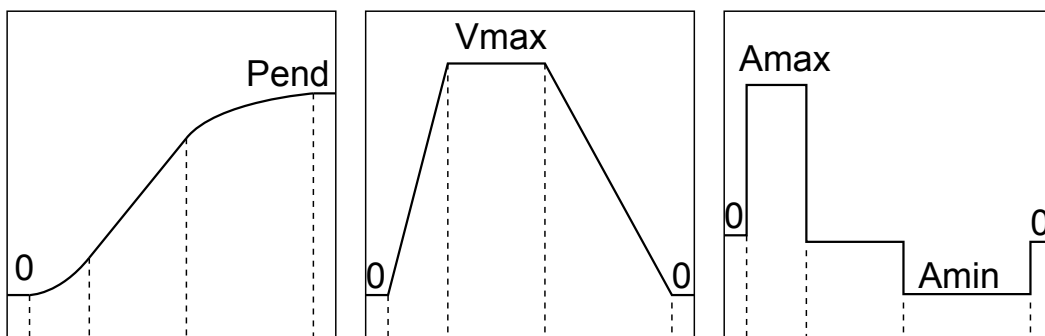


Figure 6.17: Position and Acceleration trajectories generated from trapezoidal Velocity profile

Table 6.4: Bits of Statusword and Controlword for Profile Position Mode.

(a) Controword bits in PPM.

Bit	Name	Value	Description
4	New set point	0	Not assume Target Position
		1	Assume Target Position
5	Change set point	0	Finish actual movement before start the next
		1	Interrupt actual movement and start the next
6	Absolute / Relative	0	Target Position is absolute
		1	Target Position is relative
8	Halt	0	Move
		1	Stop the axle with Profile Deceleration (6084 _h)

(b) Statusword bits in PPM.

Bit	Name	Value	Description
10	Target Reached	0	Target Position not reached
		1	Target Position reached
12	Set point acknowledge	0	Trajectory generator hasn't yet assumed position value
		1	Trajectory generator has assumed position value
13	Following error	0	Not Following error
		1	Following error

the value of the final position of the joint given in position unit Step:

$$1\text{Step} = 4 \times (\text{Encoder pulse per revolution})$$

In addition we have to use some specific bits of the Controlword, in particular the bits number 4, 5, 6 and 8 (table 6.1(b)). The response for the commands is given in terms of Statusword with the bits number 10,12 and 13 (table 6.4(b)). Note that the device must be in State "Operation Enabled" state, so Statusword bits number 0, 1, 2 and 5 must be high.

Referring to tables 6.4(a) we have to make some observations:

- Following error is the difference between Target Position and Position Demand Value⁶. This error is set when the Encoder Pulse per turn hasn't been set correctly, because the drive tries to move the motor, but it observe a different value between where the drive thinks the motor is and where the motor really is. This error turn the EPOS2 into Fault state. Or when the Following Error raises.
- Target Reached has different meanings based if the Controlword Halt bit is set

⁶Position Demand Value is the value of the joint request position exiting from the Profile Position Trajectory generator, that is a sub function of the Position Control Function implemented by MAXON[®]

Table 6.5: Different meaning for Statusword Target Reached bit based on Controlword Halt bit value. The meaning is the same bot for Profile Position mode and for Homing mode (this is in brackets.)

Target Reached Bit value	Halt Bit value	Description
0	0	Target (or Homing) Position is not reached
0	1	Axle deceleration with Profile Deceleration
1	0	Target (or Homing) Position reached
1	1	Velocity of axle is 0

or no (see table 6.5).

In order to write the program for Profile Position Mode, we can follow the sequence of instructions shown if figure 6.18. We give some clarification on the figure 6.18:

- Ready for new Set point is the state with Bit 10 High and Bit 12 Low.
- Set point Acknowledge is the state with Bit 12 High.
- Target reached is the state with Bit 10 High.
- Bit 4 in Controlword is set High when we enter a new value for Target Position, after the target has been reached, Bit 4 will be set to Low, and after is set to High again.

Also in this case the program becomes a sort of state machine that read the input and write the output, if a different state different from those expected occurs, the drive turns into Fault state.

Finally we have to observe that this Mode is just able to make a single motion per time. In fact, as we can see in figure 6.17, the velocity start from zero at the begin of motion and return back to zero at the end of the motion, this means that the motion could at least be formed of many segments. The same thing happens with the Position Mode, with the only difference that in this mode, no trajectory will be created: the velocity is always at the maximum value given by the Maximal Profile Velocity object so there isn't any trapezoidal velocity (excluding the initial and final moments of acceleration and deceleration); while the acceleration is always at the value given by Maximum Acceleration object.

To conclude, we could say that if we want to improve the smoothness of the trajectory we can PDO map both the Profile Velocity (6081_h), the Profile Acceleration (6083_h) and the Profile Deceleration (6084_h). In this way we can control all the three fundamentals motion quantities, so we can increase the number of the trajectory segments where the trajectory becomes more complex (see algorithm 5).

This is just a way to improve the smoothness, but the trajectory is still made by many segments. We can do a segmented motion also wit the Position Mode, but in this case we have to map the Maximal Profile Velocity (607F_h) and the

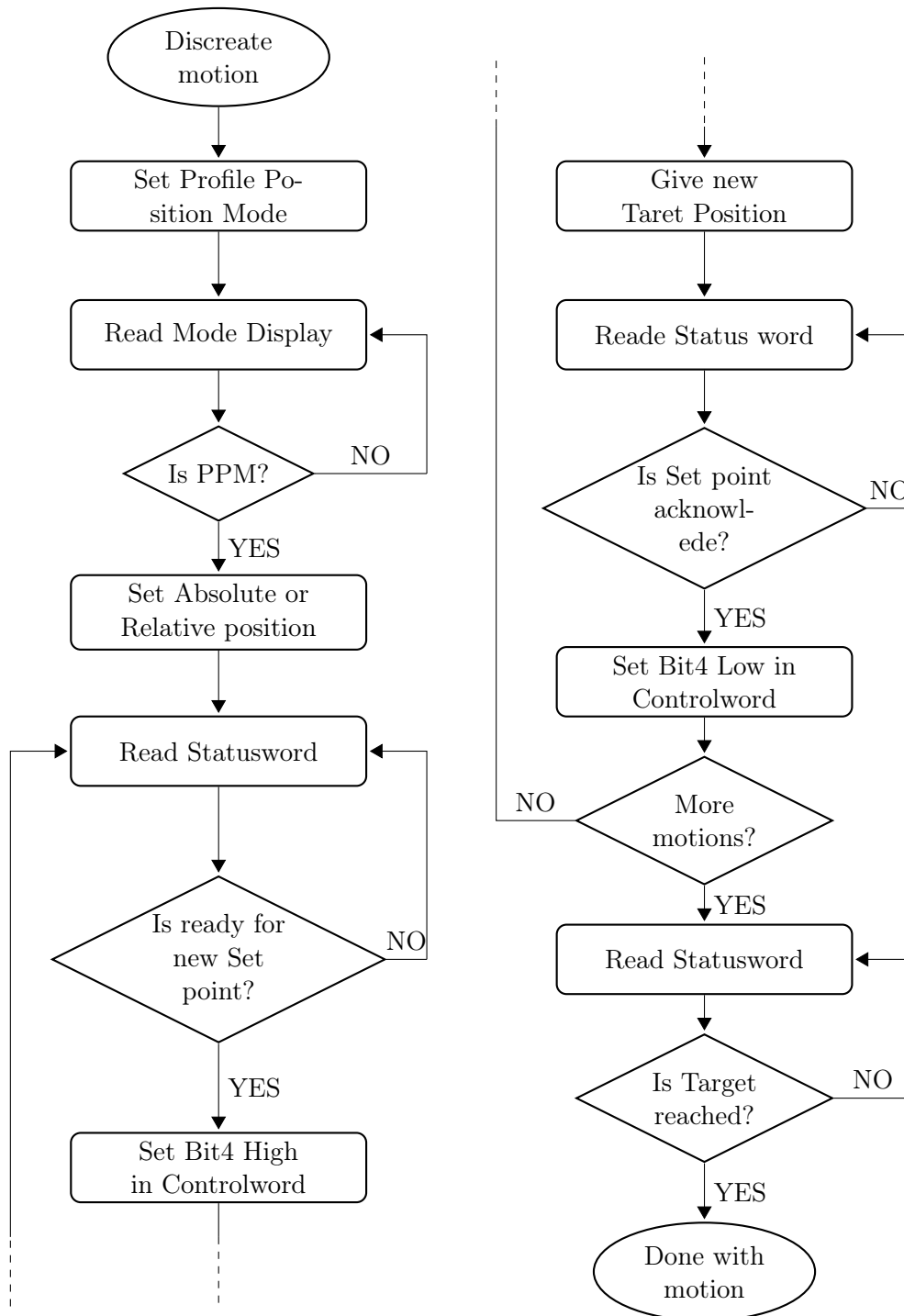


Figure 6.18: Flow diagram with the sequences of instructions to implement the Profile Position Mode for more than one consecutive motion.

```
1 Position: array with  $n$  elements ;
2 Velocity: array with  $n$  elements ;
3 Acceleration: array with  $n$  elements ;
4 switch (Step) do
5   case Step 1
6     Read Statusword ;
7     if Bit 12 = 0 then
8       | Step = step 2 ;
9     end
10  case step 2
11    Write New set point command on Controlword ;
12    Target Position = Position[i] ;
13    Profile Velocity = Velocity[i] ;
14    Profile Acceleration = Acceleration[i] ;
15    i=i+1 ;
16    Step = step 3 ;
17  case step 3
18    Read Statusword ;
19    if Set point acknowledge then
20      | Step = step 4 ;
21    end
22  case step 4
23    if  $i = n$  then
24      | Stop ;
25    end
26    Reset Controlword ;
27    if Target reached then
28      | Step = step 1 ;
29    end
30 endsw
```

Algorithm 5: PPM with discrete position, velocity and acceleration.

Max Acceleration ($60C5_h$). For creating a continuous motion we have to use the Interpolated Position Mode (section 6.4).

6.4 Interpolated Position Mode

Suppose we have already generated a joint trajectory (for example with a program whitening the PLC) composed by many reference points, Interpolated Position Mode (IPM) uses a cubic spline to interpolate those reference points. The interpolation method is the PVT method described below in section 6.4.2.

Joint trajectory must have been already calculated by the CANopen producer (PLC) and passed to controller's interpolated position buffer as a set of points, after the Interpolation Controller⁷ creates the values position, velocity and acceleration which feed the Position Control Function⁷, this give the position, velocity and acceleration to the Current Function and sends the Statusword and Interpolationstatus to the PLC.

We use only bits 4 and 8 of the Controlword in order control the IPM, in particular bit 4 enables IPM and bit 8 gives the Halt command (as what happens in Profile Position Mode). To implement PVT interpolation we use a single object called Interpolation Data Record ($20C1_h$), whose structure permits to insert both Position (P) and velocity (V) and Time (T). In fact, it is of type 64 bit complex data structure, this means that the 64 bits is divided into three sub-set in which we specify the value of the PVT parameters:

- in the first 32 bits we insert the Position with a data type of SIGNED 32.
- in the following 24 bits we insert the Velocity with a data type of SIGNED 24.
- in the last 8 bits we insert the Time with a data type of UNSIGNED 8.

Once we have created the complex structure of the object $20C1_h$ we have to insert it into a FIFO (First In First Out) object, which is implemented by a circular buffer with the length of 64 entries. The circular buffer is a memory which supplies the difference between the time for the motion and the time to write the new PVT data into the FIFO. In other words if we don't use the circular buffer, the PLC will write the PVT data into the EPOS2 faster than the time in which a move starts and ends and the motion will result segmented; so to avoid this we use one FIFO object.

FIFO principle is based on the following statement: *the first data insert into the buffer is the first data to be processed*. For this reason this method is very useful in our case, because we have a vector containing all the joint positions in order. It is easier to implement FIFO by a circular buffer due to its mode of work.

Circular buffer is a data structure which uses a single, fixed size buffer⁸. The useful property of a circular buffer is that it doesn't need to have its elements shuffled around when one is consumed (if a non-circular buffer were used then it would be necessary to shift all elements when one is consumed). This property make circular

⁷by MAXON®.

⁸Buffer is a region of a physical memory storage used to temporarily store data

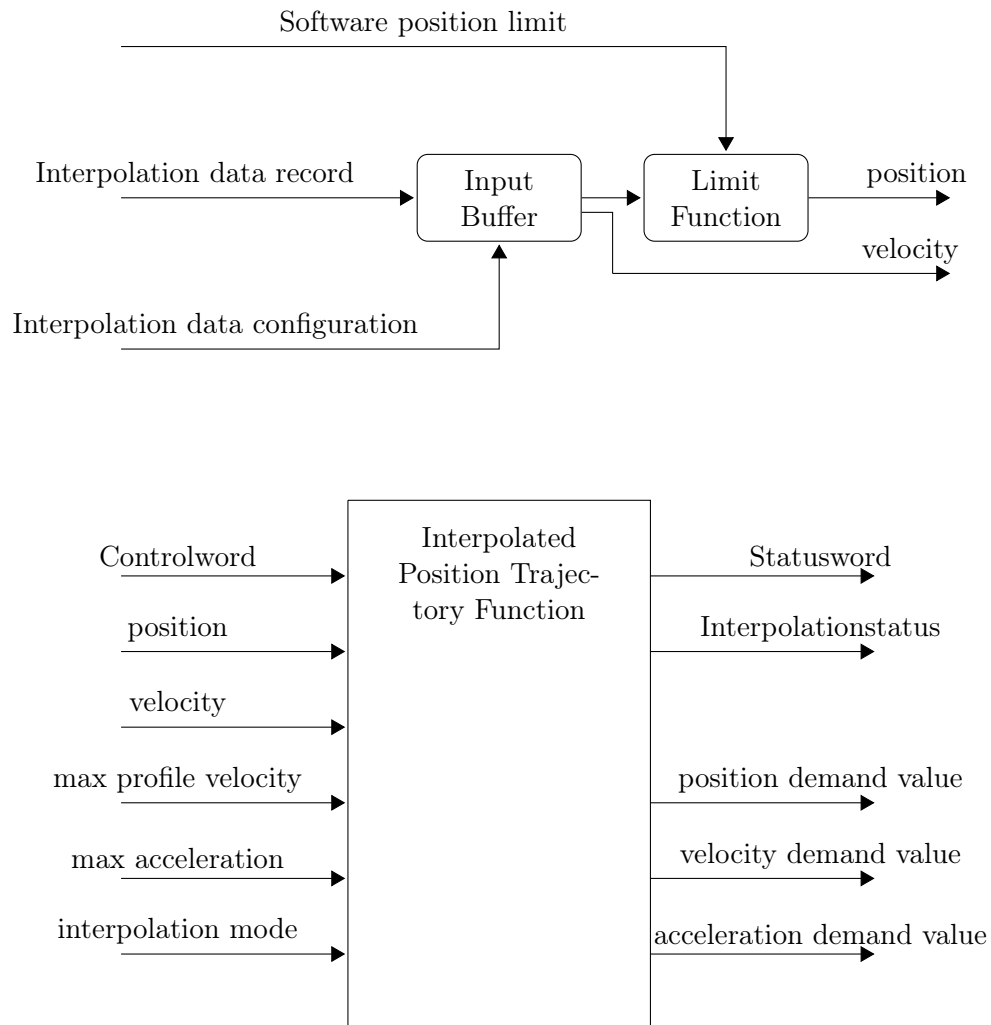


Figure 6.19: Block diagram for explaining Interpolated Position Mode.

buffer well-suited as a FIFO buffer. Referring to figure 6.20, let us explaining how a circular buffer works:

1. Suppose our buffer is 6–element circular buffer. A buffer starts empty.
2. Assume that the first value of joint position for interpolation PVT_1 is written in a place of the buffer (exact starting location doesn't matter in a circular buffer).
3. Then other two PVT are added, PVT_2 and PVT_3 , which are appended after PVT_1 (this is automatically defined).
4. If two PVT elements are the removed (processed), also the two oldest value inside the buffer are removed.
5. By adding PVT elements, the buffer will be completely full.
6. A consequence of the circular buffer is that when it is full and a subsequent write is performed, then it starts overwriting the oldest data. New elements PVT_9 and PVT_{10} are added and overwrite the PVT_3 and PVT_4 .
7. if two elements are now removed then what would be returned is not PVT_3 and PVT_4 but PVT_5 and PVT_6 .
8. Finally, when all PVT data has been processed (so the move is done), the buffer return empty.

Circular buffer has been already implemented in EPOS2[®] by MAXON[®], so we have not to implement it, circular buffer could be implemented using two pointers and two integers:

- buffer start in memory (pointer),
- buffer end in memory, or buffer capacity (pointer),
- start of valid data (integer),
- end of valid data, or amount of data currently in the buffer (integer).

When an element is overwritten, the start pointer is incremented to the next element.

We are not able to know the four values listed above that describe the status of the buffer (they are given with different names: Maximum Buffer Space $60C4_h$ sub01_h, Actual Buffer Size $60C4_h$ sub02_h, Buffer Position $60C4_h$ sub04_h and Size of Data Record $60C4_h$ sub04_h) because they are calculated inside the EPOS2 and they can't be mapped with PDO. But we can know how much the buffer is full or empty. Figure 6.21(a) and figure 6.21(b) show two extreme filling status of the buffer, one partially full and the other completely full. Both the situations raise a warning in the Interpolation Buffer Status $60C4_h$ sub01_h.

Objects Interpolation Buffer Status $20C4_h$ sub01_h is a only read, UNSIGNED 16 number whom bits gives us information about the status on the IPM input data buffer, these information are shown in table 6.6.

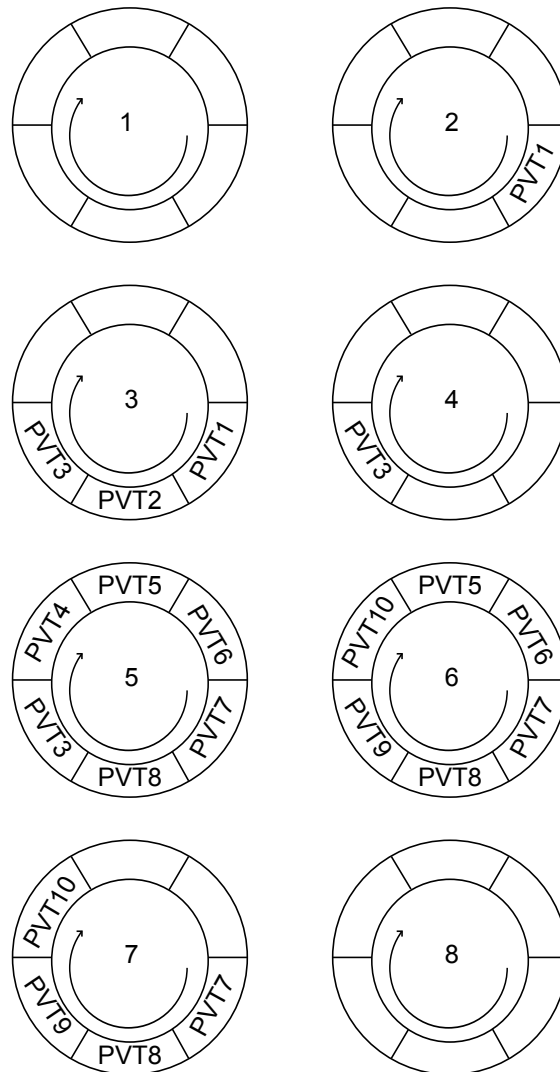


Figure 6.20: Example and explanation on how a circular buffer works. Number inside the figures are the same number of the explanation above. The arrows indicate the write and read direction.

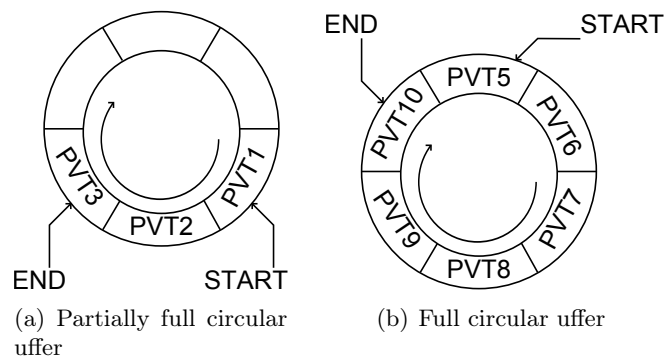


Figure 6.21: Circular buffer

Table 6.6: Interpolation Buffer status bits.

Bit	Name	Value	Description
0	Underflow warning	0	No buffer underflow warning
		1	Buffer underflow warning level reached (20C4 _h sub02 _h)
1	Overflow warning	0	No buffer overflow warning
		1	Buffer overflow warning level reached (20C4 _h sub03 _h)
2	Velocity warning	0	No velocity warning
		1	IPM velocity grater than Profile Velocity (6081 _h) detected
3	Acceleration Warning	0	No acceleration warning
		1	IPM acceleration grater than Profile Acceleration (6083 _h) detected
4 ÷ 8			reserved
8	Underflow error	0	No buffer underflow error
		1	Buffer underflow error
9	Overflow error	0	No buffer overflow error
		1	Buffer overflow error
10	Velocity error	0	No maximal velocity error
		1	IPM velocity grater than Max Profile Velocity (607F _h) detected
11	Acceleration error	0	No maximal acceleration error
		1	IPM acceleration grater than Max Profile Acceleration (60C5 _h) detected
12 ÷ 13			reserved
14	Buffer enabled	0	Disabled access to the input buffer
		1	Access to the input buffer enabled
15	IPM active	0	IPM inactive (same as bit 12 in Status-word)
		1	IPM active

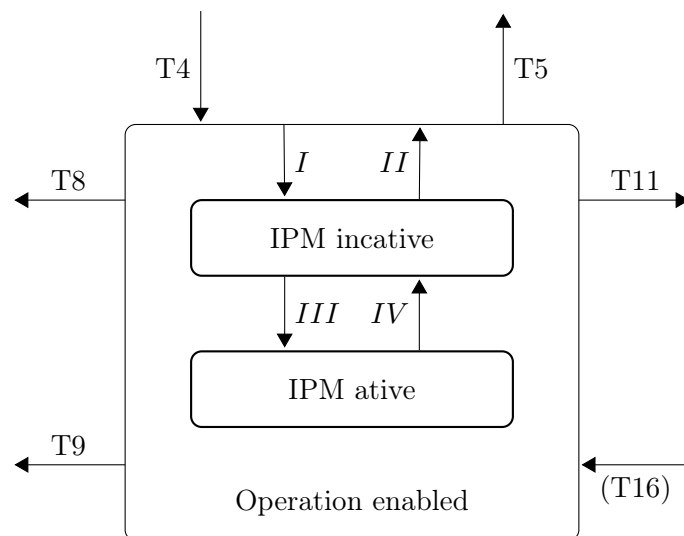


Figure 6.22: Interpolate Position Mode finite state machine. State Operation enabled is the same state with the same transitions which are showed in figure 6.11.

In particular, when the Underflow warning raises we have to insert PVT reference points into the buffer, otherwise the Underflow error occurs. While, when the Overflow warning raises we have to stop to feed the PVT reference points, otherwise the Overflow error occurs. All the errors make the device turn into Fault, the reaction to fault may be change by modifying the Fault Reaction Option Code in the Object Dictionary.

With Statusword and Interpolation Buffer Status, we know the state of the EPOS2 during the Interpolated Position mode. We can define a sub-state machine for the IPM into the Operation enabled state (figure 6.22). States and transitions are explained in tables 6.7.

Typical IPM command sequence is showed in figure 6.23. Let us give some clarification on the flux diagram that describes the command sequence :

- In Set Parameters we have to set some parameters that we find under the menu Configuration as explained in subsection 6.1. The value of these parameters depends on the motors and the application the robotic arm is called to make. The parameters to set are:
 - Max. Following Error (6065_h)
 - Software position Limit (607D_h)
 - Max. Profile Velocity (607F_h)
 - Max. Acceleration (60C5_h)
 - Profile Velocity (6081_h)
 - Profile Acceleration (6083_h)
 - Quick Stop Deceleration (6084_h)
- After we have to set the Interpolation Profile Mode by setting Mode of Operation (6060_h) = 07_h.

Table 6.7: States and transition of the Finite State machine in figure 6.22.

(a) Statusword bits.

State	Description
IPM inactive	Device accepts input data and buffers it for interpolation, but doesn't move the axis
IPM active	Device accepts input data moves the axis

(b) Controlword bits.

Transition	Event
<i>I</i>	IP mode selected ($60C0_h$) and clear buffer data writing on object ($60C4_h$ sub 06_h)
<i>II</i>	IP mode not selected ($60C0_h$)
<i>III</i>	Enable IP mode by setting Controlword bit 4 to 1
<i>IV</i>	Disable IP mode by setting Controlword bit 4 to 0

- Now we take the device into Operation Enable state by following the State Machine as shown in figure 6.11. We use Controlword (6040_h) and Statusword (6041_h) to control transition (figure 6.12).
- Enable buffer access is made with object Buffer Clear ($60C4_h$ sub (06_h)). This object have to be PDO mapped and it is write only. We write first 00_h to Disable and Clear buffer, after we write 01_h to Enable buffer.
- Feed starting PVT reference points. At least two points are required to start the trajectory generator. The other points will be inserted while motor is moving.
- Activate Interpolation with Controlword bit 4 set to 1.
- Now the motor is moving, but we can feed new PVT reference points. See Algorithm 7 on how feed reference points.
- When there aren't any new point, we feed the profile end by setting Controlword bit 4 to 0, or by inserting a PVT reference point with time = 0. If we don't make this, an error occurs and the EPOS2 turns itself into Fault state.

If an error occurs, the EPOS2 reacts as described in the object Shutdown Option Code $605B_h$. An error set to 1 the Interpolation Buffer Status $20C4_h$ sub 03_h bit 15, and the trajectory will be aborted.

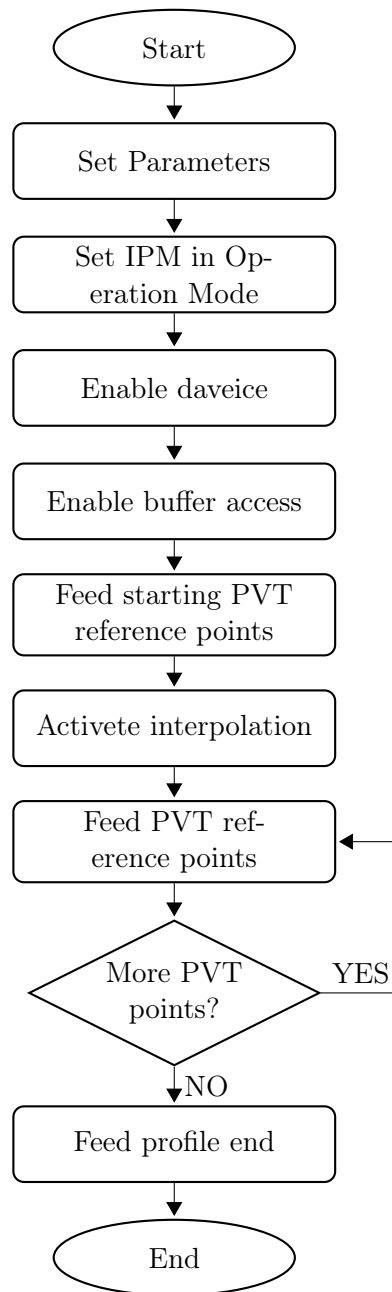


Figure 6.23: Typical command sequence to implement the Interpolated Profile Mode.

```

1 if (Bit0 == 1 AND Bit1 == 0) then
2   | Data = new PVT ;
3   | else if (Bit0 == 0 OR Bit1 == 1) then
4   |   | Stop feed new PVT ;
5   | end
6 end
7 .

```

Algorithm 6: How to feed new PVT reference points while motor is moving. Bit0 and Bit1 indicates the respectively Interpolation Buffer Status (20C4_h sub01_h) bits. And Data is the Interpolation Data Record (20C1_h)

6.4.1 Axis Synchronization

Finally we describe how coordinate the motion of multiple axes, as those of a robotic arm. The movement of a number of slave axes can be synchronized if they all run in IPM, and if they all possess the same time. To start the synchronized movement, map the Controlword to a synchronous RxPDO, then use the mapped Controlword to enable interpolation for axes; in this way all the Controlword bit 4 set to 1 at the same time in all axes, and interpolation starts at the same time. But there will no reaction until the next SYNC (because we use a Synchronous RxPDO, see Chapter 4.3). Then, all drives will enable interpolated motion at once, setting the SYNC arrival time as the path specification's "zero" time. If the axes have been synchronized by the SYNC Time Stamp Mechanism, the moving axes will run synchronous within an accuracy of microseconds.⁹

The explanation on how the time is synchronized is called CANopen Time service and it is described in CiA 301 protocol [6]; now we give a brief explanation on it. High Resolution Time Stamp contains the time stamp (date and time) of the last received SYNC Object [1 μ s] after a write access to this object, the EPOS2 calculates the difference between the received time stamp and the internal latched time stamp of the SYNC Object; this time difference is used as correction for the IPM time calculation. The SYNC will be transmitted periodically by the SYNC master. The exact time (T_{e1}) may be stored by the internal 1 μ s timer. The reception time (T_{r1}) of the SYNC message will be stored by latching the device internal motion clock timer. After the measured transmitting time (T_{e1}) will be sent to the drive using the High Resolution Time Stamp object (1013_h). The device then adjusts its internal motion clock time in relation to time given in the last SYNC. For example, let us suppose the latched EPOS2 time $T_d = 0$, after the receive of the SYNC with latched time at the time T_{e1} and the time for the SYNC reception T_{r1} , the new latched EPOS2 time become:

$$T_d = T_d + T_{e1} - T_{r1}$$

and so on with the next SYNC. By sending a CANopen Time service (by default the COB-ID Time Stamp Object 100_h, and it is immutable), the device internal

⁹The accuracy depends on the CAN bit rate. For example a resolution of [1 μ s] required a CAN bit rate of 1 Mbit/s.

motion clock timer can be reset to zero.

If CAN Producer (PLC) isn't able to produce the high resolution time stamp, an EPOS2 might be used as clock master. For this we have to map the object High Resolution Time Stamp object (1013_h) to a synchronous transmitted PDO in the clock master EPOS2, the other EPOS2s must be configured as clock slaves with the High Resolution Time Stamp object mapped to an asynchronous TxPDO with the same COB-ID as the clock master's TxPDO.

We can use this method also in the Profile Position mode, when we want to synchronized the robotic arm axis.

6.4.2 PVT Algorithm

PVT algorithm fits a Jerk¹⁰ profile after the user specified Position, Velocity and Time. PVT algorithm make sure to hit each specified position, with each specified velocity at the specified time. For each point, PVT algorithm calculates acceleration and jerk values to exactly hit the specified position and velocity at the next point. Let X_n , V_n , A_n , J_n and T_n be the position, velocity, acceleration, jerk and time at n-th point, then the algorithm is:

$$\begin{aligned} X_{n+1} &= X_n + V_n \cdot T_n + \frac{1}{2} A_n \cdot T_n^2 + \frac{1}{16} J_n \cdot T_n^3 \\ V_{n+1} &= v_n + A_n \cdot T_n + \frac{1}{2} J_n \cdot T_n^2 \end{aligned}$$

The profile between point may not be desired, but it will be accurate, but, in our case position, velocity and time are calculated with the inverse kinematics of the robotic arm, so in this case they are properly matched and the profile generated with the PVT algorithm is also the joint trajectory that has been already calculated in Simulink Simscape.

By solving the equation above, known that the position profile must be at most of third degree, we find that for two successive points n-1 and n the Position, Velocity and Acceleration laws are:

$$\begin{aligned} P(t) &= a \cdot (t - t_0)^3 + b \cdot (t - t_0)^2 + c \cdot (t - t_0) + d \\ V(t) &= 3a \cdot (t - t_0)^2 + 2b \cdot (t - t_0) + c \\ A(t) &= 6a \cdot (t - t_0) + 2b \end{aligned}$$

Where t_0 is the end time of interpolation interval and the interpolation parameters a , b , c and d are given in term of position, velocity and time in points n-1 and

¹⁰Jerk is the acceleration rate of change: $J = \frac{dA}{dT}$

n:

$$\begin{aligned}
 a &= \frac{2 \cdot (X_n - X_{n-1})}{T_n^3} + T_n \cdot (V_n - V_{n-1}) \\
 b &= \frac{3 \cdot (X_n - X_{n-1})}{T_n^2} + T_n \cdot (V_n - 2 \cdot V_{n-1}) \\
 c &= V(t_0) = V_n \\
 c &= X(t_0) = X_n
 \end{aligned}$$

6.5 The transition control method

As we saw in previous sections, the EPOS2s are driven by a finite state machines (FSM) where the changing in state are executed by transitions. The FSMs are different in base of the task the EPOS2s are making. For this reason I made an overall program which manages all the others programs (called Transition) both in case of emergency and in case of normal working, this program provides the transitions between the states of the Finite State Machine defined by CiA 402 protocol (all the transition are saved and are performed by a Switch-Case control structure (see Algorithm 7). This method also helps to have a more clear working space, a more ordered program structure, a more synchronization between the six EPOS2s and a more reliable react to the emergency. Remember that a transition is made by a set of operations (as shown in figure 6.12):

1. Create the Controlword from a given command;
2. Read the Statusword to verify the change in state;
3. if the state has not been changed, read the Statusword again;
4. if the state has been changed, the transition is done.

Hence all the commands and all the states are saved into the folder that contains the program "Transition" and by it the six Controlword and the six Statusword are managed. The program Transitions manages only one Controlword (called $T_{Controlword}$) which is not connected to any EPOS2 Controlword channel, but all the EPOS2s Controlword are equaled to it, so they are managed with only one command. While the EPOS2s Statusword are verified to be all equal (in terms of decimal number), and after the Statusword of Transition (called $T_{Statusword}$) are equaled to them, in this way only one Statusword is managed, this mean that the function which read the single bits is called only to read the bits of the $T_{Statusword}$ (see Algorithm 7). In this way, many memory is saved. $T_{Controlword}$ and $T_{Statusword}$ are scoped as Global variables, so in the other programs it results more simple to equal them to the Local Controlword and Statusword. This is due to the fact that Automation Studio doesn't allow to connect one RW process variable to more than one PLC input, and to connect one RO process variable to more than one PLC output. Furthermore, the Transition program is also able to turn all the EPOS2s into the state "Switch On Disabled" if just one of them turns into Fault state. Other feature of this method is that the Transition program controls if the state has changed after a command for a

```

1  switch (transition) do
2  |   case T1
3  |   |   automatic ;
4  |   case T2
5  |   |   command = Shut Down ;
6  |   |   new state = Switch On Disabled ;
7  |   case T3
8  |   |   command = Switch On ;
9  |   |   new state = Switched On ;
10 |   :
11 |   case T9
12 |   |   command = Disable Voltage ;
13 |   |   new state = Switch On Disabled ;
14 |   case T10
15 |   |   command = Disable Voltage ;
16 |   |   new state = Switch On Disabled ;
17 |   :
18 |   case T12
19 |   |   command = Disable Voltage ;
20 |   |   new state = Switch On Disabled ;
21 |   :
22 |   case T15
23 |   |   command = Fault Reset ;
24 |   |   new state = Switch On Disabled ;
25 |   :
26 endsw

```

Algorithm 7: The Switch-Case control structure for the Transition Program. Only the most significant transitions are showed. The commands are the same as those in table 6.2(b). States and transitions are the same as those in figure 6.10.

```

1  Read all the Statusword ;
2  if All the Statusword are equal then
3  |    $T_{Statusword} = \text{Statusword } 1$  ;
4  |    $T_{Controlword} = \text{Command}$  ;
5  |   else
6  |   |   Read again
7  |   end
8  |    $i=i+1$ ;
9  end
10 Set all the Controlword equal to  $T_{Controlword}$ ;

```

Algorithm 8: Transition method.

given number of times, if the change in state is not happened the "Disable voltage" command is given.

Also the robotic arm motion is driven by a similar program (called Motion Transition), but in this case all the transitions for the motion control are scoped as Local variables, but the principle of operation is the same as the program Transition. This choice is driven by the fact that only the CiA 402 FSM is able to react to the dangers and turn all the EPOS2s into the states "Operation Enabled" or "Switch On Disabled", that are the two main states. With this method we are sure that all the EPOS2 receive the same command at the same time. This is useful not only to synchronized all the robotic arm axes with only one Controlword, but it is very useful to react to dangers.

The command "Switch On Disabled" is also given if the Statusword of all the EPOS2s are not equal after the Transition command have compare them for a given number of times (this feature is both in the Transition and in Motion Transition).

6.5.1 React to the dangers

EPOS2s are able to react to internal error by sending an Emergency message above the CANopen network, but if the danger is from the external ambient, the operator is called to react to it and drive the robotic arm into a safe position both for the operator and for the robotic arm itself. In order to permit the operator to react to the external dangers, an emergency push button is used. The button is a typical emergency-stop mushroom push button that remains activate once pushed. In addition also the movement of links 1, 2 and 3 is limited by some limit switches.

The emergency push button and the limit switches are connected to the PLC digital input module X20 DIF 371. The program which manages the emergency push button and the limit switches gives first the Halt command (after one of them have bee triggered), and after turns all the EPOS2s into the state "Switch On Disable" by performing the Transition $T9$ ("Disable voltage" command, Algorithm 9). The Halt command is given by setting the Controlword bit8 to 1, it doesn't matter how the other bits are set. With the Transition method, we give the command only to one Controlword ($T_{Controlword}$) and the program read only one Statusword ($T_{Statusword}$). Hence the reaction to emergencies results more efficiency and more fast.

```

1 if Emergency then
2   | Command = Halt ;
3 end
4 transition = T9 ;

```

Algorithm 9: Program which manage the Emergency push button.

One may say that the motor positions have been already limited with the object "Software Position Limit" ($607D_h$), but those just set the limitation into the robotic arm work space, but if the PLC lose the robotic arm control, the software doesn't make anything and an "Halt" command is required. For Example let us suppose that one of the six axes is doing a dangerous movement, in this case one limit switches will be pushed and all the EPOS2s stop the motion (the Halt command) and all the

EPOS2s LED turn to red which indicates the status of disable. The same happens if the operator push the emergency button.

All this is for improve the security, in fact, both the Emergency push button and the limit switches have the same effect, i.e. perform first the "Halt" command and after the "Disable voltage" command.

6.6 B&R Automation[®] package for Simulink[®]

Now we see how to get a program which will run in the B&R Automation[®] PLC starting from a Simulink[®] model. B&R Automation[®] provides a package for Simulink[®] in which some blocks are included. These blocks are used to create the program in C++ language from a Simulink model, but they also allow to create the Process Variables from the Simulink model. To see the most useful blocks provided by B&R Automation[®], let us create a simple Simulink model as the one shown in figure 6.25. Referring to the figure we can see six blocks:

- **B&R IN** and **B&R OUT**. These blocks are used to declare variables for Automation Studio, after we have created the program and we have imported it into AS, we can connect the variables to the PLC channels. IN stands for PLC input and OUT stands for PLC output. We can also declare them scope (if Local or Global) and them name by double clicking on them and setting the parameters (figure 6.24(a) and figure 6.24(b)).
- **B&R EXT IN** and **B&R EXT OUT**. These two blocks are used to convert input or output variables type (for example from INT to REAL and vice versa), in fact, all the variables in Simulink are REAL, but we can't use these variables in a C++ program because they will occupy too much space, so we convert it into a INT variables (for example for the arrays index we use an INT variable and not a REAL one). We can also use these blocks to convert the measure unit of the variables. For example in our case the measure unit of the motor rotation is the Step, but Simulink uses the radians, hence we may use this blocks to make the conversion. The setting of these variables is done by double clicking on the icons and compiling the parameters.
- **B&R PARAMETER**. This let us to change a Simulink gain via the PLC (Automation Studio).
- **B&R WORKSPACE VAR**. This block allow the program to use variables from Matlab Workspace. The variables from Matlab Workspace becomes a parameters for the PLC (Automation Studio).

The Global variables are not saved into the Global.var file in Automation Studio, but they will saved into a .txt file that has the same structure as a .var file, so we have just to copy and paste it into the Global.var file. This is due to avoid to overwrite the global variables already exist, we can delete this option when we make the configuration.

The model above will not runs if we don't make the configuration. This is done with the block B&R CONFIG (figure 6.26). This block must be always present at

Block Parameter: Input Block

Information
Block to generate a B&R Input variable.

Variable parameters

Variable Name

Variable Description

Variable Scope

Variable Data Type

Memory

Initial Value

Array Size

OK Cancel Help Apply

(a) B&R IN Block Parameters.

Block Parameter: Output Block

Information
Block to generate a B&R Output variable.

Variable parameters

Variable Name

Variable Description

Variable Scope

Variable Data Type

Memory

Initial Value

Array Size

OK Cancel Help Apply

(b) B&R OUT Block Parameters.

Figure 6.24: B&R Block Parameters.

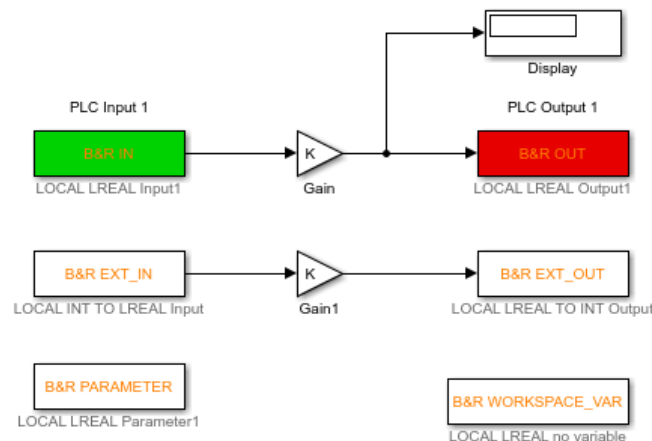


Figure 6.25: A simple Simulink model that helps us to understand some blocks of the B&R Automation[®] package for Simulink[®].



Figure 6.26: The configuration block. It must be always present (only once) in the Simulink model.

once in the Simulink model. When we click on it a window with the configuration parameters appears (from figure 6.27 to figure 6.30).

Referring to the figures we give an explanation about the configuration:

- **Model Configuration** (figure 6.27). Here we set the language of the program that we want to create from the Simulink model. We have to use C++ because of in Simulink model also Simscape[®] is used. We flag "Embedded Coder [ERT]" because this make a more slender and more efficient program. In Solver option we choose the Solver and we set the "Fixed-step size (fundamental sample time)" which must be equal to the time lapse of the Task Class in which we intend to place the program (this is an important passage).
- **Basic Setting** (figure 6.28). We intend to place the program into a task, so we flag "Task" in Target. After we put the address of the B&R project folder in which we want to place the program. We can already add the task to hardware by flagging the icon and choosing the Configuration, if we won't to do that we will make it after in Automation Studio environment, once the program will have been saved into the Logical View. Finally we choose the target Task Class, it is better not to change the Cyclic Task Class time. The "Fixed-step size (fundamental sample time)" must be the same as "Duration".
- **Advanced Settings** (figure 6.29). Starting from the bottom we flag the icon "Simscape Support" because our Simulink model uses it. We may flag the icon "Create Global Var-Files", if we do the Global variables are saved into the .var file in the Logical View and not in .txt file as explained above. The "Heap

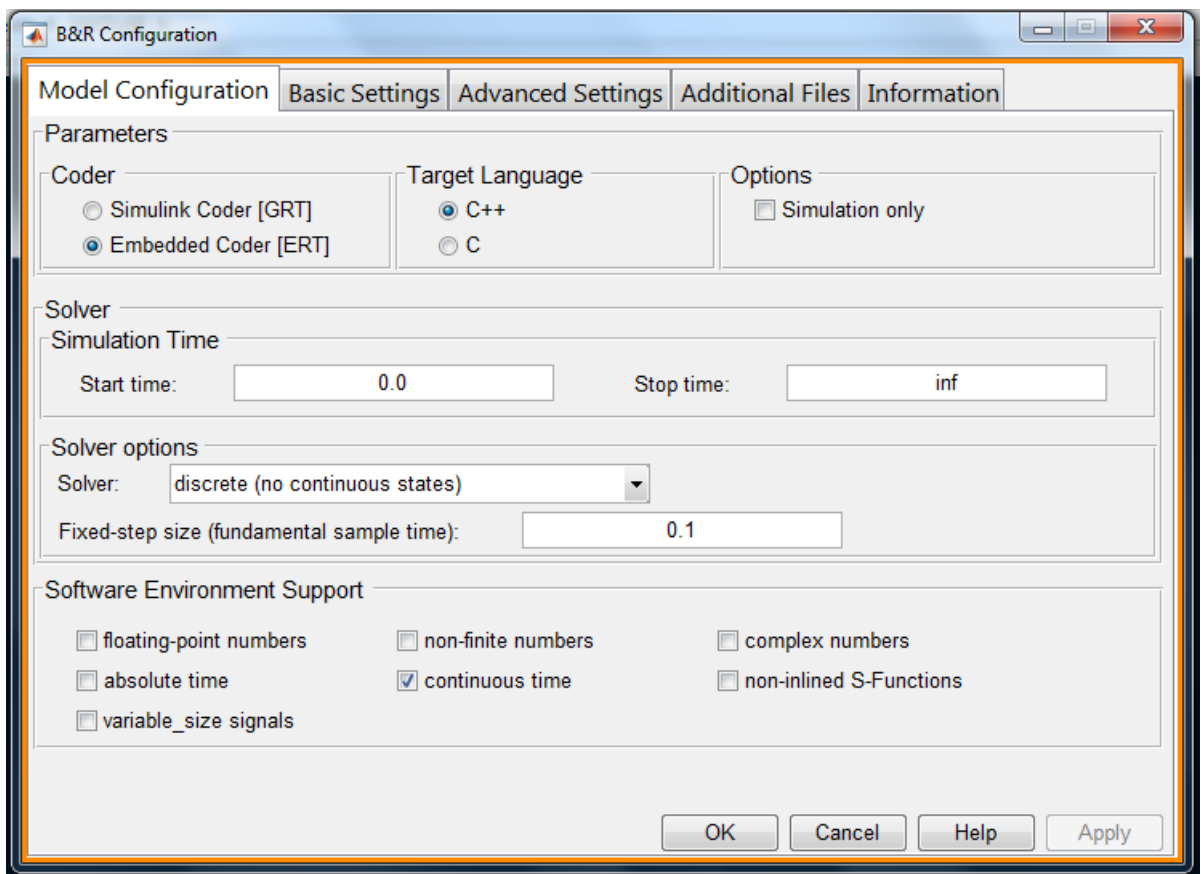


Figure 6.27: B&R CONFIG Model Configuration.

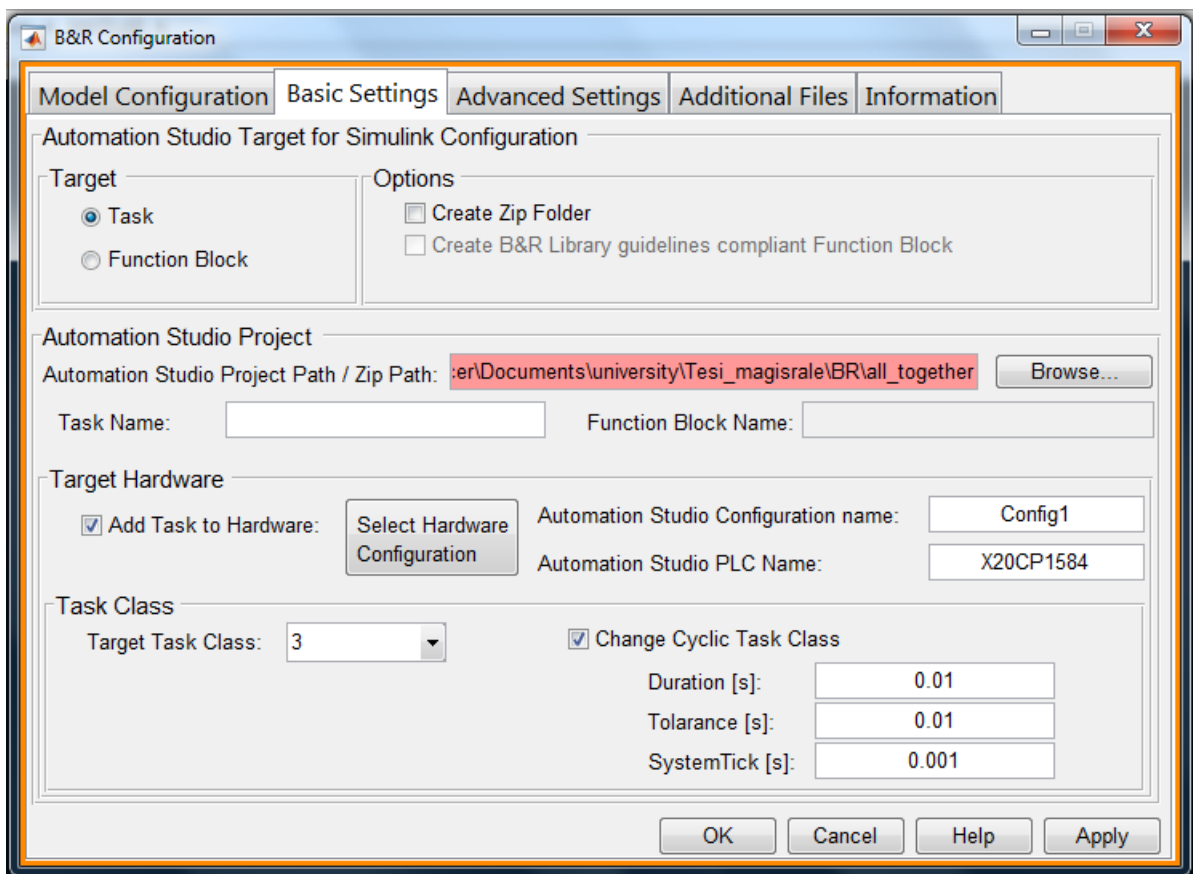


Figure 6.28: B&R CONFIG Basic Settings.

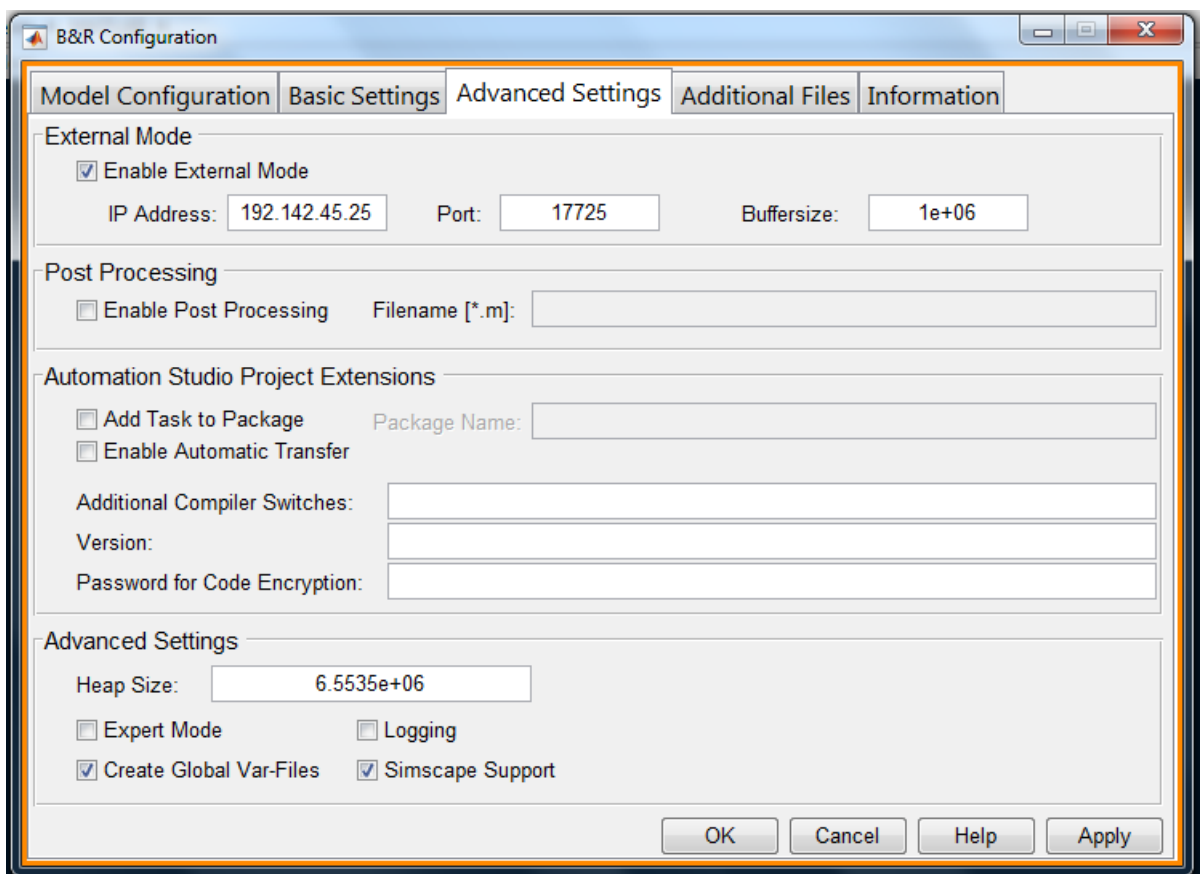


Figure 6.29: B&R CONFIG Advanced Settings.

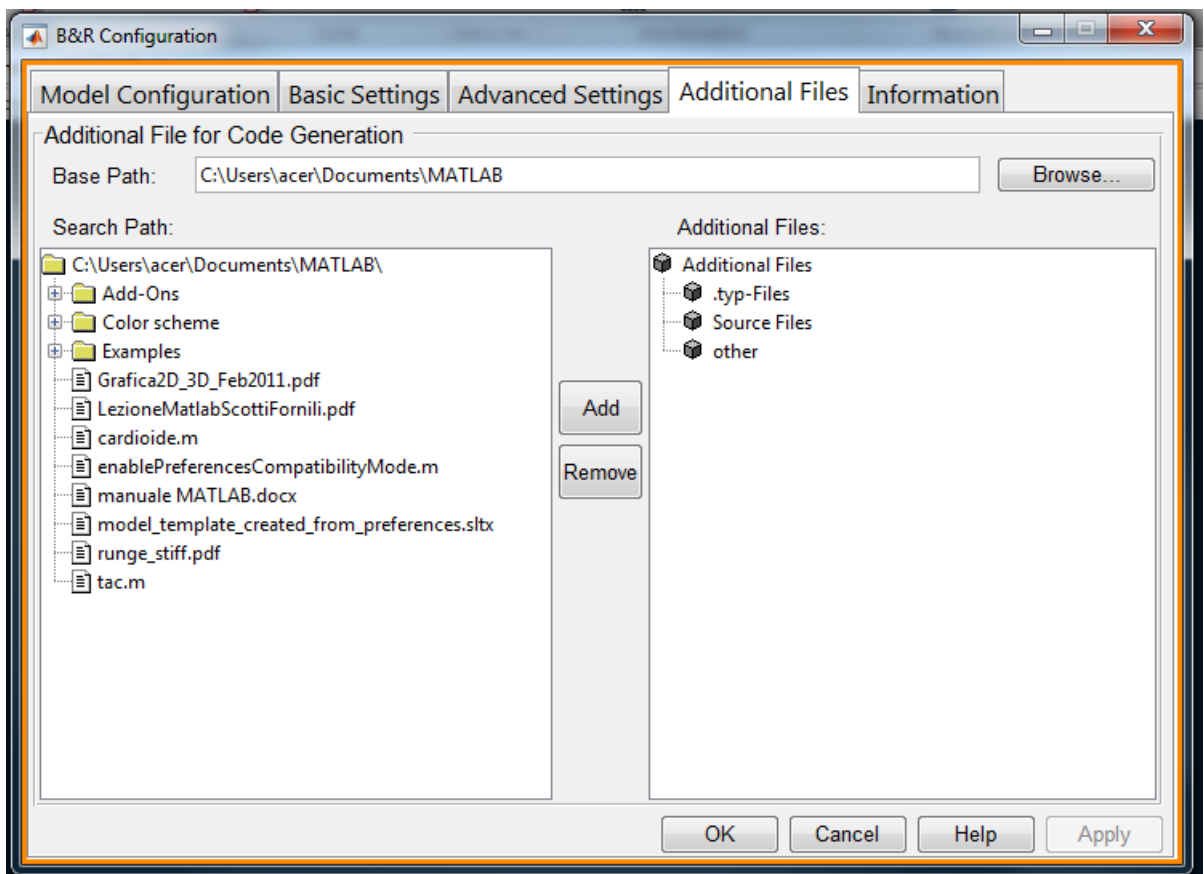


Figure 6.30: B&R CONFIG Additional Files.

Size" is the amount of memory that a C++ program will occupy, this value is always present as soon as we create a C++ program in Automation Studio, the default value is FFF_h that correspond to the value in the figure. If we decide to change this value we must change it both here and in AS. With "Enable External Mode" we can see in Simulink (after put the PLC IP-address) the variables value during the PLC is running. Note that this is not in real time, but it delayed also more the the Watch Window in AS, because AS have to convert the variables into a value which can be read by Simulink. The "Expert Mode" enables some configuration fields which are normally gray and which are not normally editable.

- **Additional Files** (figure 6.30). In addition to the ones already mentioned, there are other types of variables that the package is able to manage, these include structures. Simulink doesn't accept the structures. Hence we have to declare the structures in Automation Studio (in a .typ file), and after we can add the .typ file from AS, and the structures appear in Simulink like normal bus variables.

Finally in the Simulink window we click on the icon "Start" and the model becomes a program saved in the Logical View of Automation Studio (this process requires some time).

Chapter 7

Final Results

7.1 Discrete Motion

The Interpolated Position mode can't be implemented in the PLC. This is due to the fact that Interpolation Data Record (20C1_h) is a data with 64 bits, and the PLC isn't able to manage data of such length. Thus we are not able to implement the latched PVT algorithm in the EPOS2s. But we find a possible solution for this problem. It consists to create a function in the C++ program which implements the PVT algorithm (or any other interpolating function), this gives the right discrete values of position, velocity and acceleration which are necessary to hit all the desired points in the trajectory. The trajectory will result segmented, hence the joint motion will result discrete as well.

A segmented joint movement will be made with the Profile Position mode. As explained in 6.3.2, we map the Target Position, the Profile Velocity, the Profile Acceleration and the Profile Deceleration into the PDOs for each EPOS2 and finally we write on them the correct values of position, velocity and acceleration that we obtain from the interpolating function. The smallest joint angular motion which can be performed is given by the conversion from the position unit (Step) into the angular unit (radians or degree). If we indicate with q_{dis} the value of the smallest angular movement downstream the reduction gear, we obtain that:

$$\begin{aligned} 1q_{dis} &= \frac{2\pi}{1\text{Step}} \cdot (\text{Reduction Gear ratio}) \\ &= \frac{2\pi}{4 \cdot (\text{Encoder Pulse per Turn})} \cdot (\text{Reduction Gear ratio}) \end{aligned}$$

Instead, the Cartesian position (X_{dis}) is given by multiply q_{dis} by the length of the connected to the joint. Since the value of X_{dis} relative to the first joint depends on the configuration of the links 2 and 3, we give only the maximum value, which is obtained when the links 2 and 3 are completely extended. The values of q_{dis} and X_{dis} for all the first three joints¹ are illustrated in table 7.1.

This kind of motion has been tried with the fourth joint. The chosen positions was far from each of them, because this trial was made only to test if we are able

¹We indicate only the value for the first three joints because they are the joints which perform the trajectory. While the last three joints simulate the attitude of the chaser satellite.

Table 7.1: Values of the smallest angular movement for the first three joints and the values of the smallest Cartesian movements for the for the first three links.

Join and Link	Encoder pulse per turn	Reduction Gear ratio	Reference link length	Position Steps	q_{dis} [μrad]	X_{dis} [μm]
1	2048	126	1300	1032192	6.09	7.91
2	500	308	700	616000	10.2	7.14
3	6400	113	600	2892800	2.17	1.3

Table 7.2: The values of Target Position and Profile Velocity, performed by the fourth joint in order to try the segmented Profile Position mode.

	1	2	2	3	4	6	7	8	9	10
Target Position [deg]	0	34.7	-28.05	-65.45	2805	-37.4	9.35	56.1	93.5	0
Profile velocity [rpm]	100	400	100	400	100	400	100	400	100	400

to write on the objects Profile Velocity and Profile Acceleration during a motion. The motion is composed by a set of 10 different values of Profile Velocity and Target Position, and the Profile Acceleration was written to a value of $22959rpm/s^2$ at every iteration. The values of velocity and position, are shown in table 7.2. Position was tried both in absolute and relative reference.

An similar discrete motion was performed also with the Position mode, but it hasn't made the same smoothness than that one performed with the Profile Position mode.

7.2 Test results

Here we present some tests values. These tests are taken by moving the first three links both alone and together with the Profile Position mode and see how much is the Voltage and current consumes. From the current consume, we can find the value of the torque required for the motion. The torque is calculated by multiplying the current by the torque constant K_T reported in the motors data sheet (this method is just an approximation, but it is near to the reality). The torque at the downstream of the reduction gear is also calculated, by multiplying that value by the gear ratio:

$$\text{Motor torque} = K_T \cdot \text{Current}$$

$$\text{Joint torque} = (\text{Reduction Gear ratio}) \cdot (\text{Motor torque})$$

When we made the EPOS2 tuning with EPOS Studio, we found the value of the gains, and the value of the Profile Velocity, Profile Acceleration and Profile Deceleration in case of joints without the links. With these tests, we had also the chance to find the optimal values of Profile Velocity, Profile Acceleration and Profile Deceleration which satisfy both the overshoot and the smoothness of motion.

We first make some small motions (less than 10 degrees) and finally a larger motion was performed. The voltage and current consumption for a 10 degree movement

²This value was obtained with the tuning of the relative EPOS2, the link wasn't mounted.

are shown in table 7.3(a), while for a larger movement are shown in table 7.3(b). Many other tests was taken, but we report only these two because they are the first and last trials. The motion of the second link started from the vertical position (see figure 7.1)

Table 7.3: Voltage, Current and power consumption for two different movement of the first three links.

(a) 10 degree movement.

Joint number	Movement degree	Voltage [V]	Current [A]	Power [W]	Torque constant	Gear ratio [mNm/A]	Motor torque [mNm]	Joint torque [mNm]
1	10	24	0.09	2.16	36.9	136	3.32	418.44
2	10	48	0.1	4.8	84.9	308	8.49	2614.92
3	10	48	0.1	4.8	70.5	113	7.05	796.65

(b) Larger movement.

Joint number	Movement degree	Voltage [V]	Current [A]	Power [W]	Torque constant	Gear ratio [mNm/A]	Motor torque [mNm]	Joint torque [mNm]
1	45	24	0.12	2.88	36.9	126	4.42	557.93
2	35	48	0.17	8.16	84.9	308	14.43	4445.36
3	40	48	0.11	5.28	70.5	113	7.75	876.31

Table 7.4: Optimal values of Profile Velocity, Profile Acceleration and Profile Deceleration obtained with some trials at different values of them.

Motor number	Profile Velocity	Profile Acceleration	Profile Deceleration
1	100	200	200
2	500	300	300
3	275	300	3003

We illustrate the optimal value of the Profile Velocity, Profile Acceleration and Profile Deceleration in table 7.4. Note that these quantities, becomes the upper limits when we will make the discrete motion with the Profile Position mode, hence they have had to be set as the maximum values of velocity and acceleration, already in the interpolated function

Finally in figure 7.1, there is a sequence of the robotic arm configurations, which describes the motion shown in table 7.3(b): the joint 1, 2 and 3 motion was of 90, 40 and 45 degrees, respectively.

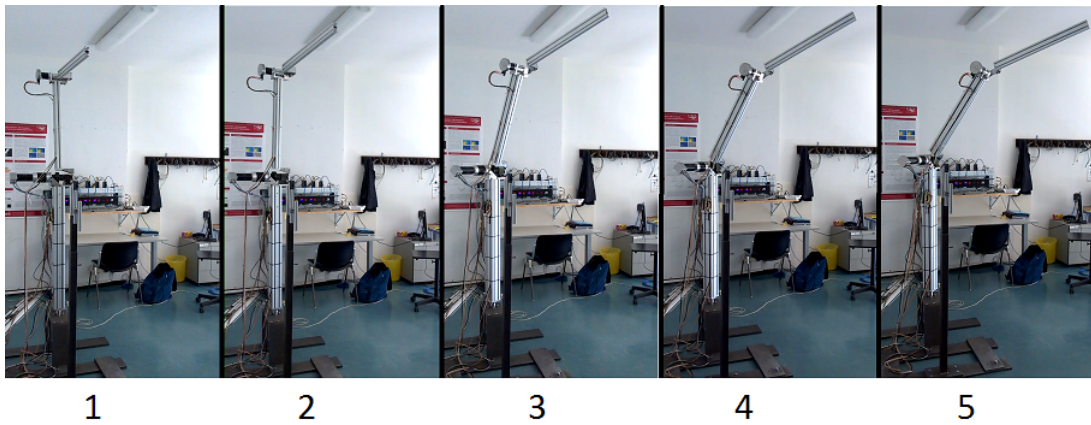


Figure 7.1: Sequence of robotic arm motion. Joint 1 performed 90 degrees, joint 2 performed 40 degrees and joint 3 performed 45 degrees,

Chapter 8

Conclusions

8.1 Comments

All the purposes of this thesis were reached. The supporting structure was made, and the software architecture is now complete and the robotic arm is able to perform some motions. Also some trials to verify if the software work were done. In addition we modify some default values we obtained with the tuning procedures.

Finally we resolves some problems which raise during the tests procedures. Some problems were referred to the fact that the tuning procedures were made without the links attached to the respective joints; but with the last tests we found the Profile Velocity, Profile Acceleration (and Deceleration) which best satisfy the smoothness of motion. Another problems was about the fact that the PLC is not able to manage a 64 – *bit* data and a continuous motion cannot be implemented. But with the aid of the discrete motion and an interpolating function, a segmented motion (with small segments) can be used.

8.2 Future works

Many work was done on this facility by Dr. Andrea Antonello and by me, but many other work is necessary to take the robotic arm into the fully operational state. The first thing to do is to make the EPOS2 tuning with the real inertia. To to this we can connect to the motors shaft a disc with the same inertia than the real. The real inertia is that in case of maximum value.

A software improvement is required. We made only a preliminary software architecture, but many others improvements are necessities in order to achieve the software optimization. For example we could create a more user-friendly software environment, with all the variables that the operator can manage.

We have also to implement a Simulink[®] model for the simulation of the micro-gravity environment into the PLC, in order to simulate the interactions between chaser and target. The interactions include both the relative motion and the forces and/or torques which the two vehicles exchange when a docking maneuver is occurring. In this way, when we make the velocity and the acceleration vectors which feed the Profile Velocity and the Profile Acceleration/Deceleration, we are sure that

trajectory is the same than that two vehicles in micro-gravity environments perform.

After these final works will be concluded, the facility is able to host the testing sensors and others instrumentation. For example a computer-based vision system could be mount on the last link, in order to simulate a real orbital docking maneuver with a fixed target. A computer vision system might be mounted in the laboratory as well, in this way we are able to implement an operational space control which monitors the end effector cartesian position despite the geometry. This is necessary if we want to make docking tests, which requires the knowledge of the relative position between chaser and target with high precision. But the manipulator is able to host position sensors which have to be tested, for example the sun sensor discussed in [2].

Bibliography

- [1] Alessandro Francesconi Ruggiero Carli Andrea Caron Andrea Antonello, Francesco Sansone. A novel approach to the simulation of on-orbit rendezvous and docking in a laboratory environment through the aid of an anthropomorphic robotic arm.
- [2] Andrea Antonello. *Design of a Robotic Arm for Laboratory Simulations of Spacecraft Proximity Navigation and Docking*. PhD Thesis, 2016.
- [3] BuR Automation. *TM210 - Working with Automation Studio V 2.3.0.3*. 2017.
- [4] John J. Craig. *Introduction to Robotics - Mechanics and Control*. 2005.
- [5] Il. D. Curtis. *Orbital Mechanics for Engineering Students*. Elsevier, 2010.
- [6] CAN in Automation. *CiA 301 - CANopen - CANopen application layer and communication profile*. February 2011.
- [7] CAN in Automation. *CiA 306 - CANopen - Electronic Data Sheet specifications*. February 2011.
- [8] CAN in Automation. *CiA Draft Standard Proposal 402 - CANopen - Device Profile Drive and Motion Control*. July 2002.
- [9] Maxon Motor. *EPOS2 24/5 Positioning Controllers - Hardware Reference*. May 2016.
- [10] Maxon Motor. *EPOS2 70/10 Positioning Controllers - Hardware Reference*. May 2016.
- [11] Maxon Motor. *EPOS2 Communication Guide V1.3*. November 2017.
- [12] Maxon Motor. *EPOS2 Positioning Controllers - Firmware Specification*. November 2017.
- [13] David Orin Roy Featherstone. *Robot dynamics: Equations and algorithms*.
- [14] M. Spong. *Robot dynamics and control*. Wiley, 1989.
- [15] K. R. Symon. *Mechanics*. Addison-Wesley, 1971.