UNIVERSITÀ DI PISA

Facoltà di Ingegneria

Laurea Specialistica in Ingegneria dell'Automazione

Tesi di laurea

Design, simulation and development of a decentralized control for a robotic manipulator

Candidate:

Giuseppe Di Napoli _____

Supervisors:

Prof. Carlo Alberto Avizzano _____

Ing. Emanuele Ruffaldi _____

Ing. Alessandro Filippeschi _____

Examiner:

Prof. Lorenzo Pollini _____

Sessione di Laurea del 26/11/2015
Archivio tesi Laurea Specialistica in Ingegneria dell'Automazione
Anno accademico 2015/2016

Consultazione consentita

Dedico questa lavoro a mia madre ed al mio defunto padre che mi hanno sostenuto sempre sino alla fine in questi lunghi anni pisani.

RINGRAZIAMENTI

Desidero ringraziare il Prof. Avizzano per l'opportunità di questa tesi e la pazienza avuta con me nel percorso che ha portato alla sua conclusione, Matteo Tanzini e Lorenzo Peppoloni per il supporto tecnico ed Alessandro Filippeschi per avermi sostenuto durante tutto l'arco della tesi e per avermi costretto sempre ad indagare i problemi con metodo scientifico.
Inoltre ringrazio mia sorella che in questi anni è stata sempre al mio fianco ed in generale la mia famiglia che mi hanno spinto sempre a dare quel qualcosa in più che mi ha permesso di superare i non pochi ostacoli avuti durante questo percorso.
In ultimo ringrazio anche Clara (la mia ragazza) per essermi stata sempre concretamente vicina seppur lontana e tutte le persone che negli anni si sono alternate al mio fianco.

ABSTRACT

The objectives of this thesis are to design, develop and test a decentralized control for the youBot platform.

This robot is composed by an omnidirectional mobile platform, on which a five-axis robot arm with a two-finger gripper is installed.

The actual performance of the youBot arm control is analyzed in terms of: jitter, system analysis responses under position and speed input, sampling time and torque characterization.

Then, a new hybrid motion/force decentralized control is presented, based on Newton-Eulero dynamics.

Finally, the control algorithm has been tested in simulation and on the real youBot arm. The obtained results have been analysed and discussed.

SOMMARIO

L'obiettivo di questa tesi è la progettazione, lo sviluppo e la valutazione di un controllo decentralizzato per il KUKA *youBot*.

Questo *robot* è composto da una piattaforma mobile omnidirezionale e da un braccio robotico a 5 DOF.

Inizialmente sono stati analizzati il *software* e l'*hardware* presenti sul manipolatore come anche le prestazioni in termini di: *jitter*, caratteristiche delle risposte del sistema alle sollecitazioni in posizione e velocità, tempo di campionamento ottimale e caratterizzazione della risposta in coppia.

Dopo di che è stato progettato un controllo decentralizzato ibrido forza/velocità basato sulla descrizione dinamica di Newton-Eulero.

L'algoritmo di controllo è stato poi testato in simulazione e sul manipolatore in esame. I risultati ottenuti sono stati analizzati e discussi.

INDEX OF CONTENTS

INDEX OF FIGURES

INDEX OF TABLES

1 INTRODUCTION

The inspiration for this thesis is KUKA program "Research & Education". In line with the slogan "enabling you to realize your potential and your ideas", KUKA aims to provide researchers, teachers and learners to develop their own applications with the products they need for their work (KUKA, 2013).

In this operational context, the PERCRO laboratory gave me the chance to work directly with the KUKA youBot. It is composed by an omnidirectional mobile platform on which a five-axis robot arm with a two-finger gripper is installed (KUKA, 2013).

In the first step the youBot arm's performance is analysed and the problems are verified and mentioned by the community. The main youBot arm's problems are the following:

1) The control system operates on a position loop with a frequency equal to $25/50\ Hz$.
2) The gravity compensation is absent from the control.
3) The system executes the commands individually not allowing to serialize, this means that if the end-effector has to go from A to B and then the operator sends the command to C, the control doesn't accept the command C if it hasn't reached B.

From the problems mentioned, the control is identified as the main cause of a bad performance.

In this operational context, I have decided to redesign through a decentralized control, suitable for the subsequent realization of haptic interfaces.

This thesis is organized in the following way:

- SECTION I: The state of the art is assessed.
- SECTION II: The existing hardware and software architectures are studied to detect their strengths and weaknesses.
  - In particular, regarding to the hardware, the focus was on features of:
    - Omnidirectional platform.
    - The youBot arm.
    - Engine's microcontrollers.
  - In particular, regarding to the software, the main focus was on:
    - Open-source computing platform (ROS).
    - The youBot driver libraries.
- SECTION III: The performance of the robotic arm is analysed.
  - In particular, the focus was on:
    - Evaluation of the system's response under specific inputs.
    - The jitter (Wikipedia, 2014), i.e. delay between command sent and engines response.
- SECTION V: The new controller is explained. This section consists of two subsections:
  - In the first, the new controller was designed in accordance with the requirement of decoupling through the Newton-Eulero algorithm.

    - In the second, unknown links' centre of mass (COM) have been identified through the least squares technique.
- SECTION VI: The new controller is tested in simulation and on the real youBot platform to verify that the required performance overlaps the real performance.
- SECTION VII: A summary of the obtained results is realized. In the same section, future development is proposed.

2 STATE OF THE ART

The bibliographic research comprehends four sections.

In the first section the decentralized and centralized control concepts are introduced. The difference between them is the cornerstone of this thesis.

In the second section the last torque control implemented for the KUKA youBot arm is shown.

In the third section the history of decentralized control, implemented for manipulator similar to the youBot arm, is shown.

In the fourth section the history of hybrid control, implemented for manipulator similar to the youBot arm, is shown.

2.1 Distributed control architecture

The centralized control (Figure 1) is an optimal solution to obtain the desired performance. Some of its characteristics are:

- It is not very flexible
- Hard-to-manage for:
  - Saturation.
  - Non – linearity.
- It's also not implementable for very large dimension systems.

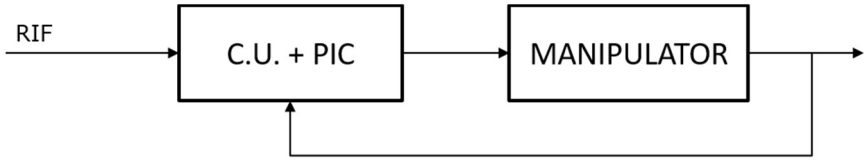

Figure 1: Scheme of centralized control

In comparison with the centralized control, the decentralized control has a simplified architecture (Figure 2), where each controller works independently improving the system's robustness.

This also allows a greater ease of installation and calibration, which results in a significant reduction of the cost.

The decentralized control doesn't get the optimum performance, compared to the centralized control, and therefore it doesn't apply to all the systems. Still, it can get close to such benefits by exploring the system's architecture, in order to fully use the decentralization control.

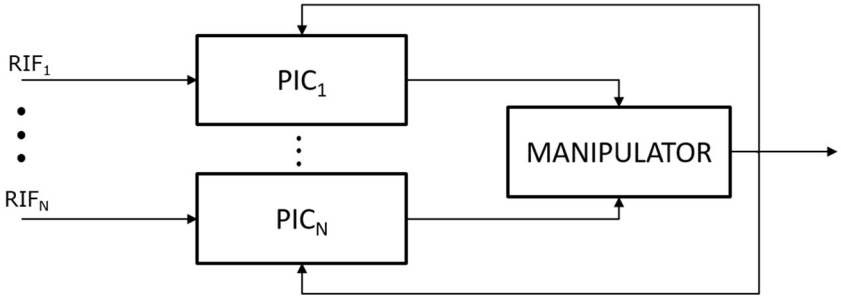

Figure 2: Scheme of decentralized control

A different option is the decentralized control with communication between controllers that combine the advantages of centralized and decentralized control. The difference is that it should be designed to not require excessive benefits of the communication system.

2.2 Torque Control of a KUKA youBot Arm

The controller described in (Keiser B., 2013) is the most advanced control existent at this time, implemented for the youBot arm.

It is an example of computed torque control application for serial manipulator.

Shashank Sharma, Gerhard K. Kraetzschmar, Christian Scheurer, and Rainer Bischoff developed a closed and unified solution for the inverse kinematics of the youBot which allows fast computation of joint positions for Cartesian end – effector positions (Sharma S., 2012). Using this approach, generated Cartesian trajectories are easily transformed to the corresponding joint space trajectories required by the torque controller, as long as all the positions are feasible. Timothy Jenkel, Richard Kelly and Paul Shepanski show an approach to object manipulation on the KUKA youBot (Jenkel T., 2013). Their system is able to detect objects autonomously in the environment, then pick them up and put them down using a simple user interface. In order to effectively control the youBot arm they make use of the existing ROS arm navigation stack and adapt it for the KUKA youBot. The stack allows to control the manipulator in Cartesian space based on the implemented inverse kinematic chain. Once objects get detected using a Microsoft Kinect attached to the arm, the youBot is guided to a position from where the objects are grasped using a set of overhead cameras. For grasping, they make use of the ROS object manipulation stack, which provides a framework for pick and place actions. In this project, the author is developing a torque controller which is designed for close tracking of trajectories and assume grasping is always optimal when attempted from directly above. In addition, the base is only controlled using the odometry messages generated by the KUKA youBot ROS stack. The object detection algorithm was developed by Paul Malmsten (Malmsten P., 2012).

2.2.1 Computed-torque control on the KUKA youBot

To evaluate the torque controller, the author looks at different trajectories. The first trajectory is the one which motivated the project, namely grasping an object by lowering the end – effector from a position directly above the object. Then it takes a much longer trajectory, which is used for grasping while driving; here the arm gets dragged parallel to the base, while lowering the manipulator and then lifting it back up. Finally, the author analyses tracking of a complicated trajectory in form of a circle where all the joints have to be moved simultaneously and the controller is operating near the joint position limits.

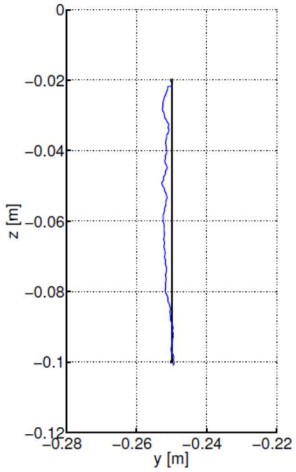

Figure 3: The actual end-effector position (blue) compared to the desired trajectory (black)

The (Figure 3) shows the actual end-effector trajectory driven by the torque controller, when attempting to grasp objects directly from above. The trajectory was generated using a maximum end – effector velocity of $0.05\ m/s$ and a maximum end – effector acceleration of $0.5\ m/s^2$. It is completed in $1.7\ s$.

The absolute error is the 2-norm of the errors in x, y, and z direction and is illustrated for the torque controller in (Figure 4).

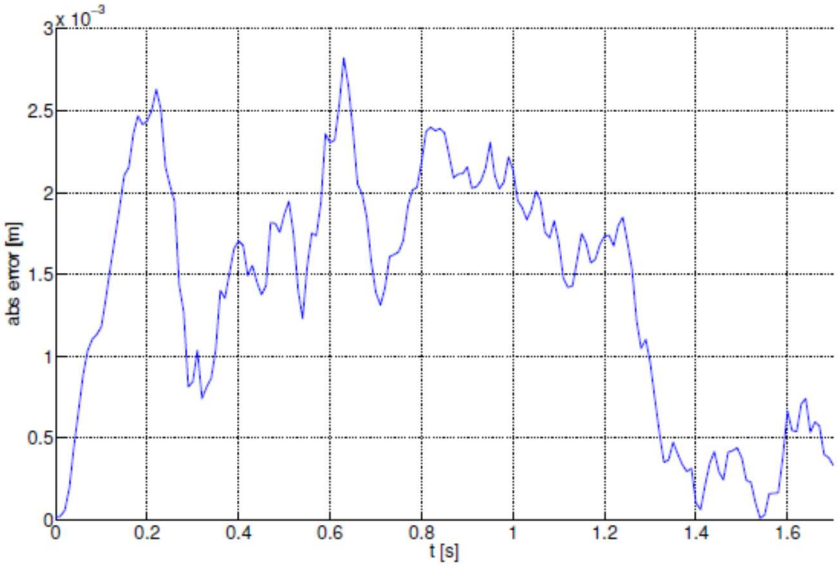

Figure 4: Absolute error during the grasping trajectory

Looking at a longer trajectory, as seen in (Figure 5), which was again generated with a maximum end-effector velocity of $0.05\ m/s$ and a maximum end-effector acceleration of $0.5\ m/s^2$, it is possible to see that much longer trajectories can still be tracked with good accuracy.

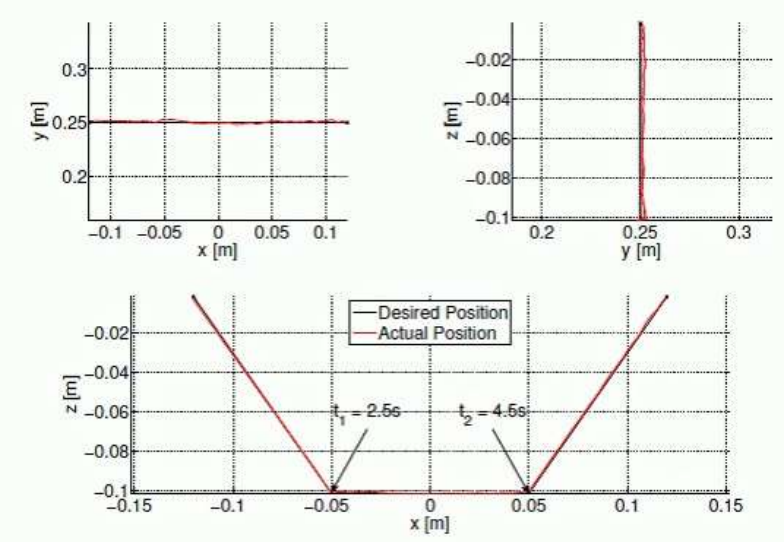

Figure 5: Tracking of a trajectory for grasp while driving

The (Figure 6) shows the absolute error in position tracking over the 7 s it took to complete the trajectory. The error peaks 4.2 mm with the average being 2.2 mm.

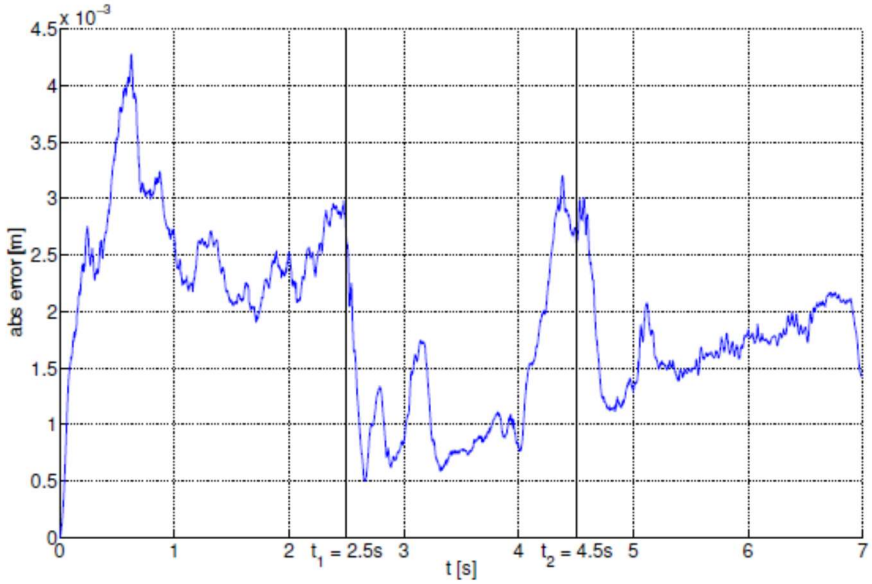

Figure 6: Absolute error while tracking the grasp and driving trajectory

So far the author only considered straight end-effector trajectories which can be used for various grasping tasks. While these paths are important, they are also quite simple. To really put the torque controller to test, a more complex trajectory was generated in the shape of a circle in Cartesian space. In order to follow a circle, all joints of the youBot have to be moved in both directions during the whole process and all joints are moving simultaneously. In addition, the torque controller is also operating near the joint position limits.

The (Figure 7) shows the desired trajectory together with the actual end-effector position of the process which took 20 s to complete.

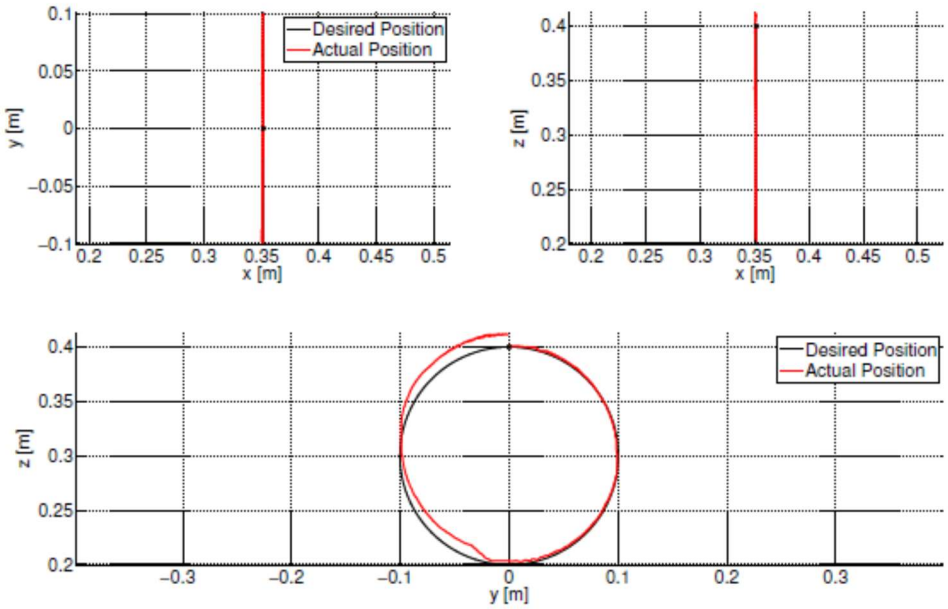

Figure 7: Tracking of a circular trajectory

The (Figure 8) shows the corresponding absolute error over the time period. During the first half of the trajectory, where the arm is lowered, tracking is accurate.

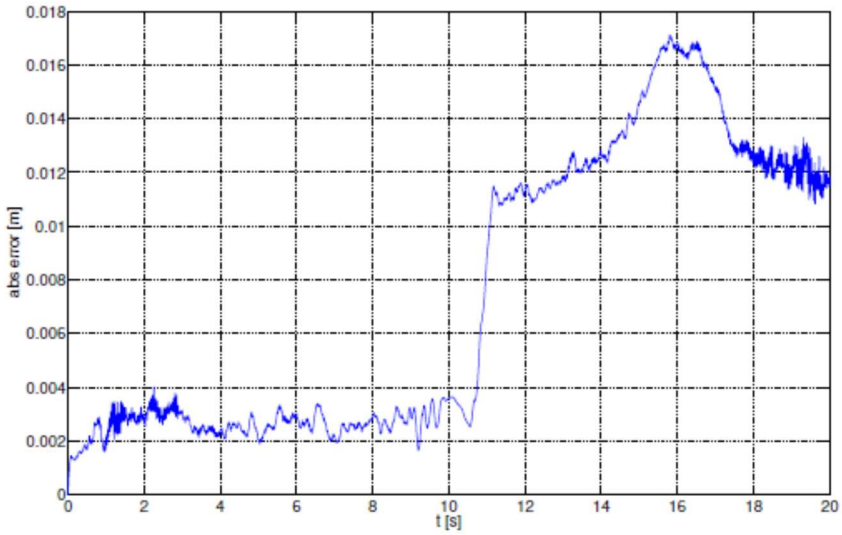

Figure 8: Absolute error while tracking a circle

As soon as the arm is lifted up again, an offset in z direction is introduced by the torque controller. The offset originates from physical limitations of the youBot manipulator and its inherent problems of lifting arm joints in certain configurations in the torque control mode. This problem is intensified by the self-locking joints of the youBot. At joint velocities close to $0\ m/s$, the joint stops completely and needs a high torque to get moved again. This effect is not included in the dynamical model of the youBot and has a negative consequence on the performance of the torque controller.

2.3 Decentralized control

The first industrial distributed systems are based on OSI standard for network protocols. It resulted in a hierarchic architecture which has its components grouped on four layers (Figure 9 (a)).

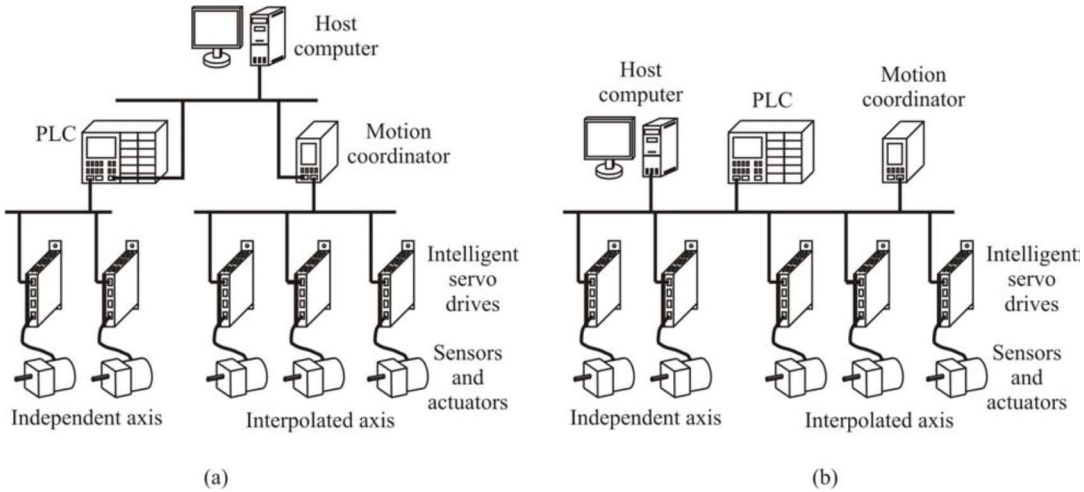

Figure 9: Hierarchic architecture based on layers (a) and architecture with communication network (b)

Each system's component receives commands from the devices of the superior layer and sends the command to inferior layers' devices (Lee K.C., 2006).

The highest layer is the one that ensures a graphic user interface; it manages:

- Downloading and uploading of programs and parameters.
- Monitoring of the stocks.
- Analysis of the working times (Lee K.C., 2006).

The next layer is the control process one, where the execution times are a critical aspect. The intelligent units from this layer have the following main tasks:

- Generation of the trajectory.
- Coordination of the axis.
- Synchronization of the axis (Lee K.C., 2006).

The Programmable Logic Controllers (PLC) or the microcontrollers from the axis control layer receive motion trajectory for the actuator and execute it (Lee K.C., 2006).

The last layer contains sensors and actuators and it is close to the hardware of the automation (Lee K.C., 2006).

This structure has a few disadvantages that make it hard to implement:

- It has several communication protocols with increase the automation price and makes it hard to debug;

- From the top of the structure to the bottom and reverse the data have to pass through several processor and the speed of transfer is decreased;
- The combination of horizontal traffic with vertical traffic is hard to be implemented for a real time multi-axis control.

An optimal solution to implement a distributed real time system is to connect, in a network with a single protocol, all the intelligent units from all the layers (Figure 9 (b)).

In a distributed control structure, the engine driver assumes more responsibility for the motion control like the hardware and software position limits supervision, the motor braking modes control and the safe low speed operation for the machine commissioning. Regarding this point, interpolated axes application and independent axes application must be distinguished (Jouve D., 2000).

Many robots and machine tools require interpolated motion control (several axes trajectories must be continuously coordinated). In this case, the axes trajectory must be calculated by the same processor at a high frequency rate in order to maintain the axes coordination (Cheng M. Y., 2007).

The distributed motion control architecture, suitable for multi-axis interpolation, is based on intelligent engine drivers; these perform the complete engine driver task, including position, speed and current loops plus the power conversion. The host motion controller performs the multi-axes trajectory calculation and sends the digital position set point values to each engine driver through the industrial network.

Most of the motion control applications in the automation area do not require interpolation between axes (the axes trajectories are independent). In this case a centralized trajectory calculation is no longer necessary and the trajectory calculation can be distributed inside the engine driver.

These solutions are much more effective than the centralized ones in terms of wiring cost reduction, setup and diagnostic facilities, thanks to the network communication. The driver's performances are also greatly improved because the current, speed and position digital server loops are all inside the engine driver (Puiu D., 2009).

2.4 Hybrid control

Combining force and position information into a single overall control scheme for end – effector motion in indefinite environment was first proposed by Raibert and Craig (Craig J. J., 1979). The control scheme made by them was called hybrid force/position control, currently called hybrid control. After Raibert and Craig's papers on hybrid control, Zhang and Paul (Zhang H., 1985) followed with a hybrid control scheme that changed from a Cartesian space form to one in joint space, using the same method for the separation of the position and force restrictions in Cartesian space for the motion of the manipulator/robot. In both cases, the advantage of position and force hybrid control is that the information are analysed independently to take advantage of well-known techniques that control the position and force, and are then combined (added-up) in the last round, when both were converted to values of joint space. More recently, An and Hollerbach (An C. H., 1987) have shown that certain methods of force control, including hybrid control are unstable. Zhang (Zhang H., 1989) showed that the hybrid control scheme can become unstable in certain configurations of manipulators using rotational joints.

Next it will be described the basic theory of hybrid control, and later will be described the developed application for hybrid control testing of a manipulator with four degrees of freedom.

To explain the basic concepts of the original hybrid control scheme, built by Raibert and Craig (Craig J. J., 1979), it will be used the simple interpretation of hybrid control, shown in (**Errore. L'origine riferimento non è stata trovata.**).

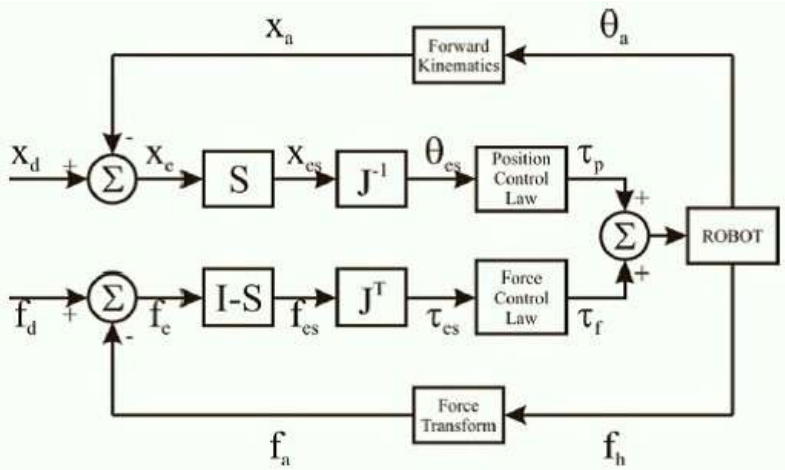

Figure 10: Hybrid control system proposed by Raibert and Craig (Craig J. J., 1979)

Control variables X in the upper loop of (Figure 10), represent Cartesian vectors of size $6\ x\ 1$ for positioning (the positioning term that will be used, will contain the position and orientation of the manipulator) and orientation of the manipulator, omitting the speed and acceleration vectors.

The control loop variables F in the bottom of (Figure 10), is the Cartesian vector of size $6\ x\ 1$ for force (the force term that will be used, will contain the force and moment of the manipulator) and moment of the manipulator.

Position and torque values of the manipulator joints are represented by θ and τ vectors of $n\ x\ 1$ size, where n is the number of joints of the manipulator. Character blocks of control scheme inFigure 10) represent a matrix of appropriate size to maintain constant size vectors. The other blocks are called by the generic terms of control blocks that varies depending on the method chosen in developing the scheme.

The most recently application of this technique is explain in (Gal A., 2011).

3 HARDWARE AND SOFTWARE

In this work I have used the KUKA youBot. This robot is composed by an omnidirectional mobile platform on which a five-axis robot arm with a two-finger gripper is installed (KUKA, 2013).

The single manipulator joints are managed by microcontrollers, and the control architecture is shown in (Figure 11).

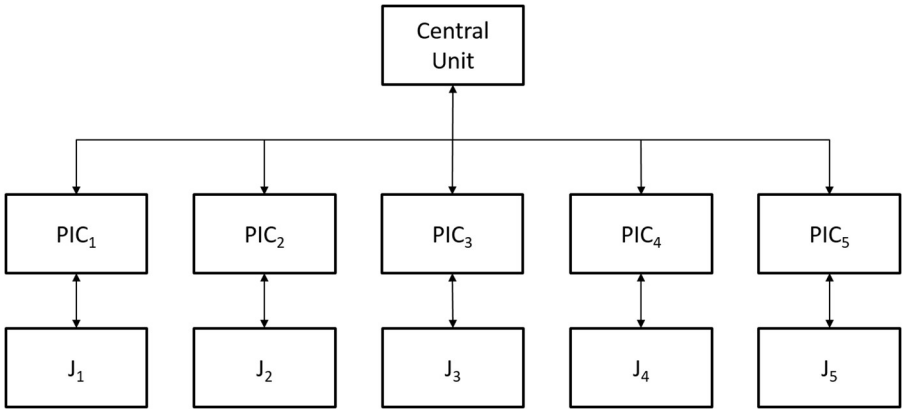

Figure 11: Centralized youBot arm control

In this control (Figure 11) there is a central unit (PC), that generated based on the control algorithm, speed, position or current set-point. It has been sent in PDO mode, from the C.U. to the microcontrollers that, according to the set-point type, manage it to generate the motor current/PWM command.

Then the microcontroller carries out the output reading, i.e. the measurements of torque, speed, current and position generate at the next step, the new set-point.

In the initial situation, the KUKA youBot is managed by ROS through the youbot_driver library.

The problems, as mentioned in the introduction, are related to the operation frequency, gravity compensation and non-implemented step by step control.

Regarding to the hardware part, i. e. the physical component of the youBot manipulator, I have focused on:

- The KUKA youBot mobile platform designed to enlarge the robotic arm's operational space (Figure 12).
- A manipulator "youBot arm" capable of carrying out pick and place operations with postures assigned (Figure 13).
- The MIC's (TMCM – 1632), provided by Trinamic, designed to set the parameter, to send the command and to read the system's responses, of the singular youBot manipulator's joints.

Regarding to the software part, i. e. any set of machine-readable instructions that directs a computer's processor to perform specific operations, I have focused on:

- The ROS architecture and the ROS function (ROS is acronym of Robot Operating System).
- The command chain, i.e. the command steps from sending and its implementation.

- The package youbot_driver, in particular the youbot library.

3.1 The omnidirectional platform

The omnidirectional holonomic platform (Figure 12) is based on a steel structure which is robust enough to carry a payload of 20 kg. It is driven by four KUKA omni-wheels, which are based on the mechanic principle. The steel structure is covered by a plastic covering, which can be demounted to provide access to the structure and mounting holes; these can be used to add additional sensors, e.g., laser scanners (Bischoff R., 2011).

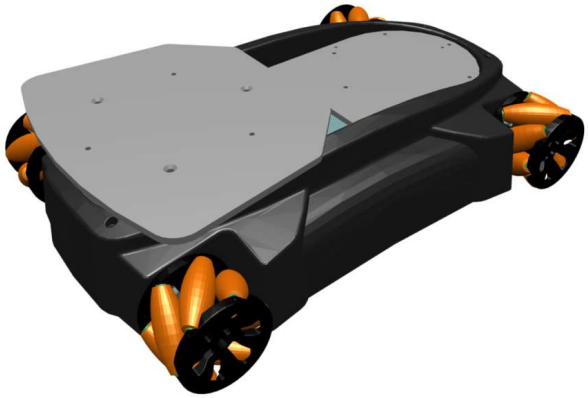

Figure 12: YouBot omnidirectional platform

3.2 The YouBot Arm

The arm of the KUKA youBot (Figure 13) consists of five rotatory joints and a two –finger gripper as an end effector. It reaches a height of 655 mm, has a weight of 6.3 kg and is designed to carry a payload of up to 0.5 kg in weight.

The rotatory joints of the arm rotate around two different axes. Joints two, three, and four are rotating in parallel around one axis, and joints one and five in parallel around the other in case the arm is pointing straight up.

The maximum rotation speed of a joint is 1.57 rad/s and the end-effector can be opened 11.5 mm per finger (Bischoff R., 2011).

Figure 13: The YouBot Arm

3.3 The microcontrollers

The microcontrollers (Trinamic, 2011) are used to manage the single joints. In them, the low-level control is implemented and handled based on the command, sent by the Central Unit.

Furthermore the microcontrollers send measurements of angular position, angular velocity, motor's current and motor's torque to the C.U.

The TMCM – 1632 is a highly integrated single axis BLDC servo controller module with an EtherCAT™ interface (E – Bus). The module has been designed in order to be plugged onto a baseboard. It offers two E – Bus ports for daisy-chaining several modules and connect them to an EtherCAT™ master (e.g. embedded PC) for multi-axes solutions. The module offers hall sensor and incremental encoder (a/b/n) inputs.

The trinamic microcontrellers are EtherCAT™ slave devices. The whole communication with an EtherCAT™ master follows a strict master-slave-relationship. Via the TMCL™ mailbox, motor parameters are written and/or read using TRINAMIC's TMCL™ protocol.

3.3.1 SyncManager

The SyncManager enables consistent and secure data exchange between the EtherCAT™ master and the local application, and it generates interrupts to inform both sides of changes. The SyncManager is configured by the EtherCAT master. The communication direction is configurable, as well as the communication mode (see par. 3. 3.3.1.1 and par. 3.3.1.2). The SyncManager uses a buffer, located in the memory area for exchanging data. Access to this buffer is controlled by the hardware of the SyncManager.

A buffer has to be accessed beginning with the start address, otherwise the access is denied. After accessing the start address, the whole buffer can be accessed, even the start address again, either as a whole or in several strokes. A buffer access finishes by accessing the end address, the buffer state changes afterwards. The end address cannot be accessed twice inside a frame. Two communication modes are supported by SyncManagers, the buffered mode and the mailbox mode.

3.3.1.1 Buffered mode

The buffered mode allows both sides, EtherCAT™ master and local application, to access the communication buffer at any time. The consumer gets always the latest consistent buffer which was written by the producer, and the producer can always update the content of the buffer. The buffered mode is used for cyclic process data.

Data transfer between EtherCAT™ master (PC etc.) and slave (TMCM – 1632) is done using the dual port memory of the ET1200 EtherCAT-IC on the slave. The buffered mode allows writing and reading data simultaneously without interference. If the buffer is written faster than it is read out, old data will be dropped. The buffered mode is also known as 3-buffermode (Figure 14):

- One buffer of the three buffers is allocated to the producer (for writing).
- Another buffer to the consumer (for reading).
- And the third buffer keeps the last consistently written data of the producer.

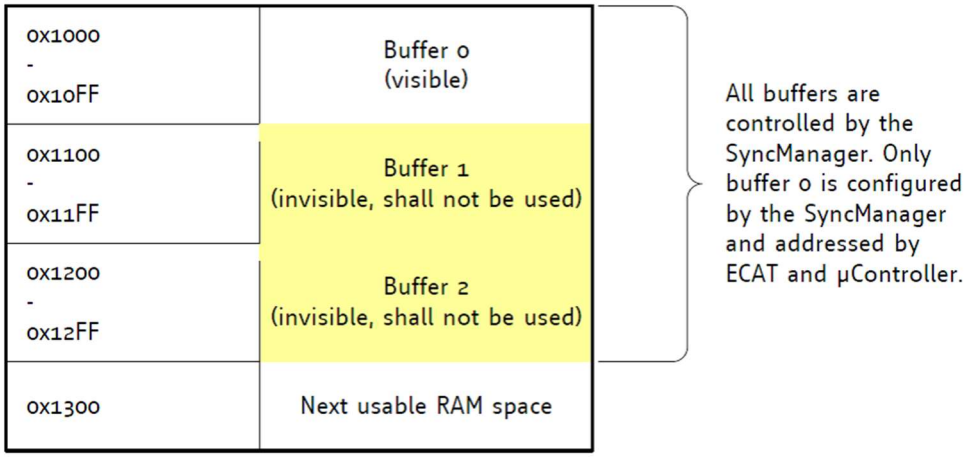

Figure 14: SyncManager buffer allocation

In (Figure 15) is an example to demonstrate a configuration with start address $0\ x\ 1000$ and Length $0\ x\ 100$. The other buffers shall not be read or written. Access to the buffer is always directed to addresses in the range of buffer 0.

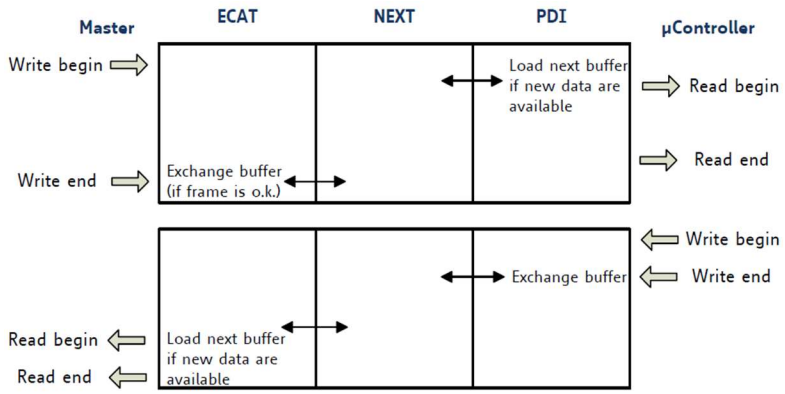

Figure 15: SyncManager buffered mode interaction

3.3.1.2 Mailbox mode

The mailbox mode implements a handshake mechanism for data exchange, so that no data will be lost. Each side, EtherCAT™ master or local application will get access to the buffer only after the other side has finished its access. The mailbox mode only allows alternating reading and writing. This assures all data that from the producer reaches the consumer. The mailbox mode uses just one buffer of the configured size. At first, after initialization/activation, the buffer (mailbox, MBX) is writeable. Once it is written completely, write access is blocked, and the buffer can be read out by the other side. After it was completely read out, it can be written again. The time it takes to read or write the mailbox does not matter. The mailbox mode is used for the application layer protocol. Via the mailbox, the motor parameters of the TMCM – 1632 can be written/read using the TMCL™ protocol (Figure 16).

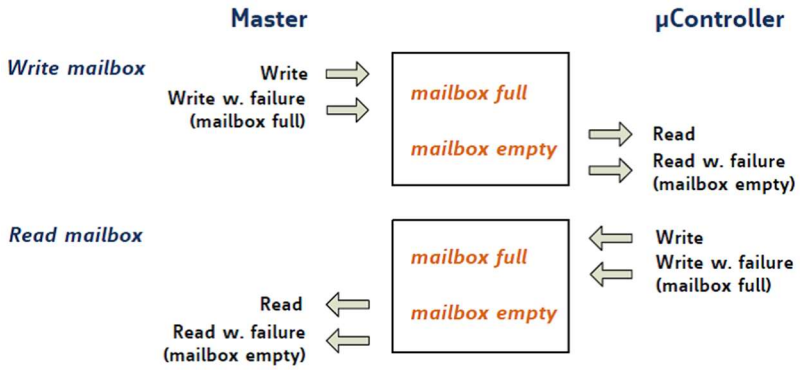

Figure 16: SyncManager mailbox interaction

3.3.2 EtherCAT™ slave state machine

The EtherCAT™ slave state machine has four states, which are shown in (Figure 17). After power ON the slave state machine is in the Init state. In this situation mailbox and process data communication is impossible. The EtherCAT™ master initializes the SyncManager channels 0 and 1 for the communication via mailbox.

While changeover from Init state to Pre-Operational state the EtherCAT™ slave checks the correct initialization of the mailbox. Afterwards mailbox communication is possible. Now, in the Pre – Operational state the master initializes the SyncManager channels for the process data and the FMMU channels. Furthermore adjustments are sent, which differ from the default values.

While changeover from Pre-Operational state to Safe-Operational state, the EtherCAT™ slave checks the correct initialization of the SyncManager channels for the process data as well as the adjustments for the Distributed Clocks. Before accepting the change of state, the EtherCAT™ slave copies actual input data into the accordant DP-RAM array of the EtherCAT™ slave controller. In the Safe-Operational state mailbox and process data communication are possible, but the slave holds its outputs in a safe situation and actualizes the input data periodically.

Before the EtherCAT™ slave changes the state to Operational, it has to transfer valid output data. In the Operational state the EtherCAT™ slave copies the output data from the EtherCAT™ master to its outputs. Process data communication and mailbox communication are possible now.

The Bootstrap state is only used for updating the firmware. This state is reachable from the Init state. During Bootstrap state mailbox communication is available over File-Access over EtherCAT. Beyond this mailbox communication or process data communication is not possible.

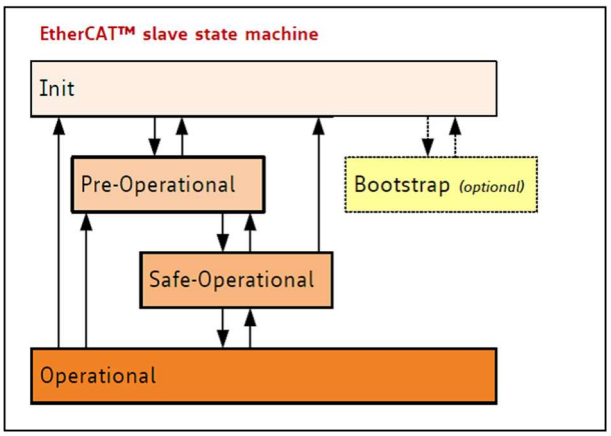

Figure 17: EtherCAT™ slave state machine

In (Table 1) there is a summary of the main mailbox action

State	Service
Init	▪ No communication on application layer ▪ Master has access to the DL-information registers
Init to Pre-Operational	▪ Master configures registers, at least: o DL address register o SyncManager channels for mailbox communication ▪ Master initializes Distributed Clock synchronization ▪ Master requests Pre-Operational state o Master sets AL control register ▪ Wait for AL status register confirmation
Pre-Operational	▪ Mailbox communication on the application layer ▪ No process data communication
Pre-Operational to Safe-Operational	▪ Master configures parameters using the mailbox: o e.g., process data mapping ▪ Master configures DL Register: o SyncManager channels for process data communication o FMMU channels ▪ Master requests Safe-Operational state ▪ Wait for AL Status register confirmation
Safe-Operational	▪ Mailbox communication on the application layer ▪ Process data communication, but only inputs are evaluated. Outputs remain in safe state
Safe-Operational to Operational	▪ Master sends valid outputs ▪ Master requests Operational state (AL Control/Status) ▪ Wait for AL Status register confirmation
Operational	▪ Inputs and outputs are valid
Bootstrap	▪ Recommended if firmware updates are necessary ▪ State changes only from and to Init ▪ No Process Data communication ▪ Mailbox communication on application layer, only FoE protocol available (possibly limited file range)

Table 1: Main mailbox action summary

3.3.3 EtherCAT™ Firmware update

For firmware updates the EtherCAT™ state machine of the slave has to be switched to Bootstrap state. The file access over EtherCAT™ protocol (FoE) is used.

The two mailboxes for data transfers have the following parameters:

- Data output buffer: Start – address: 4096, length: 268 byte
- Data input buffer: Start – address: 4364, length: 32 byte

3.3.4 Process data

In standard configuration for data transfer the following (Table 2 – Table 3) buffers are used (slave view):

Start address	End address	Data type	Data value/contents
$0\ x\ 1000$	$0\ x\ 1003$	SIGNED 32	Reference o Position o Velocity o PWM o Current
$0\ x\ 1004$	$0\ x\ 1005$	UNSIGNED 8	Controller Mode o Motor stop (0) o Positioning Mode (1) o Velocity Mode (2) o No more action (3) o Set position to reference (4) o PWM Mode (5) o Current Mode (6) o Initialize (7)

Table 2: Data output buffer/EtherCAT™ master → slave data transfer

Start address	End address	Data type	Data value/contents
$0\ x\ 1078$	$0\ x\ 107B$	SIGNED 32	Actual position (32 – Bit up – down counter)
$0\ x\ 107C$	$0\ x\ 107F$	SIGNED 32	Actual current [mA]
$0\ x\ 1080$	$0\ x\ 1083$	SIGNED 32	Actual velocity [rpm] (motor axis)
$0\ x\ 1084$	$0\ x\ 1087$	UNSIGNED 32	Error flags: o Overcurrent bit 0 o Undervoltage bit 1 o Overvoltage bit 2 o Overtemperature bit 3 o Motor halted bit 4 o Hall error flag bit 5 a 7 o PWM mode active bit 8 o Velocity mode active bit 9 o Position mode active bit 10 o Torque mode active bit 11 o Position end flags bit 14 o Module Initialized bit 15 o EtherCAT timeout flag bit 16 o I2t exceeded flag bit 17
$0\ x\ 1088$	$0\ x\ 108B$	SIGNED 32	Actual PWM

Table 3: Data input buffer/EtherCAT slave → master data transfer

3.3.5 TMCL™ mailbox

The TMCM – 1632 EtherCAT™ slave modules support the TMCL™ protocol in direct mode. The communication follows a strict master-slave-relationship. Via the TMCL™ mailbox motor – parameters can be read and/or written.

3.3.5.1 Binary command format

Every command has a mnemonic and a binary representation. When commands are sent from a host to a module, the binary format has to be used. Every command consists of a one-byte command field, a one-byte type field, a one-byte motor/bank field and a four-byte value field. So the binary representation of a command always has seven bytes (Table 4 – Table 5).

Bytes	Meaning
1	Module address 0 → drive 1 → gripper
1	Command Number
1	Type Number
1	Motor or bank number
4	Value (MSB first)

Table 4: Transmit an 8-byte command

Bytes	Meaning
1	Reply address
1	Module address
1	Status
1	Command Number
4	Value (MSB first)

Table 5: Receive an 8-byte reply

In (Table 6) is the meaning of the possible receiving values

Code	Meaning
100	Successfully executed
2	Invalid command
3	Wrong type
4	Invalid value
5	Configuration EEPROM locked
6	Command not available
8	Parameter is password protected

Table 6: Status code and meaning

3.3.5.2 Behaviour after interrupted EtherCat™ communication

In operational mode, as well as in mailbox mode, every controller module checks the available communication with the EtherCAT™ master. Therefore, the time since the last master command is measured. If the last command lasted more than the timeout defined by global parameter 90, the

module detects that either the operational mode is left, or the EtherCAT™ cable has been unplugged. Thereby the following three situations can occur:

- The communication is interrupted before the module is initialized. Thereby the module does nothing and stays in uninitialized state.
- The communication is interrupted after module initialization. Then the positioning mode is activated and the actual position is used as absolute target position and will be hold.
- The communication is interrupted during module initialization. Then the module initialization is performed and afterwards the module switches to positioning mode and holds the actual position.

3.3.6 TMCL™ Command overview

In this section a short overview of the TMCL™ commands is given.

3.3.6.1 Motion commands

These commands control the motion of the motor (Table 7). They are the most important commands and can be used in direct mode or in stand-alone mode.

Mnemonic	Command Number	Description
ROR	1	Rotate Right
ROL	2	Rotate Left
MST	3	Motor Stop
MVP	4	Move to Position

Table 7: Motion Commands

3.3.6.2 Parameter commands

These commands are used to set, read and store axis parameters or global parameters (Table 8). Axis parameters can be set independently for the axis, whereas global parameters control the behaviour of the module itself. These commands can also be used in direct mode and in stand-alone mode.

Mnemonic	Command Number	Description
SAP	5	Set axis parameter
GAP	6	Get axis parameter
STAP	7	Store axis parameter into EEPROM
RSAP	8	Restore axis parameter from EEPROM
SGP	9	Set global parameter
GGP	10	Get global parameter
STGP	11	Store global parameter into EEPROM
RSGP	12	Restore global parameter into EEPROM

Table 8: Parameter Commands

3.3.6.3 List of commands

The following commands (Table 9) are currently supported:

Mnemonic	Number	Parameter	Description
ROR	1	Motor Number, Velocity	Rotate Right with specified velocity
ROL	2	Motor Number, Velocity	Rotate Left with specified velocity
MST	3	Motor Number	Stop motor movement
MVP	4	ABS/REL, Motor Number, Position/Offset	Move to position (absolute or relative)
SAP	5	Parameter, Motor Number, Value	Set axis parameter
GAP	6	Parameter, Motor Number	Get axis parameter
STAP	7	Parameter, Motor Number	Store axis parameter permanently
RSAP	8	Parameter, Motor Number	Restore axis parameter

Table 9: Command List

3.3.7 Commands

The module specific commands are explained in more detail on the following pages. They are listed according to their command number.

3.3.7.1 ROR (rotate right)

With this command (Figure 18) the motor will be instructed to rotate with a specified velocity in right direction (increasing the position counter).

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
1	(don't care)	0	<velocity> 0... 2047

Reply in direct mode:

STATUS	COMMAND	VALUE
100 – OK	1	(don't care)

Figure 18: ROR Binary and reply representation

3.3.7.2 ROL (rotate left)

With this command (Figure 19) the motor will be instructed to rotate with a specified velocity (opposite direction compared to ROR, decreasing the position counter).

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
2	(don't care)	0	<velocity> 0... 2047

Reply in direct mode:

STATUS	COMMAND	VALUE
100 – OK	2	(don't care)

Figure 19: ROL Binary and reply representation

3.3.7.3 MST (motor stop)

With this command (Figure 20) the motor will be instructed to stop with deceleration ramp (soft stop).

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
3	(don't care)	0	(don't care)

Reply in direct mode:

STATUS	COMMAND	VALUE
100 – OK	3	(don't care)

Figure 20: MST Binary and reply representation

3.3.7.4 MVP (move to position)

With this command (Figure 21) the motor will be instructed to move to a specified relative or absolute position. It will use the acceleration/deceleration ramp and the max target velocity programmed into the unit. This command is non-blocking (like all commands) that is, a reply will be sent immediately after command interpretation and initialization of the motion controller. Further commands may follow without waiting for the motor reaching its end position. The maximum target velocity and acceleration are defined by axis parameters 4 and 5.

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
4	0 ABS – absolute	0	<position>
	1 REL – relative	0	<offset>

Reply in direct mode:

STATUS	COMMAND	VALUE
100 – OK	4	(don't care)

Figure 21: MVP Binary and reply representation

3.3.7.5 SAP (set axis parameter)

With this command (Figure 22) most of the motion control parameters of the module can be specified. The settings will be stored in SRAM and therefore are volatile. That is, information will be lost after power off. Please use command STAP (store axis parameter) in order to store any setting permanently.

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
5	<parameter number>	0	<value>

Reply in direct mode:

STATUS	COMMAND	VALUE
100 – OK	5	(don't care)

Figure 22: SAP Binary and reply representation

3.3.7.6 GAP (get axis parameter)

Most parameters of the TMCM – 1632 can be adjusted individually (Figure 23). With these parameters they can be read out (the value read is only output).

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
6	<parameter number>	0	(don't care)

Reply in direct mode:

STATUS	COMMAND	VALUE
100 – OK	6	(don't care)

Figure 23: GAP Binary and reply representation

3.3.7.7 STAP (store axis parameter)

An axis parameter previously set with a Set Axis Parameter command (SAP) will be stored permanent (Figure 24). Most parameters are automatically restored after power up.

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
7	<parameter number>	0	(don't care)*

*** The value operand of this function has no effect. Instead, the currently used value (e.g. selected by SAP) is saved.**

Reply in direct mode:

STATUS	COMMAND	VALUE
100 – OK	7	(don't care)

Figure 24: STAP Binary and reply representation

3.3.7.7 RSAP (restore axis parameter)

For all configuration related axis parameters non – volatile memory locations are provided (Figure 25). By default, most parameters are automatically restored after power up.

A single parameter that has been changed before can be reset by this instruction also.

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
8	<parameter number>	0	(don't care)

Reply in direct mode:

STATUS	COMMAND	VALUE
100 – OK	8	(don't care)

Figure 25: RSAP Binary and reply representation

3.3.8 Protected axis parameters

Some axis parameters are write-protected to prevent the normal user to set values whose changes could damage the mechanics of the robot.

To change these axis parameters, the manufacturer can use parameter 248 with the right password to approve all axis parameters. This activation can be reversed by using parameter 248 with a wrong password or reboot the axis controller. If a user tries to set a protected parameter the reply status of the TMCL command is “REPLY_WRITE_PROTECTED”.

3.3.8.1 SGP (set global parameter)

Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in "banks" to allow a larger total number for future products. Currently, only bank 0 and 1 are used for global parameters, and only bank 2 is intended to be used for user variables. Parameter 0 to 15 of bank 2 are password protected. Parameter 16 to 55 can be used for user variables without password protection (Figure 26).

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
9	<parameter number>	<bank number>	<value>

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Figure 26: SGP Binary and reply representation

3.3.8.2 GGP (get global parameter)

All global parameters can be read with this function (Figure 27).

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
10	<parameter number>	<bank number>	(don't care)

Reply in direct mode:

STATUS	VALUE
100 – OK	<value>

Figure 27: GGP Binary and reply representation

3.3.8.3 STGP (store global parameter)

Some global parameters are located in RAM memory, so modifications are lost at power down. This instruction copies a value from its RAM location to the configuration EEPROM and enables permanent storing (Figure 28). Most parameters are automatically restored after power up.

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
11	<parameter number>	<bank number>	(don't care)

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Figure 28: STGP Binary and reply representation

3.3.8.4 RSGP (restore global parameter)

This instruction copies a value from the configuration EEPROM to its RAM location and recovers the permanently stored value of a RAM-located parameter. Most parameters are automatically restored after power up (Figure 29).

Binary representation:

COMMAND	TYPE	MOT/BANK	VALUE
12	<parameter number>	<bank number>	(don't care)

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Figure 29: RSGP Binary and reply representation

3.3.9 PID regulation

In this paragraph I have described the TMCM – 1632 architecture control loop and its behaviour.

3.3.9.1 Structure of the cascaded motor regulation modes

The TMCM – 1632 support a current, velocity, and position PID regulation mode for motor control in different application areas. These regulation modes are cascaded as shown in (Figure 30).

The individual modes are explained in the following sections.

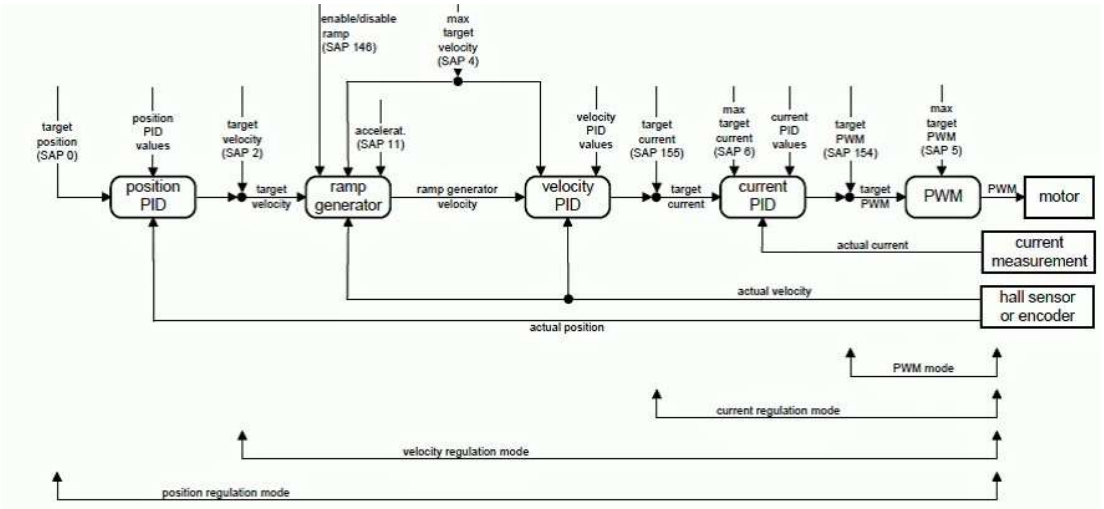

Figure 30: Cascaded PID regulation

3.3.9.2 PWM regulation

The PWM regulation mode is the most direct control mode for the TMCM – 1632. Thereby a target PWM, given by axis parameter 154, is adjusted directly without limiting the current motor. The target PWM is only limited by axis parameter 5 (max target PWM). The sign of the target PWM controls the motor rotation direction.

3.3.9.3 Current PID regulation

Based on the PWM regulation the current regulation mode uses a PID regulator to adjust a desired motor current. This target current can be set by axis parameter 155. The maximal target current is limited by axis parameter 6.

The PID regulation uses five basic parameters: The P, I, D and I – Clipping value as well as the timing control value. The timing control value (current regulation loop multiplier, axis parameter 134) determines how often the current regulation is invoked. It is given in multiple of $50\ \mu s$ as (3.1)

$$t_{PIDDELAY} = x_{PIDRLD} \cdot 50\mu s \tag{3.1}$$

Where:

- $t_{PIDDELAY}$ is resulting delay between two PID calculations.
- x_{PIDRLD} is current regulation loop multiplier parameter.

For most applications it is recommended to leave this parameter unchanged at its default of $2 \cdot 50\mu s$. Higher values may be necessary for very slow and less dynamic drives. The structure of the current PID regulator is shown in (Figure 31). It has to be parameterized with respect to a given motor.

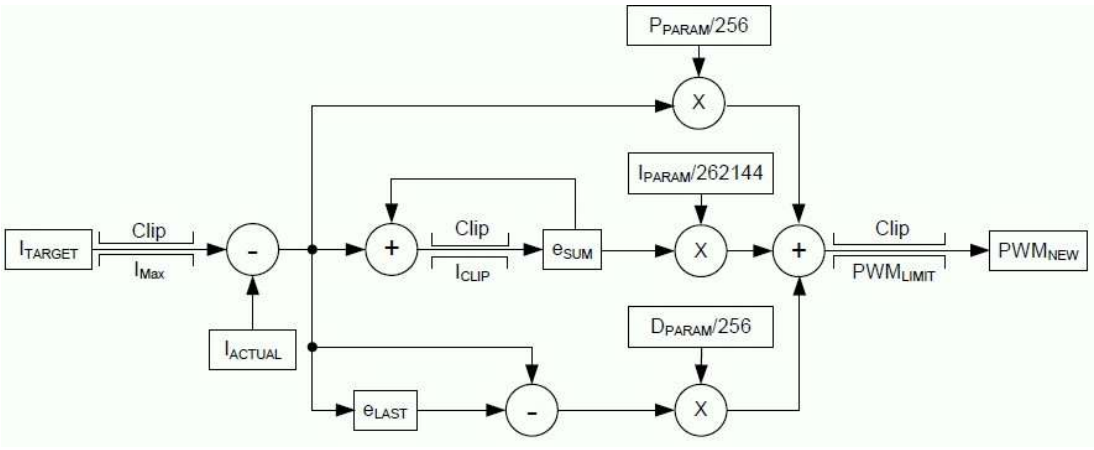

Figure 31: Current PID regulation

In the (Table 10) there are current PID parameters:

Parameter	Description
I_{ACTUAL}	Motor's actual current (GAP 150)
I_{TARGET}	Motor's target current (SAP 155)

I_{MAX}	Motor's max. target current (SAP 6)
e_{LAST}	Error value of the last PID calculation (GAP 200)
e_{SUM}	Error sum for integral calculation (GAP 201)
P_{PARAM}	Current P parameter (SAP 168 – 172)
I_{PARAM}	Current I parameter (SAP 169 – 173)
D_{PARAM}	Current D parameter (SAP 170 – 174)
I_{CLIP}	Current I – Clipping parameter (SAP 171 – 175) [1/10%] of max PWM_{LIMIT}
PWM_{LIMIT}	PWM Limit (SAP 5)
PWM_{NEW}	New target PWM value (GAP 153)

Table 10: Current PID parameter description

To parameterize the current PID regulator for a given motor, first set the P, I and D parameter of both parameter sets to zero. Then start the motor by using a low target current (e.g. $1000mA$). Then modify the current P parameter (II). This is the P parameter of parameter set 2. Start from a low value and go to a higher value, until the actual current nearly reaches the desired target current.

After that, do the same for the current I parameter (II) with the current D parameter (II) still set to zero. For the current I parameter (II), there is also a clipping value. The current I-Clipping parameter (II) should be set to a relatively low value to avoid overshooting, but high enough to reach the target current. The current D parameter (II) can still be set to zero.

After having found suitable values for parameter set 2, the first parameter set (PID Parameter Set 1) should be set to lower values, to minimize overshooting during zero-time of motor start. Then stop the motor and start again to test the current regulation settings. If the motor current is overshot during zero-time, set the PID parameter on 1 once more to lower values.

3.3.9.4 Velocity PID regulation

Based on the current's regulation the motor velocity can be controlled by the velocity PID regulator. Also, the velocity PID regulator uses a timing control value (PID regulation loop multiplier, axis parameter 133) which determines how often the PID regulator is invoked. It is given in multiple of $1\ ms$ as (3.2)

$$t_{PIDDELAY} = x_{PIDRLD} \cdot 1\text{ms} \tag{3.2}$$

Where:

- $t_{PIDDELAY}$ is resulting delay between two PID calculations.
- x_{PIDRLD} is PID regulation loop multiplier parameter.

For most applications it is recommended to leave this parameter unchanged at its default of $1\ ms$. Higher values may be necessary for very slow and less dynamic drives. The structure of the velocity PID regulator is shown in (Figure 32).

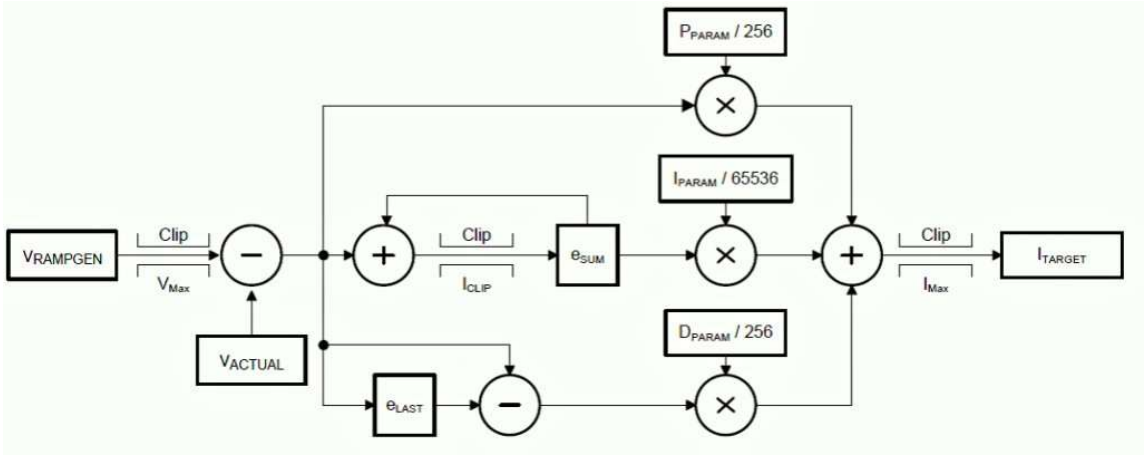

Figure 32: Velocity PID regulation

In the (Table 11) there are the following velocity PID parameters:

Parameter	Description
V_{ACTUAL}	Motor's actual velocity (GAP 3)
$V_{RAMPGEN}$	Target velocity of ramp generator (SAP 2, GAP 13)
V_{MAX}	Max. Target motor velocity (SAP 4)
e_{LAST}	Error value of the last PID calculation (GAP 228)
e_{SUM}	Error sum for integral calculation (GAP 229)
P_{PARAM}	Velocity P parameter (SAP 140 – 234)
I_{PARAM}	Velocity I parameter (SAP 141 – 235)
D_{PARAM}	Velocity D parameter (SAP 142 – 236)
I_{CLIP}	Velocity I – Clipping parameter (SAP 143 – 237) [1/10%] of I_{MAX}
I_{MAX}	Current's max. target (SAP 6)
I_{TARGET}	Current's Target for current's PID regulation (GAP 155)

Table 11: Velocity PID parameter description

3.3.9.5 Velocity ramp generator

For a controlled start-up of the motor's velocity, a velocity ramp generator can be activated/deactivated by axis parameter 146. The ramp generator uses the maximal allowed motor velocity (axis parameter 4), the acceleration (axis parameter 11) and the desired target velocity (axis parameter 2) to calculate a ramp generator velocity for the following velocity PID regulator.

3.3.9.6 Position PID regulation

Based on the current and velocity PID regulators the TMCM – 1632 support a positioning mode based on encoder or hall sensor position. During positioning the velocity ramp generator can be activated to enable motor positioning with controlled acceleration, or disabled to support motor positioning with velocity's max. target. The structure of the position PID regulator is shown in (Figure 33).

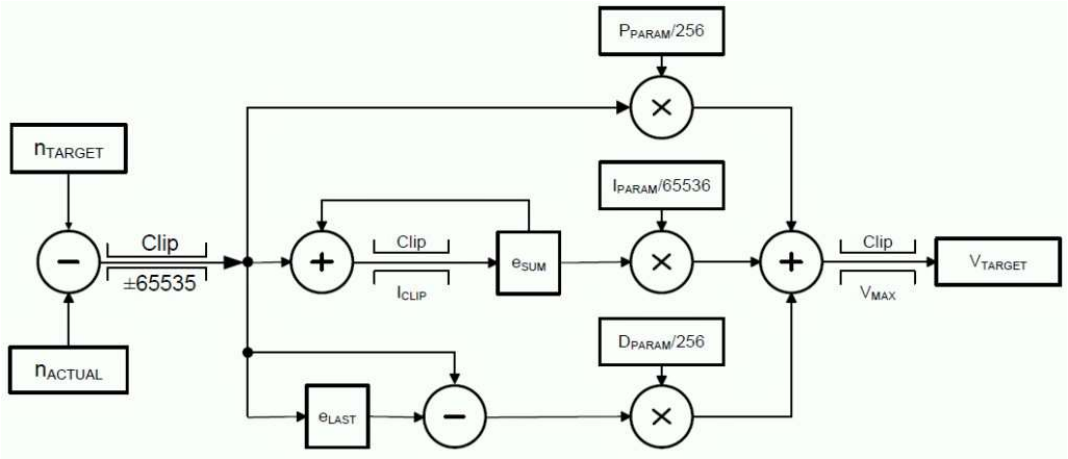

Figure 33: Positioning PID regulation

In the (Table 12) there are the following position PID parameters:

Parameter	Description
n_{ACTUAL}	Motor's actual position (GAP 1)
n_{TARGET}	Motor's target position (SAP 0)
e_{LAST}	Error value of the last PID calculation (GAP 226)
e_{SUM}	Error sum for integral calculation (GAP 227)
P_{PARAM}	Position P parameter (SAP 130 – 230)
I_{PARAM}	Position I parameter (SAP 131 – 231)
D_{PARAM}	Position D parameter (SAP 132 – 232)
I_{CLIP}	Position I – Clipping parameter (SAP 135 – 233) [1/10%] of V_{MAX}
V_{MAX}	Velocity's max. target (SAP 5)
V_{TARGET}	Velocity's new target for ramp generator (GAP 13)

Table 12: Position PID parameter description

The PID regulation uses five basic parameters. The P, I, D, and I-Clipping value as well as a timing control value. The timing control value (PID regulation loop parameter-axis parameter 133) determines how often the PID regulator is invoked. It is given in multiple of $1\ ms$ as (3.3)

$$t_{PIDDELAY} = x_{PIDRLD} \cdot 1\text{ms} \tag{3.3}$$

Where:

- $t_{PIDDELAY}$ is resulting delay between two PID calculations.
- x_{PIDRLD} is PID regulation loop multiplier parameter.

For most applications it is recommended to leave this parameter unchanged at its default of $1\ ms$. Higher values may be necessary for very slow and less dynamic drives.
Based on the velocity PID regulator the position PID regulator can be parameterized as P regulator in the simplest case. Therefore, disable the velocity ramp generator and set position P, I, and D parameters to zero. Now set a target position and increase the position P parameter until the motor reaches the target position approximately.

After P parameter finds a good position, the velocity ramp generator can be switched on again. Based on the maximum positioning velocity (axis parameter 4), as well as the acceleration (axis parameter 11) value, the ramp generator automatically calculates the slow down point, i.e. the point at which velocity is to be reduced in order to stop at the desired target position. Reaching the target position is signalled by setting the Position end flag. In order to minimize the time until this flag becomes set, a positioning tolerance (MVP target reached distance) can be chosen by axis parameter 10. Since the motor typically is assumed not to signal target reached when the target was just passed in a short moment at a high velocity, additionally a maximum target reached velocity (MVP target reached velocity) can be defined by axis parameter 7. A value of zero is the most universal, since it implies that the motor stands still at the target. But when a fast rising of the Position end flag is desired, a higher value reached for MVP target velocity parameter will save a lot of time. The best value should be tried out in the actual application. The correlation of axis parameter 10, 7, the target position and the position end flag are summarized in (Figure 34).

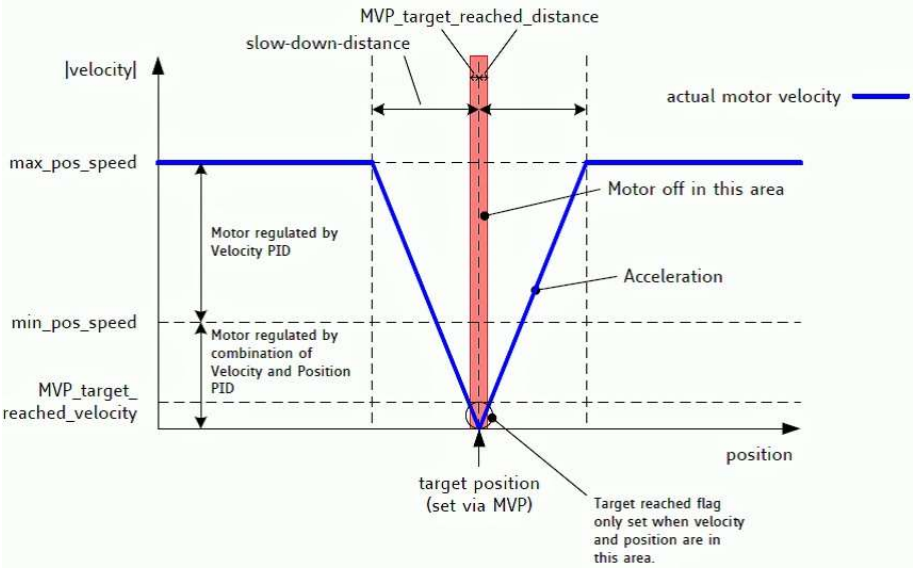

Figure 34: Positioning algorithm

Depending upon the motor and mechanics respectively, a bit of oscillation is normal; in the best of cases, it can be reduced to be at least ± 1 encoder step, because otherwise the regulation cannot keep the position.

3.3.9.7 Parameter sets for PID regulation

Every PID regulation provides two parameter sets, which are used as follows (Figure 35):

- Below a specified velocity threshold, the PID regulator uses a combination of parameter set 1 and parameter set 2.
- Above the velocity threshold the PID regulator uses only parameter set 2. If the velocity threshold is set to zero, parameter set 2 is used all the time. (The switch between both parameter sets is soft).

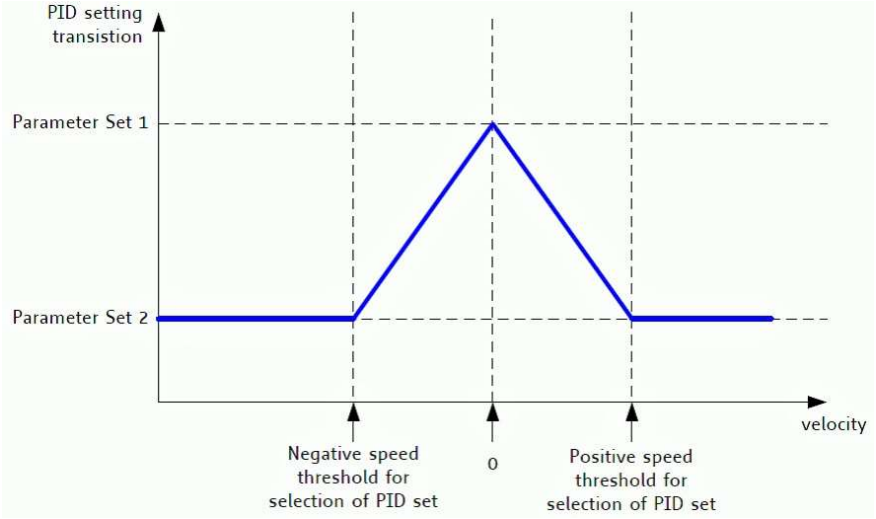

Figure 35: Transition between PID parameter sets

The velocity thresholds for current, velocity, and position PID regulation can be set as follows:

- axis parameter 176: velocity threshold for current PID regulator.
- axis parameter 8: velocity threshold for velocity PID regulator.
- axis parameter 12: velocity threshold for position PID regulator.

3.3.10 Temperature calculation

Axis parameter 152 delivers the actual ADC value of the motor driver. This ADC value can be converted to a temperature in °C as follows (3.4 – 3.5):

$$R_{NTC} = \frac{9011.2}{ADC} - 2.2 \tag{3.4}$$

$$T = \frac{B \cdot 298.16}{B + \left(\ln\left(\frac{R_{NTC}}{10}\right) \cdot 298.16\right)} - 273.16\,°C \tag{3.5}$$

Where:

- ADC is actual value of GAP 152.
- $B = 3434$ is a costant value.

3.3.11 I²t monitoring

The I^2t monitor determines the sum of the square of the motor's current over a given time. The integrating time is specific. In the datasheet of the motor this time is described as thermal winding time constant and can be set for each module using axis parameter 25. The number of measurement values within this time depends on how often the current regulation and thus the I²t monitoring is invoked. The value of the actual I^2t sum can be read by axis parameter 27. With axis parameter 26 the default value for the I²t limit can be changed (default: 211200). If the actual I^2t sum exceeds the I^2t limit of the motor, flag 17 (in axis parameter 156) is set and the motor

pwm is set to zero as long as the I^2t exceed flag is set. The actual regulation mode will not be changed.

Furthermore, the I^2t exceed counter is increased once every second, as long as the actual I^2t sum exceeds the I^2t limit. The I^2t exceed flag can be cleared manually using parameter 29, but only after the cool down time given by the thermal winding time constant has passed. The I^2t exceed flag won't be reset automatically. The I^2t limit can be determined as follows (3.6)

$$I^2t = \frac{I[mA]}{1000} \cdot \frac{I[mA]}{1000} \cdot t_{tw}[s] \tag{3.6}$$

Where:

- I is the desired average current.
- t_{tw} is the thermal winding time constant given by the motor datasheet.

3.4 The ROS

The ROS (Robot Operating System) is a set of software structures for the development of robot's software; it provides a similar functionality to that of an operating system on an heterogeneous computer cluster and standard operating system services, such as:

- Hardware abstraction.
- Low-level device control.
- Implementation of commonly used functionality
- Message-passing between processes
- Package management.

Running sets of ROS-based processes are represented in an architecture graph, where processing takes place in nodes that may receive, post and multiplex the sensor, control, state, planning, actuator and other messages. Despite the importance of reactivity and low latency in robot control, ROS, itself, is not a real time OS, though it is possible to integrate ROS with real time code (the ROS, 2011) .

Software in the ROS Ecosystem can be separated into three groups:

- Language-and platform-independent tools used for building and distributing ROS-based software;
- ROS client library implementations such as *roscpp*, *rospy*, and *roslisp*;
- Packages containing application-related code which uses one or more ROS client libraries.

Both the language-independent tools and the main client libraries (C++, Python, and LISP) are released under the terms of the BSD license, and as such are an open source software and free for both commercial and research uses. The majority of other packages are licensed under a variety of open source licenses. These other packages implement commonly used functionality and

applications such as hardware drivers, robot models, data-types, planning, perception, simultaneous localization and mapping, simulation tools, and other algorithms.

The main ROS client libraries (C++, Python, LISP) are geared toward an Unix-like system, due primarily because of their dependence on large collections of open-source software dependencies. For these client libraries, Ubuntu Linux is listed as "supported" while other variants such as Fedora Linux, Mac OSX, and Microsoft Windows are designated "experimental" and are supported by the community.

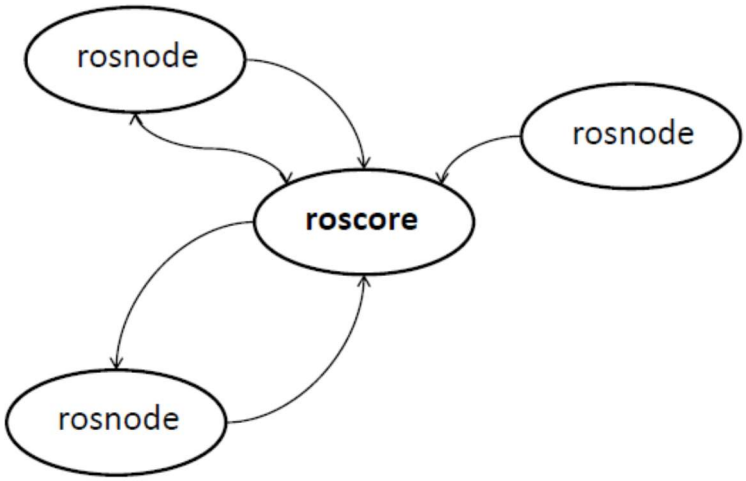

Figure 36: ROS architecture

The ROS architecture is organized as in (Figure 36), where:

- *Roscore* is a collection of nodes and programs that are pre-requisites of a ROS-based system.
- *Rosnode* is an executable file inside a ROS package that uses a library of ROS client to communicate with other *rosnodes.* It can publish or subscribe to a topic.

3.5 The command chain

The command chain is a set of significant steps between input's sending and its implementation. They are the following:

1) The "load": represents the action of sending the command from the operator through external environment interface (keyboard). The step ends when the information is published on a rostopic; in this case the rostopic is called command_current/velocity/position, according to the type of command sent.
2) The "wrapper": consists on information reading, located in rostopic by the wrapper (package youbot_oodl) and the communication of this at the youbot_driver.
3) The "send": consists in sending information from the youbot_driver to the engines.
4) The "receive": consists in receiving information from the engines to the youbot_driver.

Because it is impossible to measure the engine's implementation command time directly, my hypothesis is that it is between the third and the fourth step.

3.6 The youBot_driver

The youBot_driver consists of cpp classes that provide low-level access to the KUKA youBot hardware. It implements an EtherCAT master, based on the free SOEM library, tailored to the specific EtherCAT protocol of the youBot.

It consists of two parts:

- EtherCAT driver: on the lowest (accessible) level of control, all actuators of the KUKA youBot are driven over the EtherCAT protocol. This protocol uses standard network hardware like Ethernet adapters, respective RJ45 connectors and network cables to connect EtherCAT master and slaves and establish the communication between them (see EtherCAT specs for details. The KUKA youBot utilizes an open source driver implementation SOEM for the master.
- KUKA youBot API: on top of the EtherCAT driver stack there is a second software layer. It was developed by the Autonomous Systems Group at Bonn-Rhein-Sieg University of Applied Sciences, Germany. It builds upon SOEM EtherCAT drivers and provides additional functionality, relevant to platform kinematics and joint level commanding. Currently, this library provides the basic functionalities to set the Cartesian velocities of the base, to read odometry data from motor encoders etc. It furthermore provides the functionality to command the manipulator on joint level (setting and getting joint configurations/data, gripper). This part will be expanded very soon by libraries such as OROCOS KDL to also command the youBot arm in Cartesian space.

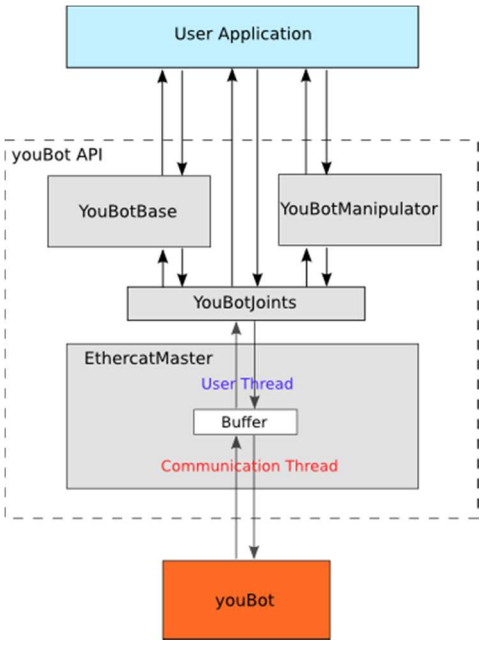

Figure 37: youBot driver sub-systems

The basic idea of the KUKA youBot API (provided in the youBot_driver) is to represent a robot system as a combination of decoupled functional subsystems as (Figure 37). This API

represents manipulator arm and base platform as the combination of several joints. At the same time each joint is defined as a combination of a motor and a gearbox.

There are three main classes in the youBot API that a user should be aware of. These are:

- **YouBotManipulator class** that represents youBot arm as the aggregation of several joints and a gripper (Paulus J., 2011).
- **YouBotBase class** that represents youBot base platform (Paulus J., 2011).
- **YouBotJoint class** that represents joints which make up both the manipulator and the base (Paulus J., 2011).

Every youBot joint is represented as a *youBot::YouBotJoint* class in the API. At this stage the API does not make a distinction if it is a base joint which powers a wheel or a manipulator joint. *youBot::YouBotBase* and *youBot::YouBotManipulator* are aggregate classes that allow access to a particular joint instance through *youBot::YouBotJoint*.

The youbot_driver specific classes that describe microcontrollers and command the motors are:

- **YouBotJointParameter class**: represents PID's position/velocity/current parameter control (Paulus J., 2011).
- **YouBotJointParameterPasswordProtected class**: represents the microcontroller's parameter, editable only by password (Paulus J., 2011).
- **YouBotJointParameterReadOnly class**: represents only the readable microcontroller's parameter (Paulus J., 2011).

4 THE EXISTING ARCHITECTURE TEST

The performance measurement is important to determine the main problems of the pre-existing process architecture.

These are divided into:

- Time measurements, such as jitter, necessary for evaluating delays, as well as the main components of it, between the sending of a command and its implementation.
- The characterization of system response, such as overshoot or rise time, necessary for evaluating the response objectively.
- The correspondence between command and response at different operating frequencies to decide which is the best sampling time.

In base of this, the performance analysis is split into four parts:

- In the first part (4.1) the jitter (Wikipedia, 2014) is measured, i.e. in terms of delay time between command sending and its implementation.
- In the second part (4.2) the position, speed, current response during a position or speed control are analysed.
- In the third part (4.3) the system responses during a position control with no constant and constant load torque are evaluated.
- In the fourth part (4.4) the maximum sampling system's frequency, i.e. the maximum value of the sampling frequency is verified; it produces no distortions, caused by the excessive number of samples sent to the microcontroller.

4.1 Jitter measurement

To estimate the jitter introduced by the system, the total delay (between command sending and its implementation) has been measured and then divided in the command chain step (see par. 3.2.2), i.e. jitter between "load" and "wrapper", between "wrapper" and "send", between "send" and "implementation".

The jitter measurements are taken during many speed control tests, and in each of them the sampling frequency has been changed in youBot OODL package.

Due to the design requirements, it was decided to have sampling frequency into the band from $1\ kHz$ to $10\ kHz$.

And for this, it was chosen to test the system with a sampling frequency respectively equal to $1\ kHz$, $10\ kHz$ and $100\ kHz$.

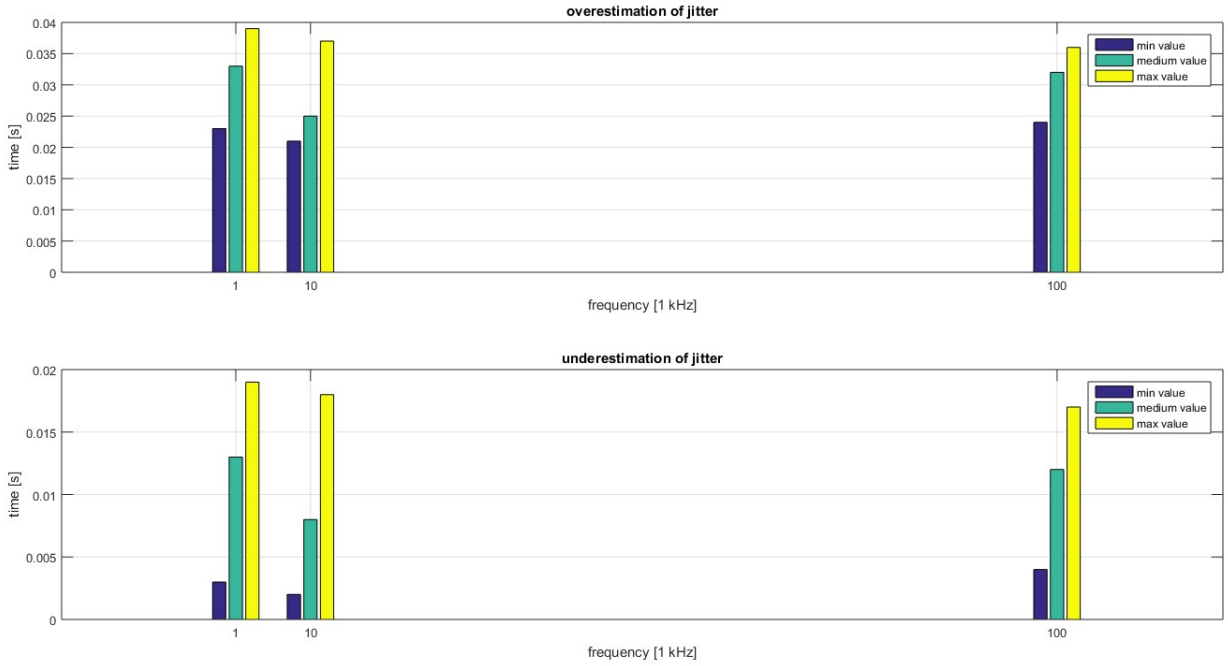

Figure 38: Underestimation and overestimation of jitter

In (Figure 38-bottom panel) the jitter (minimum value, medium value and maximum value) between "load" (see par. 3.2.2) and "send" is shown for the chosen sampling frequency.

This is an underestimation of the jitter between "load" and "implementation" and we can see that, nevertheless the changing of sampling frequency, it doesn't vary.

In (Figure 38-top panel) the jitter (minimum value, medium value and maximum value) between "load" (see par. 3.5) and "receive" is shown for the chosen sampling frequency.

This is an overestimation of the jitter between "load" and "implementation" and we can see that, nevertheless the changing of sampling frequency, it doesn't vary.

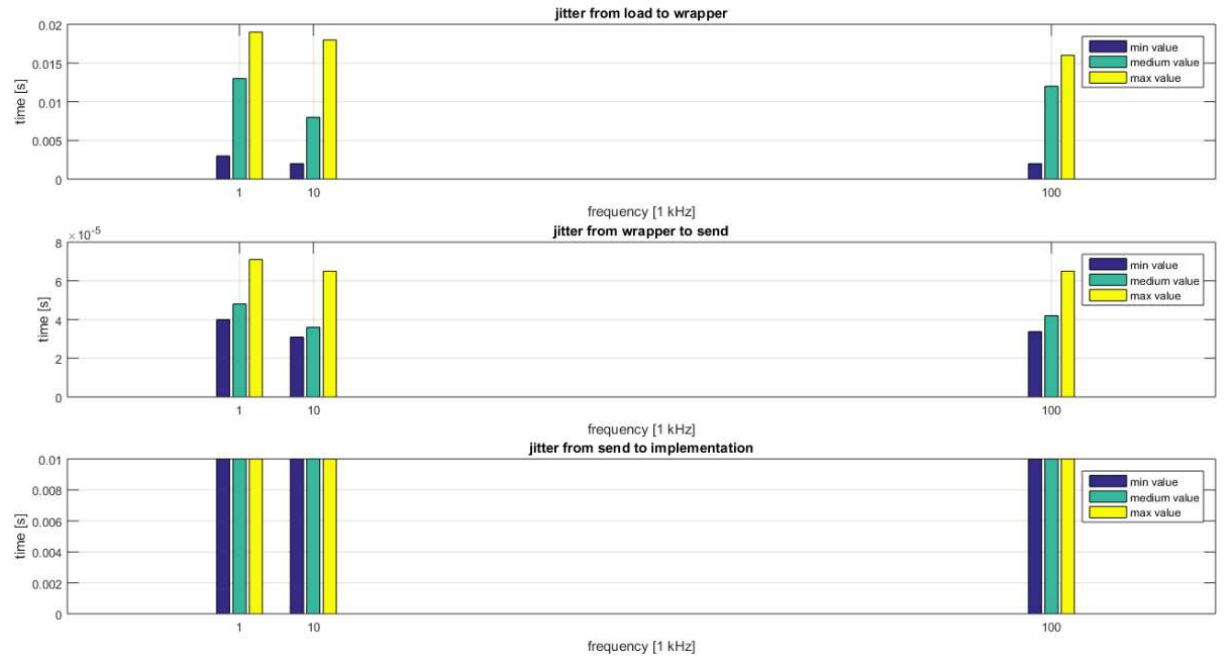

Figure 39: Partial jitter

In (Figure 39) the jitter in between “load” and “wrapper” (Figure 39-top panel), the jitter between “wrapper” and “send” (Figure 39-middle panel) and the jitter between “send” and “implementation” (Figure 39-bottom panel) are shown.

We can see that the most important contributions of jitter are in the first step (“load” to “wrapper”) and in the third step (“send” to “implementation”).

This result is important because it allows us to know which components act to reduce the delay from “load” to “implementation”.

This conclusion is the same for all sampling frequencies.

4.2 Qualitative analysis of the engine responses

This analysis is used to characterize the system’s responses when it is excited by position or speed commands.

In the following tests, the youbot_driver is interfaced with MatLab through a "mex" blocks (see Appendix A).

This operation has been made to obtain an operating frequency, real and rectified, equal to $1\ kHz$.

4.2.1 Qualitative analysis of the engines responses to the position input

In this test, the system is excited by position commands and its responses have been characterized.

This command is described as a square wave, characterized by an amplitude value equal to $0.2\ rad$ and a frequency value equal to $0.5\ Hz$.

The choice of this input is determined by two factors:

- Working with only one control parameter set because the switch, between two settings, is connected to the speed threshold.
- Obtaining an estimate of the system’s model because from the input step’s response the information (dominant poles, the system degree, etc …) is derivable.

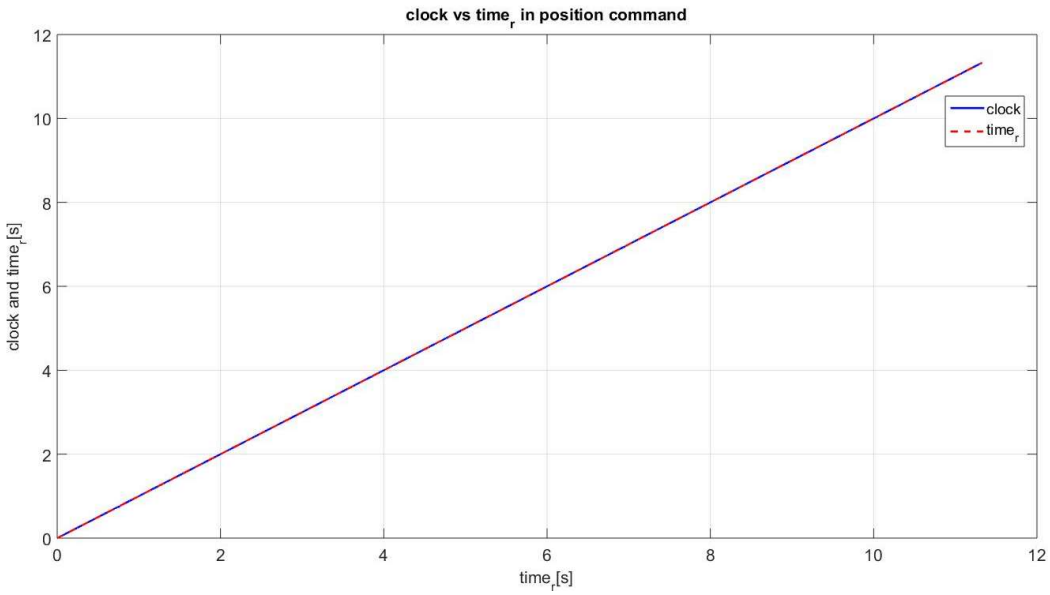

Figure 40: Real-time versus Simulation-time in position command

In the graph (Figure 40) the trend of real-time ($time_r$) and the simulation time ($clock$) in function to the real-time are shown.

The overlap of the two signals indicated the timing synchronization between the simulation time and the real time. This is a necessary condition for correct measurements.

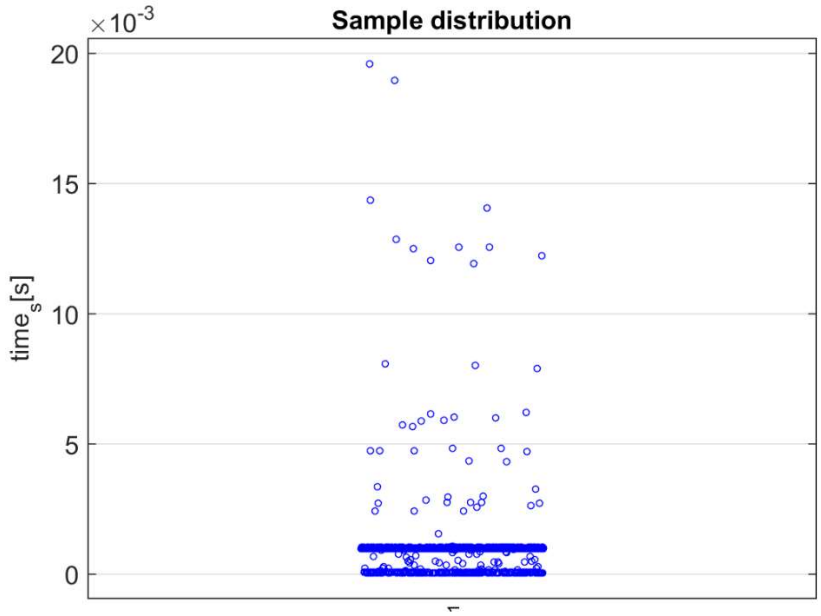

Figure 41: Sample time distribution in the position test

The sample time distribution is shown in (Figure 41) and we can see that the majority of the sample time is at 1 kHz giving a further validity to the following measurements.

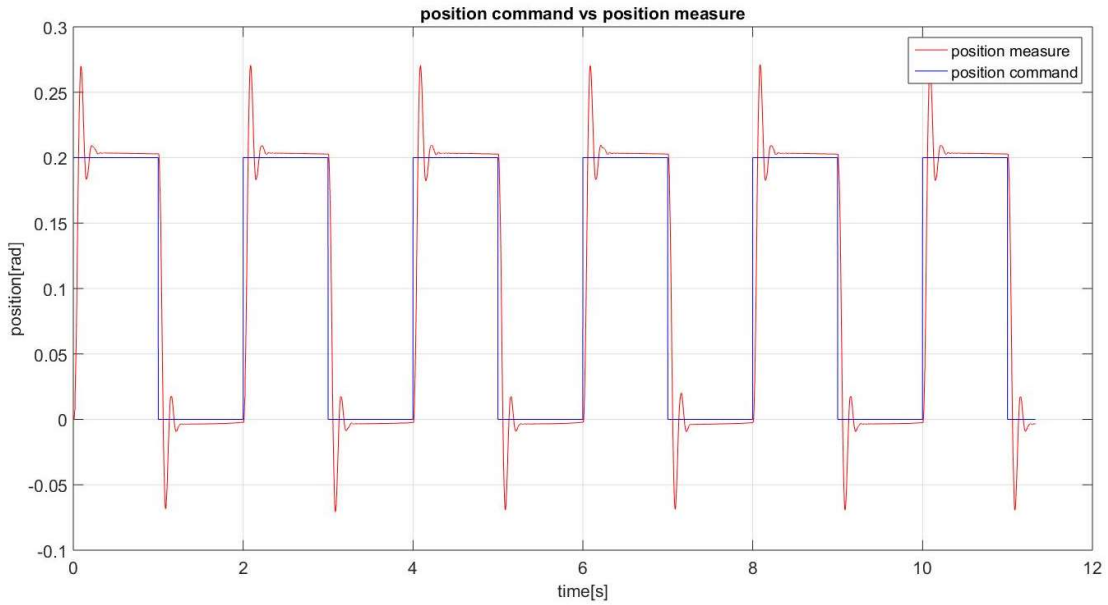

Figure 42: Position's reference versus position's measurement

The position response under the square wave position commands is shown in (Figure 42). This response is characterized by:

- Overshoot($0.07\ rad$), equal to 36% of the amplitude value.
- Rise time equal to $0.05\ sec.$
- Imperfect trajectory tracking with a maximum error equal to $0.07\ rad$ and an average error equal to $0.015\ rad$.

The study of the graph (Figure 42) shows the system's difficulty to track an assigned position trajectory, maintaining an average error of 7% compared to the amplitude value.

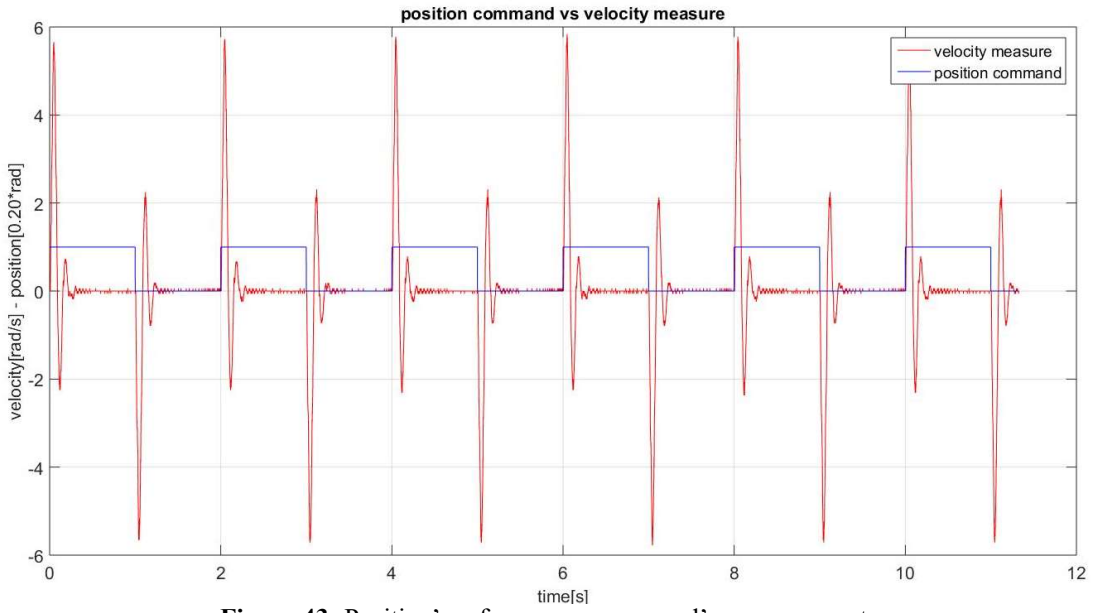

Figure 43: Position's reference versus speed's measurement

The speed's response compared to the square wave position reference is shown in (Figure 43). This response is characterized by:

- Maximum absolute value equal to $5.84\ rad/sec$.
- Two predominant peaks in response to a command position step.

The study of the graph (Figure 43) shows two predominant peaks. It can damage the manipulator and it has the main uncorrected tracking responsibility trajectory.

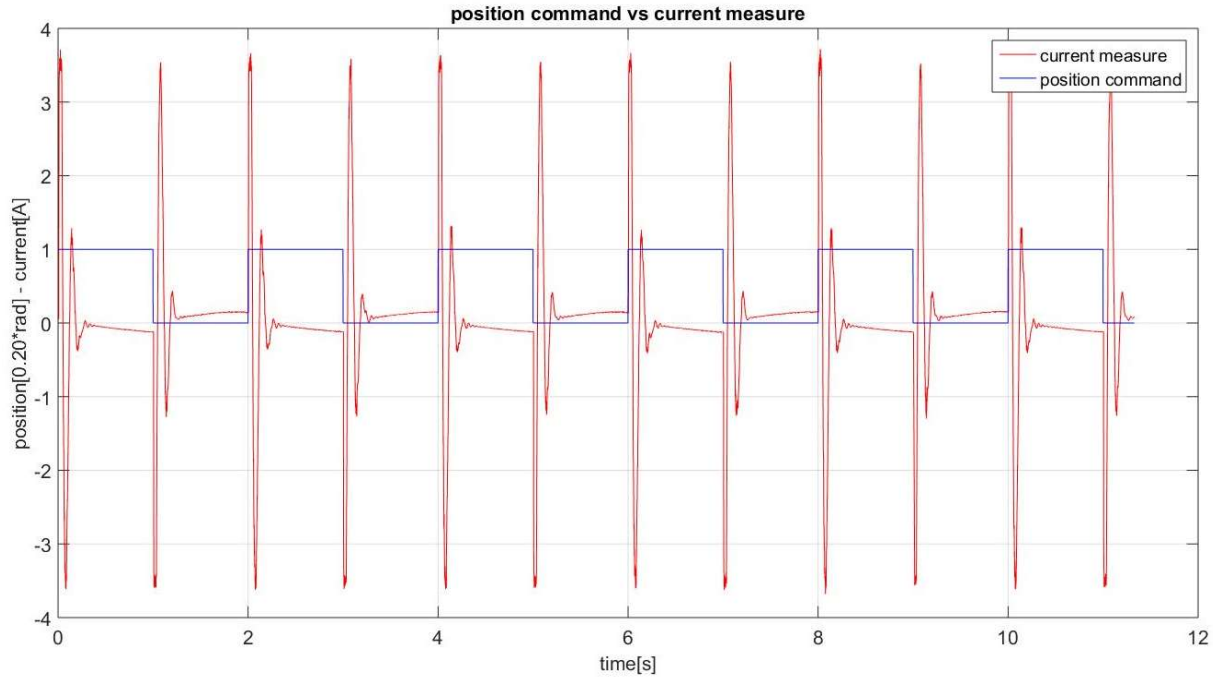

Figure 44: Position's reference versus current's measurement

The current response compared to the square wave position reference is shown in (Figure 44). This response is characterized by:

- Maximum absolute value equal to $3.71\ A$.
- Two predominant peaks in response to a position step command.
- Non-existing time when the current stays to $0\ A$.

The study of the graph (Figure 44) shows a current not always equal to $0\ A$. Especially when the current reaches the higher peaks, it can damage the manipulator and its constant presence doesn't permit the correct tracking trajectory.

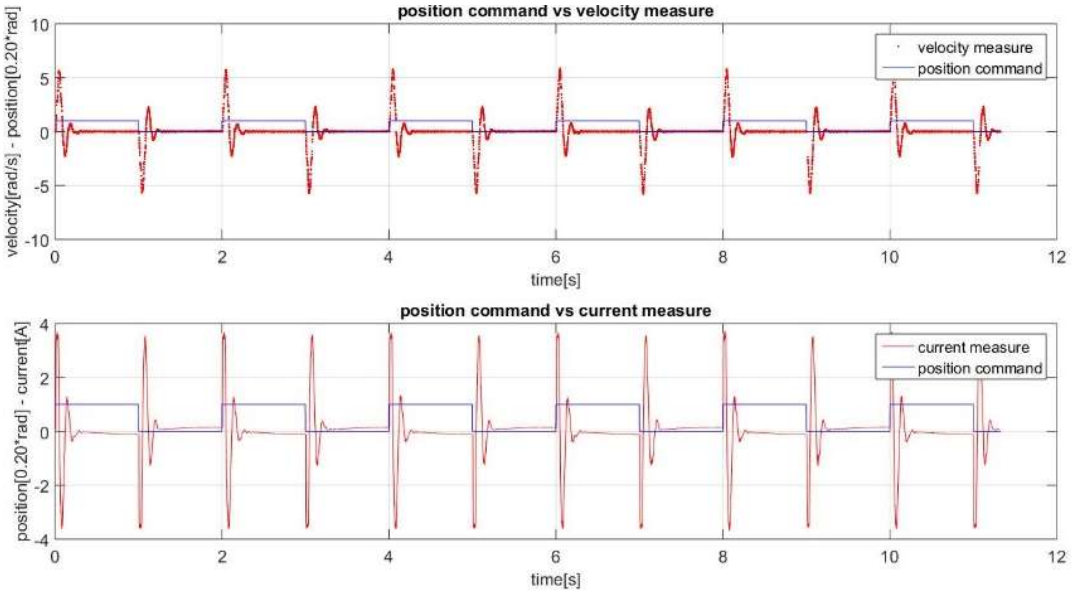

Figure 45: Position's reference versus speed's and current's measurement

The reference position compared to the velocity response (upper panel) and to the current response (lower panel) is shown in (Figure 45).

In this graph (Figure 45) it is easy to see how the current and the speed response are related. In fact, the two signals have iso-time peaks that cause the position to overshoot.

The main problems, in addition to those mentioned, are:

- Excessive current which could lead to blocking.
- Peak speed that wears mechanisms and may cause breakage.

4.2.2 Qualitative analysis of the engines responses to the speed input

In this test, the system is excited by velocity commands and its responses have been characterized.

This command is described as a square wave characterized by an amplitude value equal to $1\ rad/sec$ and a frequency value equal to $0.5\ Hz$.

The choice of this input is determined by two factors:

- Working with only one control parameter set, because the switch between two settings is connected to the speed threshold.
- Obtaining an estimate of the system model because from the input step's response the information (the dominant poles, the system degree, etc…) is derivable.

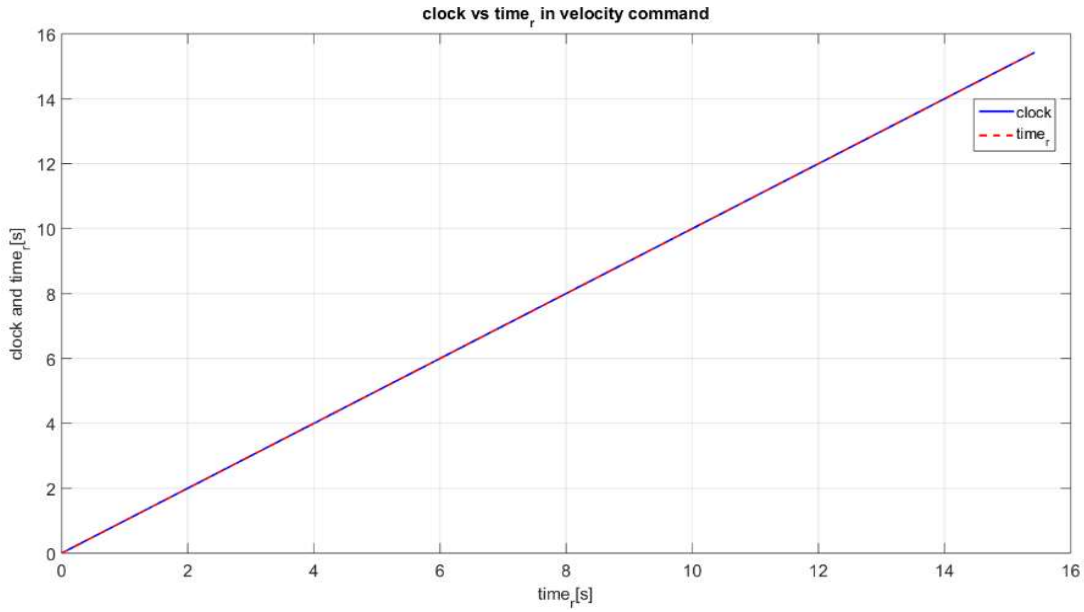

Figure 46: Real-time versus Simulation-time in speed's command

In graph (Figure 46) the trend of real-time ($time_r$) and the simulation time ($clock$) are shown in function to the real-time.

The overlap of the two signals indicated the timing synchronization between the simulation time and the real time. This is a necessary condition for correct measurements.

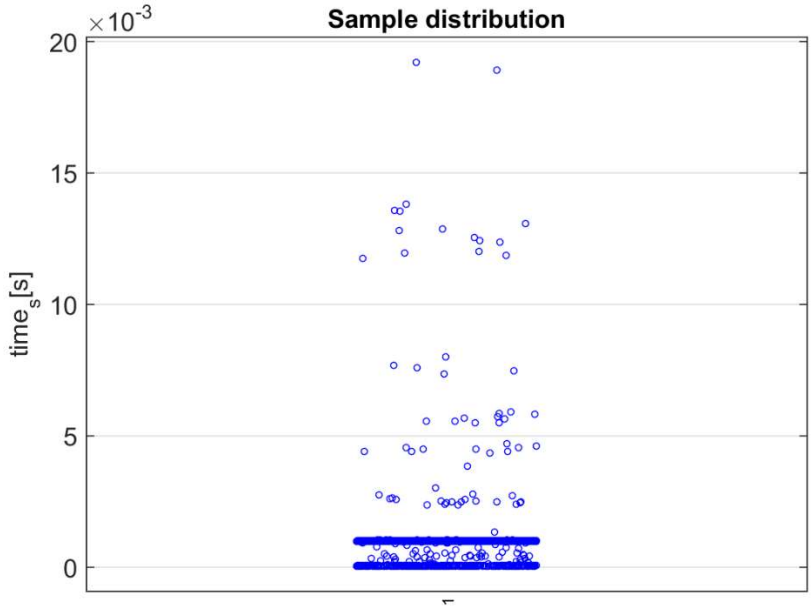

Figure 47: Sample time distribution in the velocity test

The sample time distribution is shown in (Figure 47) and we can see that the majority of the sample time is at 1 kHz giving a further validity to the following measurements.

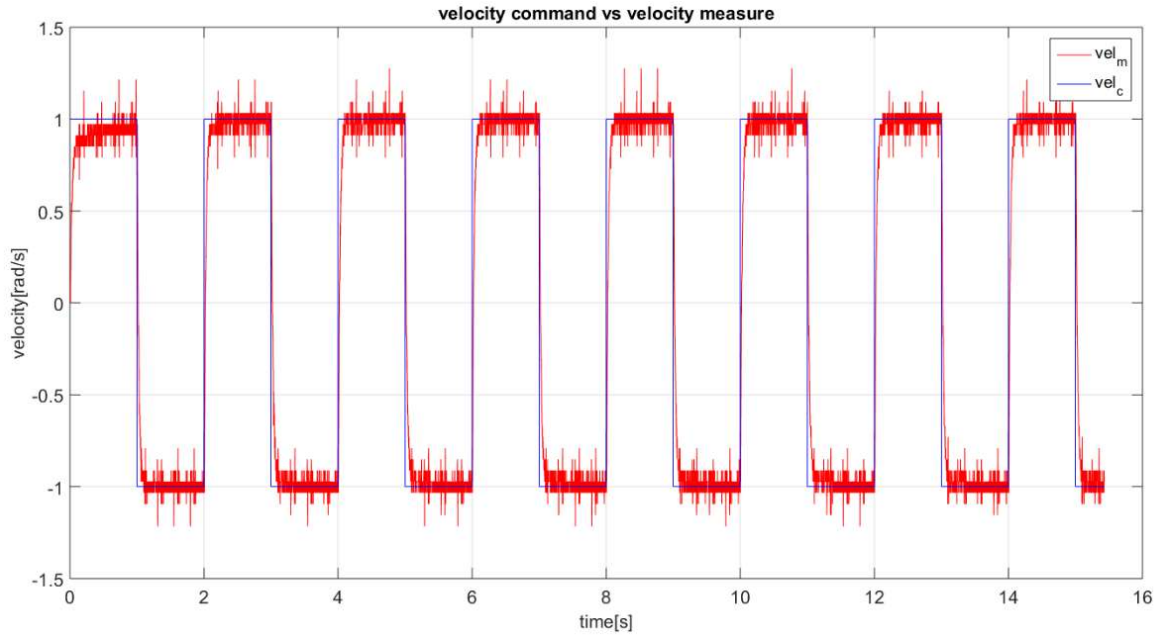

Figure 48: Speed's reference versus speed's measurement

The speed's response under the square wave speed commands is shown in (Figure 48). This response is characterized by:

- Rise time equal to 0.05 $sec.$
- Not perfect trajectory velocity tracking.

The study of the graph (Figure 48) shows the system's difficulty to track an assigned velocity trajectory because the velocity response is characterized by underdamping.

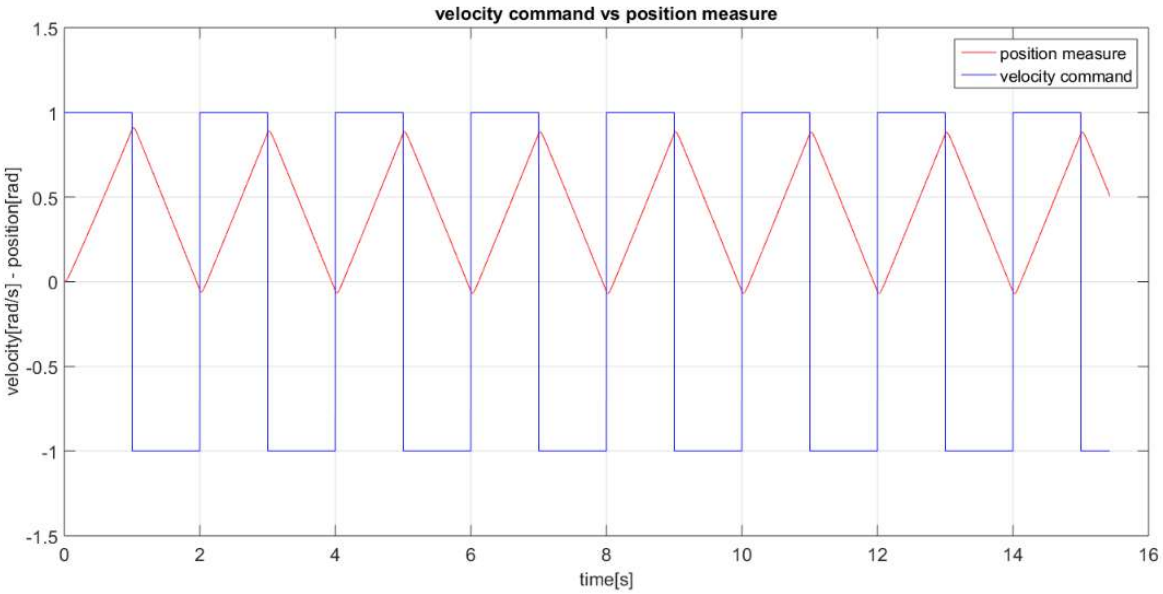

Figure 49: Speed's reference versus position's measurement

The position's response compared to the square wave velocity reference is shown in (Figure 49).

This response is characterized by:

- Not reaching the required value.

The study of the graph (Figure 49) shows the failure to reach the required value ($1\ rad$ vs $0.91\ rad$) probably due to resolution limits.

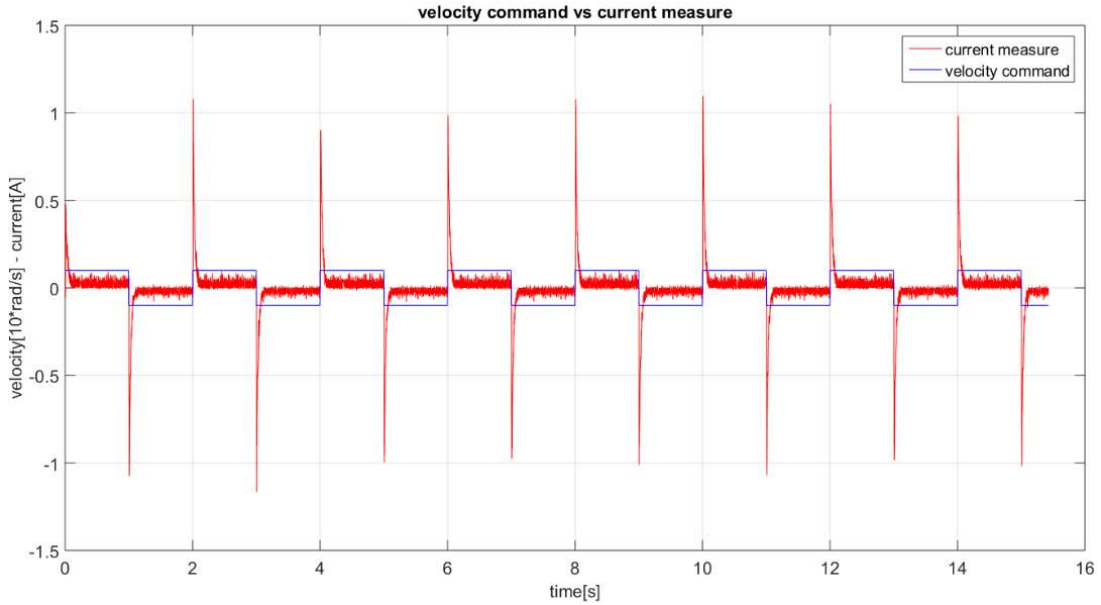

Figure 50: Speed's reference versus current's measurement

The current's response compared to the square wave velocity reference is show in (Figure 50).
This response is characterized by:

- Maximum absolute value equal to $1.09\ A$.
- Current peaks.

The study of the graph (Figure 50) shows the presence of current's peaks, always iso-time with the velocity's switch command.

This is a normal behavior because the system's response is a significant velocity variation.

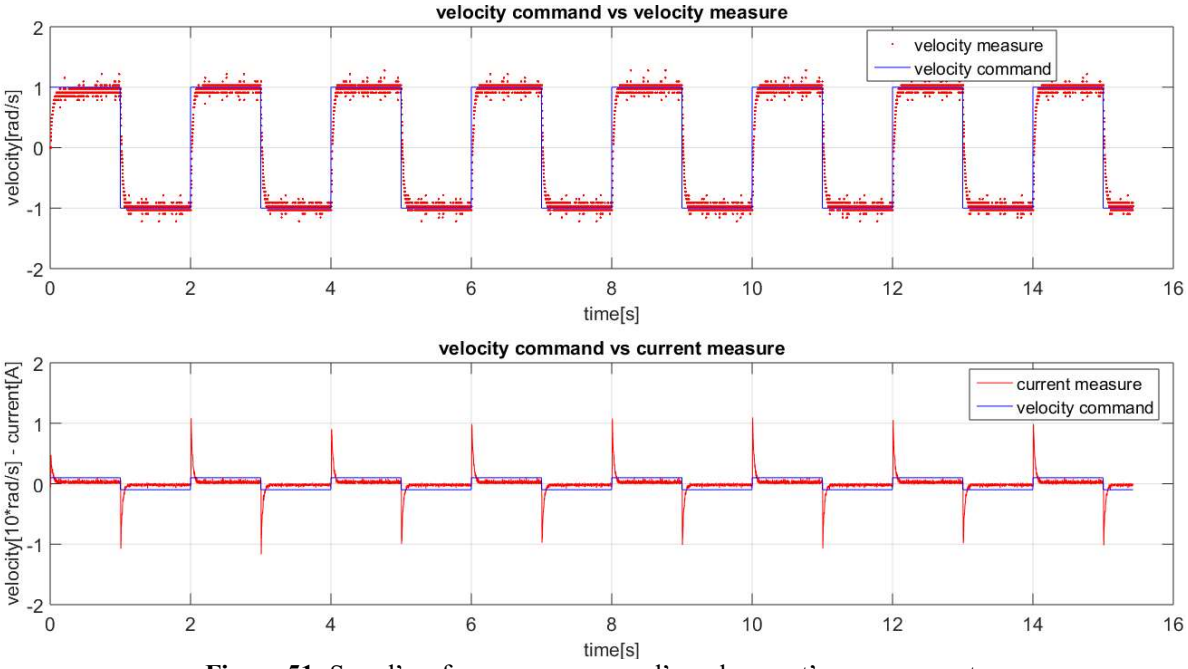

Figure 51: Speed's reference versus speed's and current's measurement

The velocity's reference compared to the velocity's response (upper panel) and the current's response (lower panel) are shown in (Figure 51).

In this plot (Figure 51) it is possible to see:

- The velocity's distribution of velocity's measurements (resolution problem).
- The iso-time connection between the current's peaks (provide energy to the system) and the speed's variation, in effect, to improve the underdamped position's response, the current's peaks should have greater magnitude.

4.3 Engine torque's response analysis and its characterization

The following analysis is used to estimate the non-linearity, its influence and the ripple's (Wikipedia, 2013) presence in the system torque's response.

It is divided into three parts:

- In the first test, the inconstant load wheel behaviour was evaluated when there is a constant speed's input module (see par. 4.3.1).
- In the second test, the constant load's wheel behaviour was evaluated when there is a constant speed input module (see 4.3.2).
- In the third test, the current's response of the youBot arm's joint was compared to the current's response of the omnidirectional platform's wheel (see par. 4.3.3).

4.3.1 Test 1: inconstant load wheel characterization in torque

In this test, at one of the omnidirectional platform wheels, a mass has been connected through a cable and positioned at the initial time, in a way that its arm, i.e. the horizontal distance from the wheels rotation axis, was equal to 0 m.

After, assuring that the mass didn't swing, this wheel has been stressed with the command shown in (Figure 52).

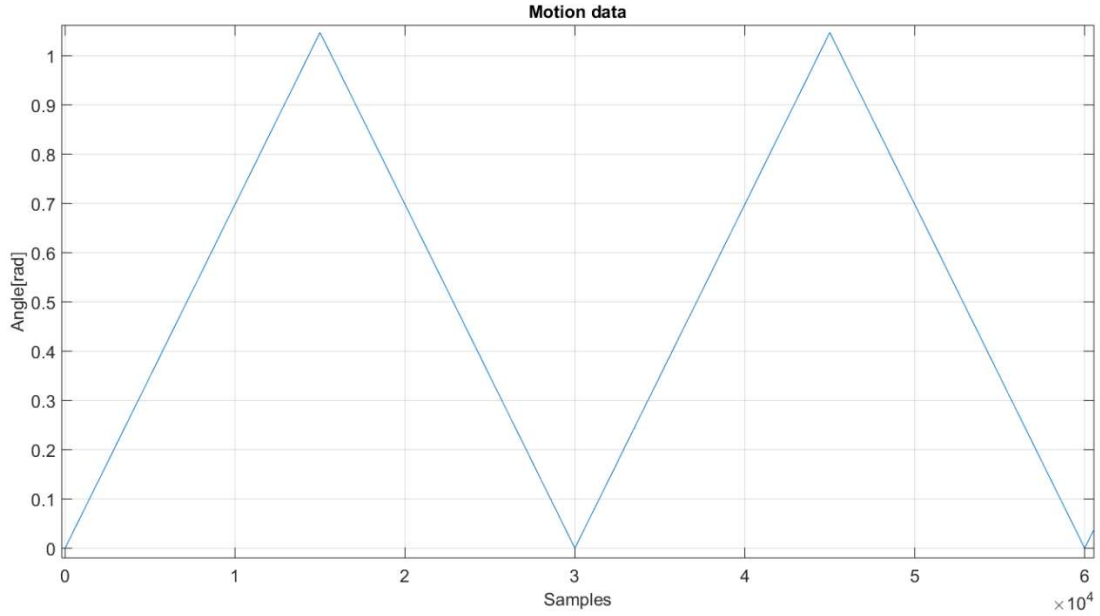

Figure 52: Trajectory path in function of sampling time

This command is represented by a double triangular wave with minimum value equal to 0 rad, maximum value equal to 1.05 rad and slope module value equal to 0.07 rad/s.

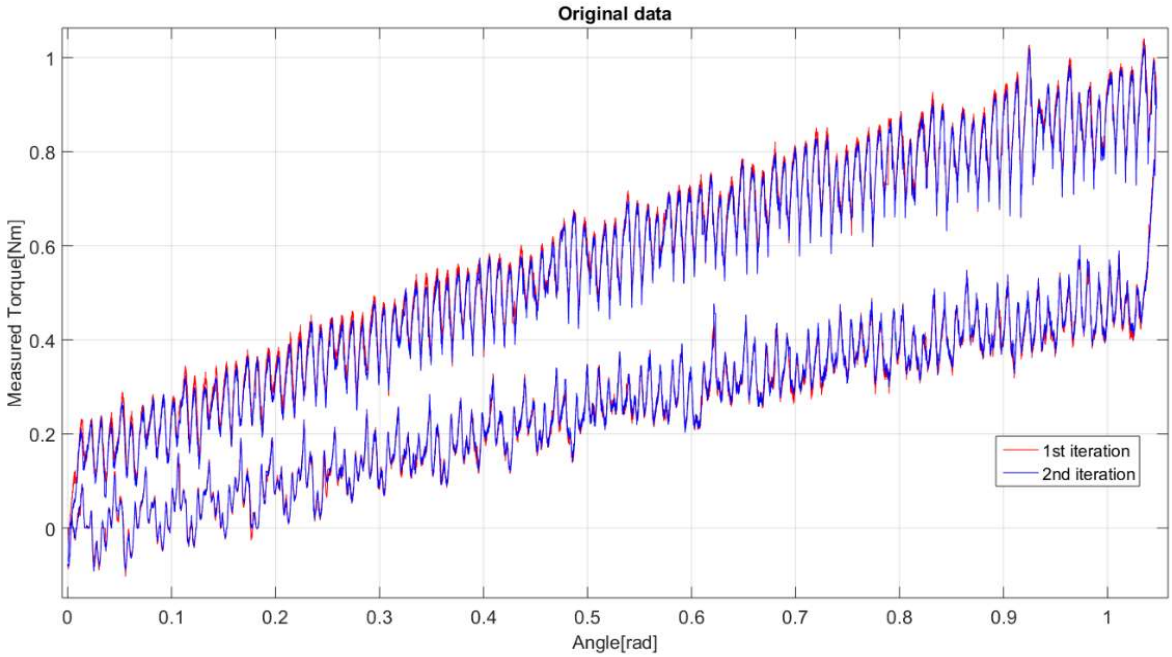

Figure 53: Measured torque in function of the measured angle (original data)

The (Figure 53) shows the trend of the measured torque in function of the measured angle.

In this graph the overlap between the two iterations, 1st in red and 2nd in blue, is visible and its presents iso-time peaks that indicate the ripple's presence.

The effect of the ripple on the wheel's response will be evaluated in the next paragraph (see par. 4.3.2).

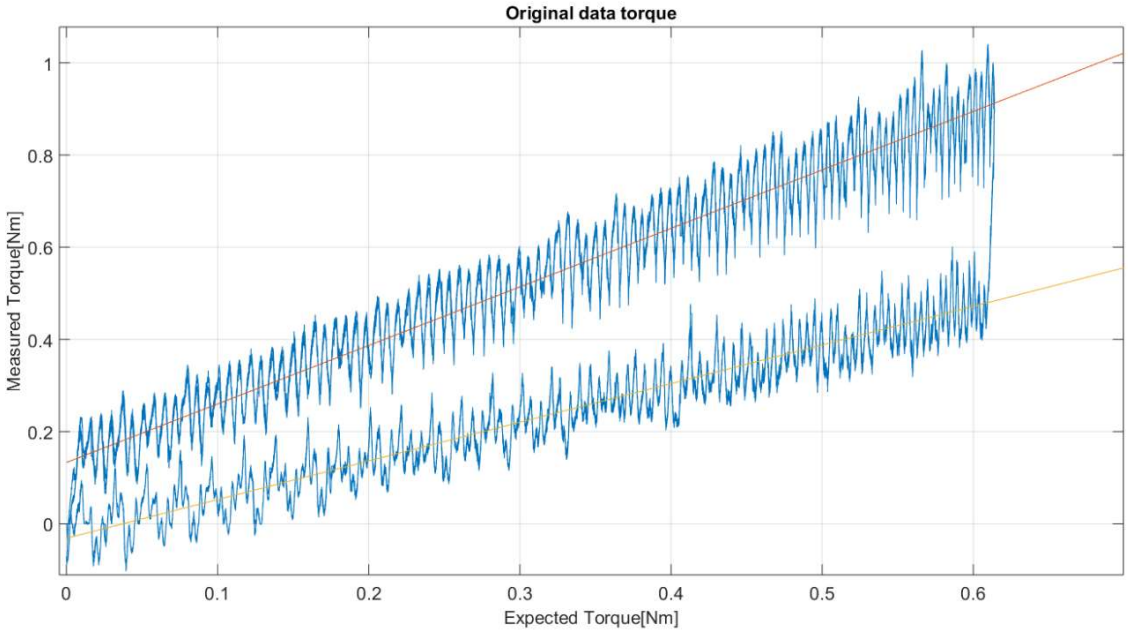

Figure 54: Measurement's torque in the estimated torque (original data)

In (Figure 54) the trend of measured engine torque is shown in function of the estimated engine torque by (3.1).

$$\tau_{est} = M \cdot g \cdot R \cdot \sin(pos_{read}) = 0.709 \cdot \sin(pos_{read}) \qquad (3.1)$$

In which:

- The value 0.709 represents the product between gravity's acceleration (g), mass (M) and radius from wheel-cable's connection point to the wheel's rotation axis (R).
- pos_{read} is the position's measurement reported in radians.
- τ_{est} is the estimated engine's torque.
- The product $R \cdot \sin(pos_{read})$ represents the arm of the force (Mg),

In the same figure, the direct (red) and reverse (yellow) efficiency trends are shown.

The efficiency trend is obtained by polynomial curve fitting, and this is represented as a split line of the trend reported.

The efficiency is important to value the motor's quality.

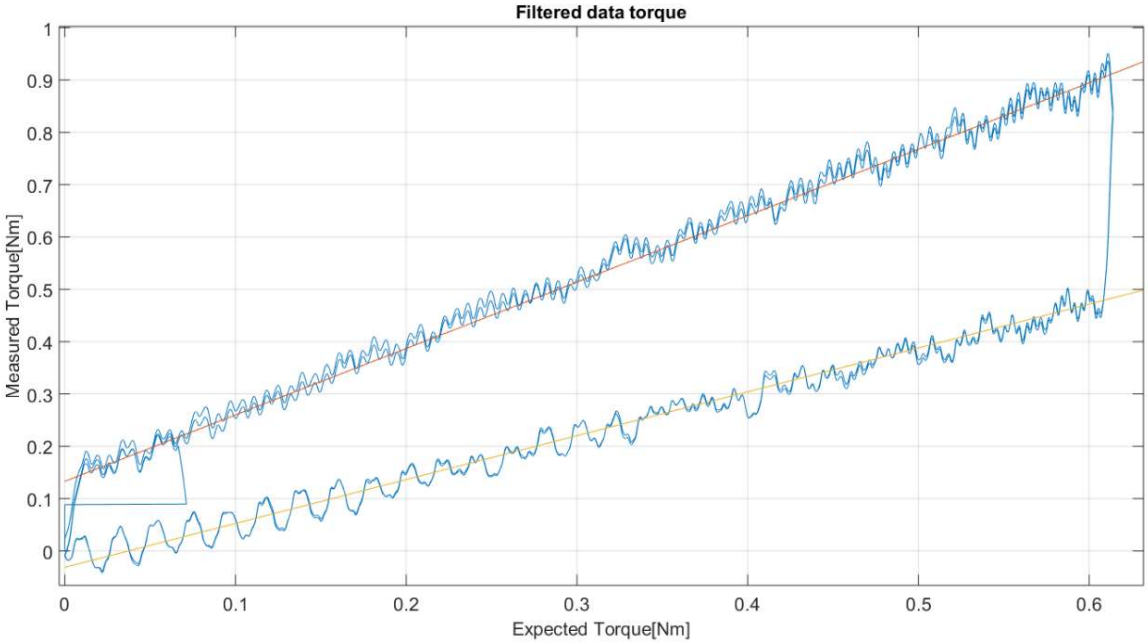

Figure 55: Measured torque in the estimated torque (filtered data)

The (Figure 55) shows the trend of the measured filtered torque in function of the expected torque calculated by (3.1).

Direct (red) and reverse (yellow) efficiencies are also reported, and this is used to assess the engine's quality in terms of performance.

This graph shows a better representation of reported split line trends.

4.3.2 Test 2: constant load wheel characterization in torque

This test is used to evaluate the presence and the magnitude of the disturbances and of the non-linearity that influences the system's behavior.

To perform this test, a pulley has been mounted on the rotation axis of a mobile platform wheel. A known load $(0.3\ kg)$ was connected to the pulley by means of cable that was winded around the pulley in order to keep the part of the cable vertical, which held the load and was out of the pulley. Therefore the torque acting on the motor axis was independent of the motor's rotation angle.

The command with which the wheel is excited in test is shown in (Figure 56).

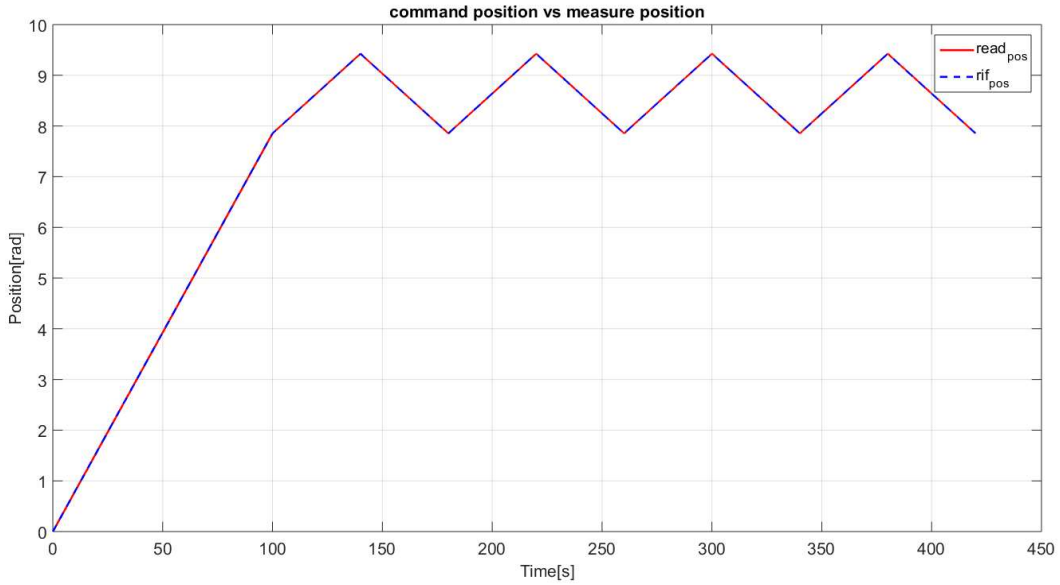

Figure 56: Command's position versus measurement's position in constant load wheel test.

This command is represented by a series of triangular waves with minimum value equal to 7.86 rad, maximum value equal to 9.43 rad and slope module value equal to 0.04 rad/s.

The cause of minimum value of the command's position different from 0 rad is the rotation to place the weight with constant distance to the wheel rotation axis.

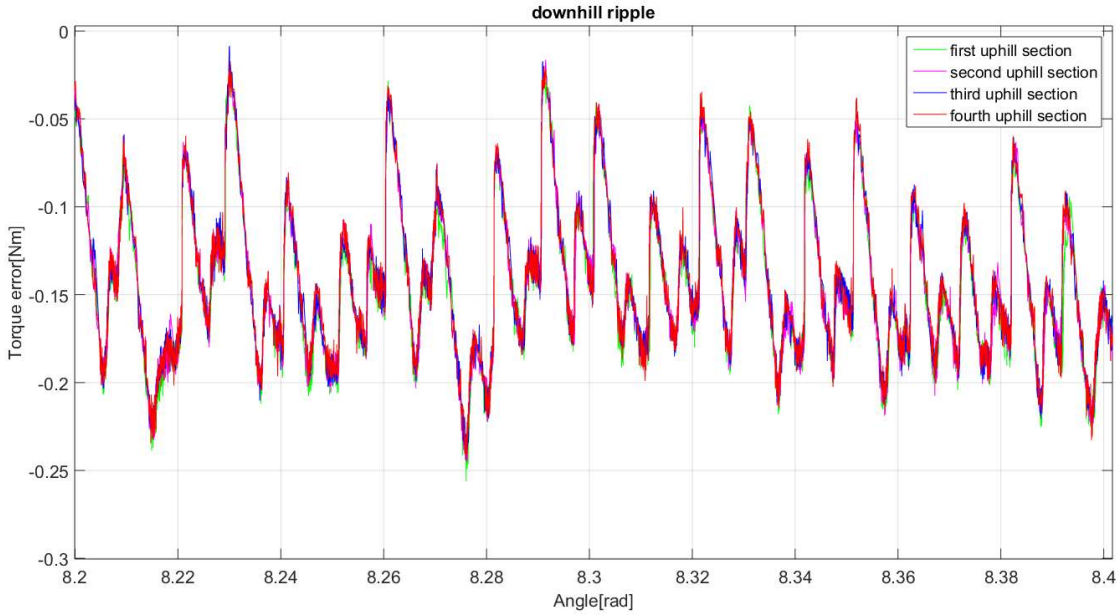

Figure 57: Downhill section error (detailed)

The graphic representation of torque's error in four downhill sections is shown in (Figure 57). This error is represented by (3.2)

$$\tau_{err} = \tau_m - M \cdot g \cdot R_P = \tau_m - 0.47 \qquad (3.2)$$

Where:

- τ_{err} is torque's error.

- τ_m is torque's measurement.
- M is load's mass.
- g is gravity's acceleration.
- R_P is pulley radius.
- 0.47 is $(M \cdot g \cdot R_P)$ value.

A portion of this error is due to the ripple, i.e. a periodic signal superimposed to the measurement, manifested through the repeatability of disturbance in the same sections.

In this case there is ripple because the signal peaks are overlapped in all the sections.

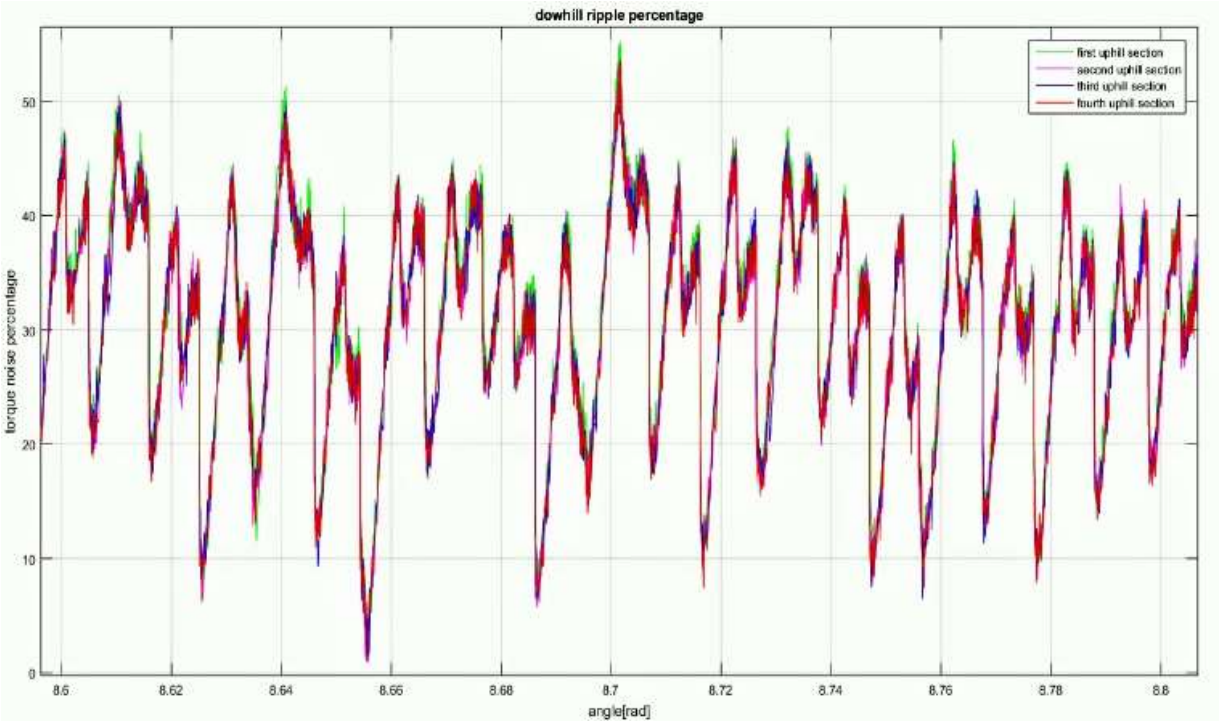

Figure 58: Percentage error of downhill section (detailed)

The trend of the torque's percentage error in four downhill sections is shown in (Figure 58).

The torque's percentage error trend is characterized by a maximum value equal to 50% and an average value equal to 30%.

A percentage error so high requires careful attention in the architecture's design control.

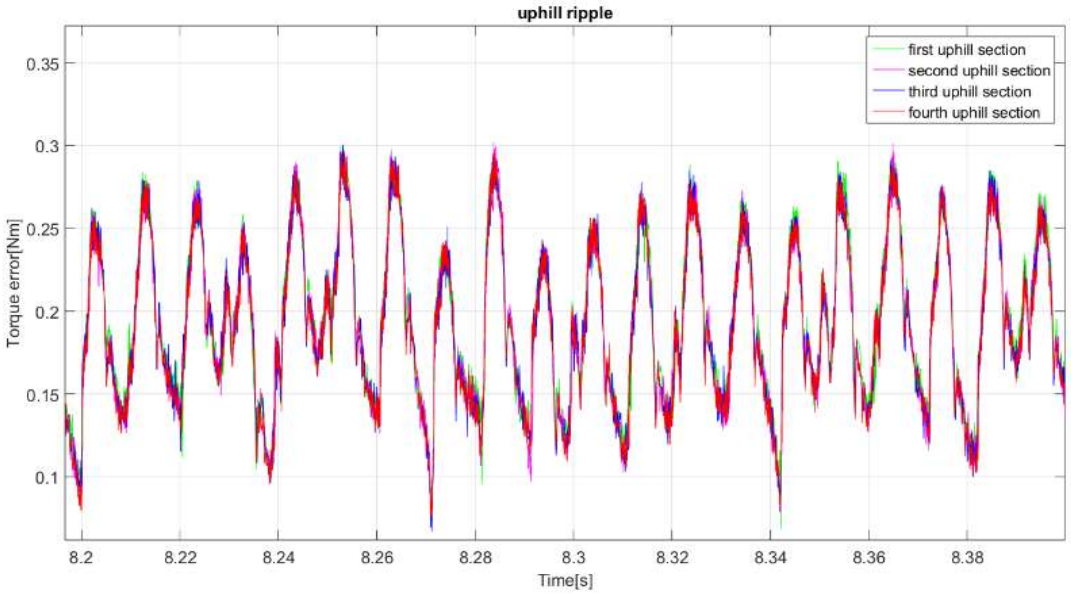

Figure 59: Uphill section error (detailed)

The graphic representation of torque's error in four uphill sections is shown in (Figure 59).

A portion of this error is due to the ripple, i.e. a periodic signal superimposed to the measurement, manifested through the repeatability of disturbance in the same sections.

In this case there is ripple because the signal peaks are overlapped in all the sections.

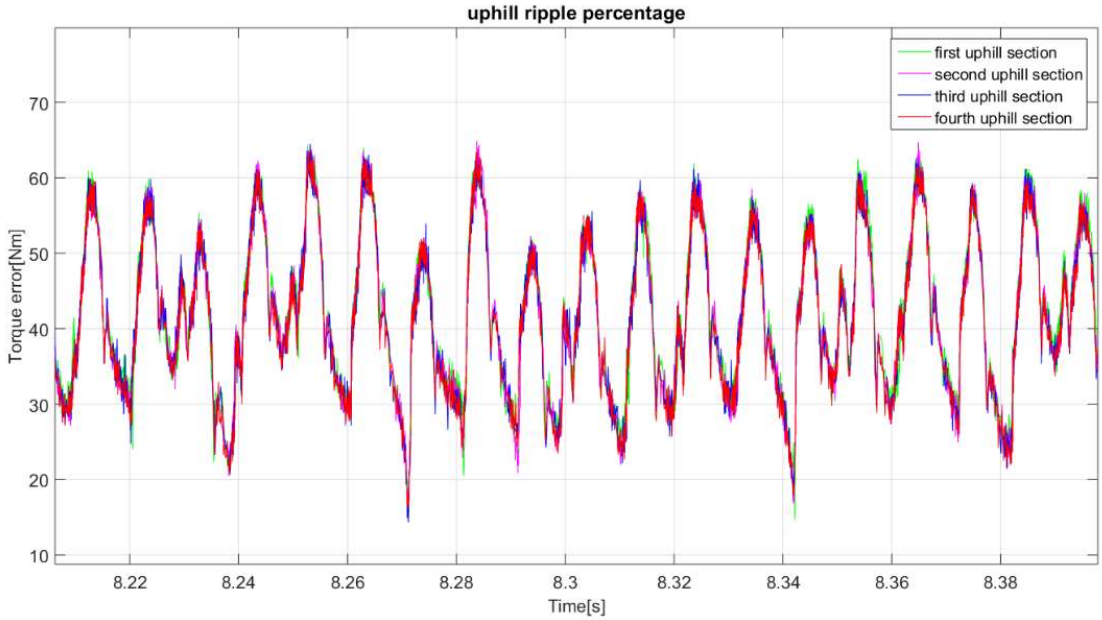

Figure 60: Percentage error of uphill section (detailed)

The trend of the torque's percentage error in four uphill sections is shown in (Figure 60).

The torque's percentage error trend is characterized by a maximum value equal to 60% and an average value equal to 40%.

A percentage error so high requires careful attention in the architecture's design control.

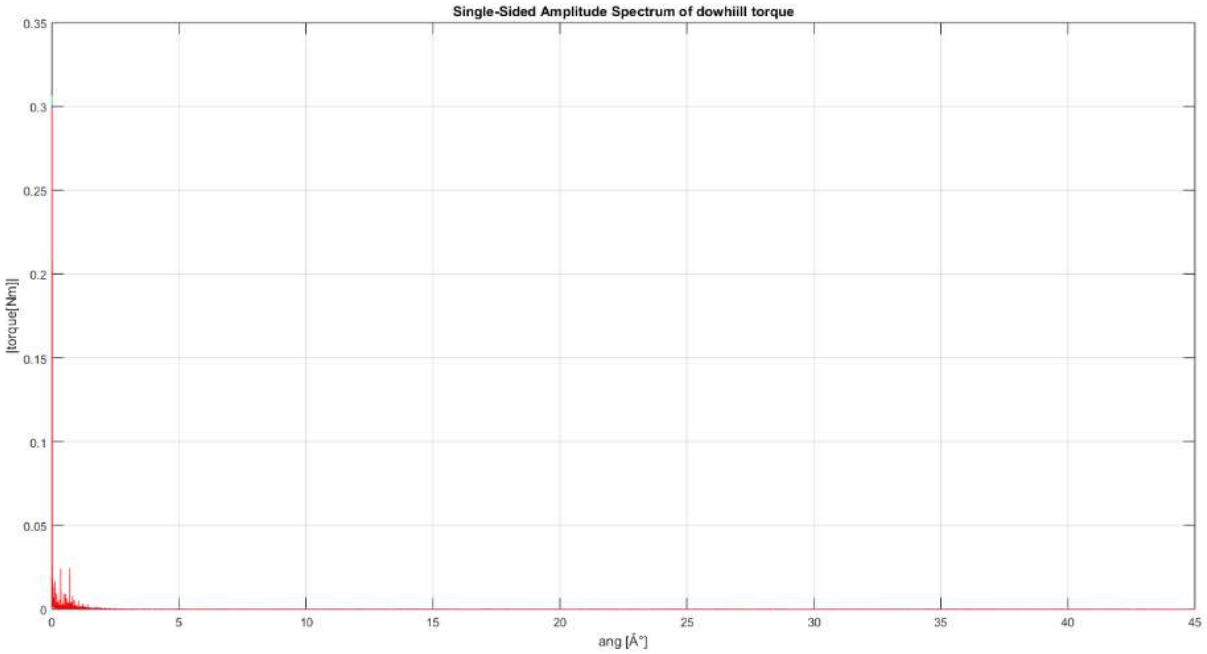

Figure 61: Downhill torque's spectral analysis

The fifth downhill torque's spectral analysis is shown in (Figure 61).
Three peaks are visible in this graph and they represent respectively:

- The first, with an approximate value of $0.3\ Nm$, the component due to the suspended mass.
- The second and the third, with an approximate value of $0.025\ Nm$, the sliding between the teeth, due to the force of gravity and to the engine (functioning as a brake).

The ripple, despite its presence, has a low value component and therefore it can be "ignored".

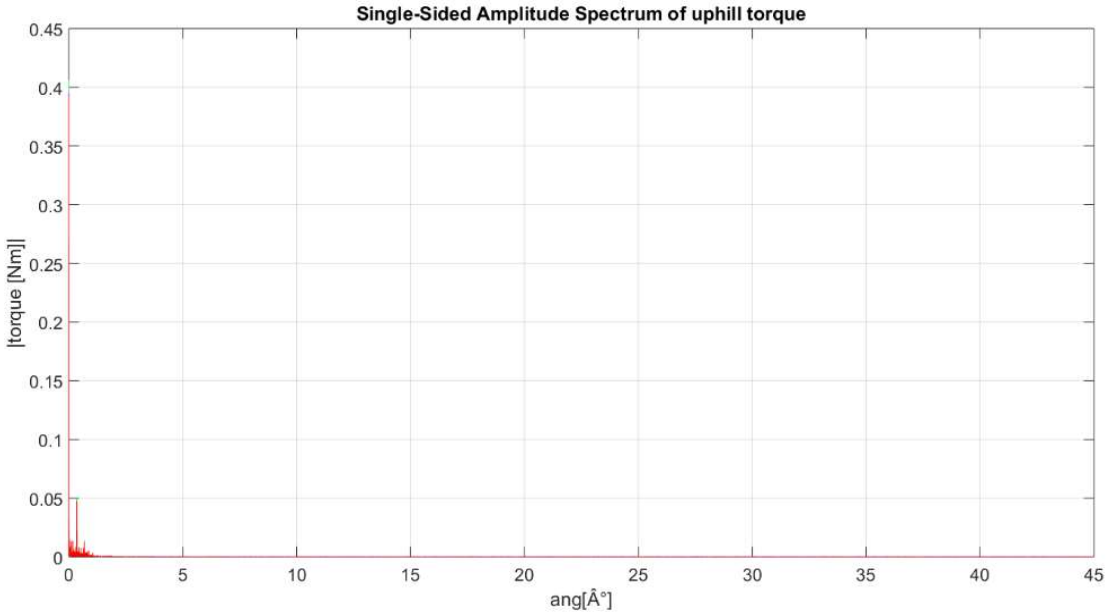

Figure 62: Uphill torque's spectral analysis

The fifth uphill torque's spectral analysis is shown in (Figure 62).

Three peaks are visible in this graph and they represent respectively:

- The first, with an approximate value of $0.4\ Nm$, the component due to the suspended mass.
- The second and the third, with an approximate value of $0.05\ Nm$ and $0.015\ Nm$, the sliding between the teeth, due to the force of gravity and to the engine.

The ripple, despite its presence, has a low value component and therefore it can be "ignored."

4.3.3 Test 3: arm's joint vs platform's joint (current's trends)

The test aim at comparing the current's trend between the arm's joint and the platform's joint (in order to evaluate a disturbing presence in the joint arm) consists in exciting each joint with a sinusoidal input, characterized by frequency equal to $0.5\ Hz$ and amplitude, respectively, equal to:

- Wheel $0.25\ A$.
- First, Second and Third arm's joint $0.25\ A$.
- Fourth arm's joint $0.175\ A$.
- Fifth arm's joint $0.15\ A$.

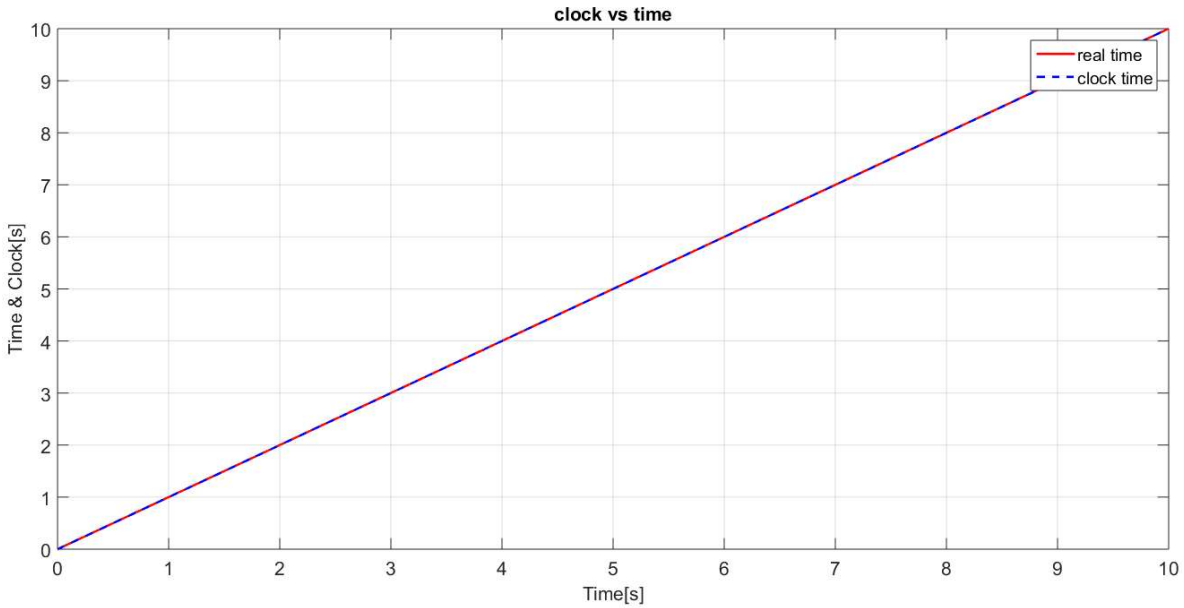

Figure 63: Clock vs real time in the compared current's test (wheel's joint)

The matlab clock and the computer time trend is shown in (Figure 63); the comparison of the two signals supports the following measurements.

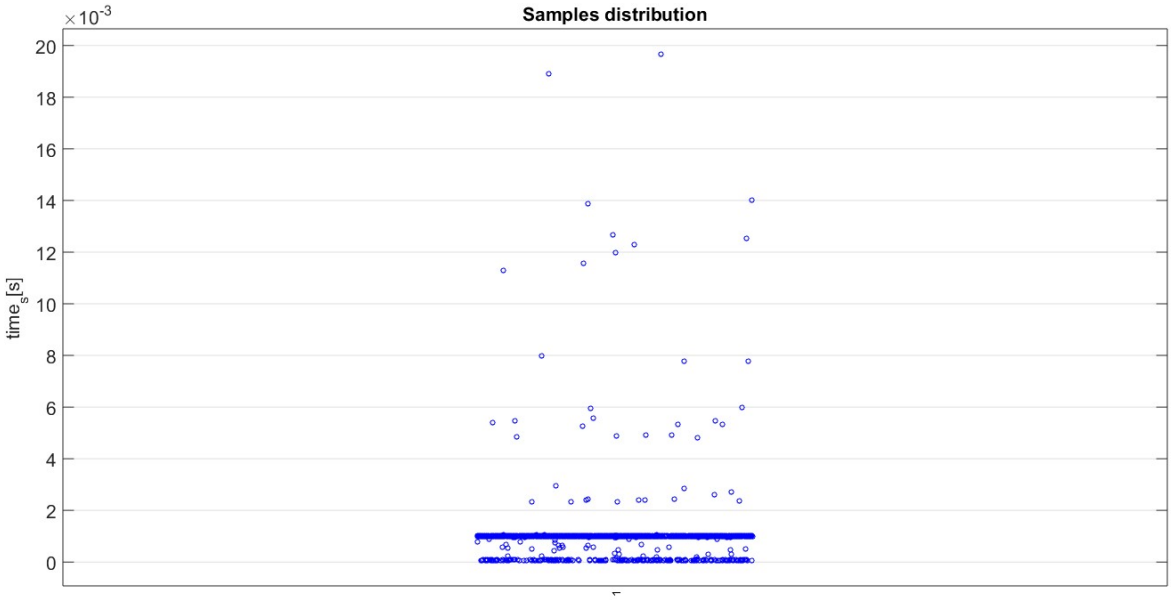

Figure 64: Sample time distribution in the compared current's test (wheel's joint)

The sample time distribution is shown in (Figure 64) and we can see that the majority of the sample time is at 1 kHz giving a further validity to the following measurements.

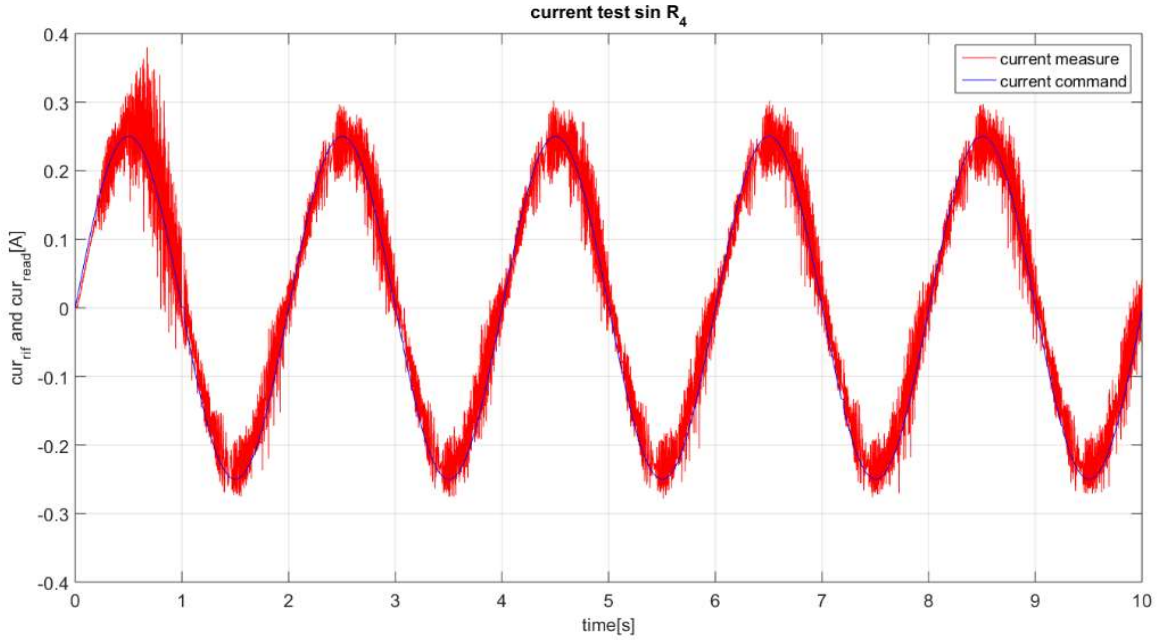

Figure 65: Current's command vs current's measurement (wheel's joint)

The current's command and the current's measurement trend, relative to a mobile platform wheel, are reported in (Figure 65).

While comparing the two trends, the presence of disturbances is visible, mainly due to the gear teeth (see par. 4.3.2).

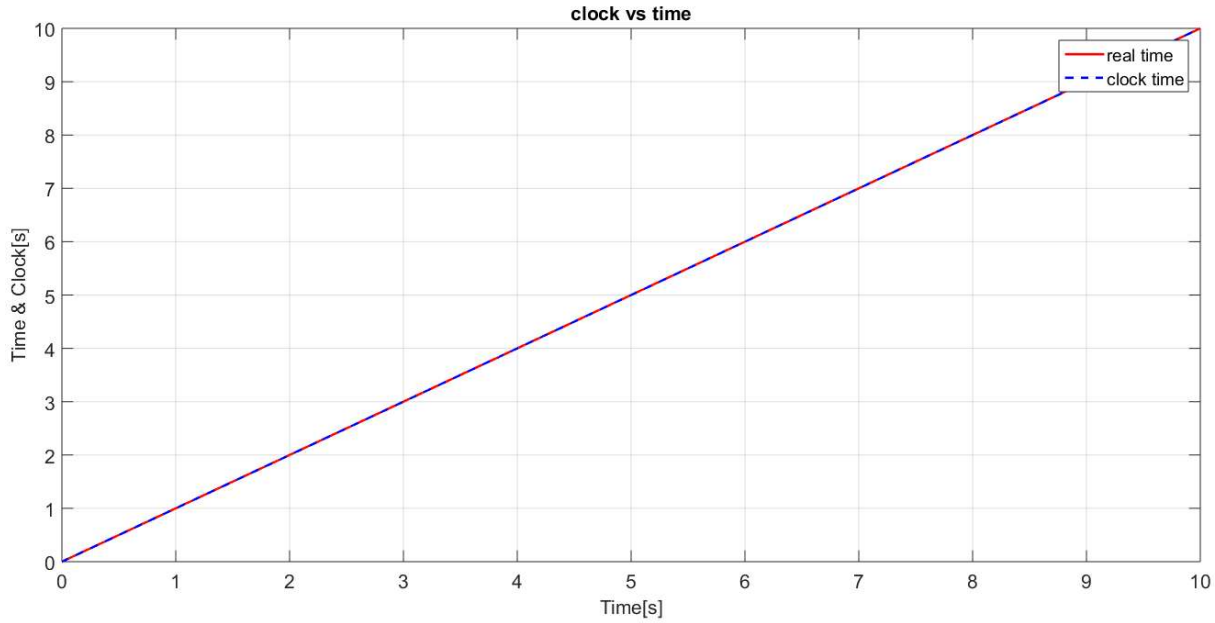

Figure 66: Clock vs real time in the compared current's test (first arm's joint)

The matlab clock and the computer time trend are shown in (Figure 66); the comparison of the two signals supports the following measurements.

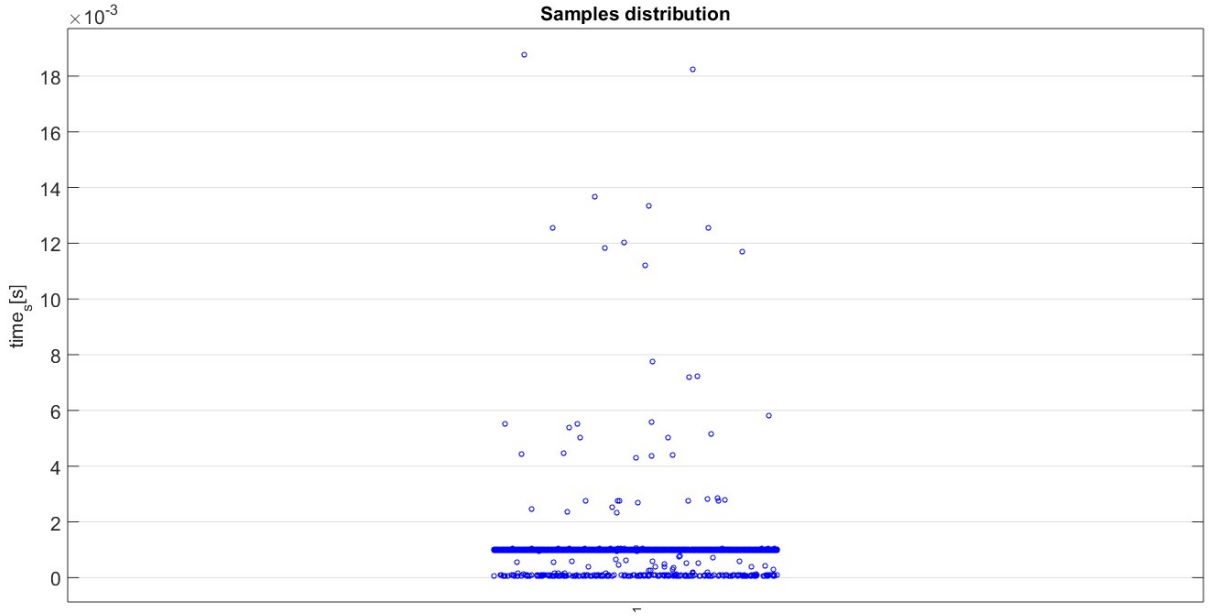

Figure 67: Sample time distribution in the compared current's test (first arm's joint)

The sample time distribution is shown in (Figure 67) and we can see that the majority of the sample time is at 1 kHz giving a further validity to the following measurements.

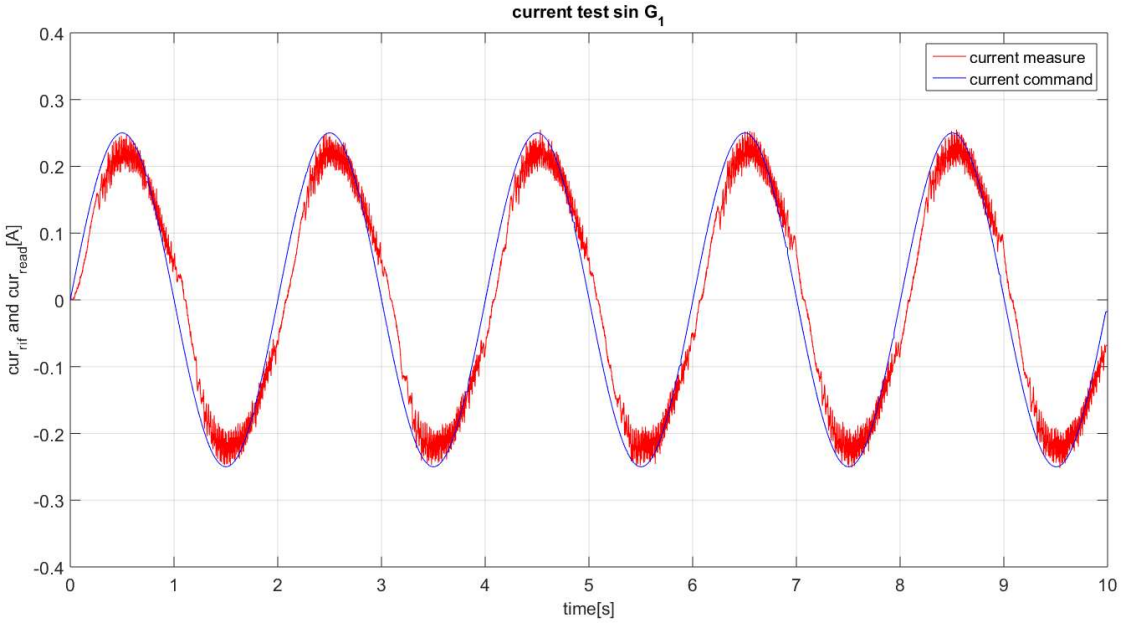

Figure 68: Current's command vs current's measurement (first arm's joint)

The current's command and the current's measurement trend, relative to the first arm's joint, are reported in (Figure 68).

By comparing the two trends (Figure 68) the presence of disturbances is visible, due to gear teeth.

If comparing the trend in (Figure 65) to that of (Figure 68), a reduction of disturbances is visible and also a better overlay.

Based on this consideration, it can be assumed that the system's behavior is better for the first arm's joint.

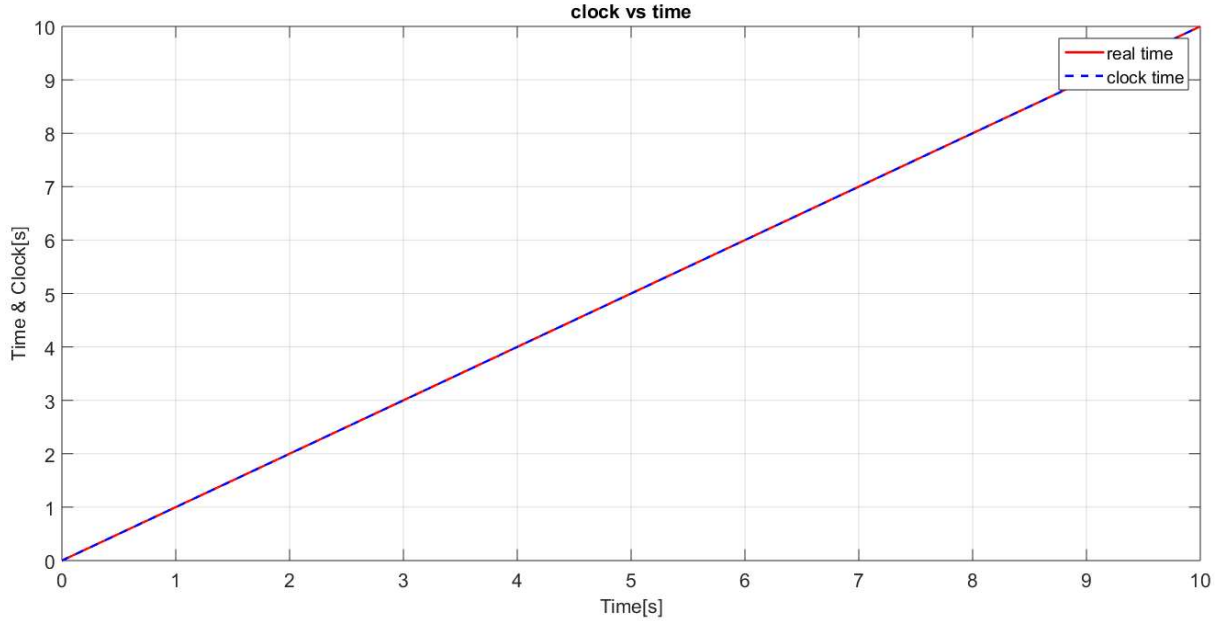

Figure 69: Clock vs real time in the compared current's test (second arm's joint)

The matlab clock and the computer time trend are shown in (Figure 69); the comparison of the two signals supports the following measurements.

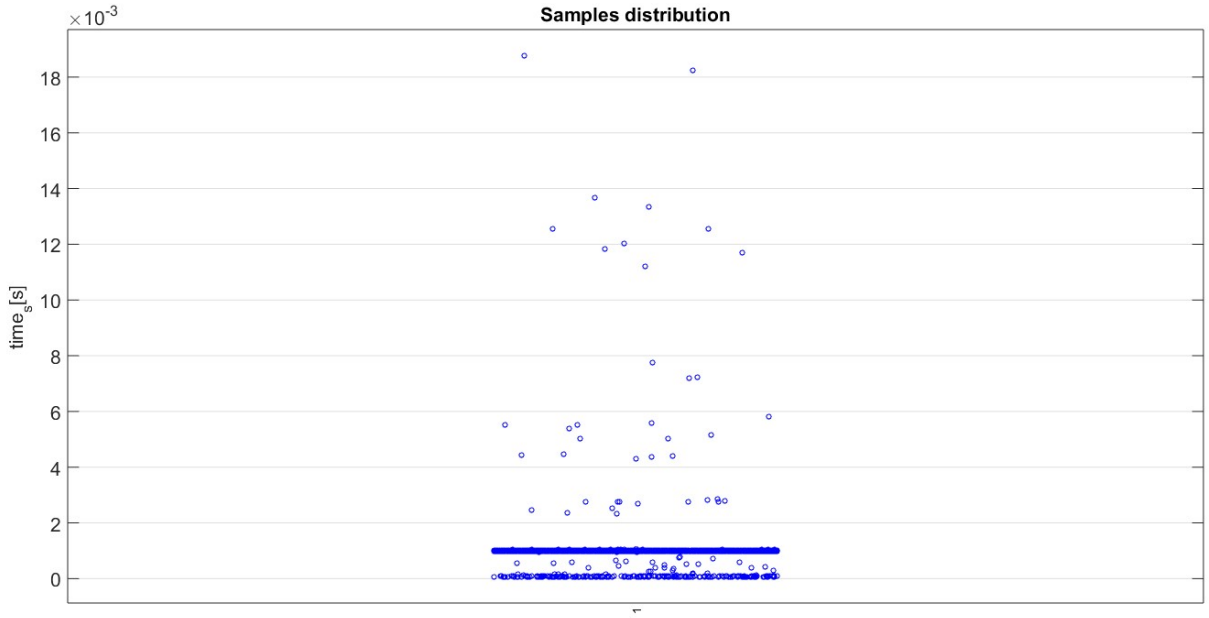

Figure 70: Sample time distribution in the compared current's test (second arm's joint)

The sample time distribution is shown in (Figure 70) and we can see that the majority of the sample time is at 1 kHz, giving a further validity to the following measurements.

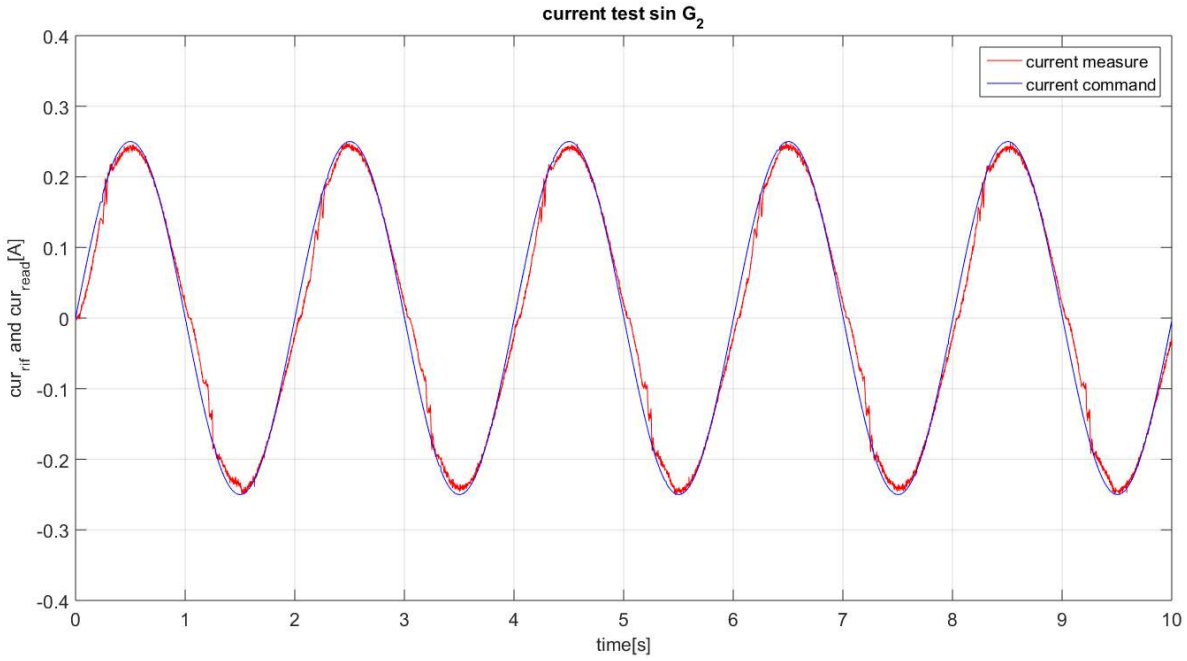

Figure 71: Current's command vs current's measurement (second arm's joint)

The current's command and the current's measurement trend, related to the second arm's joint, are reported in (Figure 71).

By comparing the two trends, a small disturbance of measurement is visible, due to the principal component: gear teeth, seen previously (see par. 4.3.2).

If comparing the trends of (Figure 65) with those of (Figure 71) a reduced disturbance's presence is visible and has a greater overlap.

Based on this consideration, it can be assumed that the system's behavior is better for the second arm's joint.

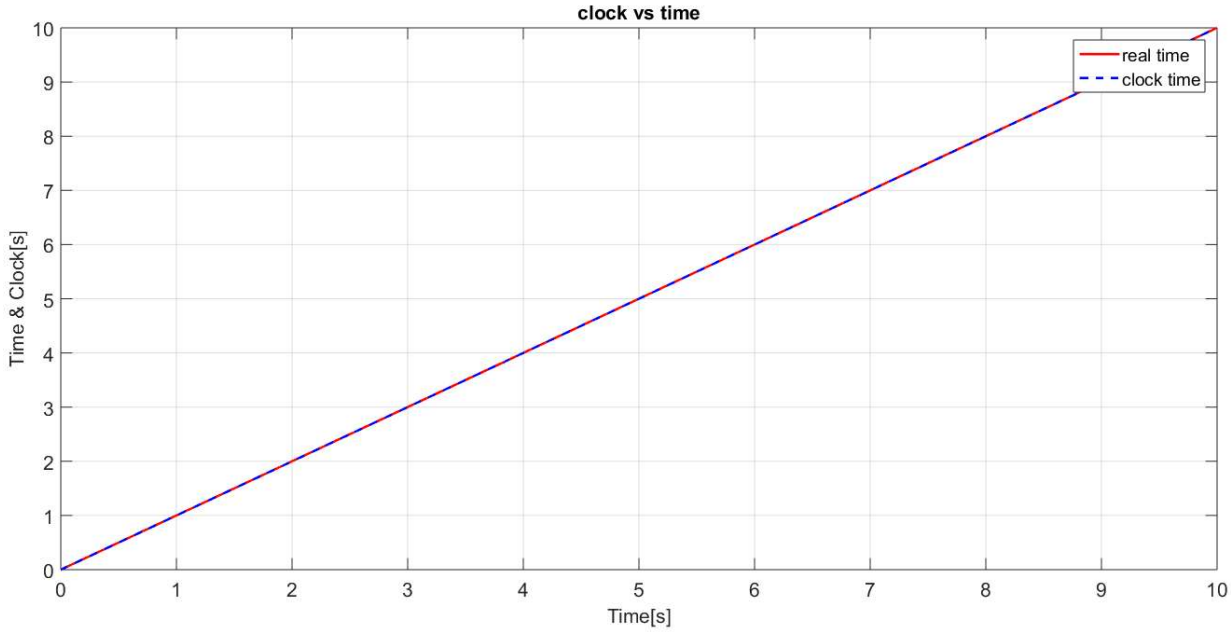

Figure 72: Clock vs real time in the compared current's test (third arm's joint)

The matlab clock and the computer time trend are shown in (Figure 72); the comparison of the two signals supports the following measurements.

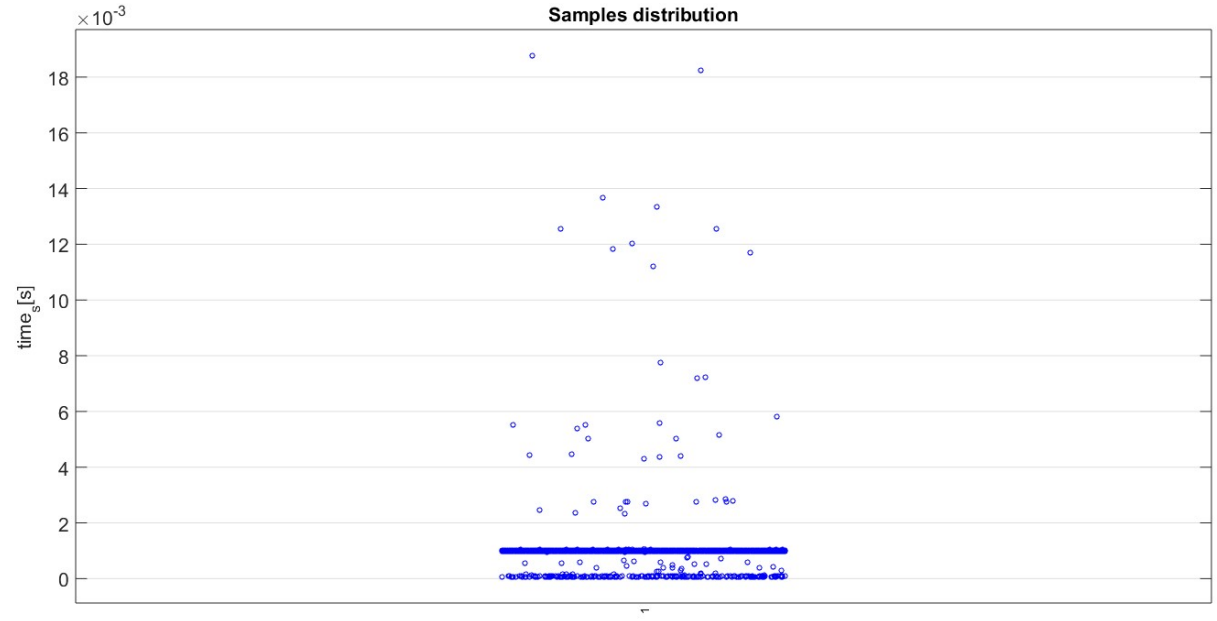

Figure 73: Sample time distribution in the compared current's test (third arm's joint)

The sample time distribution is shown in (Figure 73) and we can see that the majority of the sample time is at 1 kHz, giving a further validity to the following measurements.

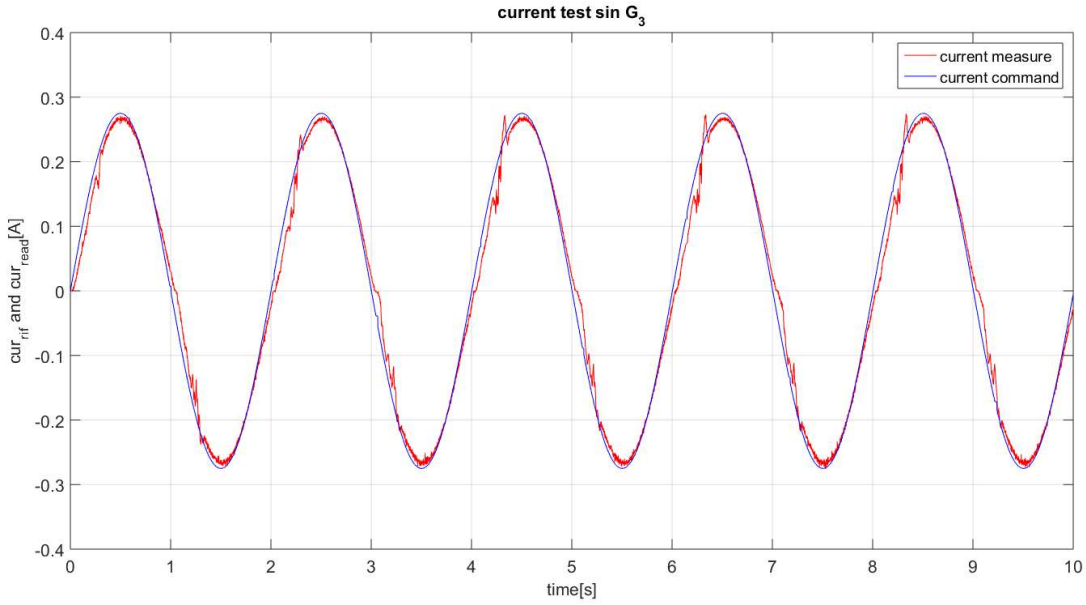

Figure 74: Current's command vs current' measurement (third arm's joint)

The current's command and the current's measure trend, related to the third arm's joint, are reported in (Figure 74).

By comparing the two trends, a small disturbance measurement is visible, mainly in the impulse form, and the principal component are the gear teeth (see par. 4.3.2).

Instead, comparing the trends of (Figure 65) with those of (Figure 74), a reduced disturbance presence is visible, as well as a greater overlap.

Based on this consideration, it can be assumed that the system's behavior is better for the third arm's joint.

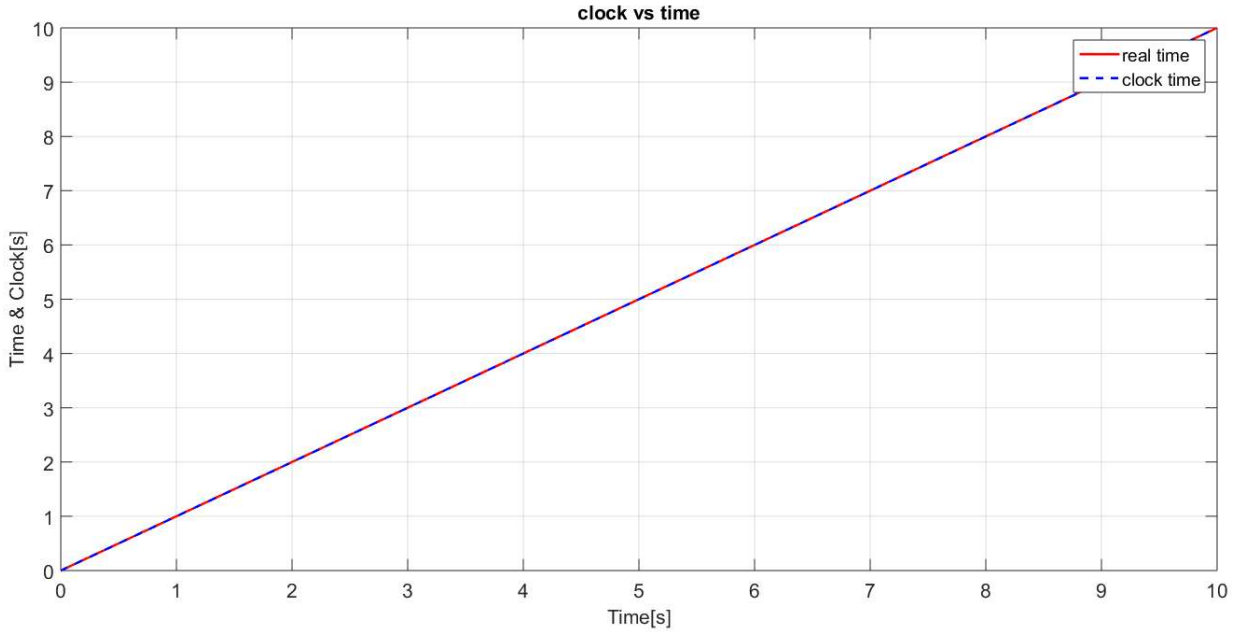

Figure 75: Clock vs real time in the compared current's test (fourth arm's joint)

The matlab clock and the computer time trend are shown in (fFigure 75); the comparison of the two signals supports the following measurements.

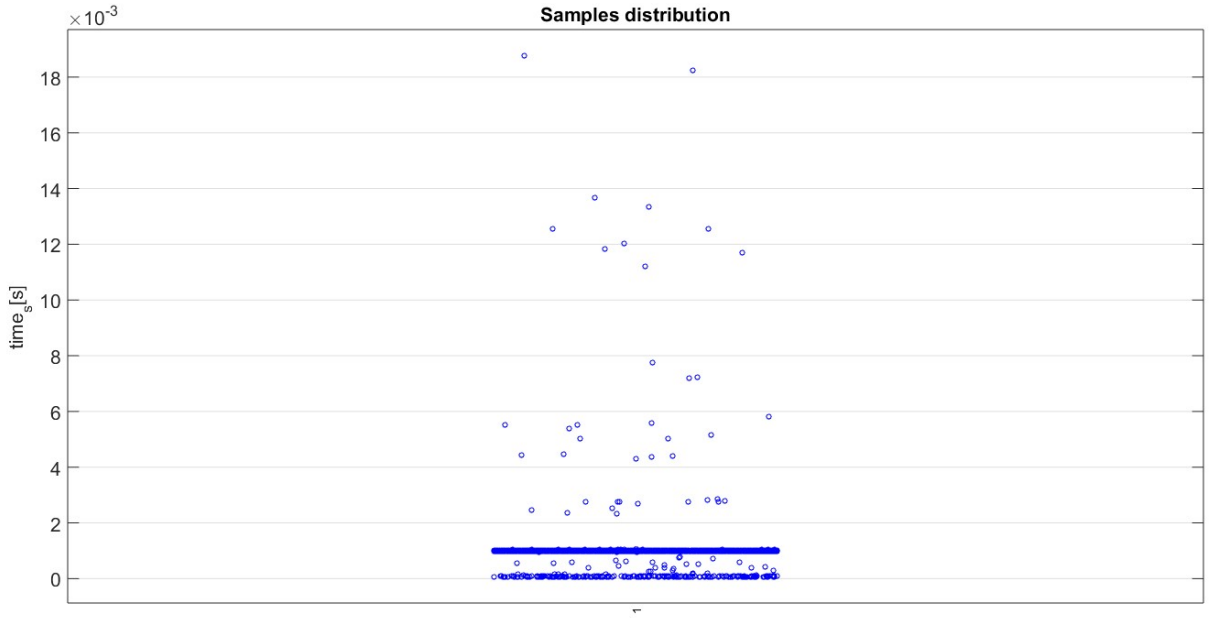

Figure 76: Sample time distribution in the compared current's test (fourth arm's joint)

The sample time distribution is shown in (Figure 76) and we can see that the majority of the sample time is at 1 kHz, giving a further validity to the following measurements.

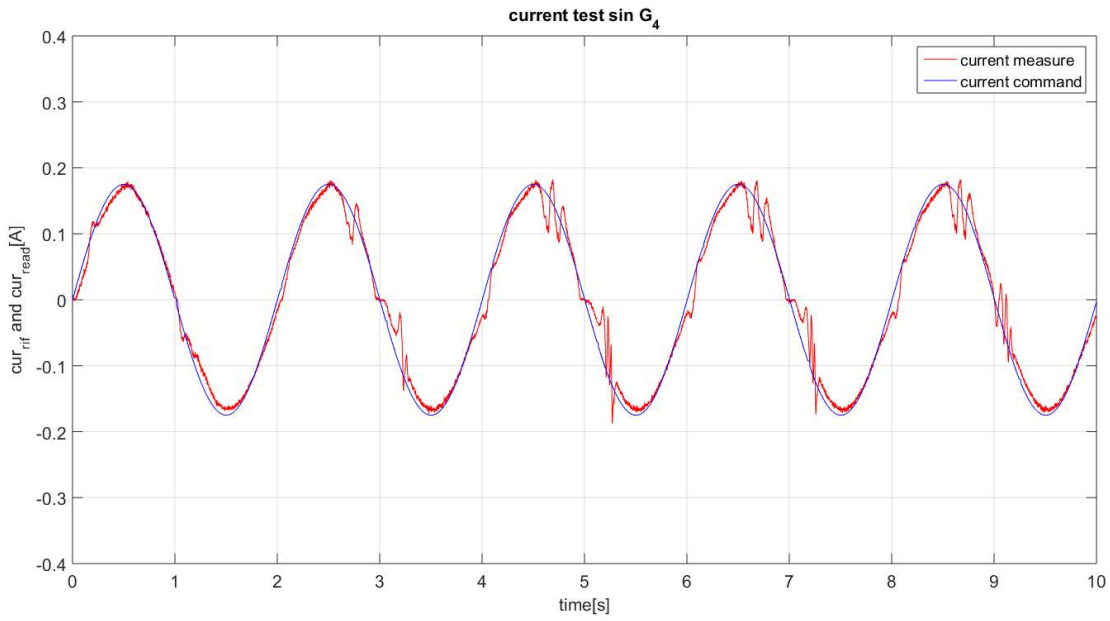

Figure 77: Current's command vs current's measurement (fourth arm's joint)

The current's command and the current's measurement trend, related to the fourth arm's joint, are reported in (Figure 77).

By comparing the two trends, a disturbance measurement is visible, mainly in the impulse form, due to lack of gravity's compensation.

In fact when it is slipping on gear teeth, there is a peak with this absence, and its magnitude is greater in correspondence with the changes of direction.

Instead, comparing the trends of (Figure 65) with those of (Figure 77), a small but significantly reduced disturbance's presence is visible, as well as a greater overlap.

Based on this consideration, it can be assumed that the system's behavior is better for the fourth arm's joint.

In this analysis, the fifth joint isn't included because it is an unessential joint for the present project.

4.4 The sampling test

In this test, the coherence between the current's command and the current's response have been evaluated to vary from sampling frequency.

This test is useful to find out the optimal sampling frequency.

In the following test, the SOEM driver (low level accessible driver) with MatLab have been interfaced through "mex" blocks (see Appendix B).

The current's command is a sine wave with an amplitude equal to $350\ mA$ and period equal to $4\ sec$.

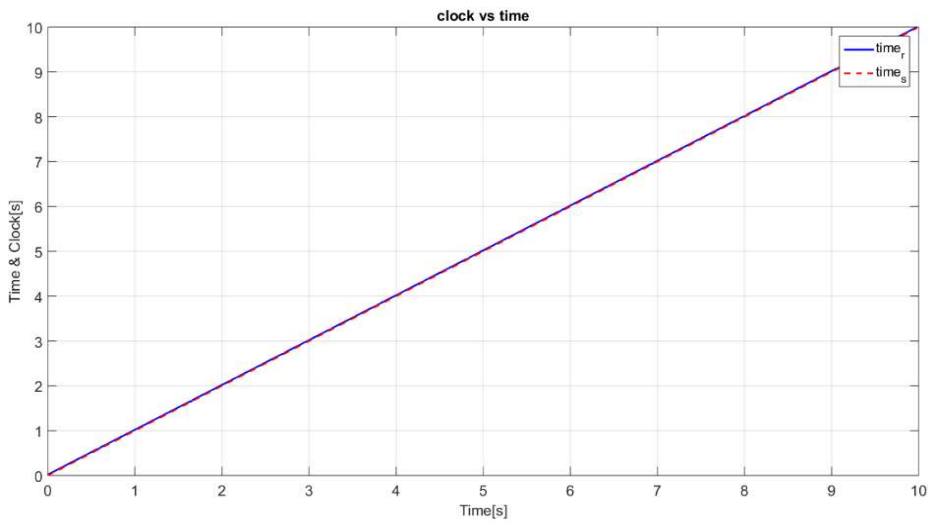

Figure 78: Clock vs real time in the sampling test (**50 *Hz***)

The matlab clock and the computer time trend are shown in (Figure 78); its overlap validates the following measurements.

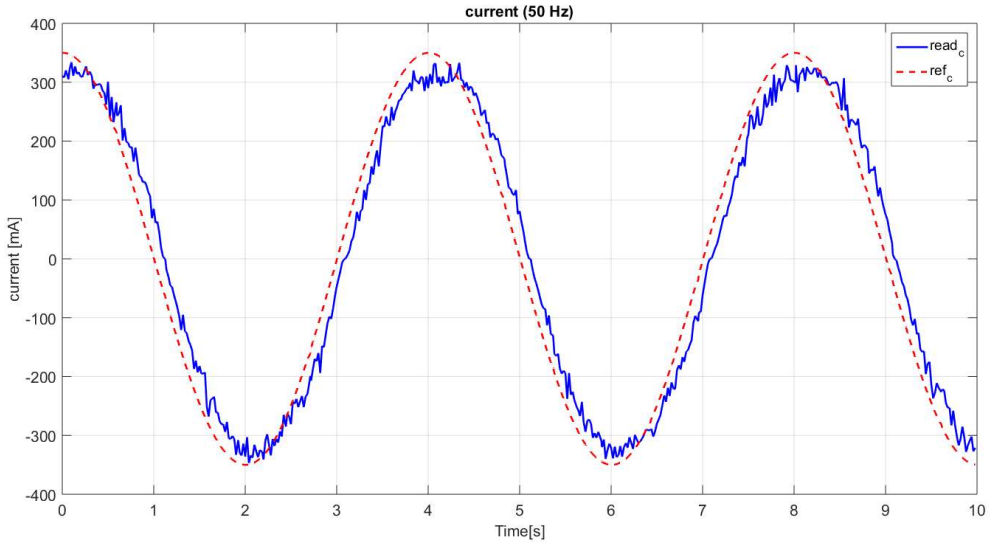

Figure 79: Current's command vs current's response in the sampling test (**50 *Hz***)

The trend of the current's command and the current's system response are shown in (Figure 78).

In the figure it is visible that:

- The consistency between the two trends is characterized by an average error of 40 mA equal to 11% of the signal amplitude (350 mA).
- An oscillating behavior around an average trend.

This oscillating behavior is due to the imperfect commands' reception from the microcontroller.

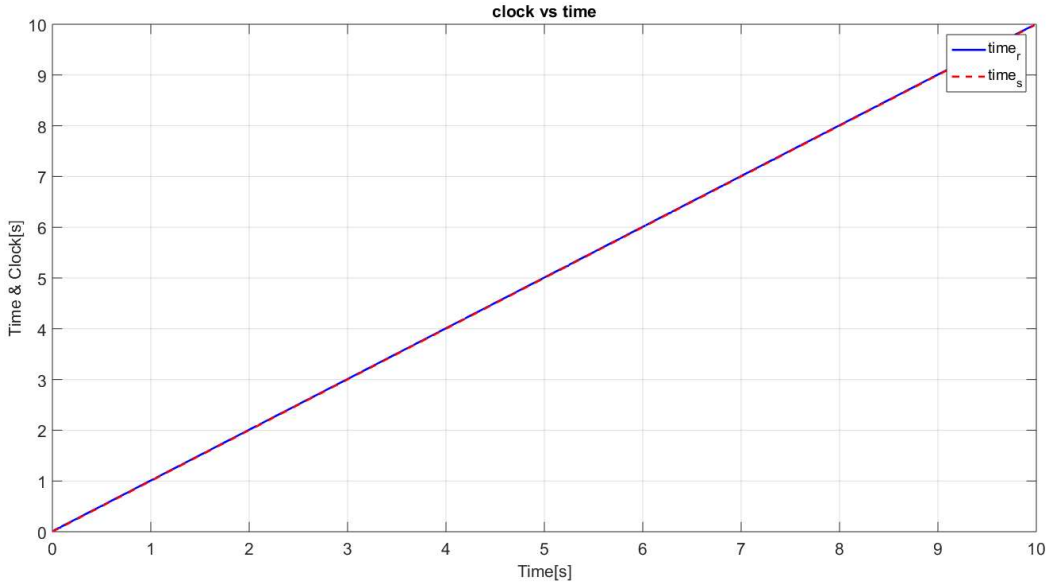

Figure 80: Clock vs real time in the sampling test (**100 *Hz***)

The matlab clock and the computer time trend are shown in (Figure 80); its overlap validates the following measurements.

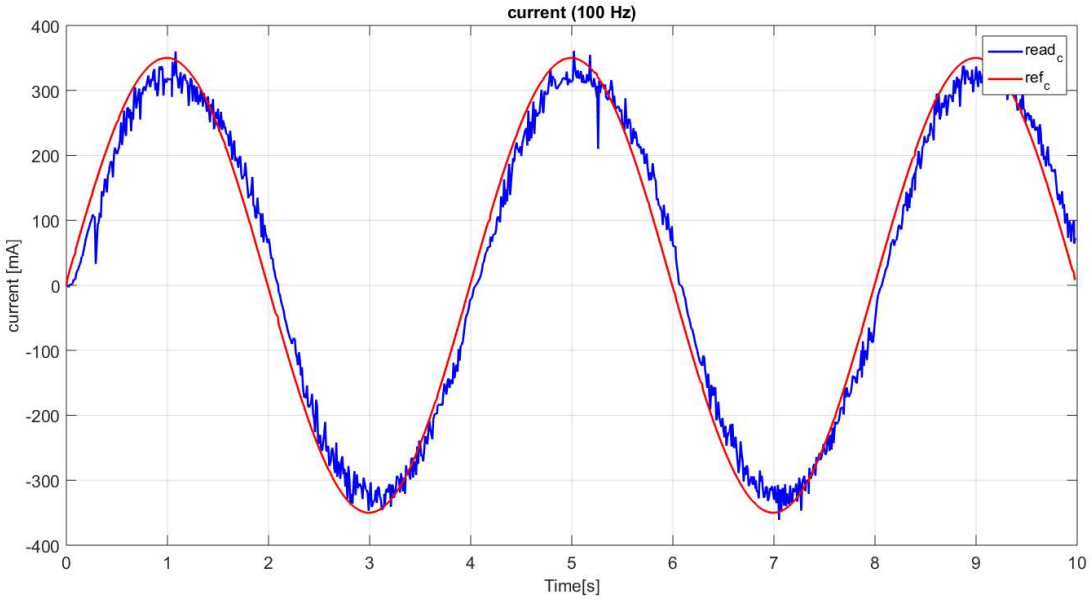

Figure 81: Current's command vs current's response in the sampling test ($\mathbf{100\ Hz}$)

The trend of the current's command and the current's system response are shown in (Figure 81).

In the figure, the following aspects are visible:

- The consistency between two trends is characterized by an average error of 32 mA equal to 9% of the signal amplitude (350 mA).
- An oscillating behavior around an average trend.

This oscillating behavior is due to the imperfect commands' reception from the microcontroller.

Between the trend in (Figure 79) and that in (Figure 81) a deterioration is visible in the latter; fortunately the current's system response is still sufficiently consistent with the current's command.

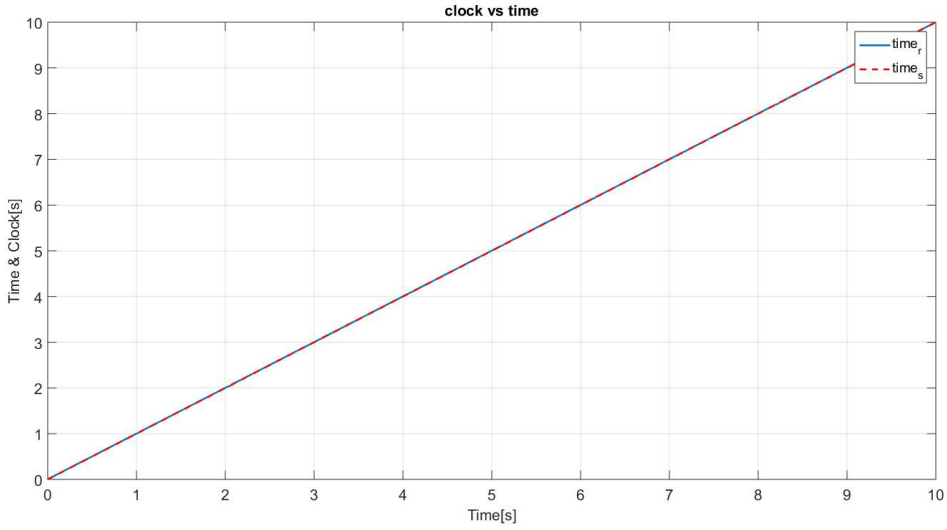

Figure 82: Clock vs real time in the sampling test ($\mathbf{1\ kHz}$)

The matlab clock and the computer time trend are shown in (Figure 82); its overlap validates the following measurements.

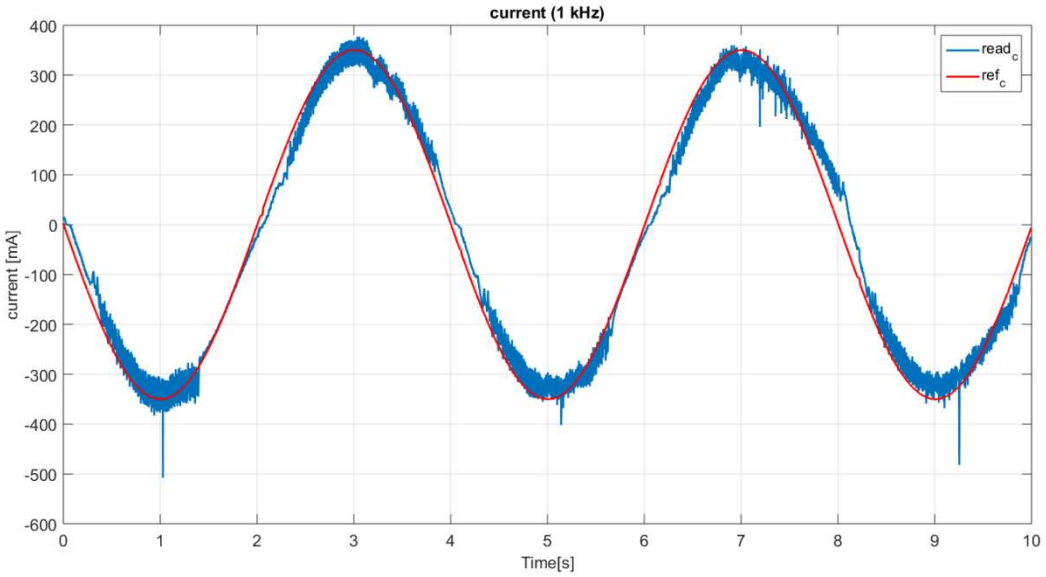

Figure 83: Current's command vs current's response in the sampling test (**1 kHz**)

The trend of the current's command and the current's system response are shown in (Figure 83).

In the figure, the following aspects are visible:

- The consistency between two trends is characterized by an average error of 25 mA equal to 7% of the signal amplitude (350 mA).
- An oscillating behavior around an average trend.
- Some very high peaks compared to the average value.

This oscillating behavior is due to the imperfect commands' reception from the microcontroller.

Between the trend in (Figure 81) and that in (Figure 83) a deterioration is visible in the latter; fortunately the current's system response is still sufficiently consistent with the current's command, nevertheless the presence of some very high peaks compared to the average value.

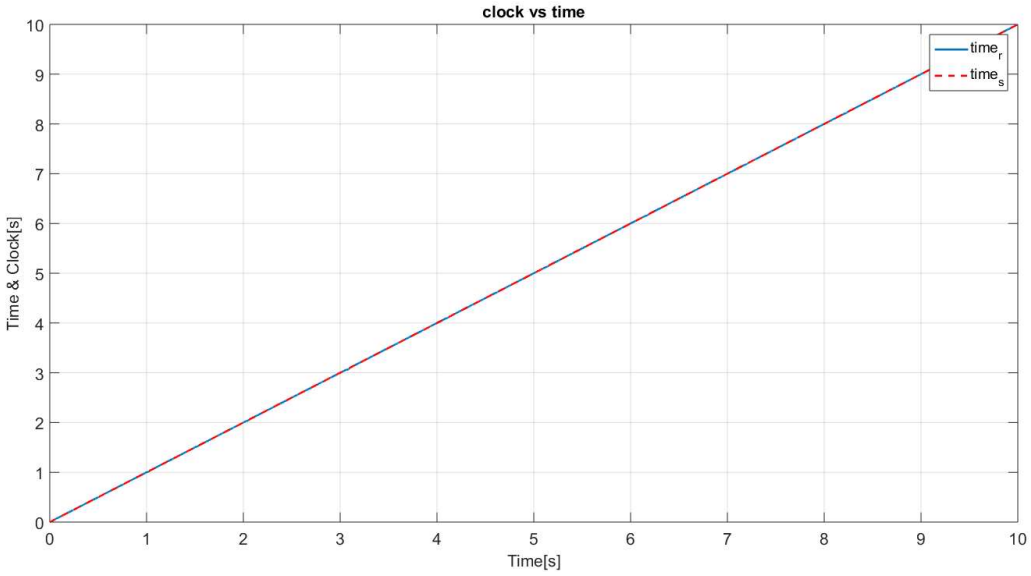

Figure 84: Clock vs real time in the sampling test (**2 kHz**)

The matlab clock and the computer time trend are shown in (Figure 84); its overlap validates the following measurements.

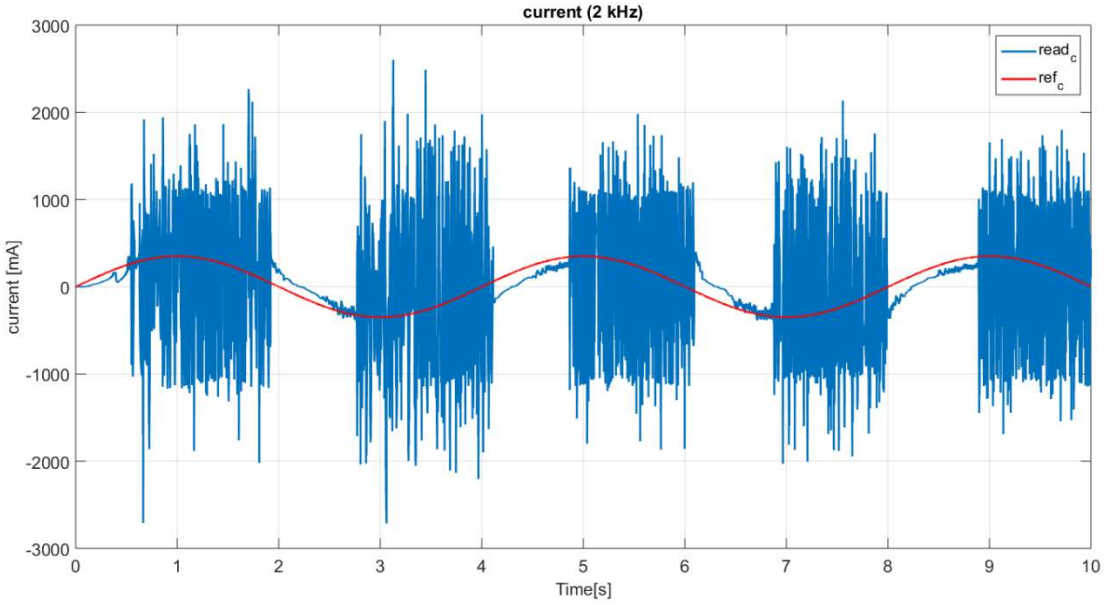

Figure 85: Current's command vs current's response in the sampling test (**2 kHz**)

The trend of the current's command and the current's system response are shown in (Figure 85).

In the figure, the following aspects are visible:

- The inconsistency between the two trends is characterized by an average error of 394 mA equal to 112% of the signal amplitude (350 mA).
- The presence of very high peaks compared to the average value.

The peaks' presence is due to the imperfect commands' reception from the microcontroller.

Between the trend in Figure 83) and that in (Figure 84) a deterioration is visible in the latter, in fact in (Figure 84) very high peaks are seen much frequently.

Because of this, a sampling frequency has been chosen equal to $1\ kHz$.

4.5 Summary of results obtained during tests

The tests show different behaviors between the omnidirectional platform's joint (wheels) and the arm's joints.

This difference is due to the use of two different mechanisms. In fact, the omnidirectional platform moves on a surface and it doesn't have the high performance request, while the arm's joints must correctly approach the object in space and they need high performance (in terms of noise rejection and control).

The transition from a control loop with a frequency of $25/50\ Hz$, to one with a frequency of $1\ kHz$ (rectified), allows to get a controller with the best of performances.

This improvement leads to an opportunity to make the most of current's control loop.

It is important to remember the presence of the jitter, as it evaluates the main components in (par. 4.1), and of the ripple, which fortunately hasn't a significant impact on the torque (see par 4.2 e 4.3).

The maximum attention should be in the sampling frequency choice (see par. 4.4), because there is a risk of producing inconsistent measurements.

5 THE NEW DECENTRALIZED CONTROL

This chapter is the project's heart and it is divided into two macro-blocks:

1) In the first macro-block, the new decentralized KUKA youBot's arm control is described in all its parts.
2) In the second macro-block, the center of mass (COM) of individual links have been identified. The mass, inertia and link length are identified by the KUKA youBot's User Manual (KUKA, 2013) and graphical representations.

5.1 Decentralized youBot's arm control

In this paragraph, the new decentralized youBot's arm control is described and it is the main topic of this work.

The initial situation, involving a centralized control, is explained before (see par 3.1).

The starting point of new control is the hybrid force/position control represented in (Figure 86).

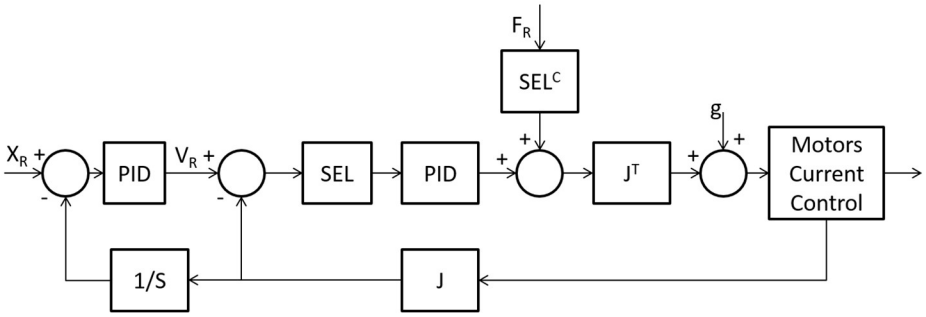

Figure 86: Hybrid force/position control

From the diagram (Figure 86), the complementary selection matrix (SEL and SEL^C) could be seen. It establishes, respectively, the amount of the position's control and the torque's control.

The gravity's compensation (g) is out of the control's loop, because if the selection matrix (SEL^C) has a value different to 1, the system isn't fully compensate.

The setpoints are:

- X_R i.e. the end-effector position respect to frame 0.
- F_R, i.e. the interaction between end-effector and the external environment.

My control solution is shown in (Figure 87)

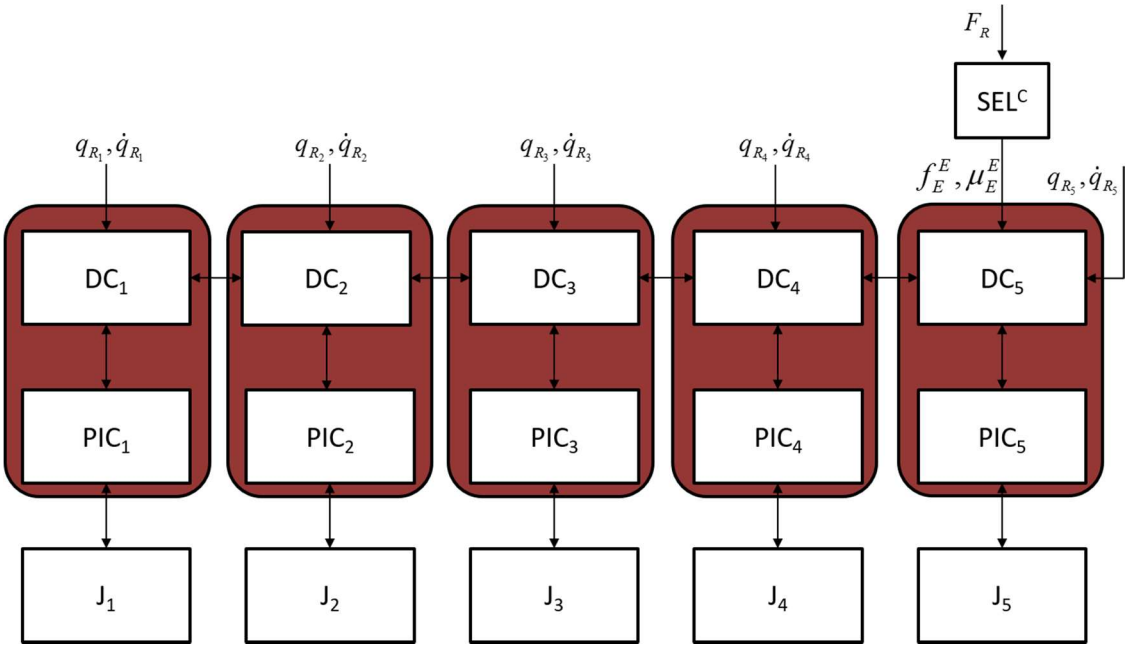

Figure 87: Decentralized youBot's arm control

The DC block represents the decentralized control that could be implemented directly on the microcontrollers, if their architecture is an open source,

The improvement, respecting the previous control, is the absence of the central unit. This reduces system's weight and improves system's performance because it simplifies communication.

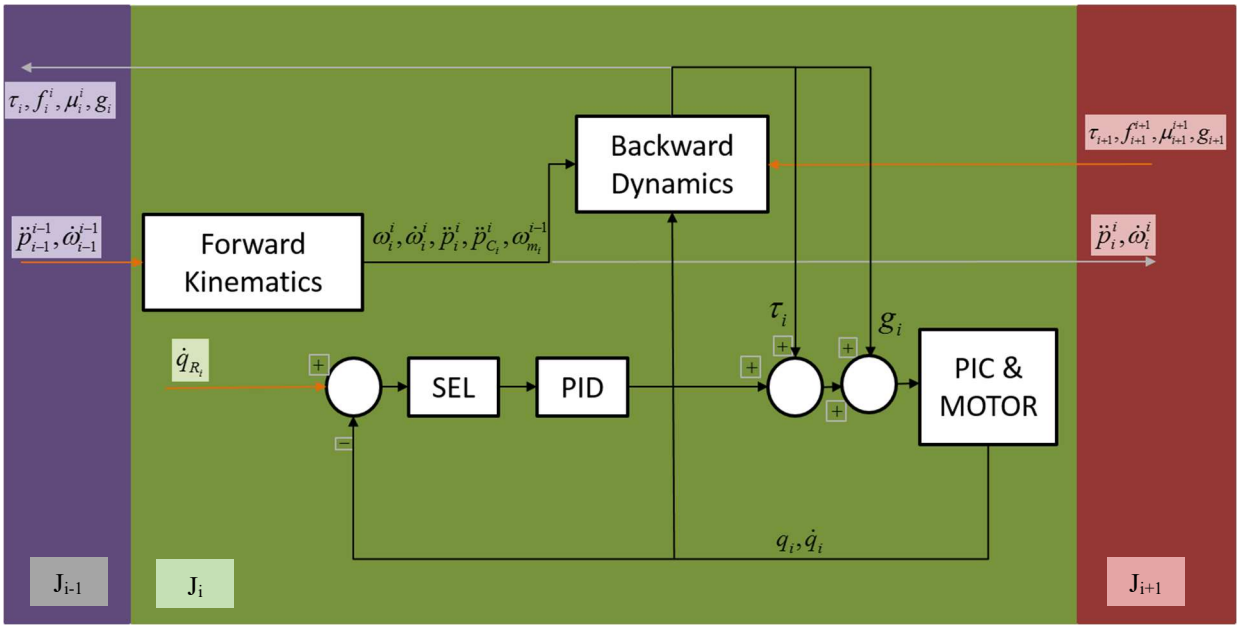

Figure 88: DC block

The architecture of DC block is shown in (Figure 88):

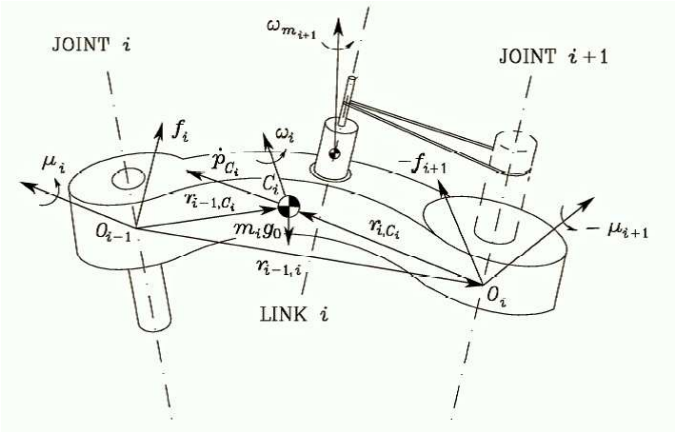

Figure 89: Characterization of link ***i.***

It is characterized, with reference in (Figure 89), by the following input (Siciliano B., 2008)

- $\ddot{p}_{i-1}^{i-1}$ linear acceleration of O_{i-1}.
- $\dot{\omega}_{i-1}^{i-1}$ angular acceleration of O_{i-1}.
- f_{i+1}^{i+1} force exerted by link $i+1$ on link i.
- μ_{i+1}^{i+1} moment exerted by link $i+1$ on link i with respect to O_{i+1}.
- τ_{i+1} generalized force at joint $i+1$.
- g_{i+1}gravity compensation at joint $i+1$.
- q_{R_i} angular position setpoint at joint i – th.
- $\dot{q}_{R_i}$ angular velocity setpoint at joint i – th.

and, with reference in (Figure 89), by the following output:

- $\ddot{p}_i^i$ linear acceleration of O_i.
- $\dot{\omega}_i^i$ angular acceleration of O_i.
- f_i^i force exerted by link i on link $i-1$.
- μ_i^i moment exerted by link i on link i with respect to O_i.
- τ_i generalized force at joint i.
- g_i gravity compensation at joint i.

The "forward kinematics" vector equations are the following (5.1).

$$\begin{bmatrix} \omega_i^i \\ \dot{\omega}_i^i \\ \ddot{p}_i^i \\ \ddot{p}_{C_i}^i \\ \dot{\omega}_{m_i}^{i-1} \end{bmatrix} = \begin{bmatrix} {R_i^{i-1}}^T\left(\dot{\omega}_{i-1}^{i-1} + \dot{\theta}_i z_0\right) \\ {R_i^{i-1}}^T\left(\dot{\omega}_{i-1}^{i-1} + \ddot{\theta}_i z_0 + \dot{\theta}_i \omega_{i-1}^{i-1}\, x\, z_0\right) \\ {R_i^{i-1}}^T \ddot{p}_{i-1}^{i-1} + \dot{\omega}_i^i\, x\, r_{i-1,i}^i + \omega_i^i\, x\, \left(\omega_i^i\, x\, r_{i-1,i}^i\right) \\ \ddot{p}_i^i + \dot{\omega}_i^i\, x\, r_{i,C_i}^i + \omega_i^i\, x\, \left(\omega_i^i\, x\, r_{i,C_i}^i\right) \\ \dot{\omega}_{i-1}^{i-1} + k_{ri}\ddot{q}_i z_{m_i}^{i-1} + k_{ri}\dot{q}_i \omega_{i-1}^{i-1}\, x\, z_{m_i}^{i-1} \end{bmatrix} \tag{5.1}$$

The "backward dynamics" vector equations are the following (5.2 – 5.4).

$$\begin{bmatrix} f_i^i \\ \mu_i^i \end{bmatrix} = \begin{bmatrix} R_{i+1}^i f_{i+1}^{i+1} + m_i \ddot{p}_{C_i}^i \\ -f_i^i \, x \, (r_{i-1,i}^i + r_{i,C_i}^i) + R_{i+1}^i \mu_{i+1}^{i+1} + R_{i+1}^i f_{i+1}^{i+1} \, x \, r_{i,C_i}^i + \bar{I}_i^i \dot{\omega}_i^i + \omega_i^i \, x \, \bar{I}_i^i \omega_i^i + k_{r,i+1} \ddot{q}_{i+1} I_{m_{i+1}} z_{m_{i+1}}^i + k_{r,i+1} \dot{q}_{i+1} I_{m_{i+1}} \, x \, z_{m_{i+1}}^i \end{bmatrix} \quad (5.2)$$

$$\tau_i = \mu_i^{i^T} R_i^{i-1^T} z_0 + k_{ri} I_{m_i} \omega_{m_i}^{i-1^T} z_{m_i}^{i-1} + F_{vi} \dot{\theta}_i + F_{si} sgn(\dot{\theta}_i) \quad (5.3)$$

$$g_i = g_{i+1} + (\sum_{k=i+1}^{n} m_k) d_{i+1} g_0 + m_i d_i g_0 \quad (5.4)$$

Where:

- m_i mass of augmented link.
- $\bar{I}_i$ inertial tensor of augmented link.
- d_i is the distance, in z_0 – component from the frame $(i - 1)$ to frame (i)
- $\bar{I}_{m_i}$ moment of inertia of rotor.
- r_{i-1,C_i} vector from the frame $(i - 1)$ origin to the COM C_i.
- r_{i,C_i} vector from the frame (i) origin to the COM C_i.
- $r_{i-1,i}$ vector from the frame $(i - 1)$ origin to the frame (i) origin.
- $\dot{p}_{C_i}$ C_i linear velocity.
- $\dot{p}_i$ frame (i) origin linear velocity.
- ω_i link angular velocity.
- ω_{m_i} rotor angular velocity.
- $\ddot{p}_{C_i}$ C_i linear acceleration.
- $\ddot{p}_i$ frame (i) origin linear acceleration.
- $\dot{\omega}_i$ link angular acceleration.
- $\dot{\omega}_{m_i}$ rotor angular acceleration.
- g_0 gravity acceleration.

With the following initial conditions:

- $\ddot{p}_0{}^0 = [0 \quad 0 \quad 0]^T$
- $\omega_0^0 = \dot{\omega}_0^0 = 0$
- $z_0{}^0 = [0 \quad 0 \quad 1]^T$

In this analysis I have decided to include the fifth joint, but in the test it is blocked (see par. 4.3.3).

5.2 COM identification

The parameter identification is an important step in the modeling system.
In this case, by datasheets and manuals, the following parameters are known:

- Engine's mass
  $m_{m_1} = 0.110\ kg$
  $m_{m_2} = 0.110\ kg$
  $m_{m_3} = 0.110\ kg$

$m_{m_4} = 0.075\ kg$
$m_{m_5} = 0.046\ kg$

- Link's mass
  $m_{l_1} = 1.390\ kg$
  $m_{l_2} = 1.318\ kg$
  $m_{l_3} = 0.821\ kg$
  $m_{l_4} = 0.769\ kg$
  $m_{l_5} = 0.687\ kg$

- Link's length
  $d_{m_3} = 0.155\ m$
  $d_{m_4} = 0.135\ m$
  $d_{m_5} = 0.113\ m$

The only unknown parameters are i-th positions of the COM's link, in respect to the frame $i-1$.

To identify these parameters, the youBot's arm has been represented as in (Figure 90).

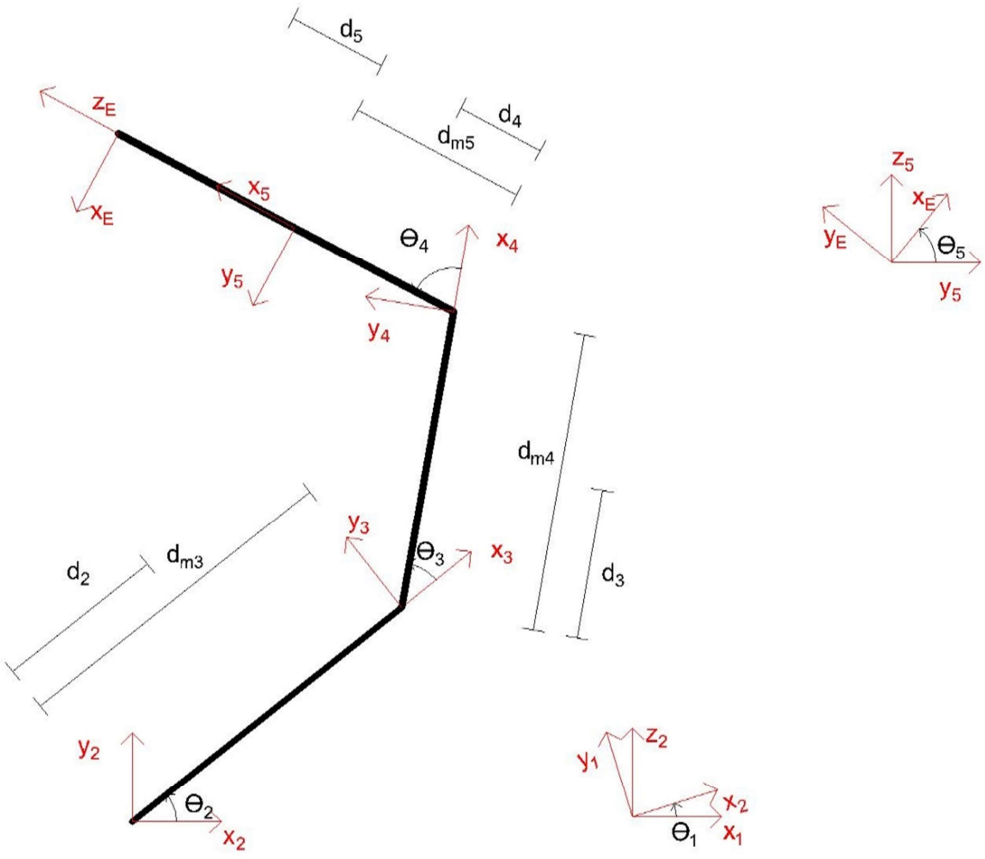

Figure 90: youBot arm's representation

In this case, the generalized coordinates' vector of the system is (5.5)

$$q = [\theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5] \tag{5.5}$$

and the system's length vector is (5.6)

$$D = [d_5 \quad d_{m_5} \quad d_4 \quad d_{m_4} \quad d_3 \quad d_{m_3} \quad d_2] \tag{5.6}$$

To identify the unknown parameters, the gravitational dynamics' component (defined by the potential energy) has been used in (5.7)

$$U = \sum_{i=1}^{5}\left(m_{l_i} g_0^T p_{l_i} + m_{m_i} g_0^T p_{m_i}\right) \tag{5.7}$$

where:

- U is potential energy.
- m_{l_i} is i – th link mass.
- p_{l_i} is i – th link centroid position respect tern 0.
- m_{m_i} is i – th engine mass.
- p_{m_i} is i – th engines position respect to frame 0.
- $g_0^T = [0 \quad 0 \quad g]^T$.

The gravitational dynamics component $G(q)$ is obtained from (5.8)

$$G(q) = \frac{\partial U}{\partial \theta} \tag{5.8}$$

And its symbolic expressions are the following (5.9 – 5.13):

$$g_5 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] \cdot D^T \tag{5.9}$$

$$g_4 = g_5 + ([m_{l_5} \quad m_{l_5} + m_{m_5} \quad m_{l_4} \quad 0 \quad 0 \quad 0 \quad 0] \cdot D^T) \cdot \cos\left(\theta_4 + \theta_3 + \theta_2 - \frac{\pi}{2}\right) \cdot g \tag{5.10}$$

$$g_3 = g_4 + ([0 \quad 0 \quad 0 \quad m_{l_5} + m_{l_4} + m_{m_5} + m_{m_4} \quad m_{l_3} \quad 0 \quad 0] \cdot D^T) \cdot \cos(\theta_3 + \theta_2) \cdot g \tag{5.11}$$

$$g_2 = g_3 + ([0 \quad 0 \quad 0 \quad 0 \quad 0 \quad m_{l_5} + m_{l_4} + m_{l_3} + m_{m_5} + m_{m_4} + m_{m_3} \quad m_{l_2}] \cdot D^T) \cdot \cos(\theta_2) \cdot g \tag{5.12}$$

$$g_1 = 0 \tag{5.13}$$

Afterwards, the joint torque's values are measured after positioning the manipulator in remarkable postures, obtaining the following results (Table 13):

postures	min joint torque	max joint torque
$q = \left[0 \quad \frac{\pi}{2} \quad 0 \quad 0 \quad 0\right]$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = -0.0\ Nm$ $\tau_3 = 0.2\ Nm$ $\tau_4 = 0.5\ Nm$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = 1.7\ Nm$ $\tau_3 = 1.3\ Nm$ $\tau_4 = 2.2\ Nm$
$q = \left[0 \quad \frac{\pi}{2} \quad 0 \quad \frac{\pi}{2} \quad 0\right]$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = \tau_3 = \tau_4 = -0.3\ Nm$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = \tau_3 = \tau_4 = 0.3\ Nm$

$q = \left[0 \quad \frac{\pi}{2} \quad -\frac{\pi}{2} \quad \frac{\pi}{2} \quad 0\right]$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = 2.0\ Nm$ $\tau_3 = 2.5\ Nm$ $\tau_4 = 0.7\ Nm$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = 6.0\ Nm$ $\tau_3 = 4.9\ Nm$ $\tau_4 = 1.8\ Nm$
$q = \left[0 \quad \frac{\pi}{2} \quad -\frac{\pi}{2} \quad \pi \quad 0\right]$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = 1.3\ Nm$ $\tau_3 = 1.4\ Nm$ $\tau_4 = -0.3\ Nm$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = 3.7\ Nm$ $\tau_3 = 3.3\ Nm$ $\tau_4 = 0.3\ Nm$
$q = \left[0 \quad \pi \quad 0 \quad -\frac{\pi}{2} \quad 0\right]$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = -4.0\ Nm$ $\tau_3 = -2.5\ Nm$ $\tau_4 = -0.7\ Nm$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = -12.0\ Nm$ $\tau_3 = -5.0\ Nm$ $\tau_4 = -1.8\ Nm$
$q = [0 \quad \pi \quad 0 \quad 0 \quad 0]$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = -4.0\ Nm$ $\tau_3 = -1.5\ Nm$ $\tau_4 = -0.3\ Nm$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = -10.5\ Nm$ $\tau_3 = -3.5\ Nm$ $\tau_4 = -0.3\ Nm$
$q = \left[0 \quad \pi \quad \frac{\pi}{2} \quad \frac{\pi}{2} \quad 0\right]$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = -2.0\ Nm$ $\tau_3 = \tau_4 = -0.3\ Nm$	$\tau_1 = \tau_5 = 0.0\ Nm$ $\tau_2 = 7.0\ Nm$ $\tau_3 = \tau_4 = 0.3\ Nm$

Table 13: Joint torque's measurements

Two joint torque's values have been measured because the friction stops the joint movements in the given range for each joint.

Based on the gravity's compensation definition, i.e. the minimum torque that allows the manipulator to be stopped in the assigned configuration, I have been decided to use only the minimum joint torque's value for the identification.

To identify the unknown parameters, a squares algorithm is used, applied to (5.9 – 5.13), obtaining the following results (5.14):

$$[d_5 \quad d_4 \quad d_3 \quad d_2] = [0.031 \quad 0.043 \quad 0.058 \quad 0.070] \qquad (5.14)$$

6 PERFOMANCE ASSESSMENT

In this chapter, the results of the new decentralized control simulation are shown.
The tests have been divided into two sections:

1) Simulation test: the control has been tested in simulation, through the aid from a mathematical model of the manipulator, obtained by the inverse dynamics formulation.
2) Real test: the control has been tested on the real youBot.

6.1 Simulation test

The tests in simulation are described in this section.
The first test is to evaluate the response of the system under input position.
The start posture of the manipulator is the following (Figure 91):

Figure 91: Start posture-simulation position test

The position's command is a square wave characterized by an amplitude value of $0.1\ rad$ and a frequency of $0.5\ Hz$.

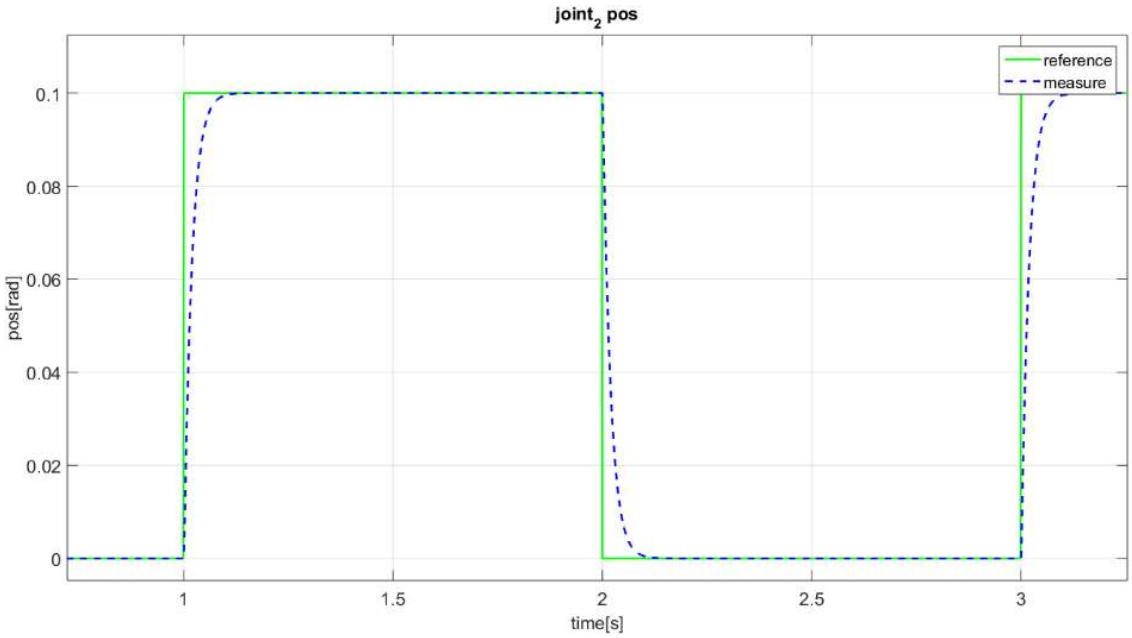

Figure 92: Simulation position's test

The position's response on (Figure 92) is characterized by:

1) Absence of overshoot.
2) Absence of steady state errors.
3) Rise time equal to $0.05\ sec$.

If we compare the response, obtained on (Figure 92) with the previous one on (Figure 42), the absence of overshoot and the steady state error is noticeable.

The second test is to evaluate the response of the system under position's input with a payload equal to $0.4\ kg$, ideally applied on the end-effector.

The position's command is a square wave characterized by an amplitude value of $0.1\ rad$ and a frequency of $0.5\ Hz$.

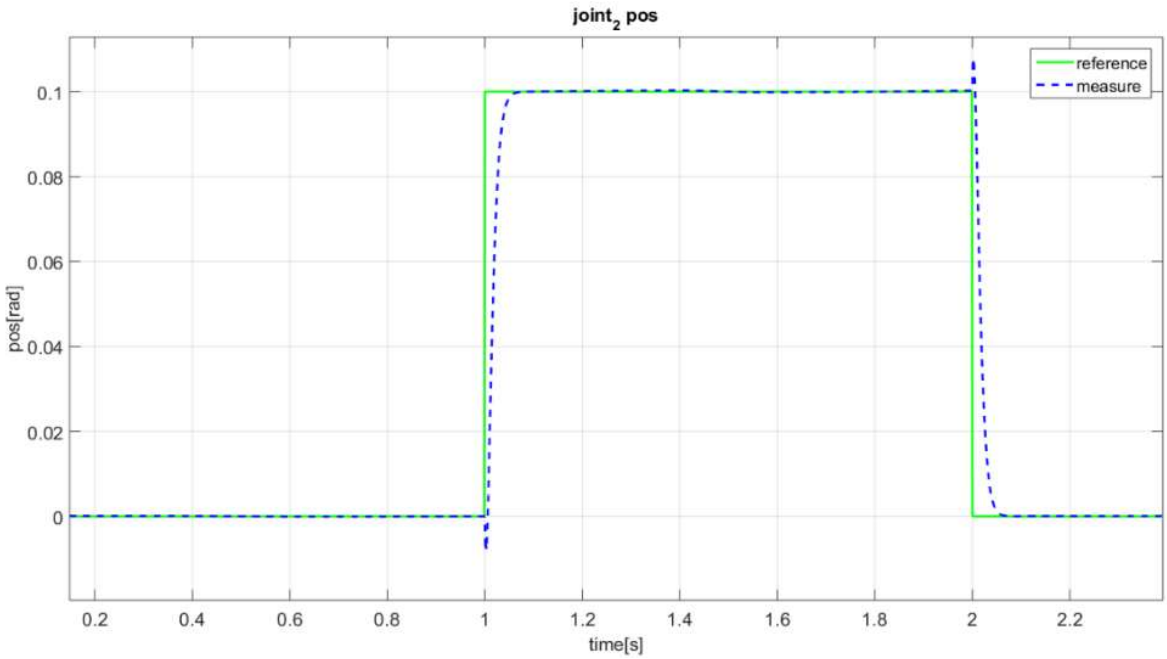

Figure 93: Simulation position's test with payload

The position's response on (Figure 93) is characterized by:

1) A little overshoot equal to $-0.008\,rad$ at $1\,sec$.
2) Absence of steady state error.
3) Rise time equal to $0.05\,sec$.

If we compare the response obtained on (Figure 93) with the previous one on (Figure 92), an overshoot equal to $-0.008\,rad$ at $1\,sec$ caused by a payload is noticeable.

This overshoot compared with the signal's amplitude is almost equal to 10%. This percentage of overshoot is considered acceptable for the case in question.

The (Figure 92) and (Figure 93) also present equal rise time and the absence of steady state error.

6.3 Real test

The tests on the youBot's arm are described in this section.

The first test is to evaluate the response of the system under position's input.

The start posture of the manipulator is shown on (Figure 91).

The position's command, used to drive the second joint, is a square wave; characterized by an amplitude value of $0.2\,rad$ and a frequency of $0.5\,Hz$ with start point equal to $0.46\,rad$.

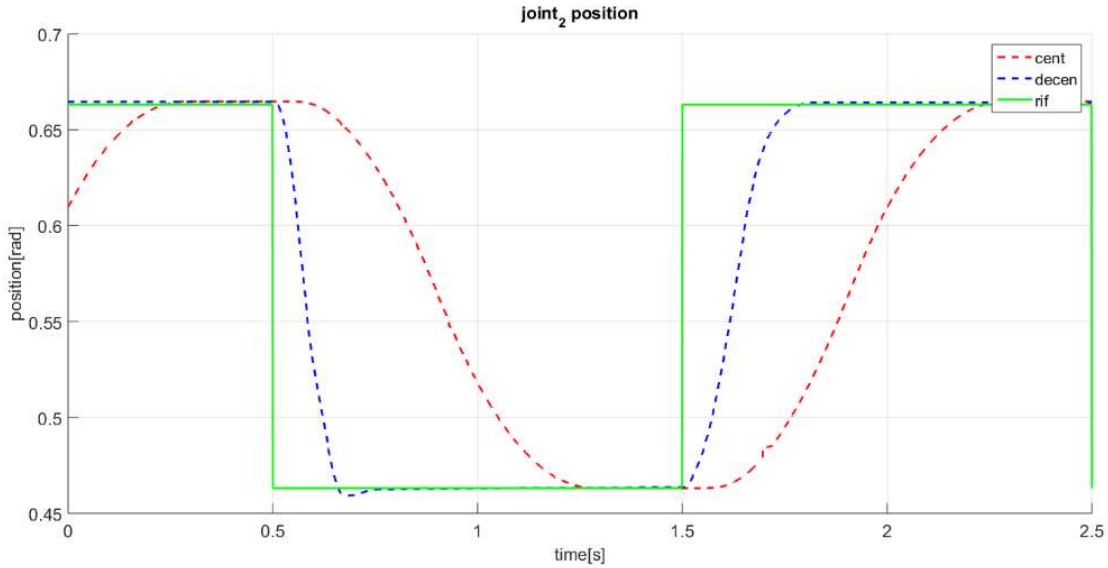

Figure 94: Centralized vs decentralized position's tracking

The centralized position's response, red in Figure 94, is characterized by:

1) Absence of overshoot
1) An insignificant steady state error ($< 0.002\ rad$)
2) Rise time equal to $0.571\ sec$.

The decentralized position's response, blue in Figure 94, is characterized by:

1) An overshoot on the downhill section
2) An insignificant steady state error ($< 0.002\ rad$)
3) Rise time equal to $0.2\ sec$.

If we compare the two responses on (Figure 94); the improvement is on the rise time reduced by a factor almost equal to 3.

The next test is to evaluate the step's response of the system under a position input with a payload equal to $0.3\ kg$, applied to the end-effector.

The position's command, used to drive the second joint, is a square wave characterized by an amplitude value of $0.1\ rad$ and a frequency of $0.5\ Hz$.

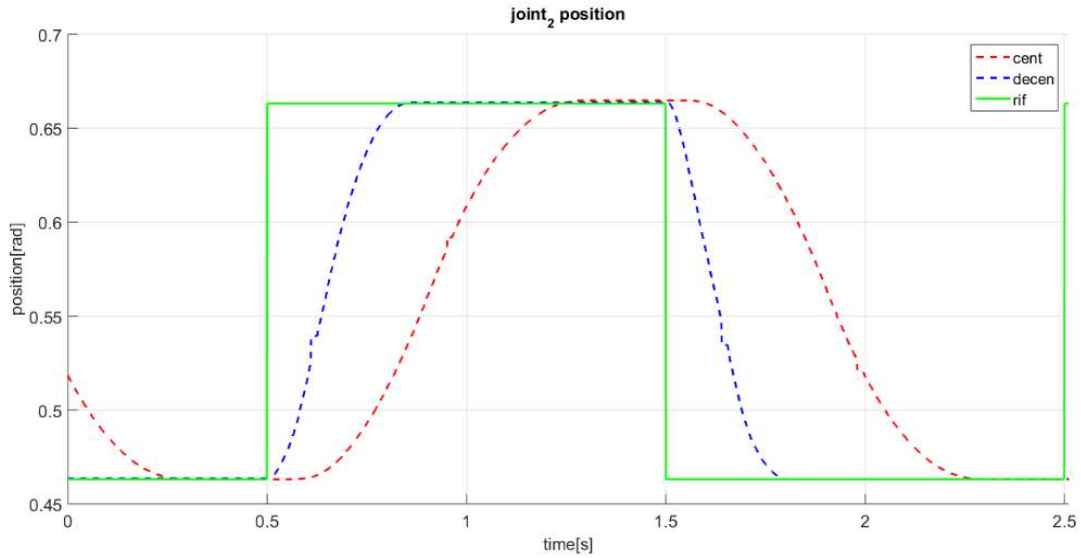

Figure 95: Centralized vs decentralized position's tracking with payload

The centralized position's response, red in Figure 95, is characterized by:

1) Absence of overshoot
2) An insignificant steady state error ($< 0.002\ rad$)
3) Rise time equal to $0.557\ sec$.

The decentralized position's response, blue in Figure 95, is characterized by:

1) Absence of overshoot
2) An insignificant steady state error ($< 0.002\ rad$)
3) Rise time equal to $0.2\ sec$.

If we compare the two responses on (Figure 94); the improvement is on the rise time reduced by a factor almost equal to 3.

If we compare the two decentralized (blue) responses on (Figure 94 - Figure 95) the payload presence has a stabilizing effect, because it eliminates the overshoot in the downhill section.

7 CONCLUSION AND FEATURE DEVELOPMENT

The goal of this work is to improve the performance of youBot's arm.

The preliminary analysis has been tested on the wheels of the mobile platform; these analysis have shown the following results:

1) Position's response is characterized by a good rise time and considerable overshoot.
2) Jitter is equal to $0.02 - 0.04\ sec$.

To improve the performance, I moved the new control from a centralized to a decentralized architecture. This decision has been taken to eliminate the central unit, in order to have a lighter platform than before, and increase the life of hypothetic battery.

The result of the new controller, for each joint, is a drastic reduction of the rise time.

Improvements of this thesis can be caused by:

1) Behavior's study of the combined position, velocity and torque's control.
2) Research on the best parameter values of position and velocity PID.
3) Haptic Interface.

REFERENCES

An C. H., a. H. J. M., 1987. *Kinematic Stability Issues in Force Control of Manipulators.* Raleigh, North Carolina, IEEE Robotics and Automation Society, pp. 897 - 203.

Bischoff R., H. U. a. P. E., 2011. KUKA youBot a mobile manipulator for research and education. *Proc. IEEE International Conference on Robotics and Automation (ICRA*.

Cheng M. Y., a. L. C. C., 2007. Motion Controller Design for Contour - Following Task Based on Real - Time Contour Error Estimation. *IEEE Trans. on Industrial Electronics*, pp. 1686 - 1695.

Craig J. J., a. R. M. H., 1979. *A Systematic Method of Hybrid Position/Force Control of a Manipulator.* Chicago, IEEE Computer Society, pp. 446 - 561.

Gal A., M. O. S. C. a. M. D., 2011. *Hybrid force - position control for manipulators with 4 degrees of freedom.* [Online]
Available at: http://www.researchgate.net/publication/268016011

Jenkel T., K. R. a. S. P., 2013. *Worcester Polytechnic Institute.* [Online]
Available at: https://www.wpi.edu/Pubs/E-project/Available/E-project-031113-133138/unrestricted/Mobile Manipulation for the KUKA youBot Platform.pdf

Jouve D., a. B. D., 2000. Distributed Intelligence Allows New Dimension in Manufacturing Processes Control. *Proc. of the Thirty-seventh Inter. Intelligent Motion Conf. – PCIM'2000*, pp. 343 - 348.

Keiser B., 2013. *Robotics & perception group.* [Online]
Available at: http://rpg.ifi.uzh.ch/docs/theses/Benjamin_Keiser_Torque_Control_2013.pdf

KUKA, 2013. *KUKA - Research & Education.* [Online]
Available at: http://www.kuka-robotics.com/usa/en/products/education/

KUKA, 2013. *KUKA youBot User Manual,.* 1.02 a cura di s.l.:KUKA.

Lee K.C., L. S. a. L. M., 2006. Worst Case Communication Delay of Real - Time Industrial Switched Ethernet With Multiple Levels. *IEEE Trans. on Industrial Electronics,* 53(5), pp. 1669 - 1670.

Malmsten P., 2012. *Worcester Polytechnic Institute.* [Online]
Available at: http://www.wpi.edu/Pubs/E-project/Available/E-project-121512-232610/unrestricted/MalmstenRAILMQP.pdf

Paulus J., 2011. *Youbot base class.* [Online]
Available at: http://janpaulus.github.io/d2/d41/classyoubot_1_1_you_bot_base.html#_details

Paulus J., 2011. *Youbot joint class.* [Online]
Available at: http://janpaulus.github.io/d6/d4c/classyoubot_1_1_you_bot_joint.html#_details

Paulus J., 2011. *Youbot joint parameter data.* [Online]
Available at: http://janpaulus.github.io/d8/d72/_you_bot_joint_parameter_8hpp.html

Paulus J., 2011. *Youbot joint parameter password protected data.* [Online]
Available at: http://janpaulus.github.io/d9/d3d/_you_bot_joint_parameter_password_protected_8hpp.html

Paulus J., 2011. *Youbot joint parameter read only data.* [Online]
Available at: http://janpaulus.github.io/d6/d61/_you_bot_joint_parameter_read_only_8hpp.html

Paulus J., 2011. *Youbot manipulator class.* [Online]
Available at: http://janpaulus.github.io/d7/d57/classyoubot_1_1_you_bot_manipulator.html

Puiu D., M. F. a. V. A. M., 2009. Distributed Control of an Articulated Arm Robot Based on a Single Fieldbuss Network. *Proceedings of the 1st International Conference on Manufacturing Engineering*, pp. 426 - 431.

Sharma S., K. G. K. S. C. a. B. R., 2012. Unifield Closed Form Inverse Kinematics for the KUKA youBot. *Proceedings of ROBOTIK*, pp. 1 - 6.

Siciliano B., S. L. V. L. a. O. G., 2008. Formulazione di Newton - Eulero. In: *Robotica - modellistica, pianificazione e controllo.* Milano: McGraw - Hill, pp. 288 - 294.
the ROS, 2011. *ROS – Introduction.* [Online]
Available at: http://wiki.ros.org/ROS/Introduction
Trinamic, 2011. *Ethercat Manual.* 2.4 a cura di s.l.:Trinamic.
Wikipedia, 2013. *Ripple.* [Online]
Available at: https://en.wikipedia.org/wiki/Ripple_%28electrical%29
Wikipedia, 2014. *Jitter.* [Online]
Available at: https://en.wikipedia.org/wiki/Jitter
Wikipedia, 2014. *Process architecture.* [Online]
Available at: https://en.wikipedia.org/wiki/Process_architecture
Wikipedia, 2015. *State of the art.* [Online]
Available at: https://en.wikipedia.org/wiki/State_of_the_art
Zhang H., 1989. *Kinematic Stability of Robot Manipulators under Force Control.* Scottsdale, Arizona,, IEEE Robotics and Automation Society, pp. 80 - 85.
Zhang H., a. P. R. P., 1985. *Hybrid Control of Robot Manipulators, in International Conference on Robotics and Automation.* St. Louis, Missouri, IEEE Computer Society, pp. 602 - 607.

APPENDIX A

Getting started

- To set a ros_workspace
  Install ROS fuerte
  Generate new workspace
  Download and install the youbot_oodl and the youbot_driver for ROS fuerte

Library preparation

- Put in the youbot_driver/youbot the gdn_lib.cpp (a cpp file, written by me, where there are the functions to manage and move the youbot).
- In the youbot_driver, edit **CMakeList.txt** file to add the new library (gdn_lib.cpp) in SET(YOUBOT_DRIVER_SRC).
- In terminal execute:
  roscd youbot_driver
  cd build
  cmake ..
  make

MatLab R2013B with compiler 4.7.x

- Open sfuntmpl_basic.c (function prototipe)
- Changing this function in according to YouBotBaseSFUN.c (youbot_driver/testing) and save it in the same path (nome_file.c).
- Write the istruction "**mex nome_file.c -ldl**" to compile the file.
- Open simulink and paste system block (S-function)
- Setting this block with the nome_file in function setup.
- Run

APPENDIX B

Getting started

- To extract from youbot_driver package the SOEM library.
- In the terminal:
  Position the current folder in the extracted folder and compile the following code lines:

In the matlab

- To write your fun.c to active the omnidirectional platform, the youbot arm.
- To write your fun.c (same file of before) to manage mailbox mode, PDO mode and acquisition of the measures
  (for example see myfun.c)

In the terminal

- To compile also this file by the following code lines (nome fun is myfun.c)
  gcc -c -fPIC myfun.c
  gcc -c -fPIC myfun.c -o myfun.o
  gcc -c -fPIC ethercatmain.c
  gcc -c -fPIC ethercatmain.c -o ethercatmain.o
  gcc -c -fPIC ethercatconfig.c
  gcc -c -fPIC ethercatconfig.c -o ethercatconfig.o
  gcc -c -fPIC ethercatcoe.c
  gcc -c -fPIC ethercatcoe.c -o ethercatcoe.o
  gcc -c -fPIC ethercatdc.c
  gcc -c -fPIC ethercatdc.c -o nicdrv.o
  gcc -c -fPIC ethercatprint.c
  gcc -c -fPIC ethercatprint.c -o ethercatprint.o
  gcc -c -fPIC ethercatsoe.c
  gcc -c -fPIC ethercatsoe.c -o ethercatsoe.o
  gcc -c -fPIC ethercatfoe.c
  gcc -c -fPIC ethercatfoe.c -o ethercatfoe.o
  gcc -c -fPIC ethercatbase.c
  gcc -c -fPIC ethercatbase.c -o ethercatbase.o
  gcc -c -fPIC nicdrv.c
  gcc -c -fPIC nicdrv.c -o nicdrv.o

MatLab R2013B with compiler 4.7.x

- Open sfuntmpl_basic.c (function prototipe)
- Changing this function in according to command_current.c, command_sync.c & co.
- Saving its in the same path.

- Write the istruction "**mex nome_file.c ……**" to compile the file.
- Open simulink and paste system block (S-function)
  example: mex command_current.c myfun.o ethercatmain.o ethercatconfig.o nicdrv.o ethercatprint.o ethercatbase.o ethercatcoe.o ethercatsoe.o
- Setting this block with the nome_file in function setup.