ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCHOOL OF ENGINEERING
Master Degree in Automation Engineering

STATIC AND DYNAMIC FORCES ACCURACY OF A LIGHTWEIGHT COLLABORATIVE ROBOT THROUGH IMPEDANCE CONTROL

Supervisor:
Chiar.mo Prof. GIANLUCA PALLI

Presented by:
GIACOMO DAVID

Assistant Supervisor:
Dott. Ing. LORENZO CASONI

Graduation Session
Academic year 2020/2021

Abstract

Industry 4.0 has become established, arousing the great interest of curiosities and fears for the impact not yet recognized. Human-robot collaboration is also being given more emphasis since collaborative robots improve safety and flexibility. However, not for all possible applications, the great initial economic effort of these machines may pay off expectations.

This dissertation aims to give useful measurements performed on a KUKA LBR iiwa, an advanced collaborative robot. By executing particular tasks, like polishing, it has been noticed that Cartesian forces reconstructed by the robot are not always such accurate as assured by the manufacturer.

The goal of this thesis is to compare the measured force by the robot sensors with the force measured by an external load cell. The focus is on the Z-axis force, considered the most relevant for the applications considered. For this reason, a commercial one axis load cell has been used.

In these experiments, both static and dynamic forces were treat.

Preface

This thesis is original, unpublished, independent work by the author, Giacomo David. The experiments reported in Chapters 4 and 5 were covered at Egicon SRL, issued by the University of Bologna.

Acknowledgments

Dopo essere arrivato alla fine di questo lungo percorso, vorrei ringraziare tutti coloro che mi sono stati vicini in questi anni di studio.

Un ringraziamento va al mio relatore Prof. Gianluca Palli per avermi dato la possibilità di scrivere questo elaborato con il suo supporto.

Un ringraziamento particolare va alla dottoressa Annalisa Bighi, che è stata tramite fondamentale per poter svolgere il tirocinio inerente alla tesi presso Egicon. Proprio di Egicon ringrazio tutti i colleghi che in qualche modo mi hanno supportato e aiutato fin dal primo giorno.

In particolare ringrazio l'Ing. Paolo Monari per avermi messo a disposizione tutte le tecnologie necessarie per svolgere una tesi sperimentale come questa, il mio correlatore Lorenzo che non finirò mai di ringraziare per l'infinita pazienza e disponibilità, Nicolas per avermi dato consigli e aiuto quando mi trovavo in difficoltà e tutti gli altri.

Inoltre, non avendolo fatto prima, vorrei ringraziare tutte le persone che a partire da quando ho cominciato l'università mi sono state accanto a Bologna. Partendo da Don Riccardo, che è stato il primo ad accogliermi tra le braccia di Villa San Giacomo, per poi ringraziare Don Marco e tutti i ragazzi che ho conosciuto nei primi anni di studio; in particolare Gianluca, Niccolò, Rebecca, Maddalena, Ignazio, Diego, Matteo R. , Matteo C., Silvio, Davide, Margherita e tutti gli altri che non posso citare diventando altrimenti una lista davvero interminabile.

Vorrei inoltre ringraziare i miei compagni di corso Lorenzo, Riccardo, Agustin, Nino, Luca, Giuseppe, Gianluca ed Angelo, perchè senza di voi non sarei arrivato così facilmente alla fine di questo percorso.

Ringrazio inoltre Vanessa che mi ha aiutato a dare sempre il meglio di me e tutti i miei amici con cui ho condiviso momenti di svago e studio. In particolare Matteo, Umberto M.T, Leonardo, Stefano, Raffaele, Marco, Umberto M., Francesco, Giovanni, Michele che per quanto lontani in questi ultimi anni un po' insoliti, hanno sempre fatto in modo di essere nella mia vita, anche se non sempre fisicamente.

Un grande grazie inoltre ad Aurora, per il costante aiuto che mi ha dato per raggiungere al meglio questo traguardo e a Brunella, per l'infinita ospitalità.

Ringrazio gli amici della John Lennon e della Filarmonica, i vecchi compagni scout, gli amici della palestra e della biblioteca per avermi fatto trascorrere momenti di svago e spensieratezza quando più ne avevo bisogno. Ringrazio anche i colleghi della bottega per i momenti faticosi e non passati insieme.

Infine, il ringraziamento più grande va a tutta la mia famiglia, in particolare ai miei genitori che oltre all' essenziale sostegno economico per completare gli studi, hanno sempre creduto in me e sono stati sempre al mio fianco in ogni situazione.

Ai miei nonni

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

"Feminine intuition? Is that what you wanted the robot for? You men."

Isaac Asimov

1.1 Focus and objectives

This work evaluates the forces accuracy of a KUKA LBR iiwa 14, a collaborative robot capable of carrying 14 kilograms of payload. The main problem to solve consists in comparing and justifying the difference of forces measured by the robot's sensors from the measurement of a load cell, an external force sensor made of strain gauges.
The thesis was realized consequently to a poor force accuracy measured during polishing of an uneven surface with the LBR.
In the experiments, it is given particular importance to the Cartesian force along the z-axis.

The robot, equipped with an I/O Pneumatic Flange and a polishing tool attached, follows the trajectory with a certain pressure over the surface to execute a satisfying quality of the operation. Thanks to impedance control, the difficulties of the operation are overcome as the robot adapts its trajectory to the surface correctly.

However, if we focus on the force accuracy of the z-axis, the measurement is not precise as expected. It may be the principal cause of the non-optimal execution of the operation. For such an application, where contact force has to be as constant as possible, the accuracy assured by the manufacturer may not be enough.

For this reason, we tried to understand the causes of this result by using an external load cell. In this way, we managed to help the user and show the best results achievable with this

robot for realizing an application like polishing, where forces accuracy is an indispensable requirement.

It was decided to compare the forces measured by the two devices with the use of a PLC, as described in Chapter 2.

Consequently, we set up the experiment as shown in Chapter 3 to perform the experiments below and find a satisfactory explanation of the results. In particular, we performed four different experiments that allowed us to validate the robot's behavior and show the reader how accurate can be the LBR for these operations and in which cases it's the best choice.

1.2 Collaborative robotics and Industry 4.0

Companies are constantly being confronted with new challenges. Platforms are changing the face of traditional markets; new technologies and working methods result in shorter innovation cycles, while climate change and resource scarcity demand solutions that are compatible with a circular economy.

It is becoming more important for companies to develop resilience, responsiveness, and adaptability competencies. Industry 4.0 marks the onset of the fourth industrial revolution. The factory of the future will be hyper-connected, smart, and autonomous. It will also be characterised by high adaptability and optimal resource utilisation.

Industry 4.0 can deliver huge benefits for the productivity and profitability of the manufacturing industry, manufacturing equipment suppliers, and enterprise software companies. The use of digital technologies and the comprehensive networking of objects, devices, and machines in the implementation of innovative Industry 4.0 solutions also contribute to improvements in the speed of reaction and the resilience of companies to better overcome unexpected developments such as the coronavirus crisis. In addition, real-time networking simplifies operations while maintaining a physical distance.(1)

More in detail, the adoption of collaborative robots in Industries worldwide is rising. Robots and human both have their strengths and limitations. Working together safely will provide a better quality product with high accuracy in less time. The main goal of Robotics and Industry 4.0 is to improve productivity, produce high quality products at low price and meet customer expectations.(2)

This new way of collaboration offers workers the prospect of more fulfilling and satisfying jobs. On the other hand, it represents to companies an advanced way to reach new goals and optimize many processes for being more competitive on the market.

Processes commonly hand-made have the possibility to be automatized with the help of these emerging devices.

1.3 Collaborative robots characteristics and applications

Collaborative robots are part of one of the most revolutionary and challenging next-generation robots. They are all characterized by physical human-robot interaction (pHRI). In pHRI, humans and robots share the same workspace and come into contact with each other. Physical interaction may occasionally happen if a regular operation is intended to be without contact or on purpose if the operator is supposed to work in physical contact with the machine, exchanging forces and cooperating with the environment. We will designate these intended forms of interaction hands-off and hands-on pHRI, respectively. pHRI robots will be designed to coexist and cooperate with humans in applications such as assisted industrial manipulation, collaborative assembly, domestic work, entertainment, rehabilitation, or medical applications. Such robots must fulfill different requirements from those typically met in conventional industrial applications. While it might be possible to relax requirements on the velocity of execution and absolute accuracy, concerns such as safety and dependability become paramount when human lives are involved. This paradigm shift entails many differences in design and control and opens up new challenging directions for research.(3)Although they still represent a small percentage in the entire robotics industry, Cobots are becoming rapidly relevant on the market for their great potential. There are forecasts about a vastly increasing demand for these solutions for the following years. The global collaborative robots market size was USD 725.8 million in 2019 and is projected to reach USD 11,684.7 million by 2027, exhibiting a CAGR of 42.0% during the forecast period.(4) To highlight the advantages and disadvantages, a comparison with Industrial robots is made in Figure 2.10, and the relevant differences are discussed.

Collaborative robots	Industrial robots
Simple programming	Harder programming
Low weigh < 30Kg	High weigh > 50Kg
Embedded safety	No safety sensors
Free operation in workspace	Cage required
Mobile	Immobile
External force sensors	No external force sensors
Faster and simpler settling	High operation speed
Flexibility of deployment	Universal use in the limited spa

Table 1.1: Collaborative and Industrial robots comparison

Simple programming

Industrial robots can usually perform only one task and require deep programming knowledge. A collaborative robot, instead, is easier to program. Using user-friendly software, Cobots can learn new actions after a teaching phase. It is also possible to set some desired positions, and save them by deploying the new configuration to the software. Consequential to the ease of programming, it is also possible to perform different tasks with different purposes.

Flexibility and free operation in workspace

The possibility of installing cobots in the workspace without cages is undoubtedly one of the most relevant benefits of this technology. Cobots are designed to work with humans and not to replace humans safely. They can work in complex human environments without being located inside considerable safety barriers and sensors that often accompany them, leaving more floor space for free. Flexibility is high thanks to the variety of tools that can be mounted on the end effector.

(a) Collaborative robot (7)

(b) Industrial robot
https://padgettechnologies.com/

Figure 1.1: Robot examples

Operation speed

For having the possibility of working without physical barriers, collaborative robots are designed to work with a speed lower than the high speed reachable with Industrial robots. On the other hand, the speed of cobots is usually enough for the purpose for which are made.

External force sensors

In conclusion, the presence of external force sensors certainly needs to be mentioned. Thanks to these devices, flexibility improves, and operations where force measurement is fundamental like polishing. Directly linked to the presence of external force sensors, there is the possibility for Cobots to be guided with Impedance control, a fundamental requirement for many operations.

Chapter 2

General Robotics concepts

2.1 Kinematic analysis

In describing the kinematics of a robot, there are two types of problems:

- Direct (forward) kinematic model: once the position, velocity, acceleration values are known in the joint space, the corresponding values in workspace (in a proper reference frame, e.g. Cartesian) are determined.

$$x = f(q) \quad q \in \mathbb{R}^n$$

- Inverse kinematic model: determine the inverse mapping of the kinematic variables from the workspace to the joint space

$$q = g(x) = f^{-1}(x) \quad q \in \mathbb{R}^n, x \in \mathbb{R}^m$$

It is possible to define different kinematic models (i.e. different functions f (q)) for a given manipulator, although equivalent from a mathematical point of view.
In robotics, it is of interest to define the relationships between:

- end-effector velocity and joint velocities

$$\begin{bmatrix} v \\ \omega \end{bmatrix} \Longleftrightarrow \dot{q} \tag{2.1}$$

- between force applied on the environment by the manipulator and corresponding joint torques

$$\begin{bmatrix} f \\ n \end{bmatrix} \Longleftrightarrow \tau \tag{2.2}$$

These two relationships are based on a matrix, defining a linear mapping between joint- and work-space, known as the Jacobian of the manipulator. Jacobian matrix J(q) is fundamental for solving the inverse kinematic problem among its different uses in robotics. It is defined with an analytic or a geometric expression and is a m x n matrix.(5)

The forward mapping between joint- and work-space is given by

$$\dot{x} = J(q)\dot{q} \qquad J(q) \in \mathbb{R}^{m\,x\,n} \tag{2.3}$$

Then, the inverse mapping is given as the solution of a linear (matrix) algebraic equation:

- With m = n, if the manipulator is not in a singular configuration, it is possible to use the inverse of the Jacobian:

$$\dot{q} = J^{-1}(q)\dot{x} \tag{2.4}$$

- With m ≠ n, the Moore-Penrose pseudoinverse of the Jacobian is used:

$$\dot{q} = J^{+}(q)\dot{x} \tag{2.5}$$

In our case specifically, the number of DOF (n) is higher than the dimension of the workspace (m) and we will have a fat matrix. Consequently, we will have a RIGHT-pseudo-inverse formulation: $J^{+} = J^{T}(J\,J^{T})^{-1}$

This will lead to a subspace inside the whole joint space where all the joint velocities belonging to that space would be mapped by the Jacobian to the null workspace velocity. Thus it means that there exist a kernel of J(q) inside the Joint space, which is different from the vector q = 0.

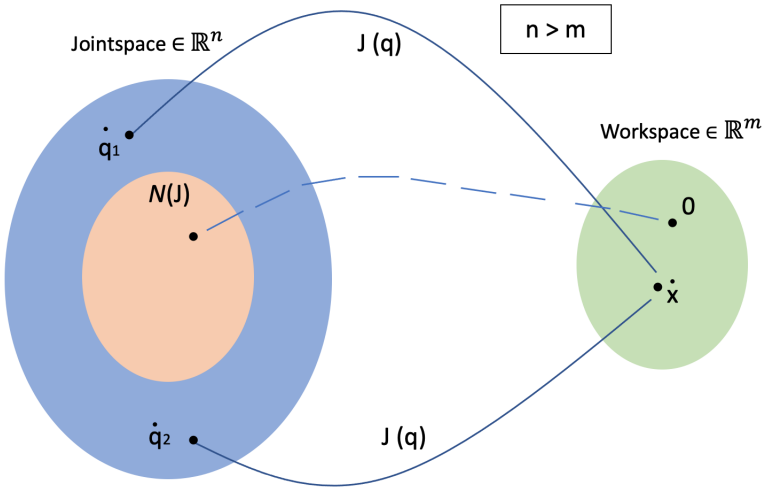


Figure 2.1: Joint/Work space relation

Therefore we will have that: $\forall \dot{x} \quad \exists \dot{q} \quad s.t. \quad \dot{x} = J(q)\dot{q}$ is not unique
We will also write that $\dot{q} = J^{+}\dot{x} \quad \exists N(J) \quad s.t. \quad \forall \dot{q} \in N(J) \quad \rightarrow \quad \dot{x} = J\dot{q} = 0$

In practice, this will result in a not unique solution. This property of redundant manipulators, and thus valid also for the LBR, will be better described in section 3.1

2.2 Base and tool transforms

Base transform

Tool and Base transformations are two important types of coordinate frames. They are fundamental concepts required for developing efficient and reusable applications.

Let's consider a standard industrial robot with six rotational joints. We want to add a coordinate frame on link zero and a coordinate frame on link 6, the end-effector frame. As already known, there exist a kinematic model which relates the pose of frame 6 with respect to frame zero, and it's the K function of the six joint angles of the robot. If this robot is said to be located in a factory, where its world coordinate frame is called {W}, we can write the second left formulation in figure 2.2.We call base transform the pose of the robot link zero concerning the world coordinate frame. The pose of the robot's end effector in the world coordinate frame can then be obtained simply by compounding the base transform with the forward kinematic transform of the robot, which is a function of the robot's joint angles.
Imagine introducing another robot hanging upside down from the roof with a known pose with respect to the world coordinate frame. Also, in this case, we can write an expression for the end-effector pose of the second robot with respect to the world, and another Base transform will be obtained. (2.2) We got in this way two base transformations.

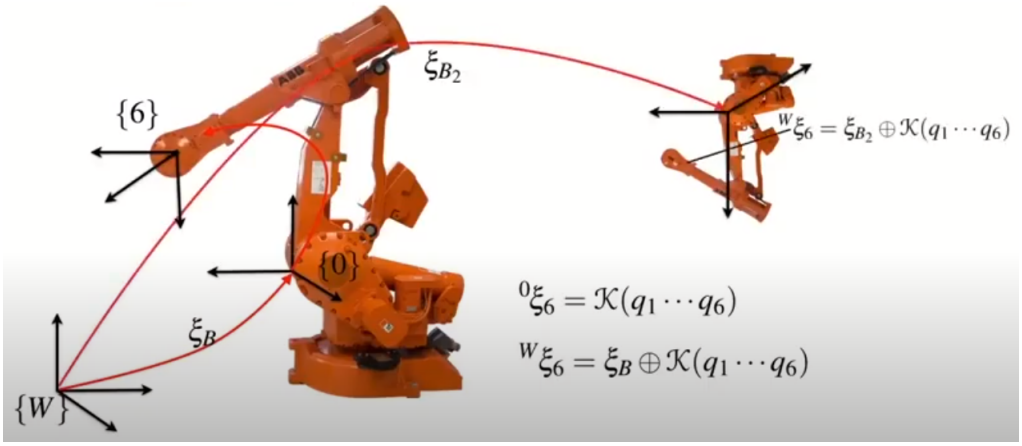


Figure 2.2: Base Transformation (6)

Tool transform

If we now look at the tool end, the kinematics will give us the pose of the robot's end-effector. However, a flange is typically mounted at the end effector, at which a tool is usually attached. Let's consider, for instance, a grinding tool. We would like to know the pose of a new coordinate frame {E}. With some careful measurement, it can be determined the relative pose from frame 6 to frame {E}, and this would be constant relative pose. So, now the pose of the end-effector, with respect to the world coordinate frame, can be written by this set of compounded poses. We will then define a Tool transform as the base transform compounded with the kinematic model of the arm, compounded with the constant tool transform from the mounting point to the end of the tool.(6)

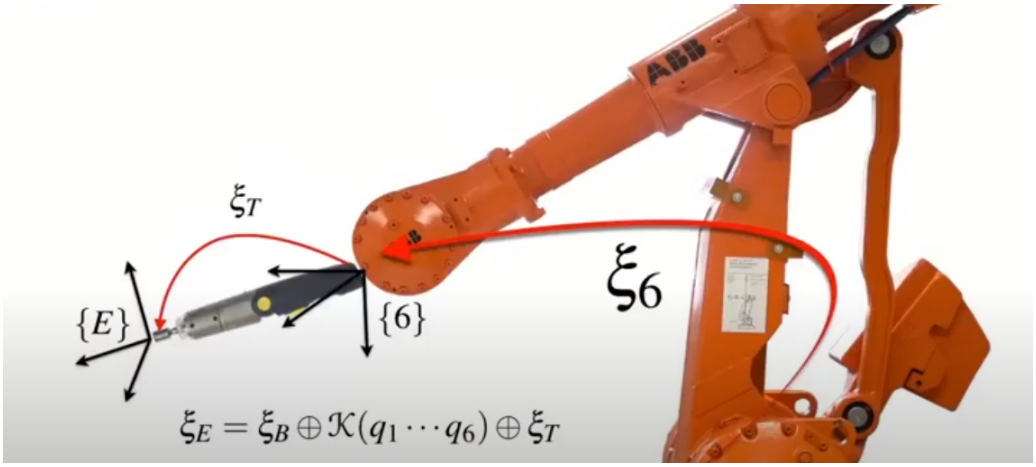


Figure 2.3: Tool Transformation (6)

In our particular case with LBR iiwa, each frame created for a tool can be programmed in the robot application as the reference point for motions. After the project is synchronized, the frames of a tool can be selected as the TCP for Cartesian jogging on the smartHMI. (Figure 3.6)

We will use tool transform for our goal, but for more complex applications, with a different workbench, a base transform might be necessary.

In conclusion, these transforms are also really important for programming applications. Thanks to them, if a tool has to be changed or needs to be executed over a different base, it will not be necessary to change all the relative frames required for motion. Instead, we will need to simply change the selected base/tool transformation with the desired one.

2.3 Motion types programming

The manipulator can move according to different motion types.The start point of a motion is always the end point of the previous motion. The main motion types, that can be classified in different manners, include: PTP, LIN, CIRC,SPL and Spline motion.

PTP motion type

In this case the robot guides the Tool Center Point (TCP) along the fastest path to the end point. The fastest path is generally not the shortest path in space and is thus not a straight line. As the motions of the robot axes are simultaneous and rotational, curved paths can be executed faster than straight paths. PTP is a fast positioning motion. The exact path of the motion is not predictable, but is always the same, as long as the general conditions are not changed.(7)

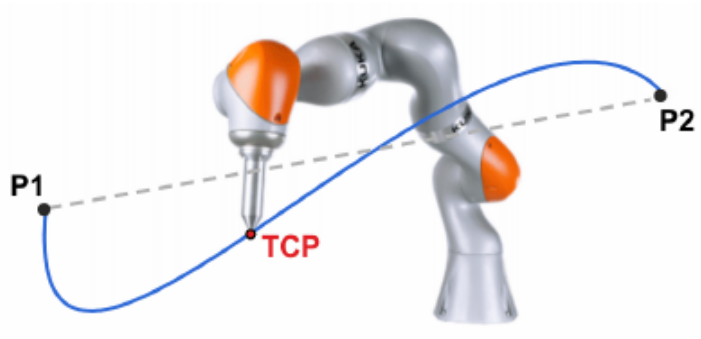


Figure 2.4: PTP Motion (7)

LIN motion type

. The robot guides the TCP at the defined velocity along a straight path in space to the end point. In a LIN motion, the robot configuration of the end pose is not taken into account.

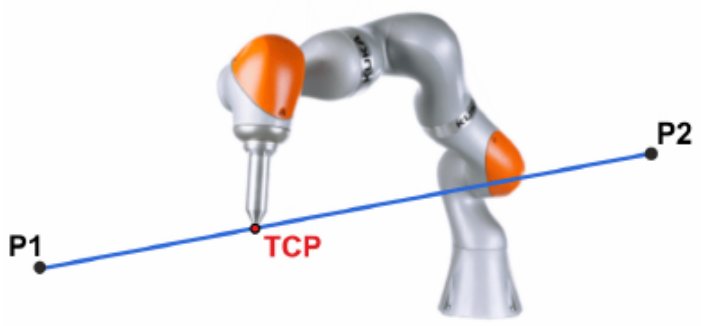


Figure 2.5: LIN Motion (7)

CIRC motion type

. The robot guides the TCP at the defined velocity along a circular path to the end point. The circular path is defined by a start point, auxiliary point and end point. In a CIRC motion, the robot configuration of the end pose is not taken into account.

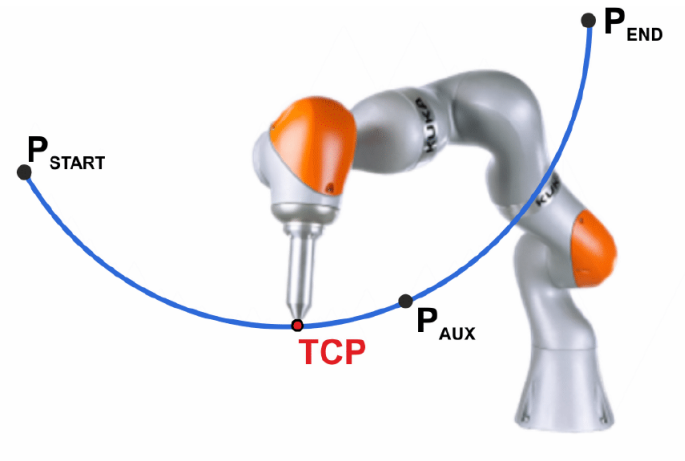


Figure 2.6: CIRC Motion (7)

SPL motion type

The motion type SPL enables the generation of curved paths. SPL motions are always grouped together in spline blocks. The resulting paths run smoothly through the end points of the SPL motion. In an SPL motion, the robot configuration of the end pose is not taken into account. Curved lines are achieved by grouping together 2 or more SPL segments. If a single SPL segment is executed, the result is the same as for a LIN command.(7)

Spline motion type

Spline is a motion type that is particularly suitable for complex, curved paths. With a spline motion, the robot can execute these complex paths continuously. Such paths can also be generated using approximated LIN and CIRC motions, but splines have advantages, however. Splines are programmed in spline blocks. A spline block is used to group together several individual motions as an overall motion. The robot controller planned and executed the spline block as a single motion block. The motions contained in a spline block are called spline segments. In a Cartesian spline motion, the robot configuration of the end pose is not taken into account. The configuration of the end pose of a spline segment depends on the robot configuration at the start of the spline segment.

The path is defined by means of points located on the path and passed through without exact positioning. These points are the end points of the individual spline segments.(7)

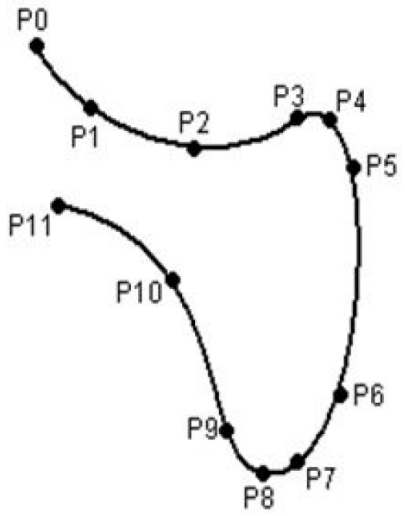


Figure 2.7: Spline block example (7)

2.4 Force and Impedance control

Interaction Control

The control of physical interaction between a robot manipulator and the environment is crucial for successfully executing several practical tasks. The robot end-effector has to manipulate an object or perform some operation on a surface. Typical examples in industrial settings include polishing, deburring, machining or assembly.(3)
In all these cases, using a pure motion control strategy may lead to failure. To solve this problem, force control is taken into account to handle the forces and the torques that the robot exchanges with the environment.
In particular, the quantity that describes the state of interaction more effectively is the contact force at the manipulator's end-effector. High contact force values are generally undesirable since they may stress both the manipulator and the manipulated object.(8)
There exist two types of force control:

Passive force control

In this case the compliance between the robot and the environment is provided by a compliance device mounted on the end effector, or in general, in the wrist of the robot. A compliance tool widely used in industrial applications is the Remote Center of Compliance (RCC). The advantages of this approach are the simplicity and the low cost due to the absence of force/torque sensors. On the other hand, this solution is not flexible since a compliance device should be designed and selected for every different task the manipulator has to perform. Additionally,

since no forces are measured, it can not guarantee that high contact forces will never occur.(3)

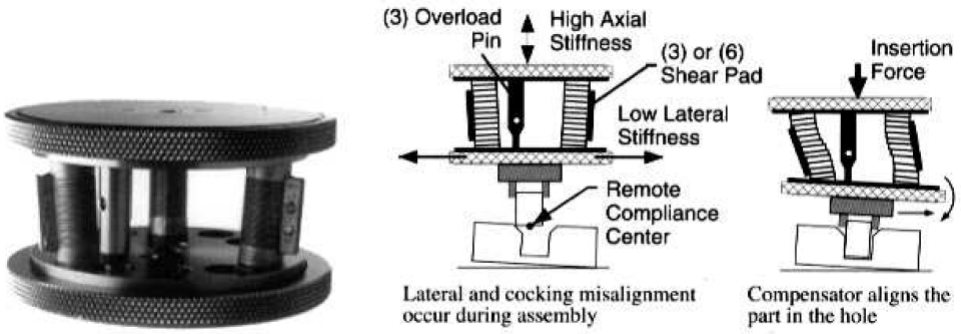


Figure 2.8: A Remote Center of Compliance (5)

Active force control

With active force control, the compliance of the manipulator is introduced by the control action. Therefore, a model of the environment or a proper sensor is needed to measure the external wrenches and act according to a control action that would introduce compliance in the dynamic behavior of the manipulator itself. Then, if a model of the surrounding environment is needed, computational complexity could be high. However, for the flexibility and task independence assured, it is worth considering this type of control when implementable. With this approach, the manipulator can be maintained. Moreover, the manipulator can switch from one task into another immediately, without any mechanical intervention.

Impedance control

Traditional industrial robots are usually driven by gear-head motors and they are purely stiff positioning devices that perform specific tasks such as welding, painting, and palletizing.(8) On the other hand, with collaborative robots designed for compliance manipulation, impedance control is preferred for the required interaction between a robot arm and an environment, which demands a robot arm capable of producing accurate joint torques such that the appropriate force is realized at the end-effector.(9)

Impedance control is not truly an absolute control, in fact, the goal is not to explicitly control the interaction forces applied to the environment. Instead, it is to define the robot's dynamic behavior in terms of stiffness, damping, and masses of the system. For example, we would like the robot to be rigid or compliant depending on the environment.

The appropriate manipulator impedance for a given the situation depends on the task to be performed. In most manipulatory tasks, there is a tradeoff to be made between allowable interface forces and acceptable deviations from desired motions.

Whether it has been rationally chosen or not, the manipulator impedance specifies a relation between interface forces and imposed motions. If the implicit tradeoff in the task is expressed as a performance index to be maximized or minimized, the impedance appropriate for that task may be determined using optimization theory.(10)

To some extent, impedance control is a generalization of the stiffness control. It generalizes the relationship:

$$F = k\Delta x \quad \longrightarrow \quad \frac{F}{\Delta x} = k \qquad \text{(constant ratio)}$$

$$\frac{W(s)}{\dot{X}(s)} = \frac{W(s)}{sX(s)} = Z(s) \qquad \text{(transfer function ratio)}$$

The terminology of impedance comes from the electrical domain, then a similarity between mechanical and electrical domains can be made:

$$\text{Force} \iff \text{Voltage} \quad \text{(both seen as effort)}$$
$$\text{Velocity} \iff \text{Current} \quad \text{(both seen as flow)}$$

The two pairs have the same meaning from an energetical point of view.

What we want to do is to cancel the dynamics of the robot itself and to replace it with the equation $M_d\,\ddot{x} + D_d\,\dot{x} + K_d\,x = -w$
where depending on the choices of the matrices we can choose the behaviour of the robot in relation to the environment we have.

Consequently, a two steps procedure is considered:

- Linearization in work-space: as the robot's original behaviour is far from the desired one, we need to replace its original dynamics with a linearized dynamic model in the workspace. This step is similar to what is performed in the inverse dynamics control.
  In this case, the interaction force F_a exerted to the robot is measured by a Force/Torque sensor and then subtracted in the force control input F.

  $\hat{M}(x)\ddot{x} + \hat{C}(x,\dot{x})\dot{x} + \hat{g}(x) = F + F_a$

  $F = [\hat{M}(x)y + \hat{C}(x,\dot{x})\dot{x} + \hat{g}(x) - F_a] = [\hat{M}(x)y + \hat{n}(x,\dot{x}) - F_a]$

  The linear model obtained is $\ddot{x} = y$

- Definition of the desired impedance behaviour: After the model has been linearized we can impose the desired impedance behaviour.
  $y = \ddot{x}_d + M_d^{-1}[D_d(\dot{x}_d - \dot{x}) + K_d(x_d - x) + F_a]$

  from which $M_d(\ddot{x}_d - \ddot{x}) + D_d(\dot{x}_d - \dot{x}) + K_d(x_d - x) = -F_a$

  Where:

  - M_d is the resulting inertia matrix;
  - D_d is the desired damping matrix;
  - K_d is the desired stiffness matrix.

The final control law is

$$F = \hat{M}\ddot{x}_d + \hat{M}M_d^{-1}[D_d(\dot{x}_d - \dot{x}) + K_d(x_d - x) + F_a] + \hat{n}(x, \dot{x}) - F_a$$

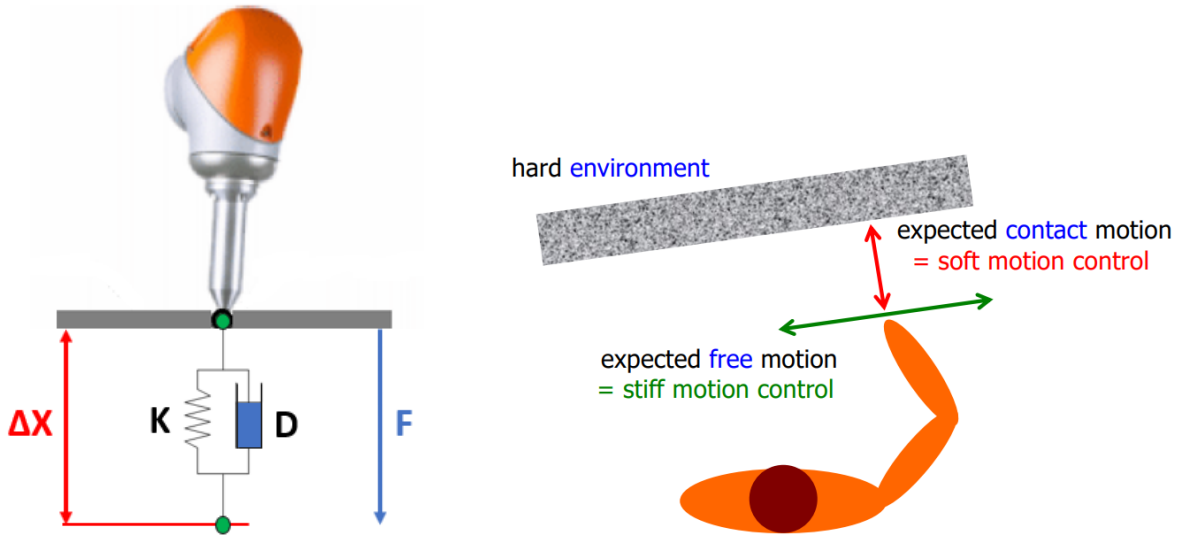
In this case, by properly shaping the desired inertia matrix M_d we can decouple the system and have the possibility to behave faster or slower depending on the direction we consider. Generally, we select the impedance as follows:

- **soft motion control**: large $M_{d,i}$ and small $K_{d,i}$ along Cartesian directions where contact is expected → low contact forces;
- **stiff motion control**: large $K_{d,i}$ and small $M_{d,i}$ along Cartesian directions supposed to be free → good trajectory tracking;
- damping coefficients $D_{d,i}$ are used to modify the transient phases.

It is noticeable that by choosing $M_d = \hat{M}$ the control law is simplified and a F/T sensor is not required for feedback as F_a is simplified:

$$F = \hat{M}\ddot{x}_d + D_d(\dot{x}_d - \dot{x}) + K_d(x_d - x) + \hat{n}(x, \dot{x})$$

On the other hand, in this case, the inertia matrix will not be diagonal anymore, leaving the system coupled. However, it is usually enough to properly select the matrices K_d and D_d to get a satisfactory result. The stiffer is the environment, the softer are the stiffness coefficients $K_{d,i}$.(5)(11)
For implementing classical impedance control, a precise torque control interface is imperative. While this can be made possible with some effort by equipping the robot with a torque sensor



(a) (b) (5)

Figure 2.9: Impedance control modeling

and using an inner loop torque control, many commercial robots still do not provide a torque control interface. In such a case, a compliant impedance behaviour could be imposed by admittance control via additional measurement of the external forces using a force/torque sensor (FTS) mounted at the end-effector. In this case, the compliant behaviour can clearly be achieved only with respect to forces acting on the robot's tip, where the FTS perceives them. All other forces acting along the robot's structure are not considered.(12)

In that case, as we don't have access to low-level robot torque (or motor current) commands with a consequently closed control architecture, then for handling the interaction with the environment, contact forces or velocity commands are used.(13)In our case, as we have torque sensors at each joint, will be much easier to implement impedence control by simply acting on damping and stiffness coefficents, without the necessity of a F/T sensor.

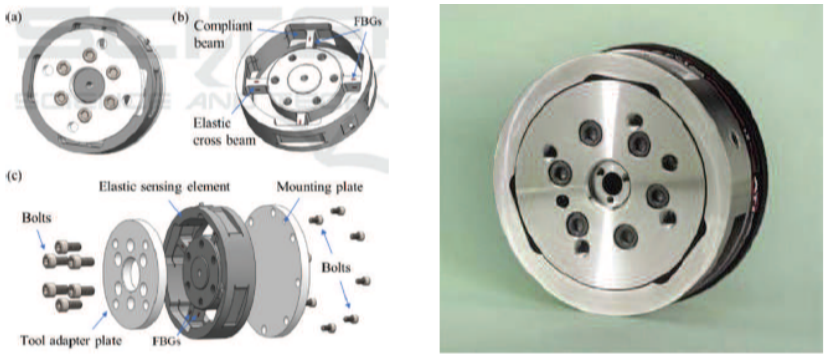


Figure 2.10: Force/Torque sensor structure (5)

Chapter 3

Main components and framework

3.1 KUKA LBR iiwa

The LBR iiwa is produced by KUKA and is the world's first series-produced sensitive, therefore, Human-Robot Collaboration(HRC)-compatible robot. It is one of the best collaborative robots on the market and has a 7 or 14 kg payload. It is the last version of the LBR series, born in 2004 with the first model of LBR. LBR stands for "Leichtbauroboter" (German for lightweight robot), while iiwa stands for "intelligent industrial work assistant". It can also adapt to its environment and adjust its motion by 'learning' from previous experiences.
The LBR can be considered as part of a robot system that comprises the following components:

1. Connecting cable to smartPAD
2. KUKA smartPAD control panel
3. Manipulator
4. Connecting cable to Sunrise Cabinet robot controller
5. KUKA Sunrise Cabinet robot controller
6. Software and accessories

Here are described the main elements of the list and some other relevant concepts. Further descriptions are present in KUKA Manual.(12)

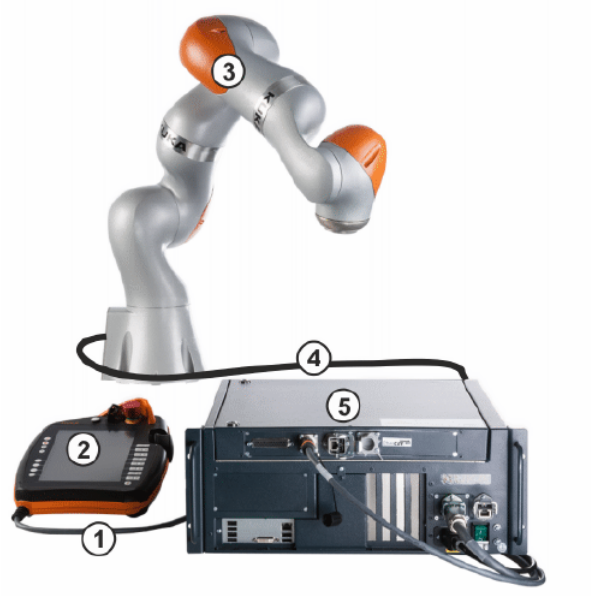


Figure 3.1: KUKA Robot System (7)

Manipulator

The manipulator used in this work is the KUKA Lbr iiwa 14 820R. It is the model with a higher payload, equal to 14 kilograms. Unlike other cobots like the Universal Robot series, this robot has 7 degrees of freedom. Consequently, it is kinematically redundant, guaranteeing the feature of moving to every point in the work envelope with an infinite number of axis configurations. With the right-hand rule, we can find the positive direction of rotation of the robot axes 3.2(a). Notice that the vertical configuration is the default home position of the robot.

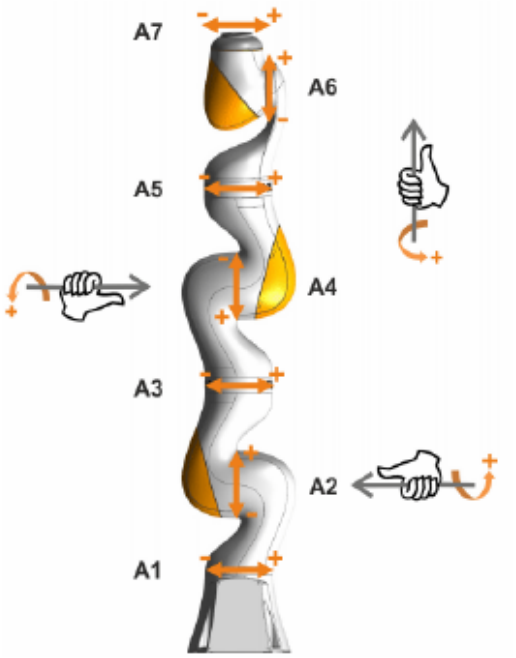
Sensitivity

LBR iiwa has a payload-weight ratio of 0.47, an high value that makes it very efficient, sensitive, and responsive. The presence of a torque sensor for each of the seven joints guarantees sensitivity property. These sensors, implemented using safe technology, enable the automation of delicate assembly tasks for force-controlled joining operations and process monitoring. Moreover, contact detection capabilities and programmable compliance are assured. Also, continuous-path processes are greatly mastered by this manipulator, and fragile objects can be handled without damaging them.

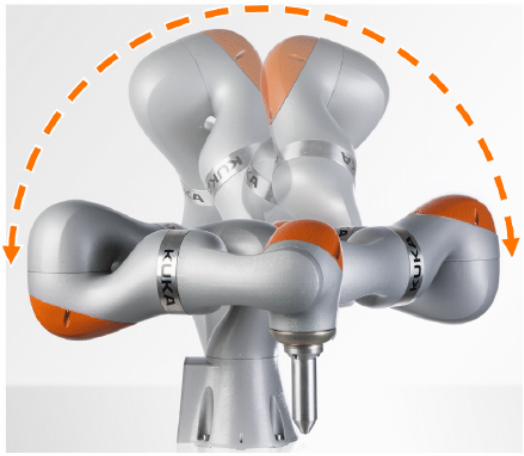
Safety

Torque sensors allow the user to change and control the execution of a robot application with even a simple touch, without the need for additional control. The housing is made of aluminum

and, coupled with the smooth, curved and streamlined design without edges, increases safety. All crushing and shearing hazards are then eliminated, and risks are minimized.



(a) Manipulator axes orientation



(b) Null space motion

Figure 3.2: Manipulator characteristics (7)

Flexibility and null space motion

With seven actuated joints guaranteeing redundancy, LBR iiwa can easily achieve the position and orientation of the tool with different configurations. The manipulator can move within its workspace, which is derived from the individual axis ranges. Especially with complicated installations or the presence of obstacles, the LBR proves its high flexibility. A way to exploit redundancy property is to keep the end effector fixed while varying the LBR configuration. We consequently have a null space motion during Cartesian jogging. In this type of motion 3.2(b), the axes are rotated in such a way that the position and orientation of the set TCP (Tool Center Point) are retained during motion. This property is helpful for some applications like obstacle avoidance and manipulability measures, during which the absence of redundancy may cause some errors. (12)

KUKA SmartPAD

The smartPAD is another relevant device of the robot system. It allows to move the robot as desired and access the robot's controller through a graphical interface. Thanks to the manual

supplied by KUKA (12) it is pretty easy to understand its proper use. By looking at Figure 3.3, there are important elements to highlight:

- 2) Keyswitch to lock/unlock the PAD
- 3)Emergency stop button
- 5) Jog keys to move the 7 Joints manually
- 8) User keys freely programmable by the user
- 9)Start Button for playing the application

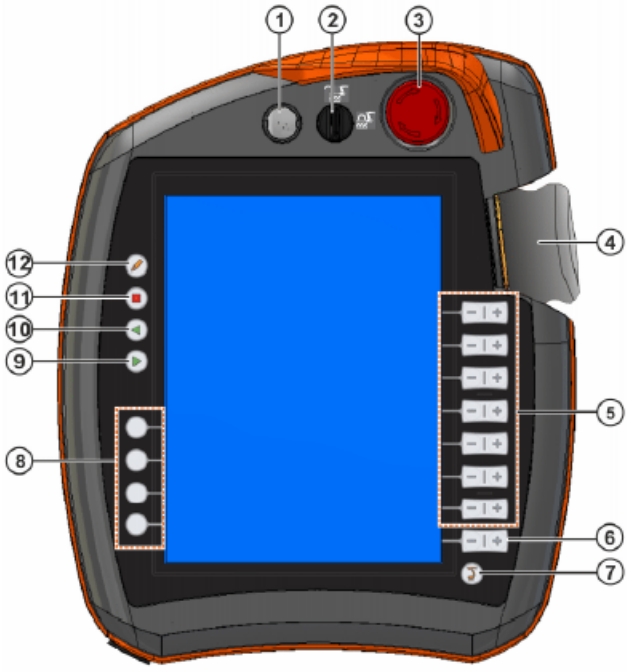


Figure 3.3: Frontal view of KUKA SmartPAD (7)

Working modes

By turning right the keyswitch 2) is not only possible to lock the smartPAD but also to select the working modality of the robot. There are 4 possible modes:

- (T1) Manual Reduced Velocity: used for programming, teaching and testing of programs. In this modality, manual guidance is allowed with a reduced jog velocity to a maximum of 250 mm/s;
- (T2) Manual High Velocity: similarly to T1 is used for testing programs, but the speed is not reduced, then has to be used more carefully;

- (AUT) Automatic: used for the Automatic execution of programs. Industrial robots with and without higher-level controllers;
- (CRR): operating mode which can be selected when the industrial robot is stopped by the safety controller when for example, the robot violates a safety condition, a position sensor is not mastered/referenced, or a joint torque sensor is not referenced.
  After changing to CRR mode, the industrial robot may once again be moved.

Jog mode

In the operating modes T1, T2 and CRR, the robot controller can only execute programs in jog mode. This means that it is necessary to slightly hold down an enabling switch (1, 3 or 5 of Figure 3.4) and the Start key to execute a program:

- Releasing the enabling switch on the smartPAD triggers a safety stop 1
- Pressing fully down on the enabling switch on the smartPAD triggers a safety stop 1 (a Man Down system is triggered)
- Releasing the Start key triggers a stop of Stop category 1

Figure 3.4: Rear view of KUKA SmartPAD (7)

SmartHMI user interface

If we turn on the smartPAD, the screen will show up the following main page. It includes:

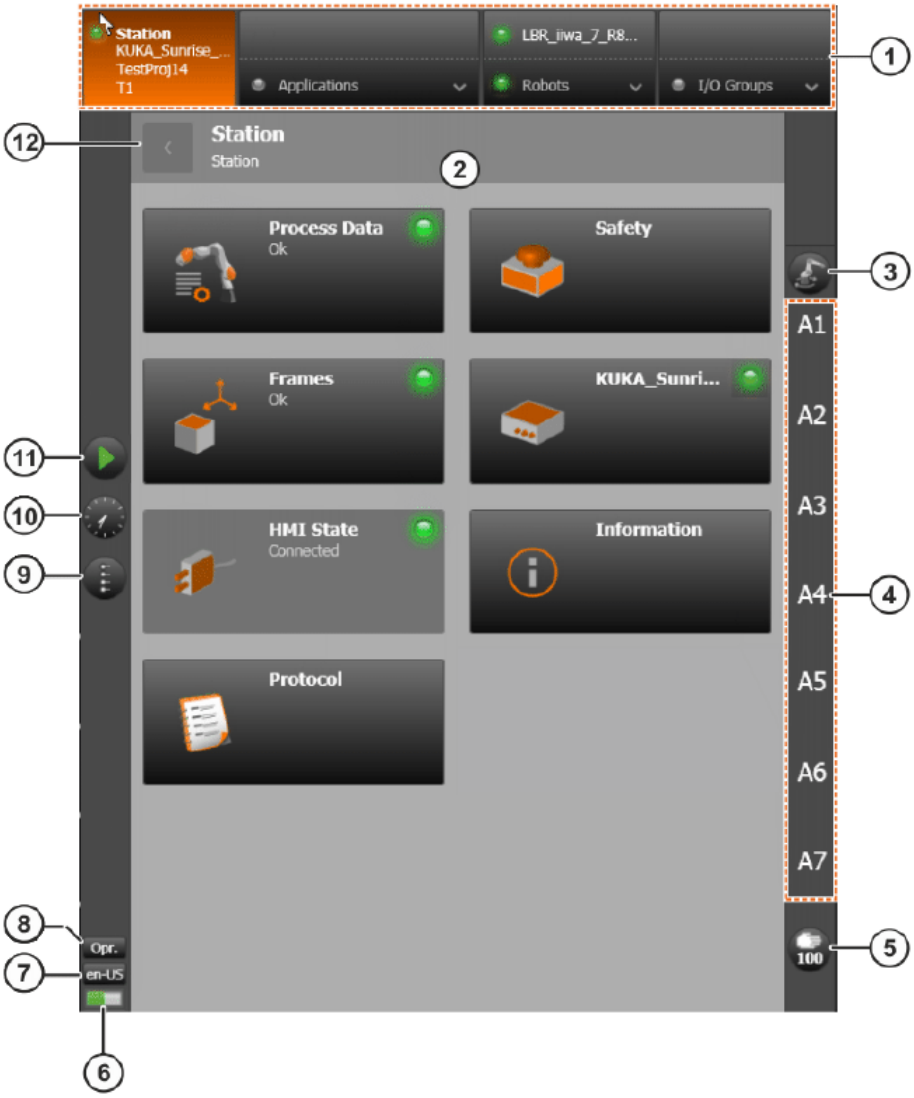


Figure 3.5: Station level of User interface (7)

1. A Navigation bar divided in 4 levels. Starting from left it contains:

   - Station level displaying the controller name and the selected operating mode;
   - Applications level displaying the selected robot application;
   - Robot level displaying the selected robot;
   - I/O groups level displaying the selected I/O group.

   Colored circles on the smartHMI the status of the system components: Red for error, Yellow for warning, Green for status OK, Grey for Status unknown.

2. A Display area of the level selected in the navigation bar, here the Station level. The main subsections of this level are:

   - Process Data, where are listed data processed by the controller;
   - Safety, containing the safety configuration to be activated/deactivated
   - Frames, including all the memorised frames.

3. Jogging options button displays the current coordinate system for jogging with the jog keys. Touching the button 1) opens the Jogging options window in, which can be set:
   2)the reference coordinate system;
   3)the desired Tool;
   4)TCP of the tool;
   5)Base selection.

Figure 3.6: Jogging options (7)

4. Jog keys display: shows different letters depending on the selected jogging.

5. Override button: indicates the current override, touching the button opens the Override window, in which the override and other velocity options can be set.

6. Life sign display: a steadily flashing life sign indicates that the smartHMI is active.

7. Language selection button

8. User group button: indicates the currently logged-on user group.

9. User key selection button

10. System time button
11. Jogging type button: displays the currently set mode of the Start key. Touching the button opens the Jogging type window, in which the mode can be changed.
12. Back button: return to the previous view by touching this button.

Robot level HMI

The Robot level in Figure 5.25 gives access to information and functionalities which affect the selected robot. Among the sections displayed:

- Load Data opens the Load data view for automatic load data determination;
- Calibration opens the calibration sublevel which contains the Base calibration and Tool calibration tiles.

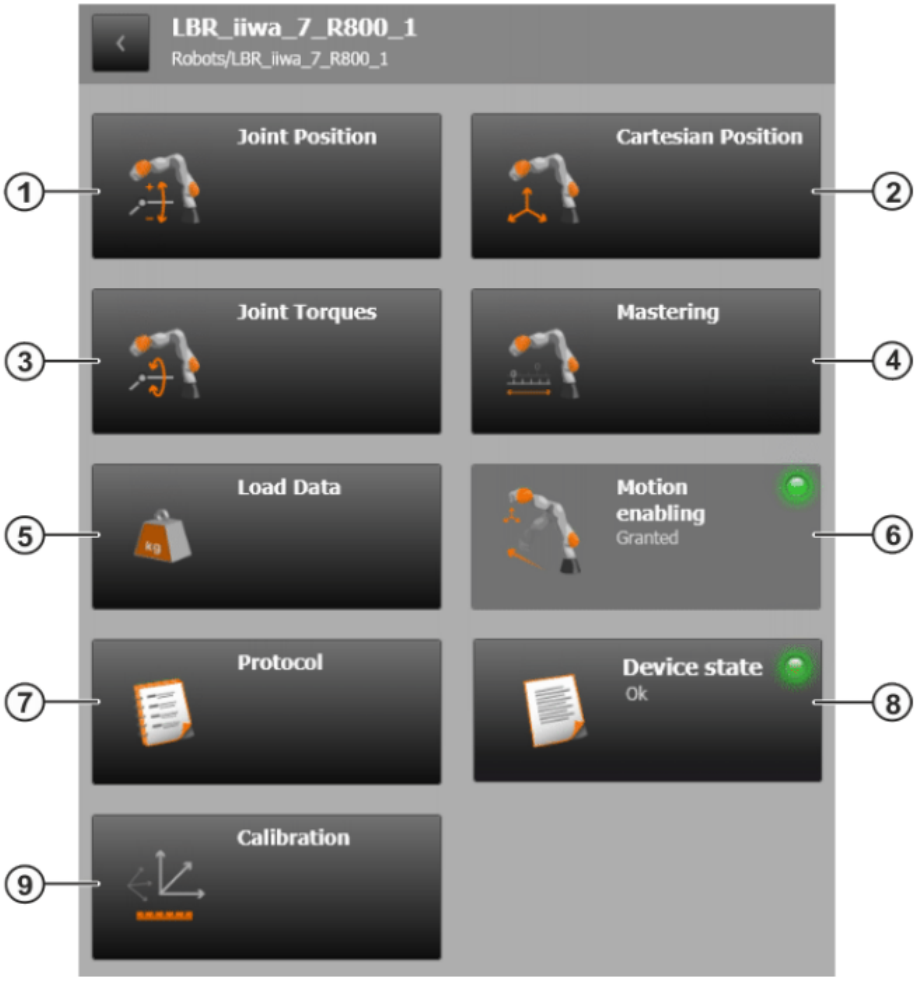


Figure 3.7: Robot level of User interface (7)

Coordinate systems

Coordinate systems or frames determine the position and orientation of an object in space. There are 5 possible types of coordinate system:

- World: is a permanently defined Cartesian coordinate system, located at the robot base by default. It is the root for all other coordinate systems;
- Robot base: is a Cartesian coordinate system originally coincident with the world coordinate system, as always located at the robot base, but can have a different mounting orientation;
- Base: is a Cartesian coordinate system required for defining Cartesian motions. It is usually an additional Coordinate system located in a relative position with respect to the World coordinate system. relative to the world coordinate system;
- flange: describes the current position and orientation of the robot flange center point. It does not have a fixed location and is moved with the robot. It is used as an origin for coordinate systems which describe tools mounted on the flange;
- Tool: a Cartesian coordinate system which is located at the working point of the mounted tool (TCP). Any number of frames can be defined for a tool and can be selected as the TCP. The origin of the tool coordinate system is generally identical to the flange coordinate system.

Position and orientation

To determine the position and orientation of an object, translation and rotation relative to a reference coordinate system are specified. 6 coordinates are used for this purpose:

- Distance X, Y and Z: Translations along the X, Y and Z axis of the reference system respectively;
- Angle A, B and C: Rotations about the Z, Y, X axis of the reference system respectively.

Frames overview

Frames are visible at station level in Figure 3.5. If we select that section, frames view shows up:

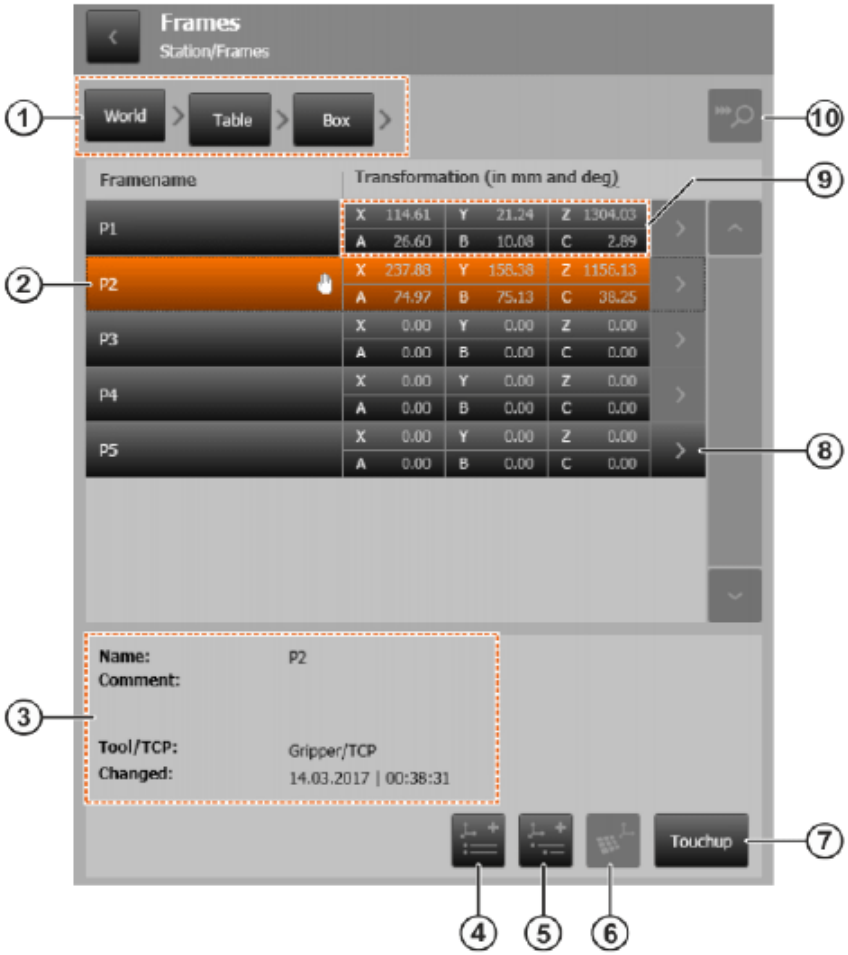


Figure 3.8: Frames view (7)

In 2) we see the different frames present. If we want to modify the selected orange frame with the robot's actual position, we can use the TouchUp button 7). 4) and 5) are used to create new frames or sub-frames, respectively. Frames are, in fact, hierarchically structured with a parent-child relationship. We can understand which part of the tree we are by looking at 1). 3) shows the information of the selected frame.

3.2 Introduction to Java

For programming the robot sequence and integrating external sensors, the LBR uses Java, a very popular object-oriented programming language. Object-oriented programming organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data. An object-oriented program can be characterized as data controlling access to code.(14) Among the advantages of this programming language are a worldwide distribution, a high recognition level, and rapid expandability through external libraries, some of which are open source.(15)

All object-oriented programming languages provide mechanisms that help you implement the object-oriented model. They are encapsulation, inheritance, and polymorphism:

- encapsulation is the mechanism that binds together code and the data it manipulates and keeps both safe from outside interference and misuse;
- inheritance is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification;
- polymorphism is a feature that allows one interface to be used for a general class of actions. The exact nature of the situation determines the specific action.

When properly applied, polymorphism, encapsulation, and inheritance combine to produce a programming environment that supports the development of far more robust and scalable programs than does the process-oriented model. A well-designed hierarchy of classes is the basis for reusing the code you have invested time and effort in developing and testing. Encapsulation allows you to migrate your implementations over time without breaking the code that depends on the public interface of your classes. Polymorphism allows you to create clean, sensible, readable, and resilient code.(14)

Interpreted and compiled languages

There are two main groups of programming languages, depending on how the machine executes the code.

- In a compiled language, the target machine directly translates the program. Compiled languages are converted directly into machine code that the processor can execute. As a result, they tend to be faster and more efficient to execute than interpreted languages. They also give the developer more control over hardware aspects, like memory management and CPU usage. However, compiled languages need a "build" step – they need to be manually compiled first. You need to "rebuild" the program every time you need to

make a change. Examples of purely compiled languages are C, C++, Erlang, Haskell, Rust, and Go. (14)

- Interpreted languages are significantly slower than compiled languages. But, with the development of just-in-time compilation, that gap shrinks. Interpreted languages generate an intermediary instruction set that is not recognizable as source code. The intermediary is not architecture-specific as machine code, either.

Java calls this intermediary form bytecode. This intermediary deployment can run anywhere. But a preinstalled interpreter is required to convert the intermediary code into machine code at runtime. The Java Virtual Machine (JVM) is the interpreter of Java for applications packaged and deployed as bytecode to run. The benefit of applications built with an interpreted language is that they can run in any environment. The drawback of an interpreted language is that the interpretation step consumes additional clock cycles, especially in comparison to applications packaged and deployed as machine code. The need for bytecode to be interpreted on a JVM before machine instructions can be fed to the CPU is often given as a reason why a Java application might be performing poorly.
Java is the most popular interpreted language. Others include Python, Scala, Ceylon, and Groovy.(16)

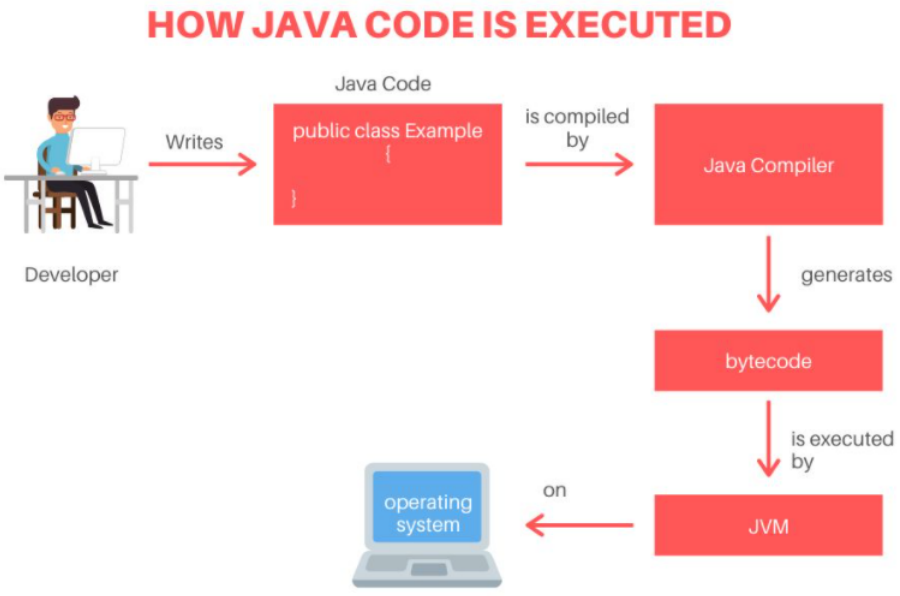


Figure 3.9: Java working principle
https://python-tricks.com/how-java-works/

Class concept

The class is at the core of Java. It is the logical construct upon which the entire Java language is built because it defines the shape and nature of an object. As such, the class forms the basis for object-oriented programming in Java. Any concept you wish to implement in a Java program must be encapsulated within a class. When you define a class, you declare its exact form and nature. You do this by specifying the data that it contains and the code that operates on that data. While elementary classes may have only code or data, most real-world classes include both. As you will see, a class code defines the interface to its data. The use of the class keyword declares a class. The data or variables defined within a class are called instance variables. The code is contained within methods. Collectively, the methods and variables defined within a class are called members of the class.
A simple example with two instances of a class is presented in Figure 3.12

```
// This program declares two Box objects.

class Box {
  double width;
  double height;
  double depth;
}

class BoxDemo2 {
  public static void main(String args[]) {
    Box mybox1 = new Box();
    Box mybox2 = new Box();
    double vol;

    // assign values to mybox1's instance variables
    mybox1.width = 10;
    mybox1.height = 20;
    mybox1.depth = 15;

    /* assign different values to mybox2's
       instance variables */
    mybox2.width = 3;
    mybox2.height = 6;
    mybox2.depth = 9;

    // compute volume of first box
    vol = mybox1.width * mybox1.height * mybox1.depth;
    System.out.println("Volume is " + vol);

    // compute volume of second box
    vol = mybox2.width * mybox2.height * mybox2.depth;
    System.out.println("Volume is " + vol);
  }
}
```

Figure 3.10: Example of class (14)

Classes are categorized in some containers called packages. For example, a package allows you to create a class named List, which you can store it in your package without concern that it will collide with another class called List stored elsewhere. Packages are stored hierarchically and are explicitly imported into new class definitions.

Now that we got the basic idea of Java, we will focus on using this programming language in the operating system dedicated to the applications development of the LBR iiwa: KUKA Sunrise OS. For more theoretical information about Java, source (14) is recommended.

3.3 KUKA Sunsrise OS

KUKA Sunrise OS is a system software package for industrial KUKA robots in which programming and operator control tasks are strictly separated from one another. It includes:

- KUKA Sunrise Workbench for developing Robot applications;
- WorkVisual for bus configuration and mapping.

KUKA Sunrise Workbench

KUKA Sunrise Workbench is the development environment for the robot cell (station). It offers the following functionalities:

- Start-up: system software installation, configuration of the robotic cell, changing the safety configuration, creating I / O configuration, transfer of the project to the robot controller;
- Application development: programming of robot applications in Java, project management and programs, runtime data modification and management, project synchronization and remote debugging.

By opening this software programming interface, the following page is shown:

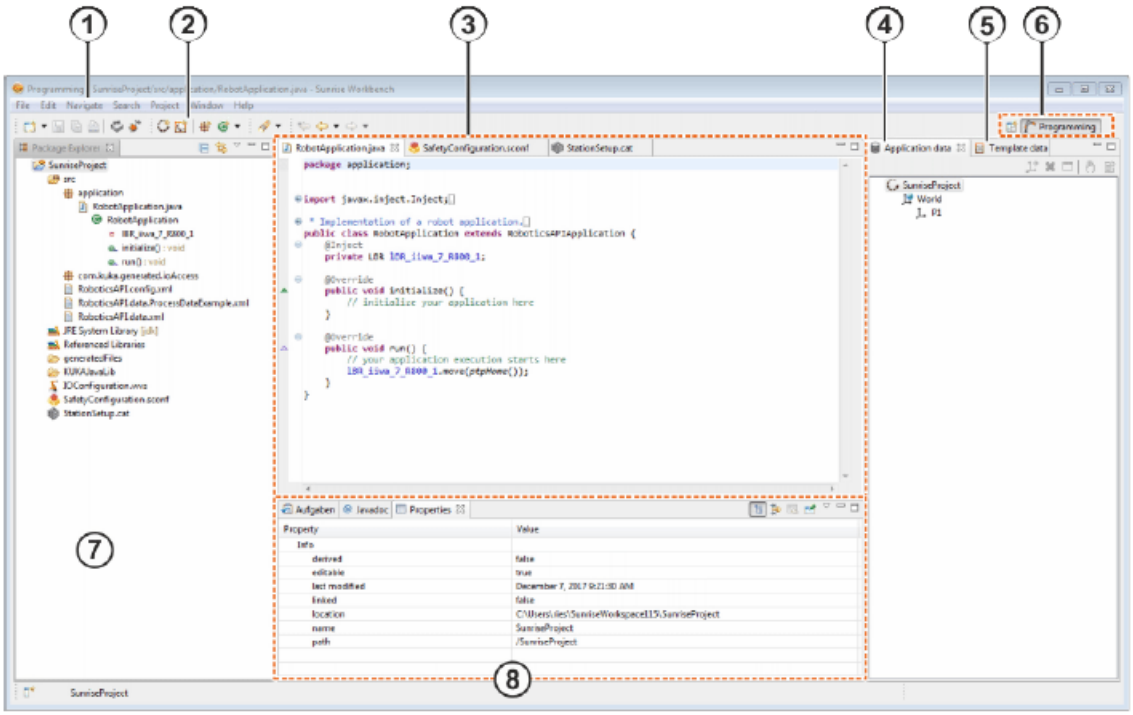


Figure 3.11: WorkBench overview (7)

Item	Description
1	Menu bar
2	Toolbars
3	Editor area
4	Application data view: Displays the frames created for a project in a tree structure.
5	Object templates view: displays the geometrical objects, tools and workpieces created for a project in a tree structure.
6	Perspective selection
7	Package Explorer view
8	Tasks/Javadoc/Properties view

Table 3.1: Workbench main page

Project overview

When a new project is created, it is visible in the Package Explorer view 7).

A project contains different items:

- One or many main applications developed in Java containing the desired task for the robot;

- One or many background Tasks (optional) running cyclically with a certain frequency (min 1ms) and gives for instance, information about the position of the robot or measured Cartesian forces;
- an API for Process data, where are present data of all the frames and the variables of applications/Background tasks you want to show on the smartPAD;
- I/O Access files automatically generated;
- Several types of libraries;
- A station Setup file;
- A safety configuration file
- IOConfiguration.wvs WorkVisual file

Safety configuration

When a new project is created, safety is the first thing it has to be assured. There is a default safety configuration in a new project that can be modified. Safety options are listed in a table composed by different rows. Each row is associated with a different safety functionality.

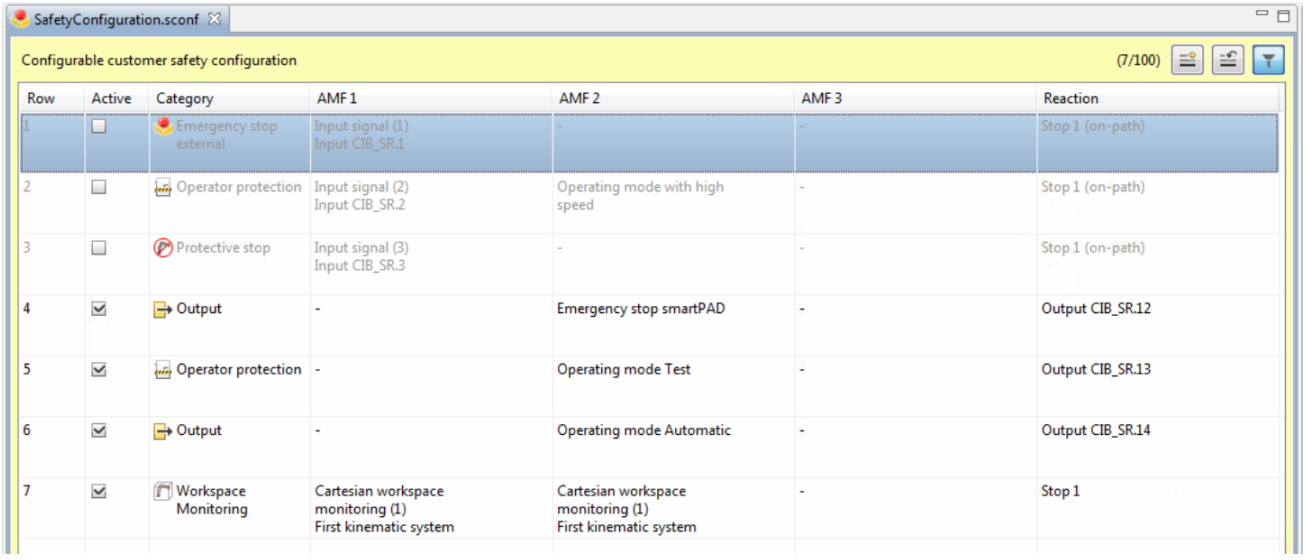


Figure 3.12: General view of safety rows

In addition to the default safety, we decided to add a row for avoiding the robot to touch the Workbench for our experiments. By adding a Workspace Monitoring condition in Figure 3.13 we put a limit on z-axis for stopping the robot when is too close to the Workbench. If the limit is exc eeded, a small red light will appear on the upper bar on the smartPAD and an RCC movement to a safe position will be required to move the robot again in a safe position.

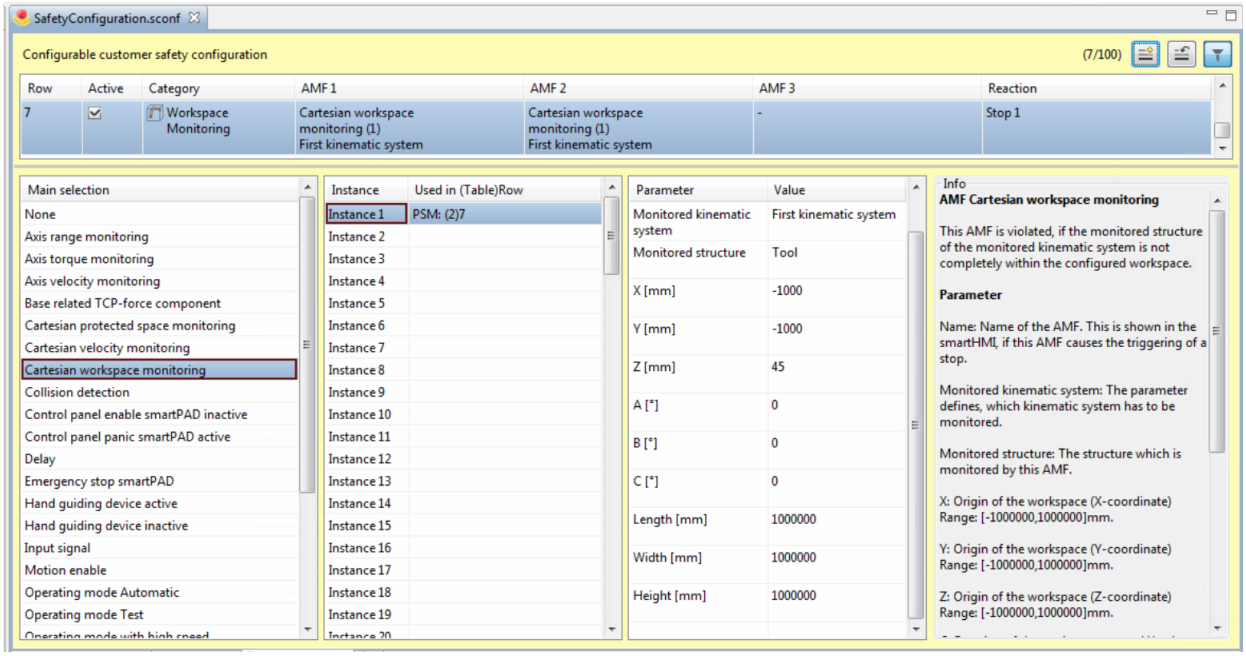


Figure 3.13: Row 7 setting

Workvisual

WorkVisual is another fundamental software for KUKA Sunrise OS where you have the possibility to create and export an IOConfiguration.wvs file that applications of your controller will use. In this place, it is possible to handle the different Inputs/Outputs connected to the robot controller and, in general, create a bus configuration and its mapping.

A robot cell (station) can then operate by using the KUKA smartPAD control panel programmed with this software. The smartPAD is required only in the start-up phase for activities that cannot be performed using KUKA Sunrise Workbench for practical or safety reasons. The smartPAD is used to master axes, calibration tools, and teaching points. After the start-up and development of the application, the operator can perform simple maintenance operations and operational activities using the smartPAD. The operator cannot change the station, safety configuration, or programming.

Compliant robot programming

For determining the current joint position, the KUKA LBR iiwa also has joint torque sensors in every axis, which allow the current joint torques to be measured. These data enable the use of an impedance controller and position control, thus making it possible to implement compliant behavior of the robot.(17) Control strategies for a robot manipulator in contact with an environment can be grouped in two categories: those performing open-loop force control and those performing direct closed-loop force control (5). To the first category belong the impedance control schemes (6) that achieve indirect force control by means of closed-loop position control; the position error is related to the contact force through a mechanical impedance of adjustable parameters.(7) In KUKA Sunrise environment, Impedance control is done by considering the robot as a model virtual spring-damper system with configurable values for stiffness and damping. Furthermore, additional forces and force oscillations can be overlaid. The special sensor technology and the available controller mechanisms make the KUKA LBR iiwa highly sensitive and compliant. This enables it to react quickly to process forces and makes it particularly suitable for a wide range of joining tasks and interaction with humans. In robot applications, the controller to be used is set separately for every motion command. As standard, the following steps are required for this:

- Create the controller object of the desired controller data type;
- Parameterize the controller object to define the control response;
- Set the controller as the motion parameter for a motion command.

One example of Cartesian impedance controller used in the experiments is the following:

```
//Defining controller parameters for individual degrees of freedom
private static final int stiffnessZ   = 50; //Small value along expected contact motion (Soft motion control)
private static final int stiffnessY   = 2500; //Large values along expected free motion (Stiff motion control)
private static final int stiffnessX   = 2500;
private static final int stiffnessROT = 300;

//Creating a Cartesian impedance controller
CartesianImpedanceControlMode ImpMode = new CartesianImpedanceControlMode();
ImpMode.parametrize(CartDOF.X).setStiffness(stiffnessX);
ImpMode.parametrize(CartDOF.Y).setStiffness(stiffnessY);
ImpMode.parametrize(CartDOF.Z).setStiffness(stiffnessZ);
ImpMode.parametrize(CartDOF.ROT).setStiffness(stiffnessROT);
ImpMode.parametrize(CartDOF.Z).setAdditionalControlForce(10);
ImpMode.parametrize(CartDOF.ALL).setDamping(0.9);

//Using Impedance Control Mode while performing the desired task
toolcp.move(lin(getApplicationData().getFrame("/P1")).setCartVelocity(2).breakWhen(ContactForceReached));
getLogger().info("Hold position in impedance control mode");
toolcp.move((new PositionHold(ImpMode, 7, TimeUnit.SECONDS)));
```

Figure 3.14: Cartesian impedance controller programming

3.4 Introduction to ROS

The Robotic Operating System (ROS) is an open-source, meta-operating system for your robot. It provides hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It provides libraries and tools to help software developers create robot applications. It supports all major host operating systems, even if Ubuntu is recommended.
ROS is not real-time, but this characteristic can be implemented within a single node.

Nodes
ROS is a distributed framework of processes (aka nodes) that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into Packages, which can be easily shared and distributed. A node is a process (application) that performs some computation. The software functionalities are implemented in different modules, each one running over a single or multiple nodes. Nodes may reside in different machines transparently. They operate at a fine-grained scale and a robot control system will usually comprise many nodes.(5)

Nodes communication
Communication between nodes is made by using streaming topics, RPC services, and the Parameter Server.

- Topics are named buses over which nodes exchange messages. They have anonymous publish/subscribe semantics. There can be multiple publishers and subscribers to a topic as we have a many-to-many communication. Transport uses TCP or UDP;
- Nodes communicate with each other by publishing messages to topics. A message is a simple data structure comprising typed fields. All messages are defined in text files;

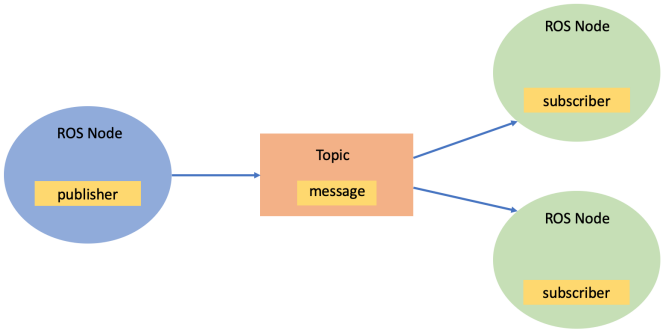


Figure 3.15: Topics and messages communication

- Service is a mechanism for a node to send a request to another node and receive a response in return. It follows a request-response design pattern. A service is called with a request structure, and in return, a response structure is returned.

Figure 3.16: Service communication

3.5 PLC Beckhoff TwinCAT 3

TwinCAT, short for The Windows Control and Automation Technology, is part of a PC based platform for industrial automation that 'transforms' a normal PC into a real time system based on three elements:

- a hardware, that is the PC;
- a field bus, typically EtherCAT;
- a software platform, the TwinCAT application.

The software integrates a highly modularized development environment, totally implementable on VisualStudio. Each element has a uniform behavior and acts as part of a state machine.
To assimilate the operation of the PC to an industrial controller, it contains components suitable for a real-time extension of Windows, required to manage control tasks. However, a large difference in performance and operating mode is still present between a normal PC and an Industrial PC. There will be some limitations in the PC, called the Local device, especially if online simulations are started; in this case, the expected results may not be guaranteed, unlike what happens with the IPC, called the Target device.

The PLC programming part supports the languages of IEC-61131-3, among which Structured Text is preferred overall. It is possible to manage virtual PLC machines that can control n tasks (with a maximum of 60). Also, Motion Control for real and virtual axes is possible.
Several bus types are supported, but EtherCAT remains the best choice.

TwinCAT platform can be summarized on two pillars:

- Real-time management;

- ADS protocol (Automation Device Specification).

Real-time management is carried out by a low-level scheduler that coexist with the Kernel and works in parallel with it. This scheduler is equipped with hardware components that make it able to perform preemptive tasks in a deterministic way (maximum jitter of 12 microsec), with also priority control. Each task has two main parameters. Priority is a number that can be assigned automatically or by an operator, from 1 to 61, where 1 is the maximum value. Cycle time is an integer multiple of the base time, relative to the CPU to which the task has been assigned.

The second pillar of Twincat is the ADS Protocol. It outlines a system architecture based on two elements: XAE (eXtended Automation Engineering): the development engineering part in Visual Studio contained locally, i.e. on the user's PC; XAR (eXtended Automation Runtime): the runtime part present on the Target device, i.e. the Beckhoff controller.

The Twincat XAE 3.17 project template is divided into:

- system configuration;
- motion control configuration;
- PLC configuration;
- Safety PLC configuration;
- C ++ module configuration;
- I/O configuration.

ADS is the TwinCAT protocol used to make servers communicate with their respective clients and servers with each other. Beckhoff owns the protocol. It is implementable by anyone and is the system's opening point towards any other device/software equipped with an ADS interface. Any module equipped with an ADS interface can interact with all the others. The protocol works through a typical Client-Server model and relies on TCP-IP network protocol.(18)

The icon indicating the operating status of the module is shown at the bottom of the PC notification area and has three possible configurations:

- red, all high-level features are inactive;
- blue, the scheduling part (real time) does not work, but the high-level functions are enabled and in this case, only Windows is working;
- green, the real-time system is running, you have the normal operation of a TwinCAT project in which all the tasks are fragmented and interposed to the execution of the operating system.

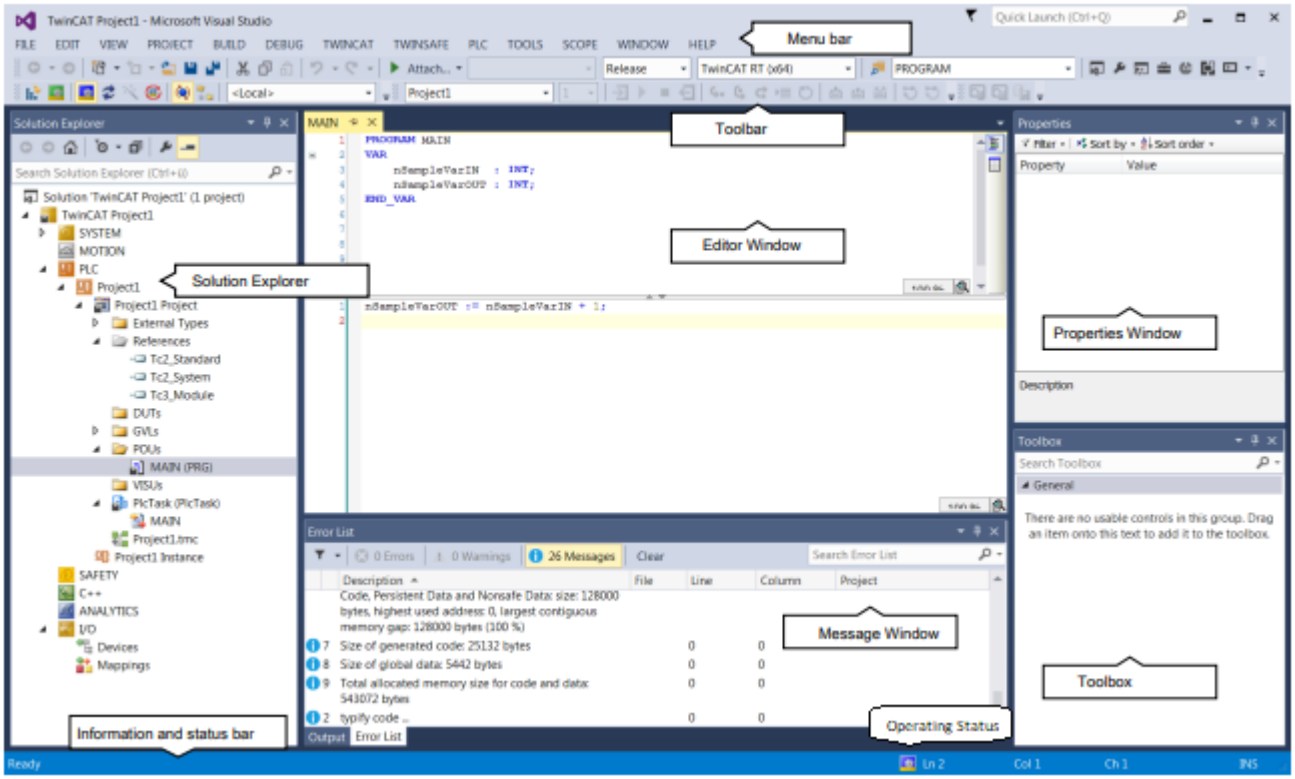


Figure 3.17: Twincat XAE project template (18)

EtherCAT fieldbus

EtherCAT is a fast Ethernet-derived bus. The network bit-rate is fixed at 100 Mbit/s. Typical network scan times are tens/hundreds of microseconds.

EtherCAT is a master-slave line in which there are 2 protocols:

- EDP protocol (EtherCAT Device Protocol) real-time and deterministic, as it possesses the qualities of data precision and repetition (data latency in a band of 100ns thanks to the mechanism of distributed clocks). The master side uses a standard network port with a specific real-time driver.
  The slave side relies on a specific controller giving the protocol the property of being exclusive, thus making it run only with its specific frames;

- EAP protocol (EtherCAT Automation Protocol): created to exchange data between different EtherCAT masters, it requires a standard LAN network and is not real-time, but still guarantees good response times. However, the TwinCAT Ethernet Protocol driver is required.

EtherCAT uses a specific RJ45 cable with a minimum specification of 5-B. The network topology is free. It is essential to exactly reconstruct the path of the EtherCAT frame on the

network for diagnostics. With ports in auto-close mode, any information arriving from an upstream network segment will travel around the network downstream from that port. The EtherCAT network is always a logical ring. The high speed of EtherCAT is also given by the on-the-fly updating of the data, which will therefore not be buffered. The master has a logic designed to save bandwidth and performs an automatic mapping.

All EtherCAT terminals communicate via a cyclic channel, which ends within a cycle, and an acyclic channel (mailbox), in which parameterization, diagnostic services or in general actions that may require a certain number of cycles are done.

ISO/OSI Layer		EtherCAT	
Host layers	7. Application	HTTP*, FTP*	• Cyclic Data Exchange • Mailbox Acyclic Data Access
	6. Presentation	—	—
	5. Session	—	—
	4. Transport	TCP*	—
Media layers	3. Network	IP*	—
	2. Data link	• Mailbox/Buffer Handling • Process Data Mapping • Extreme Fast Auto-Forwarder	
		Ethernet MAC	
	1. Physical	100BASE-TX, 100BASE-FX	

*optional, the TCP/IP Stack shown is not needed for typical fieldbus devices.
EtherCAT master can access all data including name and data types of an EtherCAT slave without complex tools.
EtherCAT uses Standard Ethernet (IEEE 802.3 - Ethernet MAC and PHY) without modifications.

Figure 3.18:
ISO/OSI Reference model for EtherCAT
https://en.wikipedia.org/wiki/EtherCAT

I/O Modules

Among the powerful features of Beckhoff there is a wide range of I/O modules. There are different categories of terminals that can be mounted on a Beckhoff PLC. For our experiments we will use two types of modules, both mounted on an EK1100 coupler and communicating with EtherCAT: EL3008 and EL6692. The EL2004 output module visible in Figure is not used, while EL9011 is a bus end cover.

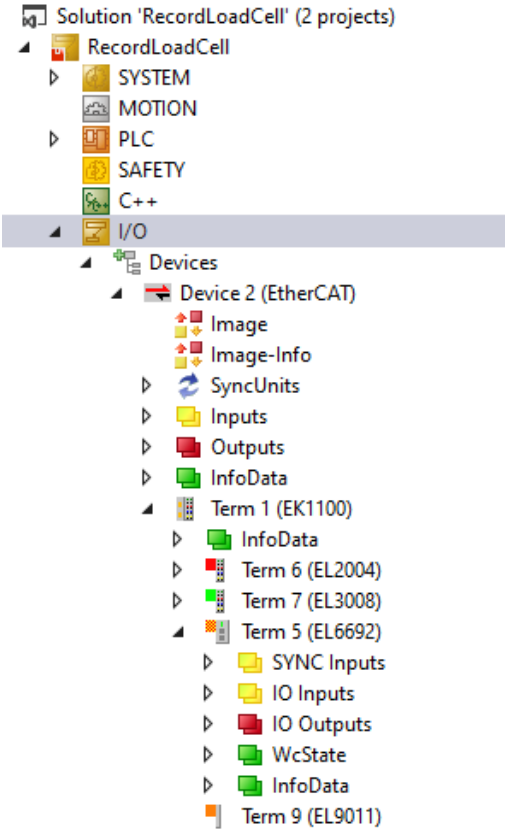


Figure 3.19: I/O Configuration

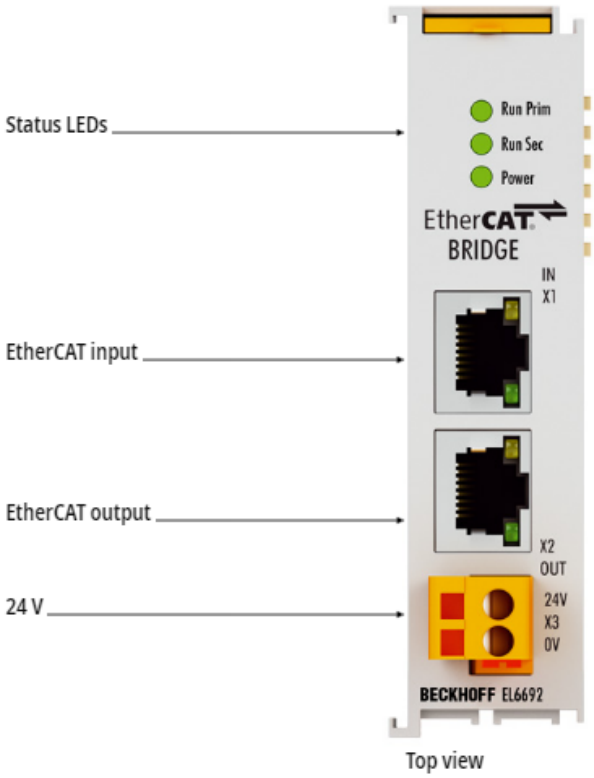


Figure 3.20: EL6692 Ethercat Bridge (18)

.

EL6692 EtherCAT Bridge

The EtherCAT bridge terminal EL6692 enables real-time data exchange between EtherCAT strands with different masters. It also enables synchronization of the distributed clocks of the individual strands. The power supply on the primary side (E-bus) comes from the E-bus, on the secondary side (RJ45) via an external connection. (18)

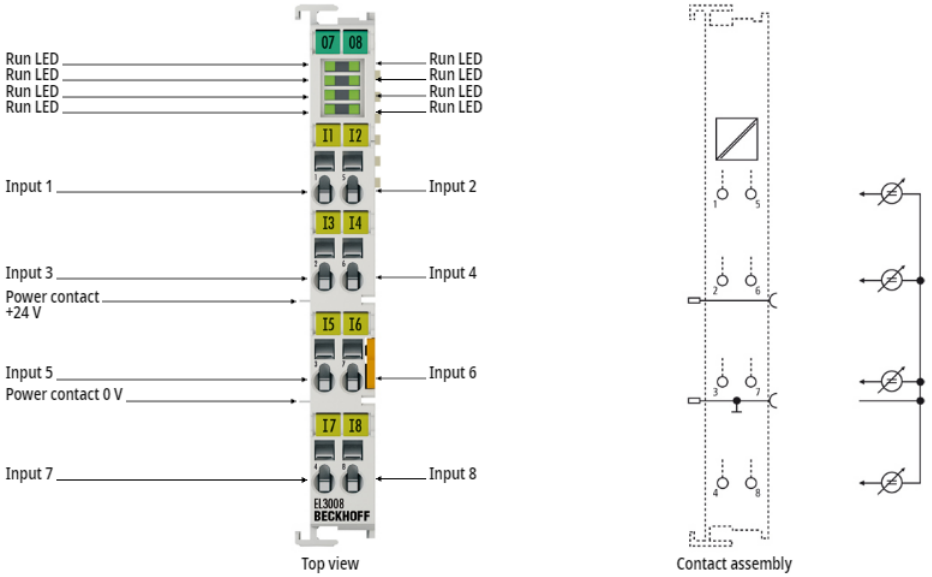


Figure 3.21: EL3008 Terminal (18)

EL3008 Input Module

The EL3008 analog input terminal processes signals in the range between -10 and +10 V. The voltage is digitized to a resolution of 12 bits, and is transmitted, electrically isolated, to the higher-level automation device. The power contacts are connected through. The reference ground for all inputs is the 0 V power contact. Light-emitting diodes indicate the signal state of the EtherCAT Terminal. (18)

I/O Mapping

Mapping is the association between logical and physical elements in the field that will be allowed through link property. Since the connection between the two components is established, whatever happens on the physical component will be reflected on the logical object and vice versa. Once verified, the link property is invariant to code and bus; it will be possible to import and export without modifying it.

3.6 Load cell

Force Sensors

There are several types of force sensors used in robotics. Among them, Force/Torque (F/T) are multi-axis sensors that measure forces and torques in the three cartesian directions. They are a potent tool that can be used for impedance control. However, they can be very expensive.
For this reason, simpler solutions as load cells can be preferred. A load cell is a type of force sensor that returns a signal proportional to the mechanical force applied to the system when connected to appropriate electronics. It can be hydraulic, pneumatic, or, most commonly, based on strain gauges.[7] In the last case, the calculation is reduced to the measurement of the strain-induced by force applied to an extensible element of convenient features. Therefore, an indirect measurement of force is obtained employing measurements of small displacements. The basic component of a load cell is the strain gauge which uses the change of electric resistance of a wire under strain.(19)

Strain Gauges

A strain gauge is a flexible backing or carrier material made of plastic or paper, on which a zig-zag pattern of resistance wire is installed. The nominal gauge resistances are typically 60, 120, 240, 350, 500, and 1000 Ω. The most common value is 120 Ω.
The resistance is connected to two twin wires at the extremities. When the strain gauge is stretched or compressed, the resistance slightly increases or decreases.
However, we cannot guarantee any good measurement with a strain gauge directly glued to the surface. There are two critical effects in the signal-conditioning techniques used for SGs. The first is the small, fractional changes in resistance that require carefully designed resistance measurement circuits, usually very difficult to build. The second effect is the need to compensate for temperature effects to eliminate masking changes in strain.(19)

Wheatstone Bridge

To solve these undesired effects, the strain gauge is inserted in one arm of a Wheatstone bridge which is balanced in the absence of stress on the strain gauge itself. By looking at Figure 2.1, the voltage balance in the bridge can be easily calculated as:(20)

$$V_o = \Big(\frac{R_2}{R_1 + R_2} - \frac{R_s}{R_3 + R_s}\Big) V_i \tag{3.1}$$

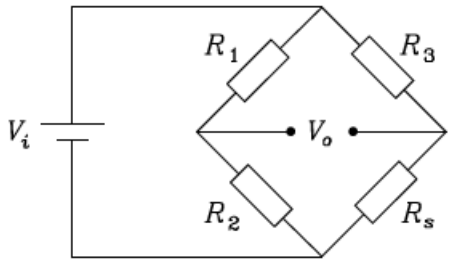


Figure 3.22: Wheatstone Bridge

Using a dummy gauge, we can also provide the required temperature compensation. In particular, the dummy is mounted in an insensitive orientation but in the same proximity as the active SG. Then, both gauges change in resistance from temperature effects, but the bridge does not respond to a change in both strain gauges. Only the active SG responds to strain effects. This is called a one-arm bridge.(20)

Load Cell DataSheet

Every load cell has its data sheet where all the main mechanical and electrical characteristics are summarized. In Figure 3.25 is shown the datasheet of the single-axis load cell we decided to use for the experiments. Notice that there are several types of load cell on the market. For our purpose, this one is the most suitable solution, especially for its simplicity to be mounted on the pneumatic flange we have at the end effector of the LBR. A three-axis load cell would have been a better solution for recording data on the three Cartesian coordinates contemporary. However, it's not easy to find this solution and it's often required to rely on expensive custom components, difficult to find, and long production times.

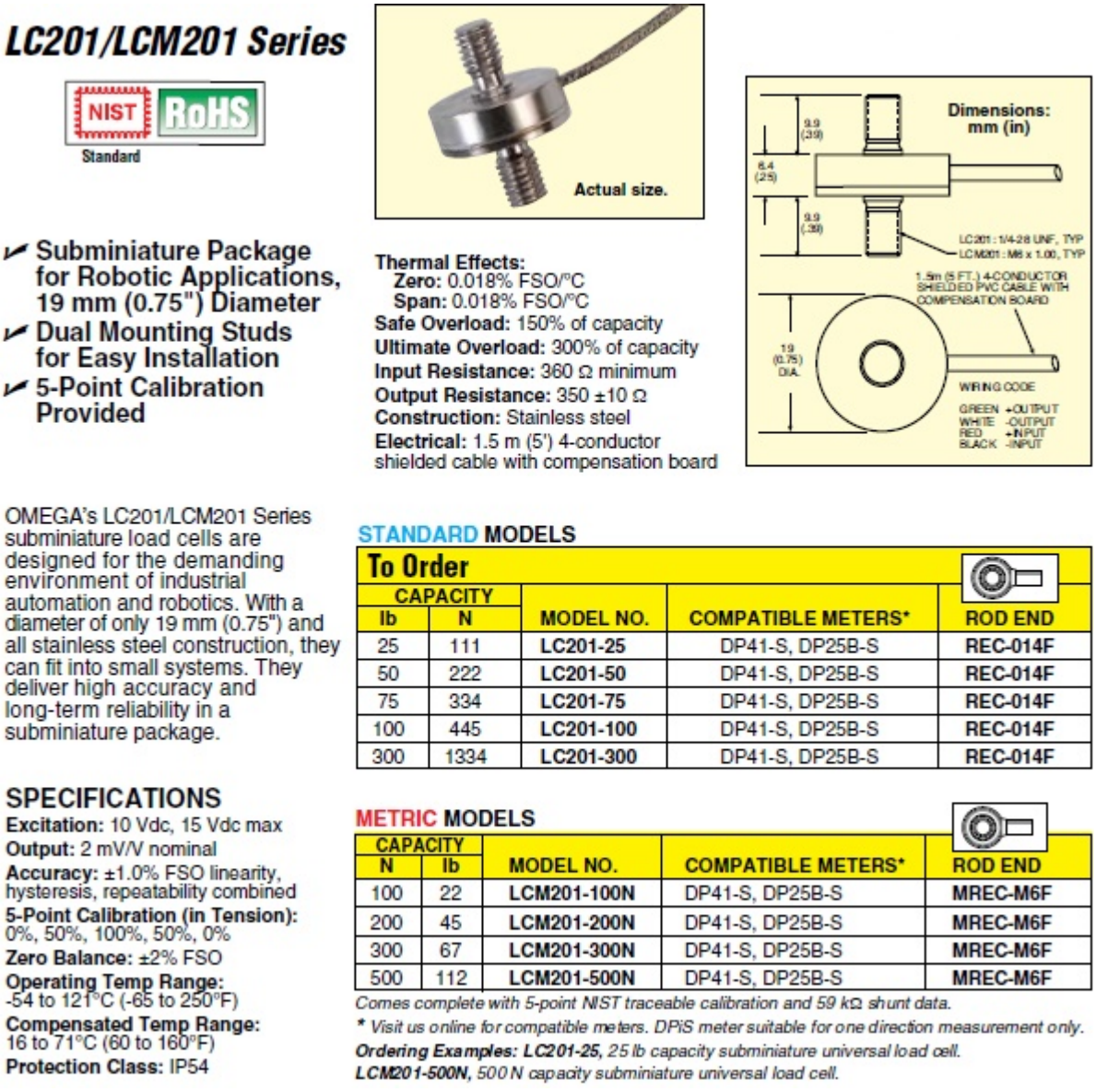
Mechanical Characteristics

The dimensions, weight and other features as the material are present in the datasheet. We understand that we choosed a very compact solution suitable for robotic applications by looking further in detail. It can be easily mounted to our robot by designing an interflange with only one threaded hole, seen by looking at the technical design attached to the Data Sheet.

Electrical Characteristics

There exist two types of wiring, depending on the cable that can be made of four or six wires. Besides having + and - signal and + and - excitation lines, a six-wire load cell also has + and

- sense lines. These sense lines are connected to the sense connections of the indicator and tell the indicator the actual voltage at the load cell. Sometimes there is a voltage drop between the indicator and load cell. The indicator either adjusts its voltage to make up for the voltage loss or amplifies the return signal to compensate for the loss of power to the cell. Load cell wires are color coded to help with proper connections. The data sheet contains the color code information. (21) In this case we consider a four wire load cell coloured as shown in figure 2.2.



LC201/LCM201 Series

NIST RoHS Standard

- Subminiature Package for Robotic Applications, 19 mm (0.75") Diameter
- Dual Mounting Studs for Easy Installation
- 5-Point Calibration Provided

Actual size.

Thermal Effects:
Zero: 0.018% FSO/°C
Span: 0.018% FSO/°C
Safe Overload: 150% of capacity
Ultimate Overload: 300% of capacity
Input Resistance: 360 Ω minimum
Output Resistance: 350 ±10 Ω
Construction: Stainless steel
Electrical: 1.5 m (5') 4-conductor shielded cable with compensation board

Dimensions: mm (in)
9.9 (.39)
6.4 (.25)
9.9 (.39)
LC201 : 1/4-28 UNF, TYP
LCM201 : M6 x 1.00, TYP
1.5m (5 FT.) 4-CONDUCTOR SHIELDED PVC CABLE WITH COMPENSATION BOARD
19 (0.75) DIA.
WIRING CODE
GREEN +OUTPUT
WHITE -OUTPUT
RED +INPUT
BLACK -INPUT

OMEGA's LC201/LCM201 Series subminiature load cells are designed for the demanding environment of industrial automation and robotics. With a diameter of only 19 mm (0.75") and all stainless steel construction, they can fit into small systems. They deliver high accuracy and long-term reliability in a subminiature package.

STANDARD MODELS

To Order

CAPACITY lb	CAPACITY N	MODEL NO.	COMPATIBLE METERS*	ROD END
25	111	LC201-25	DP41-S, DP25B-S	REC-014F
50	222	LC201-50	DP41-S, DP25B-S	REC-014F
75	334	LC201-75	DP41-S, DP25B-S	REC-014F
100	445	LC201-100	DP41-S, DP25B-S	REC-014F
300	1334	LC201-300	DP41-S, DP25B-S	REC-014F

SPECIFICATIONS

Excitation: 10 Vdc, 15 Vdc max
Output: 2 mV/V nominal
Accuracy: ±1.0% FSO linearity, hysteresis, repeatability combined
5-Point Calibration (in Tension): 0%, 50%, 100%, 50%, 0%
Zero Balance: ±2% FSO
Operating Temp Range: -54 to 121°C (-65 to 250°F)
Compensated Temp Range: 16 to 71°C (60 to 160°F)
Protection Class: IP54

METRIC MODELS

CAPACITY N	CAPACITY lb	MODEL NO.	COMPATIBLE METERS*	ROD END
100	22	LCM201-100N	DP41-S, DP25B-S	MREC-M6F
200	45	LCM201-200N	DP41-S, DP25B-S	MREC-M6F
300	67	LCM201-300N	DP41-S, DP25B-S	MREC-M6F
500	112	LCM201-500N	DP41-S, DP25B-S	MREC-M6F

Comes complete with 5-point NIST traceable calibration and 59 kΩ shunt data.
** Visit us online for compatible meters. DPiS meter suitable for one direction measurement only.*
***Ordering Examples: LC201-25**, 25 lb capacity subminiature universal load cell.*
***LCM201-500N**, 500 N capacity subminiature universal load cell.*

Figure 3.23: Load Cell DataSheet (21)

Specifications

Specifications are the main section of a load cell DS; the most relevant are the following:

- Excitation: Recommended DC input voltage to the excitation circuit;
- Output (Full Scale): The output voltage the load cell produces at rated capacity per excitation volt at the input terminals, minus the output voltage at minimum load. The output voltage at rated capacity is calculated by multiplying the full-scale output by the excitation voltage.
- Accuracy: percentage of the most important performance parameters as the hysteresis, linearity and repeatability;
- Zero Balance: The output voltage the load cell produces per volt of excitation at no load.;
- Compensated Temperature Range: The temperature range over which the load cell is compensated to maintain rated capacity and zero balance within specification limits. (19)

In the experiments, we made use of LC201-25. It is important to highlight that we don't have measurement units belonging to the SI for this model. But, we have a maximum force capacity of 25 lb and a thread measuring 1/4 28 UNF.

3.7 Load Cell amplifier

Strain gauge sensors are low-impedance devices that require exact excitation for the input due to their signal output in milli-volt range. Unfortunately, the range of output values is not suitable for measurements. Load cells need to be conditioned for the next stages with low cost, high precision, reliability, amplification, and noise reduction capability to have acceptable values. The low output signal after amplification, after being filtered, will be the input of an I/O model of the PLC.

SEM1600/B Load Cell conditioner

For our purposes we decided to use SEM1600/B, a "smart" powered bridge amplifier (or conditioner) for use with strain gauges or load cell signals. The product has a built in capability to scale the input signal to a process value while the output stage offers either voltage, bipolar voltage or active / passive current re-transmission signals. The product comes with an AC/DC power supply that will operate in the range (10 to 48) V DC and (10 to 32) V AC making the

device ideal for battery operation. An additional volt free contact input is available using a remote switch for tare setting. The high precision input stage of the device allows for a bridge excitation voltage of 5V DC to be used as opposed to the traditional 10V DC. This reduces the bridge supply's power requirement, and up to four bridges (cells) may be connected to the input.

This product uses a USB port for configuration, together with a simple to use menu-driven software configuration tool, allowing the user to take advantage of the product's comprehensive specification. Additionally, the user may read live process data when connected to the PC, allowing for offset and span calibration.

If a configuration is not specified at the time of order, the product will be shipped with the default range 2 mV/V input (4 to 20) mA output.(22)

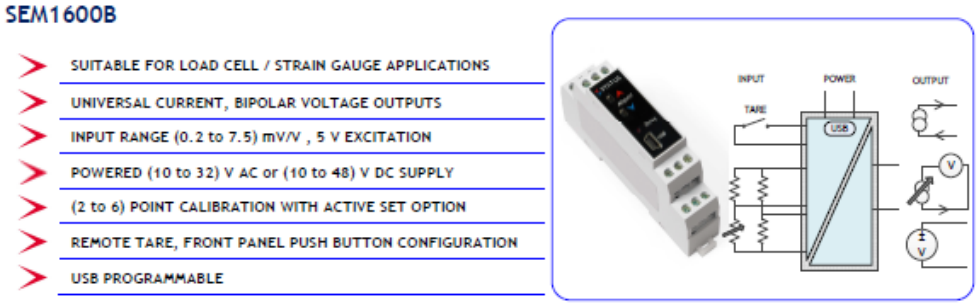


Figure 3.24: Amplifier schematic (22)

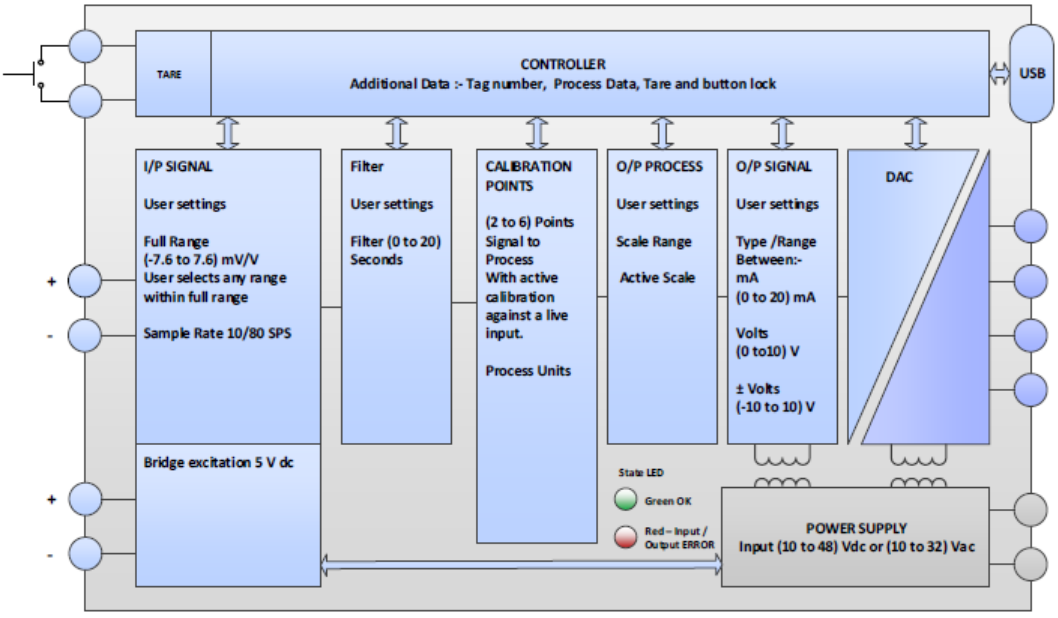


Figure 3.25: Amplifier characteristics (22)

Chapter 4

Experimental setup

For performing the desired tests and experiments we considered the following experimental setup:

- A KUKA LBR iiwa 14 r820 with a I/O pneumatic flange installed;
- A load cell LC201-25 mounted on the robot for measuring external force on Z-axis;
- A mono-axis load cell amplifier SEM1600B for conditioning the load cell output value
- Four custom tools designed for mounting the load cell on the robot flange and to perform the experiments;
- A Beckhoff PLC with the I/O modules required for communicating with both robot and load cell

4.1 Tools Design

We used Autodesk Fusion 360 for designing the required components:

- An interflange to be inserted between the load cell and the I/O pneumatic flange at the end effector of the robot;
- A spherical tool in order to have a point contact
- A planar tool in order to have a planar surface creating a different contact surface during dynamic experiments;
- A plastic holder to mount a ball castor wheel tool.

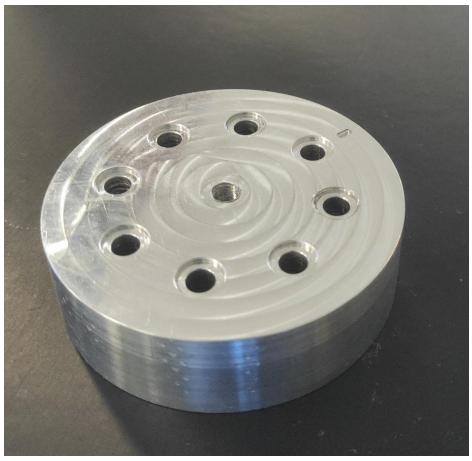


Figure 4.1: Interflange

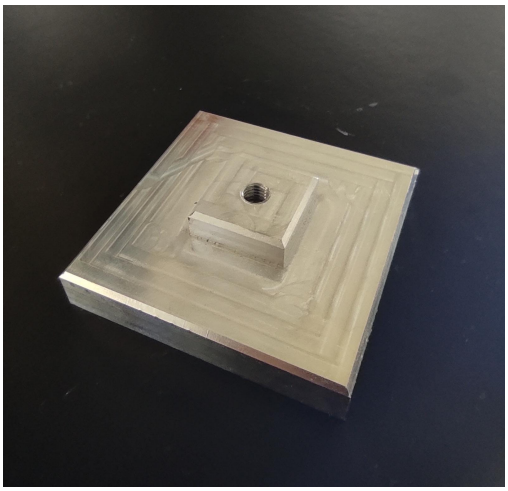


Figure 4.2: Planar Tool

.

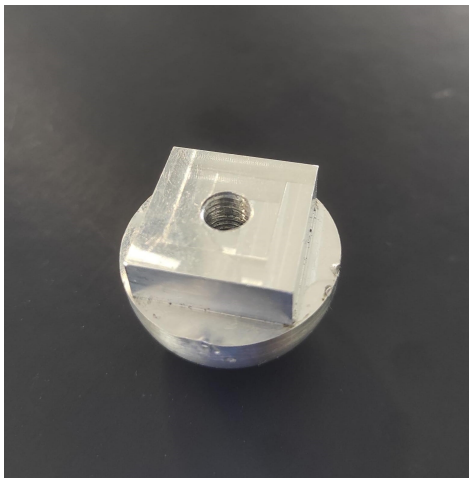


Figure 4.3: Spherical tool

Figure 4.4: Ball castor wheel tool

.

Thanks to Fusion360, it was possible to make an assembly with the designed tools and an open source 3D model of the LBR for testing the correct fixing of the components before printing them. It is shown the example with the spherical tool mounted (for the planar tool is similar).

All four components were first made in plastic materials with a 3D printer to save money in case of minor mistakes. Later on, the first three were reproduced in aluminium by CNC machining. In this way, it was possible to have good tools with less elastic deformation for accurate measurements. Subsequently, a small cable tie was also printed for keeping the load cell cable to the interflange.

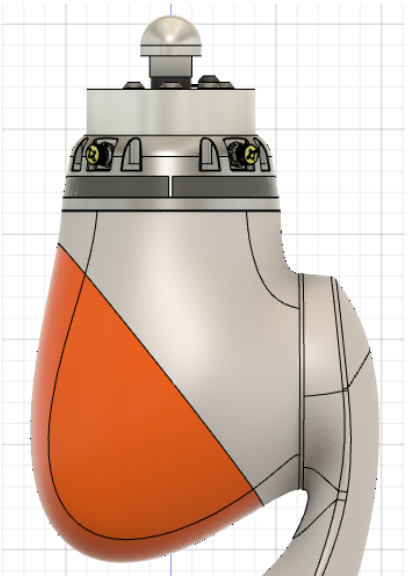


Figure 4.5: Lateral view with mounted tools

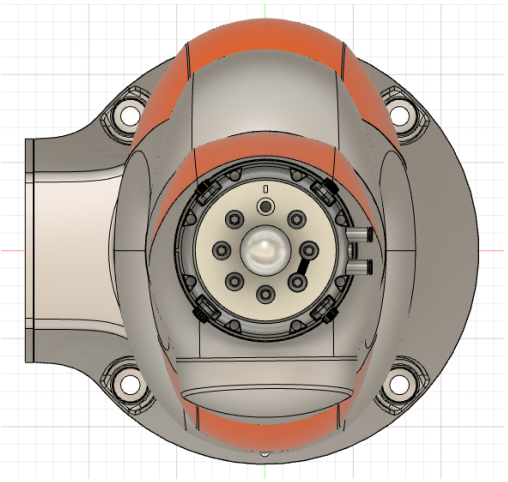


Figure 4.6: Vertical view of the LBR

.

4.2 Load cell to PLC software interface and setting

Before calibrating and using a Load cell for measurement, it is advisable to follow some preliminary steps to verify its correct functioning. We connect a multimeter to check if the measured values of Input and Output resistance match approximately with the values in the calibration certificate.

Balance	0.094	mVdc
Sensitivity	22.515	mVdc
In Resist	375.60	Ohms
Out Resist	351.90	Ohms
59K Shunt	14.847	mVdc

Change at 0.00 LBS (-INPUT to -OUTPUT)

Calibration Factors: Sensitivity = 2.252 mV/V 59K Shunt = 1.485 mV/V

Figure 4.7: Calibration certificate

Input resistance is the resistance across the two input leads that go into the Load cell. In this case, it is measured between the black and red wires.

The output resistance is the measured resistance between the signal leads coming from the load cell, green and white in this component.

As we can see, compared to Figure, the values are almost equal. We proved that the load cell correctly works and the Wheatstone bridge is balanced.

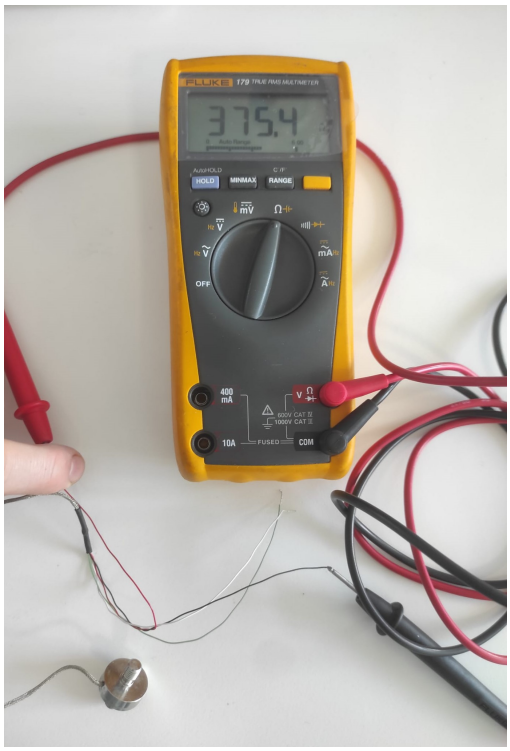


Figure 4.8: Input resistance check

Figure 4.9: Output resistance check

.

Amplifier Connection and configuration

As the measured value of a load cell has a range of output values in milliVolts, we have to connect the load cell to an amplifier. By looking at the datasheet of the SEM1600 B Conditioner, we correctly connect the load cell wires, and we then excite the amplifier with 5VDC as required. A green light will appear when the cables are all connected properly. Other two wires will be connected to the amplifier output, which will be the input of one port of the I/O module EL3008 of Beckhoff PLC.

In this case, the configuration is not specified, and the product will be shipped with a default range 2 mV/V input (4 to 20) mA output. As our desired output has to be an analog voltage, we connect to the USB port of the conditioner to be able to change some settings as desired. We use Omega's USB SpeedLink software to set the output to be between 10/-10 Volts.

We also have to select the update frequency higher than the default 20 Samples per Second (SPS). It can be raised up to 80 SPS, and in this way, we will have enough samples to collect.

Last, the Sensitivity calibration factor (4.7) introduced. voltage, we connect to the USB port of the conditioner to be able to change some settings as desired. We use Omega's USB SpeedLink software to set the output to be between 10/-10 Volts. We also have to select the update frequency higher than the default 20 Samples per Second (SPS). It can be raised up to 80 SPS, and in this way, we will have enough samples to collect. Last, the Sensitivity calibration

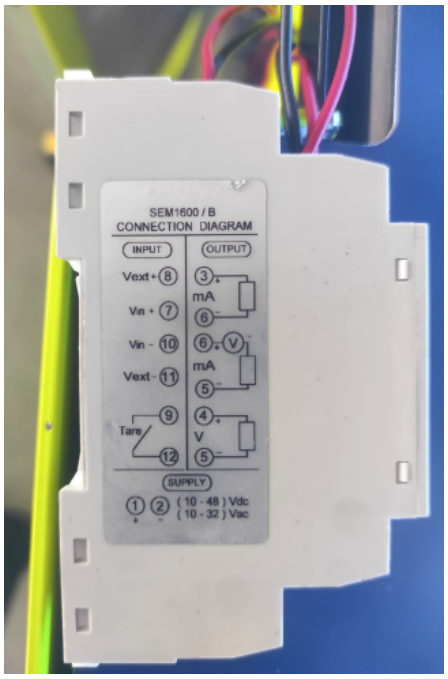


Figure 4.10: Conditioner schematic

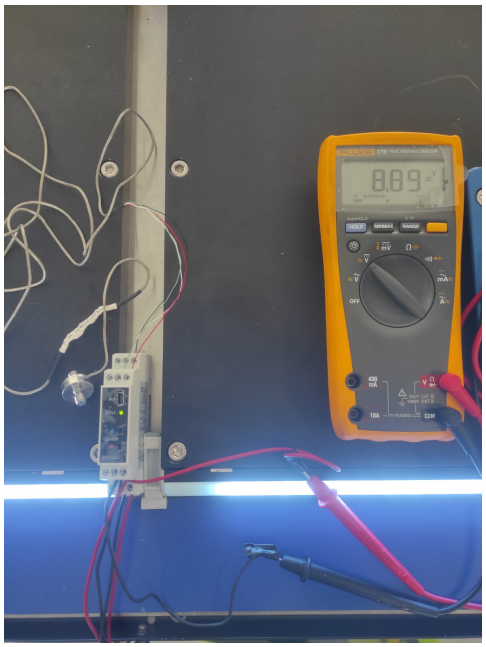


Figure 4.11: Conditioner at zero load .

factor (4.7) is introduced. The output is now supposed to be in the Volt range, and it is verified by using a multimeter. As expected, the value is in the volt range (4.11), and we can now proceed with the setting of the load cell.

The output is now supposed to be in the Volt range, and it is verified by using a multimeter. As expected, the value is in the volt range (4.11), and we can now proceed with the setting of the load cell.

Load cell setting

It has to be highlighted that the measured output is at zero load in this case. For the settings of the amplifier, this value measures 8.89 Volts. In our specific case, this value is acceptable because we are mainly interested in compression forces. The only tension forces will be given by the tool weight that is designed to be less than one kilogram.

We set the cell's sensitivity (calibration factor) and balance value. With the chosen setting, we have almost all the voltage range +/-10 Volts for compression and consequently a higher sensitivity. The calibration was done on two points; therefore considering the behavior of the linear cell in our case.

Notice that the conditioner should be differently set if we had heavier tools.
If we try to press the load cell, we will see at the multimeter that the voltage measured will be

lower and is proportional to the pressure we impose.

PLC/Load cell communication

To check this proportionality and set the load cell ready to record a force value, we use the analog input module EL3008 already described, connected to a PLC Beckhoff. We create a new TwinCAT project called RecordLoadCell, and we map to the first channel of the I/O module an INT variable aInputN1.

The load cell will have an approximately linear behaviour due to an increasing pressure.

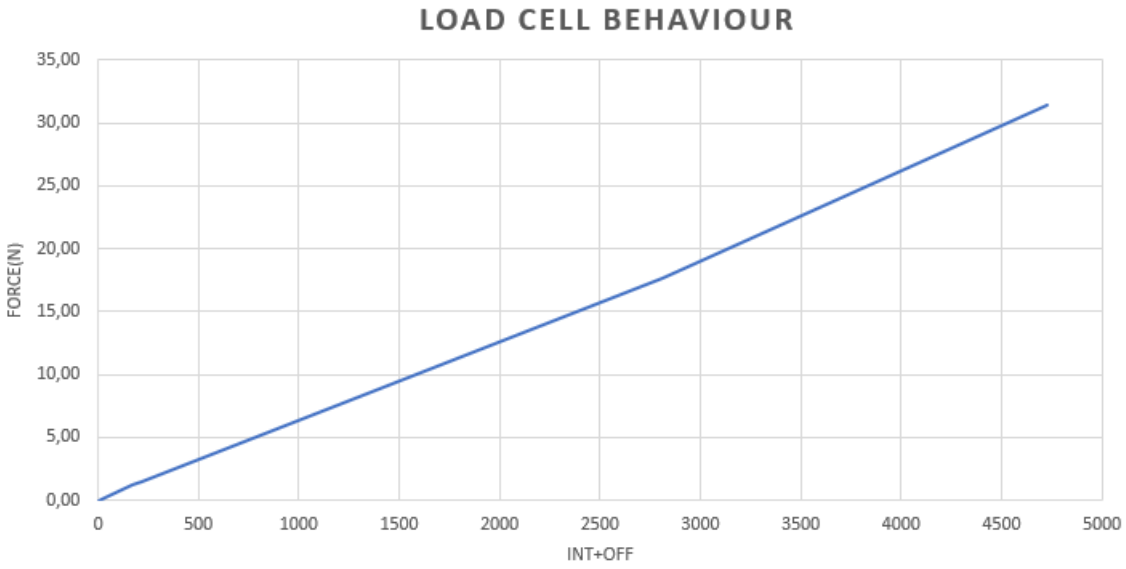


Figure 4.12: Load cell characteristic

We were then able to collect these data and find an angular coefficient associated with this linear behaviour. As the output at zero load is set to be 8.89V, we need a zero Offset value. We will then find the value converted in force and weight with the following passages as in figure.

```
PROGRAM MAIN
VAR
outValue     : INT; //value read by EL3008 from Load cell
zeroOff      : INT    := 14625; //Changes depending on tool
angCoeff    : REAL   := 0.00715; //average angular coefficient
LdCellForceZ : REAL; // out value converted in force(N)
g            : LREAL  := 9.81;
outWeight    : REAL; //out value converted in Kg
```

```
outValue := aInputN1-zeroOff;

LdCellForceZ := ABS((INT_TO_REAL(outValue))*angCoeff); //abs as only interested in compressions

outWeight := LdCellForceZ/g;
```

Figure 4.13: Load cell conversion

The force value in Newton given by *LdCellForceZ* will be used for the experiments.

On the other hand, *outWeight* will be used for a check of the correct calibration.

4.3 Robot to PLC software interface via EtherCAT

There exist different possibilities to interface the LBR to a PLC. We opted for the most complex but complete way, which relies on EtherCAT Bridge technology. With an EL6692 module, it was then possible to make two masters to communicate: The robot and the PLC. The first master, in our case the PLC, is connected to the Input X1 port with an Ethernet cable. In contrast, the second master is connected to the Output X2 with another Ethernet cable. For correct functioning, it is crucial to turn on the second master (the robot) only after the first one is running. If all the three status LEDs are active, the connection works correctly. This powerful module will exchange data in real-time at a very high frequency, guaranteeing synchronism.

Workvisual configuration

To use the EtherCAT bridge, we first need to change the I/O configuration seen by the robot. We open Workvisual, and we modify the configuration as follows. Once the hardware is selected, we will see that there are two sides in the IO mapping window. On one side, we select the PlcIO group that was previously created in the sunrise application. While on the other side, we have to choose the EL6692 terminal.

We will have now to create the desired values we want to exchange with the PLC through the EtherCAT Fieldbus. We also need to allocate the values in the Signal editor window that will appear by pressing the pencil icon. We will then connect the variables on the two sides. Once the signals are linked, they will become green, as in figure.

It is important to point out that not all data types can be chosen. We can directly pass Boolean, Integers, or Bytes. But it is not possible to exchange real or double values! To do that, a conversion was necessary.

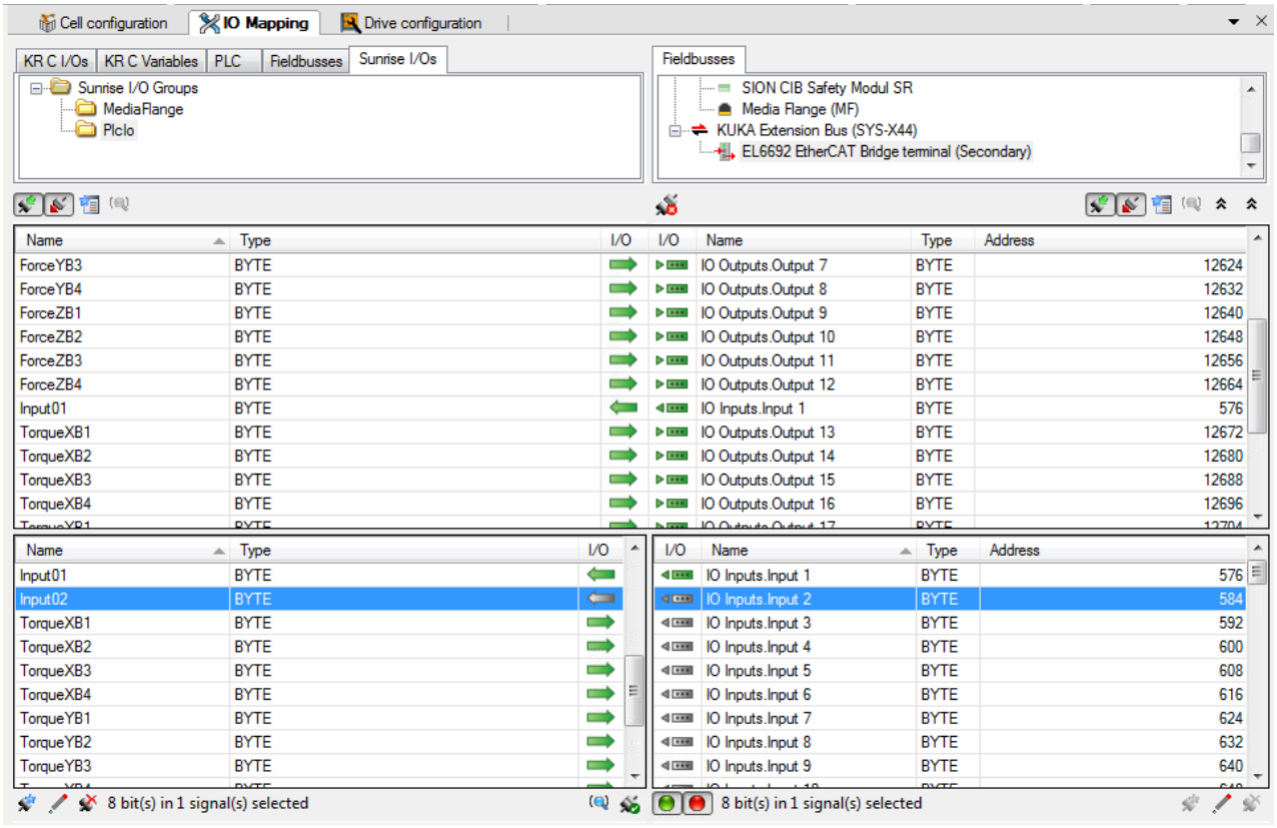


Figure 4.14: Load cell calibration with LBR

Double data types conversion

As the force values measured by the LBR are double values, it was not possible to exchange these values directly with the PLC.
We recall that a double data type is composed of 8 bytes. However, if, like in our case, there is much double data we want to exchange with the PLC, it is better first to convert the double values in float variables. We did this by simply casting the variables in the background application in our Sunrise project. It was then necessary to convert these float variables into 4 bytes each. In this way, we were able exchange all the data required with an acceptable precision.

Subsequently, it was also necessary to reconstruct the values at the PLC side with another conversion aimed at turning back into float values, called Reals in TwinCAT3. To do that, we used a UNION (a User-specified data type, DUT) called *U_convert* and we were able to reconstruct the values.

If using EtherCAT communication like this one gives the possibility to exchange much data in real-time, some problems during the conversion may arise as it happened.

Unfortunately, due to the slow execution of the background task developed in Java, we could

not run the PLC task and the Java at 1 ms.

A tradeoff between the update speed of values and correctness of the actual value was required. Furthermore, we noticed that the more variables the application needed to run, the more time was required to update the values.

For this reason, we decided to put in the PlcIO group only the essential robot values we wanted to pass to the PLC.

Once reconstructed, we could use the desired values from the robot size to do our experiments. We also assured the correctness of the conversion without the presence of spikes by doing the conversion operations only atomically.

4.4 TwinCAT3 projects

Within the twinCAT3 environment, we developed two different projects: one aimed at the correct functioning of the load cell, the other for comparing the force values of load cell and robot.

Thermal drift project

Before using the load cell, we further assured its correct functioning by verifying the absence of thermal drift. We then created a project called LoadCellDrift. Using the writing data function block (FB), we then run the program for a few hours and saved the sample values of the Int value and of the force measured by the sensor on a .csv file.

TwinCAT3 recording project

To make the comparison between the force read by the robot and the force value of the load cell, we created a project with TwinCAT3. We structured the project in a way that guaranteed us the possibility of:

- measuring real-time both the values of force measured by the load cell and by the LBR;
- showing that values and starting a recording for a quick view of the results in a Scope project;
- saving in a *.csv* file data for a certain recording time for offline analysis.

For the simultaneous measurement of robot force (particularly on the z-axis with the end effector of the robot perpendicular to the floor) and load cell force, we used the EtherCAT

bridge to exchange data in real-time with a frequency of 10 ms. The Force difference was also calculated for each instant, as in figure 4.15.

For looking at these values, but also at the position of the X and Y of the robot (for these measurements, we kept Z constant as we mainly worked on a plane surface, but it can be added)

To collect data offline for further analysis, we used a function block called writing data. We were able to save data relative to the dynamic experiments and study the thermal drift of the load cell. This function block makes use of the *PLC_TO_CSV_Array* STRUCT (a type of DUT) we defined to save the values in an array of desired dimension *arrDim* defined in the IO Global variables list.

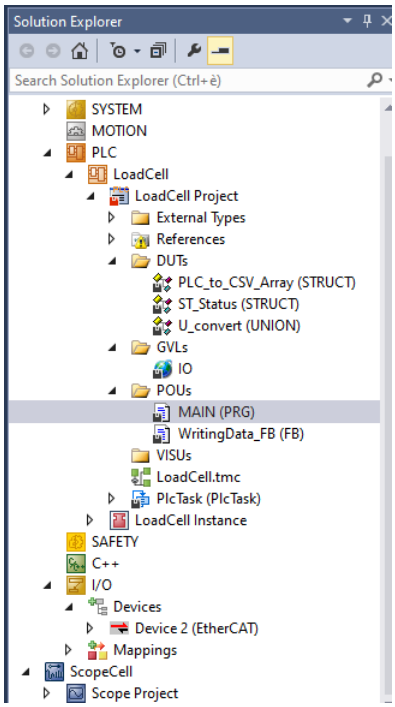


Figure 4.15: Record load cell project structure

4.5 Sunrise applications for measurements

Thanks to Sunrise Workbench OS, three different applications were developed within the same project:

- Force Static: RoboticsAPI Application for measuring the static forces of the robot
- Point sliding: RoboticsAPI Application for measuring the dynamic forces of the robot
- ForceLog: Background application for cyclical logging the desired forces and exchange via EtherCAT the desired variables with the PLC.

The source code of the applications can be seen in detail in Appendix A. The imported libraries at the beginning of the applications are omitted for saving space but can be easily imported on Sunrise OS.
When a new project is created, preliminary steps of considering the right IP address for the smartPAD connection and attaching the right flange if present (IO Pneumatic in our case) are made. Secondly, before developing the applications, a safety configuration has to be set like already presented. After that, depending on the application we want to perform, we have to set the I/O groups that have to be previously exported with WorkVisual. In the package explorer, we also have to insert in the ProcessData.xml file all the data we want to show in the smartPAD's process data window. After that, the desired applications can be made depending on the desired task to perform.
Notice that only one RoboticsAPI application per time is active, while one or more background tasks can be executed in parallel with the desired application selected.

Frames selection

For making the robot able to follow one trajectory, at least the start, and the endpoint need to be defined by a frame.
A home position has been defined as a frame for all the applications. That position will always be the first and the last command of an application. Before an application is finished, the robot will always go in the home position, waiting for the next task.

Frames can be saved with the smartPAD or can be defined in the project. In the first case, it is possible to add a new frame in the robot's current position or overwrite on the same frame the current position with the touch-up command.
In the second case within Sunrise Workbench, on the Application Data window on the right will be possible to create new frames. When a new frame is created, the RoboticsAPI.data.xml file will be automatically updated and also that frame will be present.
Frames present different properties listed in the bottom window of the figure. They are also related to the tool attached.

Load data determination

Load data determination is another command to perform in the presence of a tool. It is essential if forces are treated, especially when we need to measure the accuracy of small forces. When this command is launched, the robot will automatically perform some movements to measure mass, centers of mass, and inertial properties of the attached tool.When a tool is selected with the Injection in an application, the frames and the load properties will be related to it.

After making these steps, it will be possible to develop the applications and transfer them to the controller with Project Synchronization.

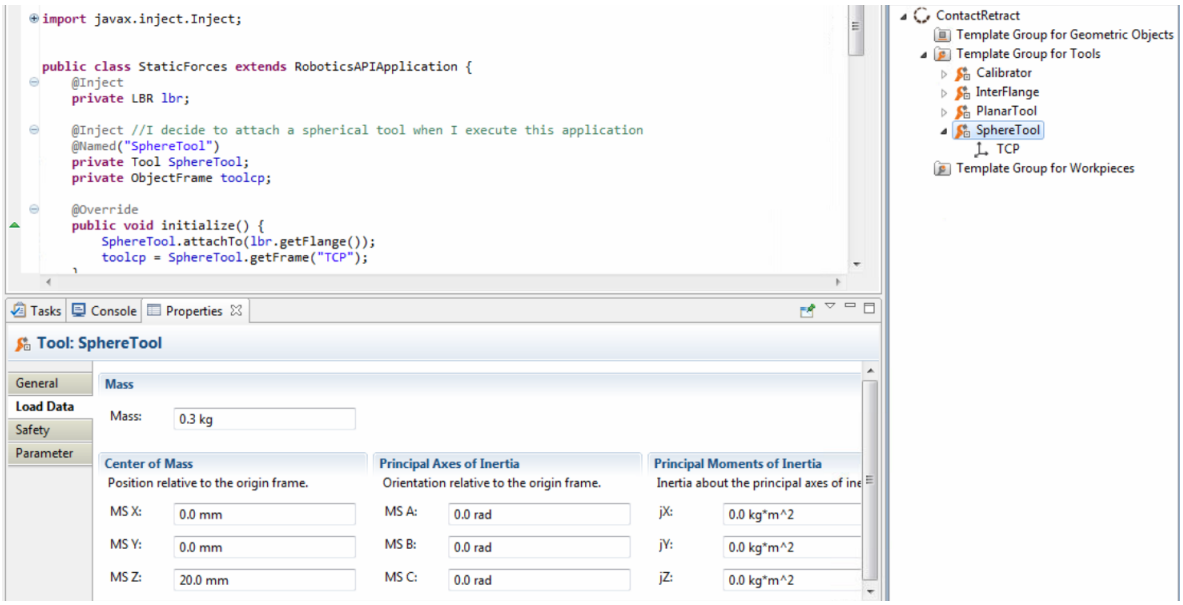


Figure 4.16: Selection of frames

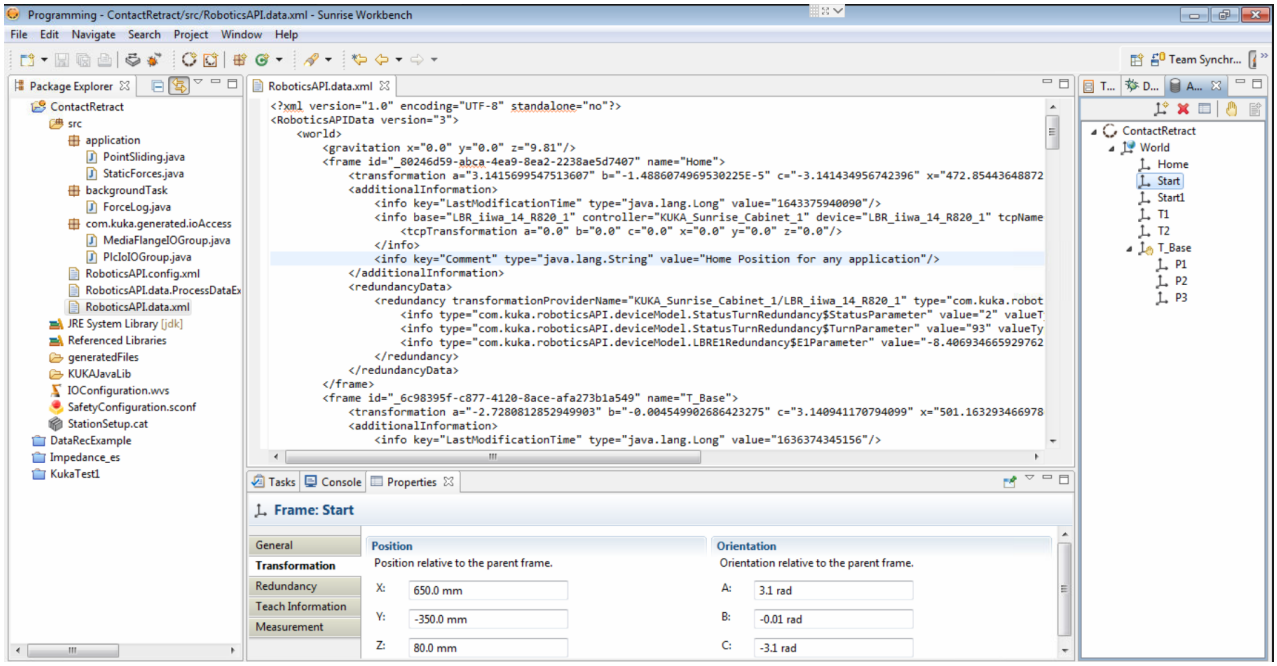


Figure 4.17: Load data determination

Chapter 5

Tests and Experiments

The performed tests and experiments were developed in the following order:

1. The mono-axial load cell was tested considering the eventual presence of thermal drift and the measurement was validated;
2. Static forces measurement was performed with the robot left free in the workspace to understand if the robot measurement was accurate or not and show the best working region of the workspace;
3. Dynamic forces measurement was performed in the identified best working region. The robot was moved over a planar surface with a certain contact force. The different tools designed were used to make also comparisons;
4. Impedance control was tested with different values of the setting parameters to understand how they affect this kind of control;
5. An integration with ROS was made with the aim of trying to control the robot considering the force measurement of the validated load cell instead of the Cartesian z force evaluated by the robot.

To perform these experiments, different custom tools were screwed to the load cell depending on the application to complete. Notice that while the load cell measures the forces in only one direction (defined to be the z-axis), the LBR is able to measure all the 3 Cartesian forces along x, y and z (and also the 3 torques).

5.1 Load cell validation

In order to validate the measurements of the load cell and proceed with the aforementioned experiments, we chose an experimental approach. By using a digital balance previously calibrated, we proved that up to 8Kg, for increasing pressure, the value in Kg of the load cell match with the value of the balance.

We mounted the sphere tool (in this case, the one made with the 3D printer is suitable) on our LBR robot. As a tool was mounted, the offset for the zero value was previously modified in the twinCAT project and the tool was considered attached to the flange at the robot side after load data determination was performed.
Then, we slowly moved the robot in T1 mode to make it able to press the balance with increasing pressures at the interval of 100 grams.

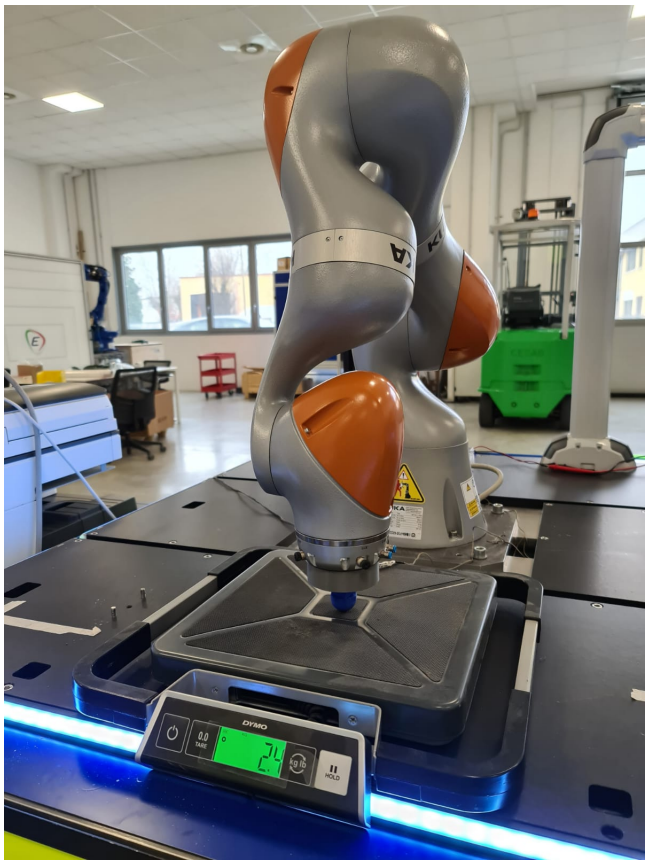


Figure 5.1: Load cell validation with digital balance

In this way, it was possible to compare different values of *outWeight* with the corresponding values measured by the balance. It was proved that the values match each other for a considerable range. However, if we would like to press more than 8 Kilograms, more than an angular

coefficient should be used to have an efficient measurement. In fact, the characteristics cannot be considered linear anymore if we want to preserve a high accuracy.

5.2 Load cell thermal drift test

Before using the load cell for the experiments, we tested its correct functioning in terms of thermal drift. As assured by the manufacturer in the datasheet, we proved that a thermal drift is not present if we have an ambient temperature slightly variable as it was for the working environment.
We run the project LoadCellDrift for 15 hours and collected 450 samples (one sample every 2 minutes). The average force measurement recorded has an average value of -0,027N. This value is very small, and we concluded that the behaviour during time does not change enough to consider the presence of thermal drift.

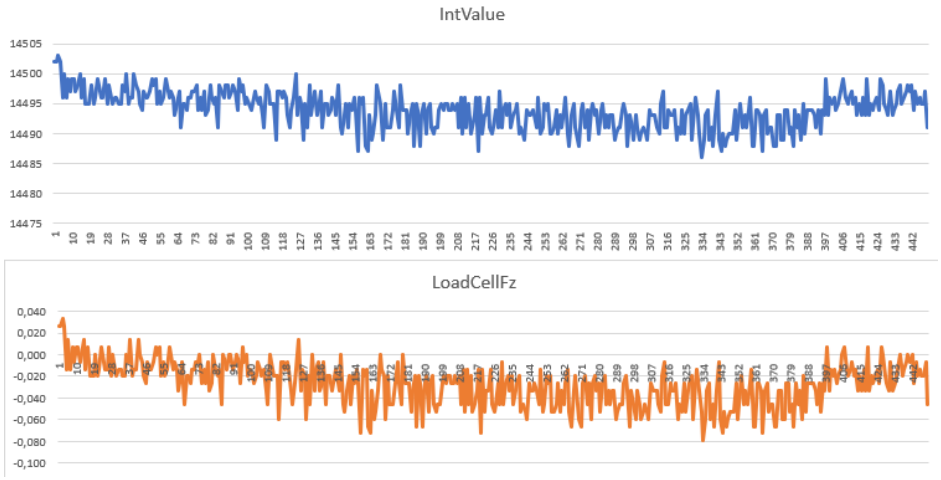


Figure 5.2: Thermal drift project

Now that the load cell has been validated, we can then proceed with the experiments.

5.3 Static Forces acquisition with spherical tool

The first experiment was developed with the aim of measuring the Cartesian Z static force seen by the robot in different positions. In this case we attached the spherical tool designed in aluminium to consider a more realistic case in which a flange is not left free without anything mounted. Since the mass of the tool is not negligible, it was considered in both measures:

- modifying the zero offset at the PLC side for the load cell measurement

- attaching the tool after having performed the load data determination on the robot side

Even if the robot is not in contact with any object in the workspace, the force that measures will be slightly different depending on the position, introducing an error with respect to the force direct measure of the load cell that in the absence of contact will always be zero. This is due to the fact that the measured force of the LBR is the result of a mathematical combination of measurements of all the robot torque sensors. The robot has a torque sensor measuring one axis of effort at each of the seven joints. Consequently, the application we developed called *StaticForces* was designed to measure this error affecting the static Z-force seen by the LBR in a discrete number of positions, with five different fixed heights.
Figure 5.3 describes the positions that the robot covers by running *StaticForces* application. In the beginning, the robot starts moving from the green position towards the first blue square. Once it is reached, the robot holds his position for a certain time and the value of z force is measured. Then it moves towards the next position, until all the zig-zag pattern indicated by the arrows is followed. At the end of the application, 5 rectangular blue grids are filled with the z force measurements. The remaining force measurements for the positions indicated by the red squares are recorded manually. The robot has been moved manually to reach these positions in T1 mode. Especially with the first height of 600mm, the robot will be close to singular configurations, and a tiny further change in position would mean reaching a singularity.

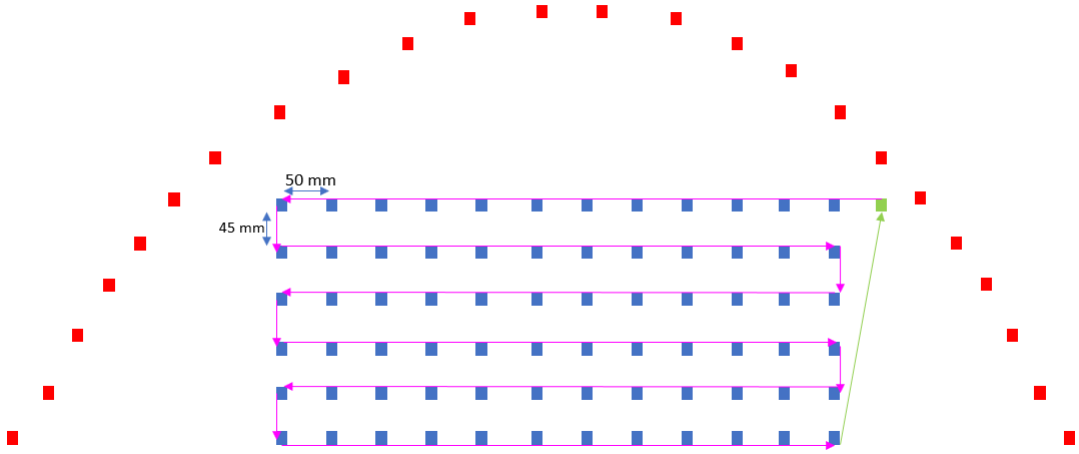


Figure 5.3: Recording Pattern

Same points with different heights

In this case, the pattern described is kept fixed at different heights with the same grid of (x,y) coordinates, also for red positions. In this way, it is clearly visible how much the height may affect the force value. By decreasing the height, we will not have huge errors like in 5.4 and 5.5.

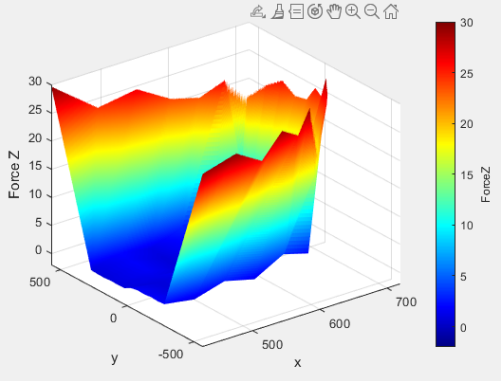


Figure 5.4: height 600mm

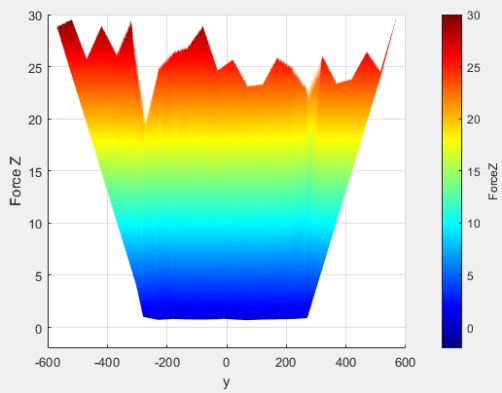


Figure 5.5: height 600mm, frontal view

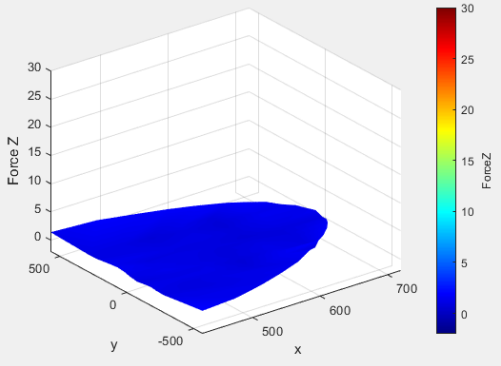


Figure 5.6: height 465mm

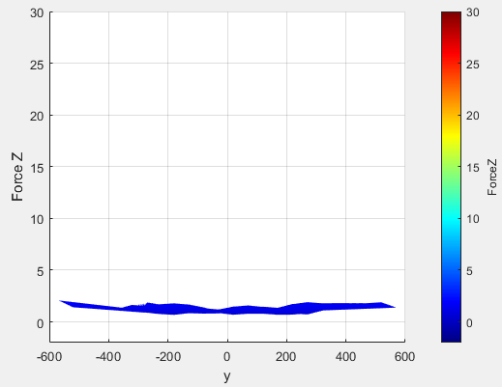


Figure 5.7: height 465mm, frontal view

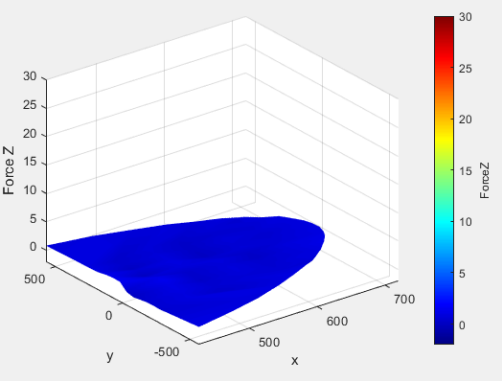


Figure 5.8: height 330mm

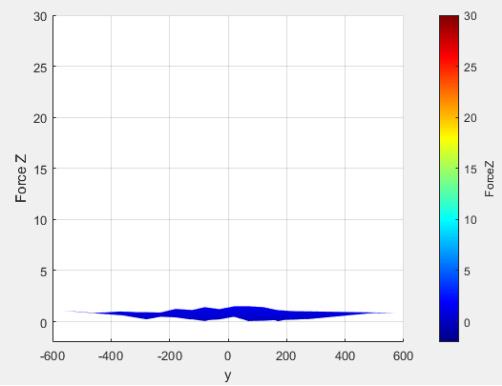


Figure 5.9: height 330mm, frontal view

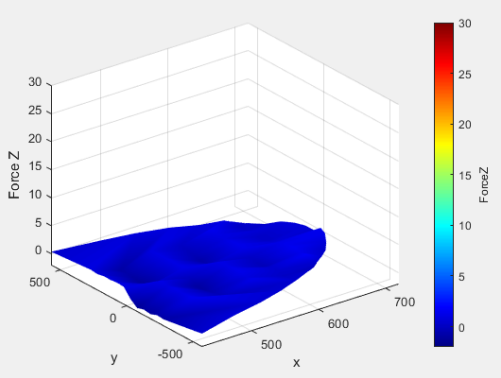


Figure 5.10: height 195mm

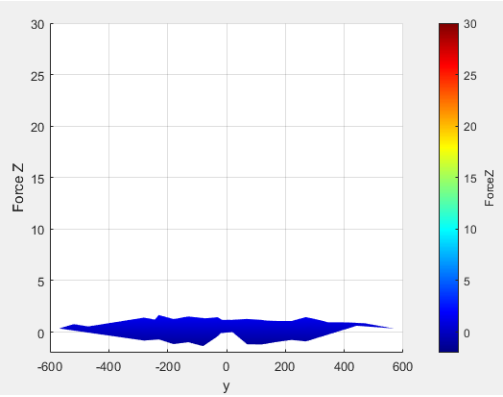


Figure 5.11: height 195mm, frontal view

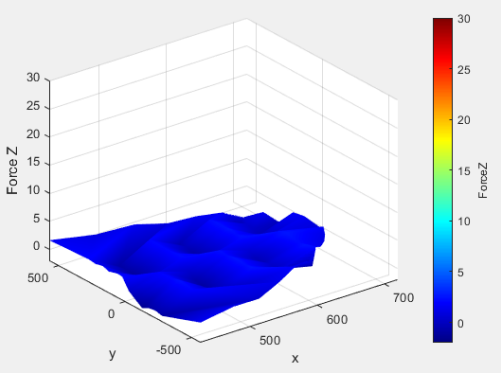


Figure 5.12: height 60mm

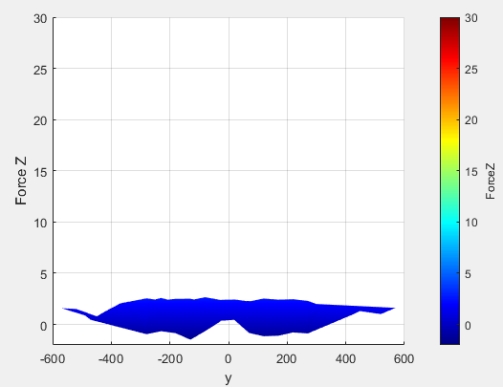


Figure 5.13: height 60mm, frontal view

As the results show, if we want to keep the error of the force as close as possible at zero, a suitable height z will be between 330mm and 465mm. As the height further decreases, the accuracy of the measured z force will degrade, and the error will assume also negative values when we reach 195mm or less.

Maximum reachable points close to singularities

Another static experiment was performed to understand the best workspace region better. To further verify how the robot is afflicted by a not always precise estimate of the forces, we paused near to the singularities and at the end of the workspace. As an example, we compared the heights of 600 to 465 mm until reaching areas close to the singularity. In these positions, the arm will be extended almost completely, thus not accurately estimating the forces. In our case we have considered in the graphs only the measure of the force on z, since the available load cell used in the following experiments is mono-axial.

In this case, we did not keep fixed the (x,y) coordinates of the positions indicated by the red squares (5.3). Instead, we decided to move the robot close to singular configurations in both cases. We also preferred to keep the number of samples recorded in the 5 cases constant. For this reason, for height 465mm there are present some cuts. The y coordinate of the samples remained unvaried, and we tried to reach the maximum x close to singular configurations in both cases.

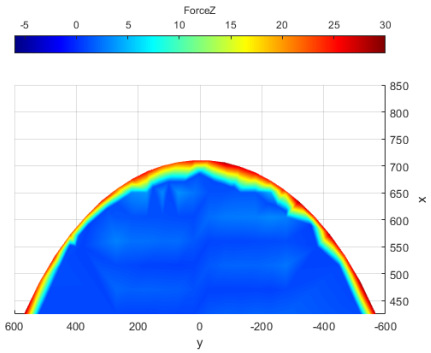


Figure 5.14: z height 600mm

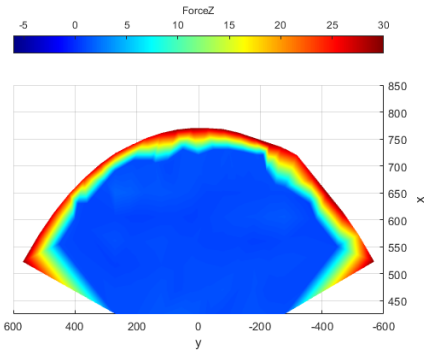


Figure 5.15: z height 465mm, frontal view

As we can expect, the reachable x and y in Figure 5.15 will be higher, due to a workspace that can be considered spherical. These regions in fact, can be seen as different slices of an almost spherical workspace. If these slices, after having collected the samples for all possible x,y positions, would be overlapped for all the heights, the working envelope in figure 5.19 will be obtained.
Thanks to these static experiments, it was then possible to understand the best working region in which the z-Cartesian force has the closest value to zero. Notice that for a matter of time, assuming that accuracy proceeds linearly away from singularities, we did not collect a large number of samples. We focused instead on the workspace that will be predominantly used in robotics applications (the grid region in 5.3). For this experiment, they were enough to understand the workspace region commonly used for classical applications performed with this robot.

5.4 Dynamic Forces acquisition with spherical tool

Once the most suitable working area was discovered, we then performed some dynamic tasks, ensuring that the measured z force was the one with minimal error.

We decided to execute the same task with all the tools to make possible a conclusive comparison. It was then decided to perform a linear segment, with impedance control mode kept

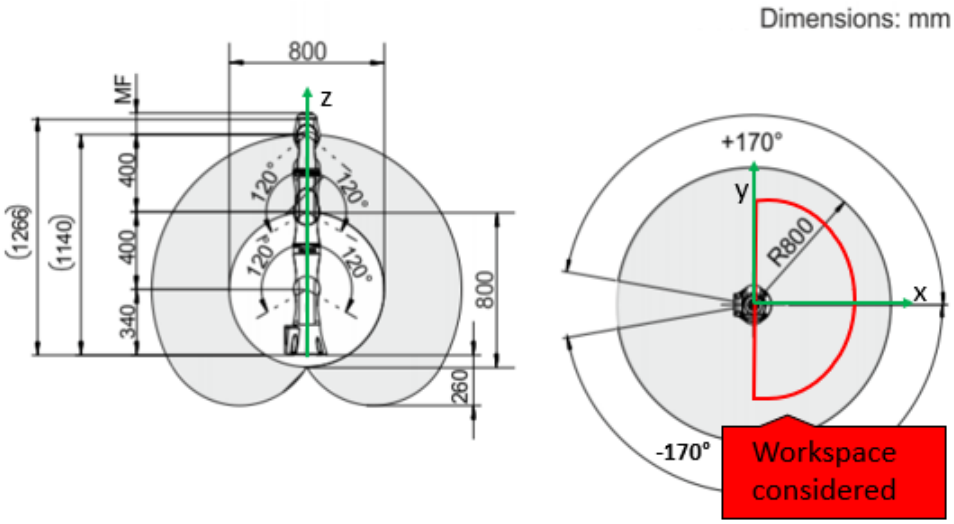


Figure 5.16: LBR working envelope

active for the entire trajectory. We also decided to stop and hold the robot in impedance mode in three different positions: the start, the middle, and the endpoint. We introduced a constant contact force of 14 Newton to keep contact with the surface, widely inside the validated range. After some trials, we found this value reasonable with all three tools. We also had to set a velocity limit to keeping the vibrations limited. The trajectory we decided to follow is schematized in the following figure 5.20. Notice that the recordings of the load cell force measurement and the

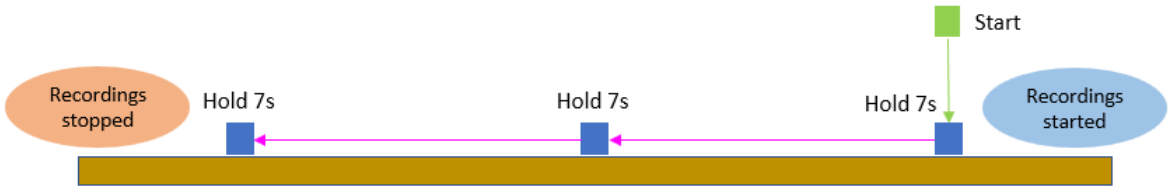


Figure 5.17: trajectory of dynamic forces experiment

LBR force z start as soon as the robot touches the first position, where it will hold for 7 seconds. The recordings will stop once the robot lasts to hold in the end position.

We then kept the same spherical tool previously mounted and launched the *PointSliding* application developed. Using a spherical tool, we will have a single point contact with sliding friction.

Figure 5.18: Planar surface with spherical tool mounted

As we can see in figure 5.22, the difference of the two values becomes larger during the sliding, while it is reduced when we stop in a certain position. During the three stops, the value measured by the LBR is slightly different as it depends on the position, as mentioned in the static analysis. This will be reflected to the load cell measurements as in figure 5.22. We can also notice that, differently from what the LBR measurement says, the measure is more variable during the sliding. The maximum error will be around 5/6N, which is acceptable for applications like this one.

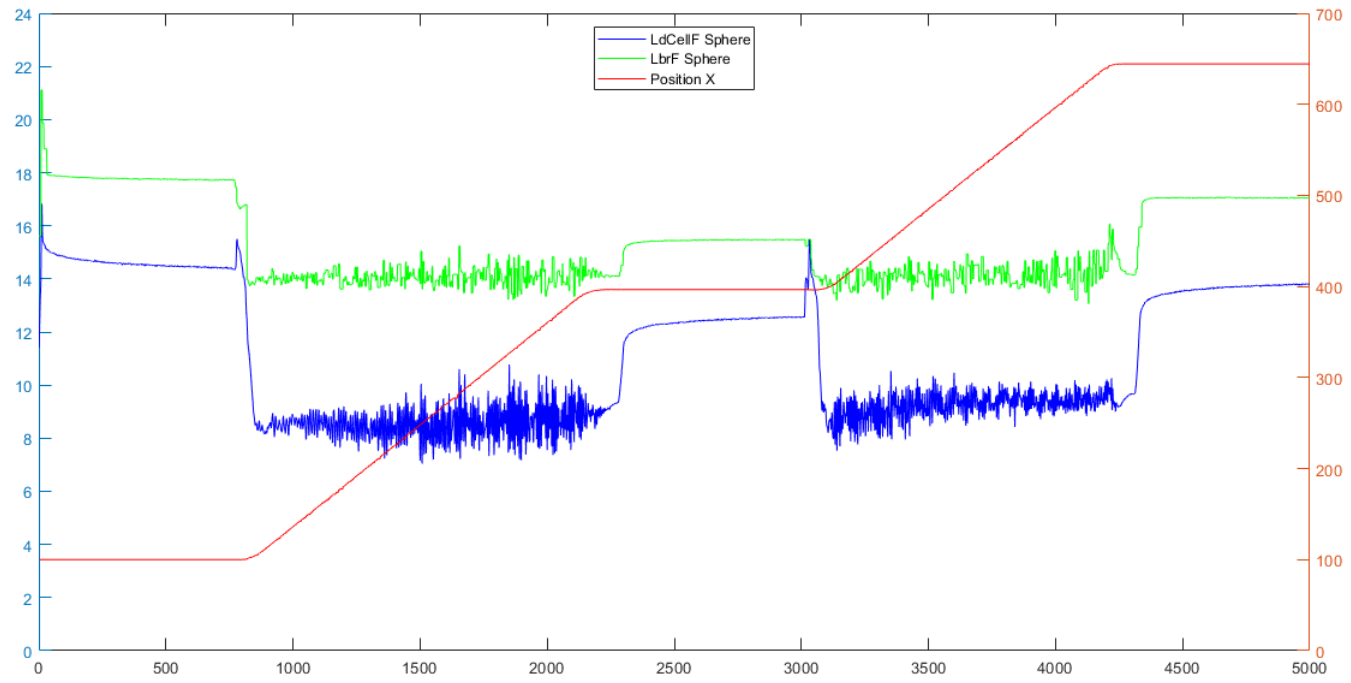


Figure 5.19: Sphere Point sliding with 14N pressure

5.5 Dynamic Forces acquisition with ball castor wheel

To try to have less friction, we tried to always rely on single point contact, but with a ball castor wheel. In this case, we tried to rely on rolling friction, much lower than the sliding friction previously present.

With a different type of friction, we could expect that vibrations of the load cell values during dynamic phases might be lower than before.

Unfortunately, the results show that is not true. In fact, if we keep the tool in contact with the surface with a certain contact force, there will be a normal force high enough to not make able the ball to roll on the surface perfectly. On the other hand, if we tried to reduce this pressing force, it would result not be enough to keep the tool in contact with the surface for the whole time.

We can then conclude that using a ball castor wheel with a specific contact force, required higher than a threshold, will change only slightly the measurement of z-force for this application.

Similarly to the sphere tool case, the results were acceptable with always an error of around 4 N when the robot holds in a position and around 5/6N during the sliding on the surface.

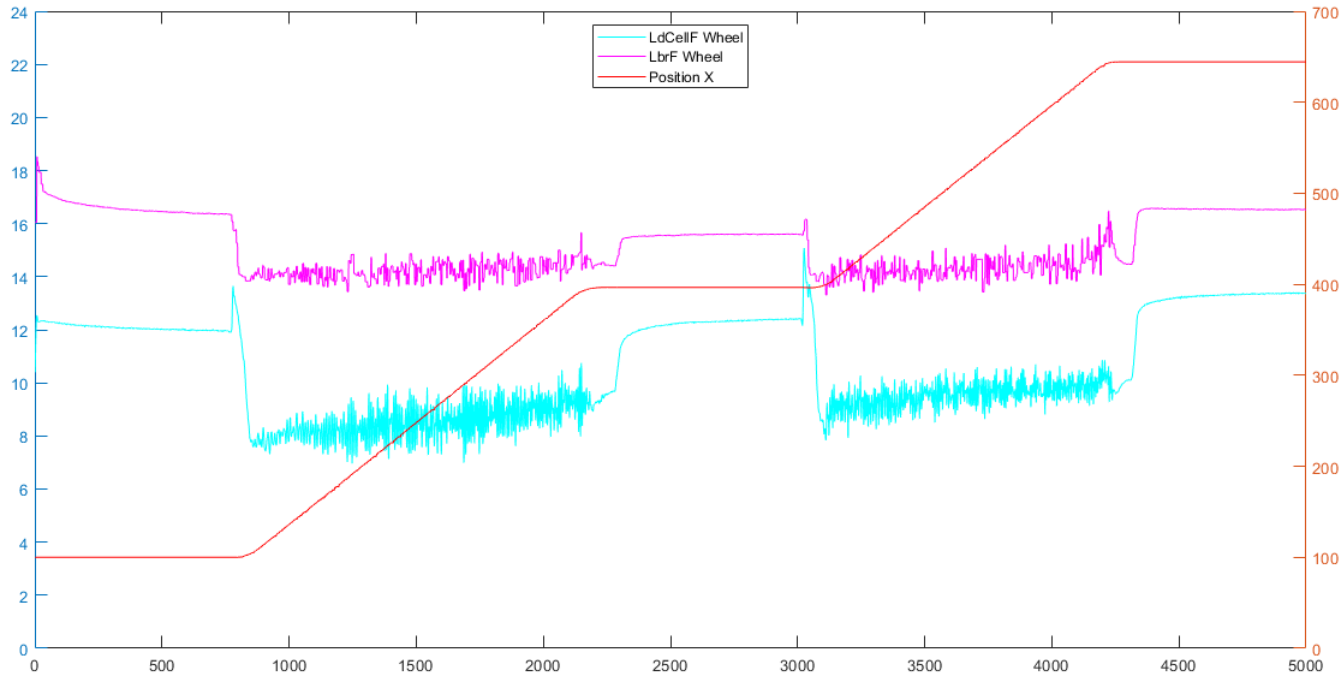


Figure 5.20: Ball castor wheel point sliding with 14N pressure

5.6 Dynamic Forces acquisition with planar tool

The third experiment was performed with the planar tool mounted on the flange. Since this tool is heavier than the two other tools (that have the same mass), also in this case, we modified the zero offset on the TwinCAT project, and we considered the different tools mounted in the Java applications after the load data determination. Then, we recorded the forces for the path followed in the two previous dynamic experiments.
As we can see, the difference between the two force values is higher than the previous cases and leads to a more considerable error. The cause of this result has to be found in friction.

In this case, the friction present between the tool and the surface is no longer point-like. But instead, we have an entire contact area. Therefore, the sliding friction present in greater quantities will cause the generation of forces in all the contact positions. These forces can be considered components of an equivalent friction force, which, when multiplied by a given arm, will generate a moment acting on the cell and on the robot.

Furthermore, as visible after the central stop, the difference increases between the two measurements increases. In fact, the impedance control does not guarantee an utterly rigid behavior in the desired directions and will therefore determine a slightly variable angle of the tool. This will generate a reaction force with two components as no longer horizontal. This will cause further undesirable components that will affect the load cell measurement.

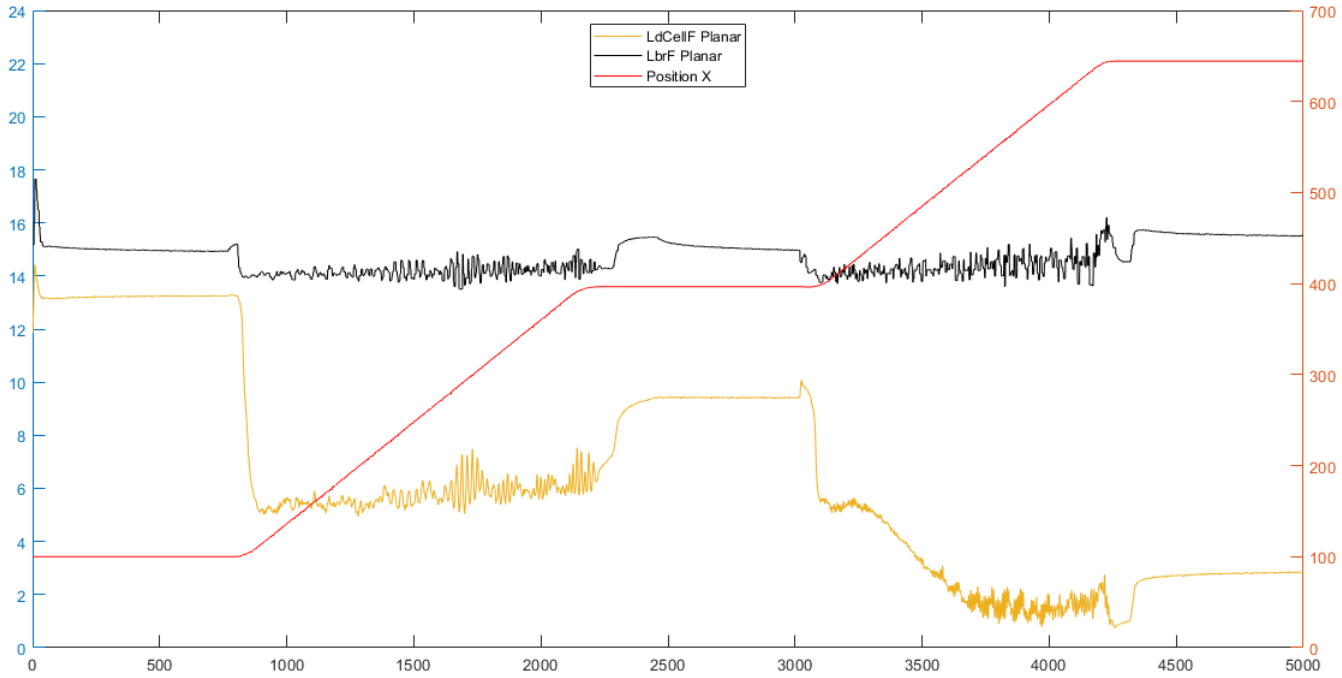


Figure 5.21: Planar surface sliding with 14N pressure

We limited this phenomenon by maximizing the stiffness in the directions where no contact force was expected, but it was impossible to eliminate it.

5.7 Comparison of the three tools

We can now compare and overlap the three dynamic experiments performed with a contact pressure of 14N. What we see in figure 5.29 is that, thanks to impedance control, the z-force measured by the LBR in all the 3 cases has almost equal shape. On the other hand, this is not the real force perceived externally by the robot. Instead, that actual force will be given by the load cell output, with the spherical tool mounted first and then with the ball castor wheel tool. With the planar tool at a certain point the value will decrease, determining a different measurement already justified.

5.8 Effect of Impedance control parameters variation

Contact force variation

We also did some further experiments, with always the spherical tool attached, to illustrate the effect of different settings of the parameters with impedance control law implemented. Firstly we compared the use of different contact force values. We then plotted the error (the force difference between the measured z-force of robot and load cell) with a contact force of 10, 12 and 14 Newton.

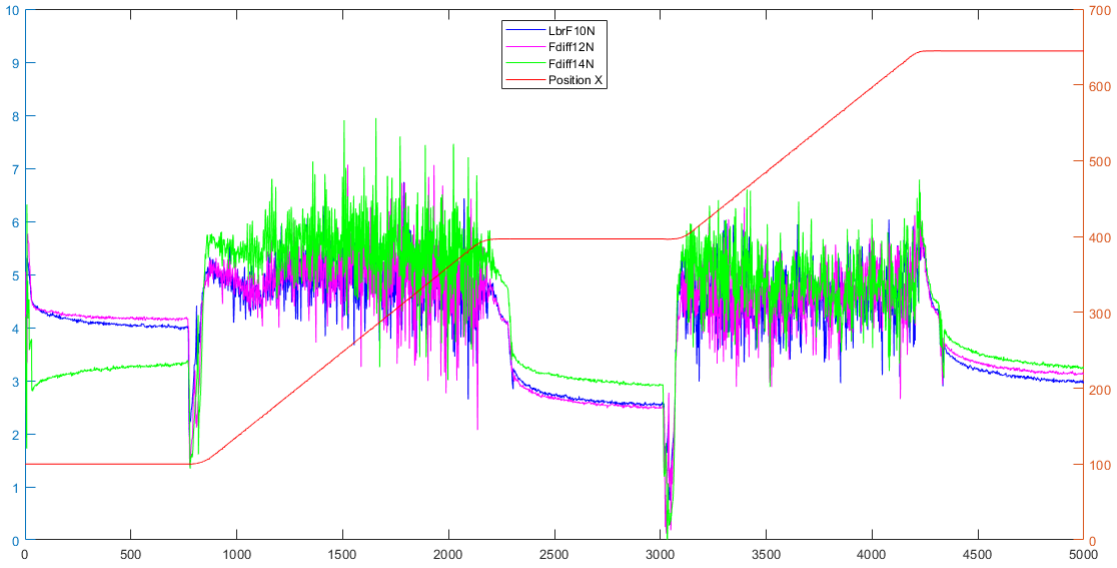


Figure 5.22: Force error with different contact forces

We can conclude that the error is almost invariant for the contact force used. Notice that if this force is too low, the impedance will not perform well as the robot will lose contact with the surface. Even if the error with a 14N pressure is slightly higher in this case, it was considered the tradeoff contact force suitable for performing the experiments with the planar tool and the ball castor wheel.

Damping variation

The damping parameter is used in impedance control to act on the transient phases. A more significant value of damping will decrease oscillations. We then tried to use two different damping values for the same application.

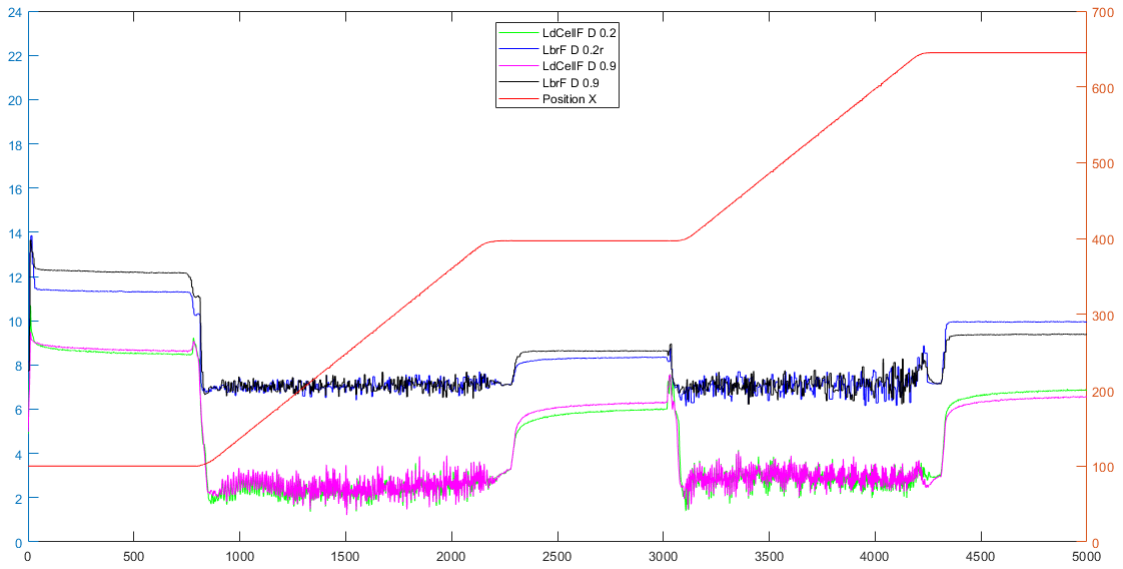


Figure 5.23: Damping variation comparison

We can conclude that a change in damping is not really relevant for our application in particular. Anyways, a high value will be used for the experiments to assure that especially the load cell will not have high peaks at the beginning of the contact with a surface.

Stiffness variation over inclined surface

More important to study is the change of the stiffness values. To better understand how a change in stiffness modifies the impedance control, we tried to follow the same previous trajectory but we inclined the surface. We then made two different experiments.

Firstly, all the stiffness values were equal to the previous case with a flat surface. In particular, the K value along the Z direction where contact is expected was kept equal to 50. Secondly, the stiffness was reduced at $K = 3$.

In both cases, the robot will try to follow the desired red trajectory (figure 5.24). However, due to the soft stiffness behaviour along the Z axis, it will follow a different real green trajectory. In the first case with $Kz = 50$ the value will be very stiff, and then, the pressure over the surface inclined will increase. In fact, the robot will always try to follow the desired trajectory and while will try to reach that, it will exert a more significant force F on the surface.

Only by using a lower Kz like in the second case, impedance control was performed with an almost constant force over the inclined surface.

In this way we were able to show how a good stiffness value, along directions where contact is expected, has to be chosen depending on the surface we want to follow. The more irregular is the surface, the softer will be the Kz stiffness we will set for having a constant contact force F.

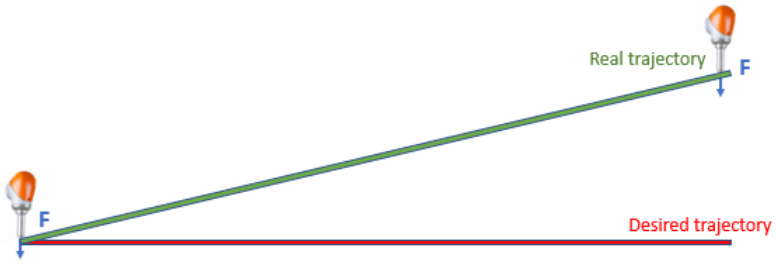


Figure 5.24: Stiffness variation in theory

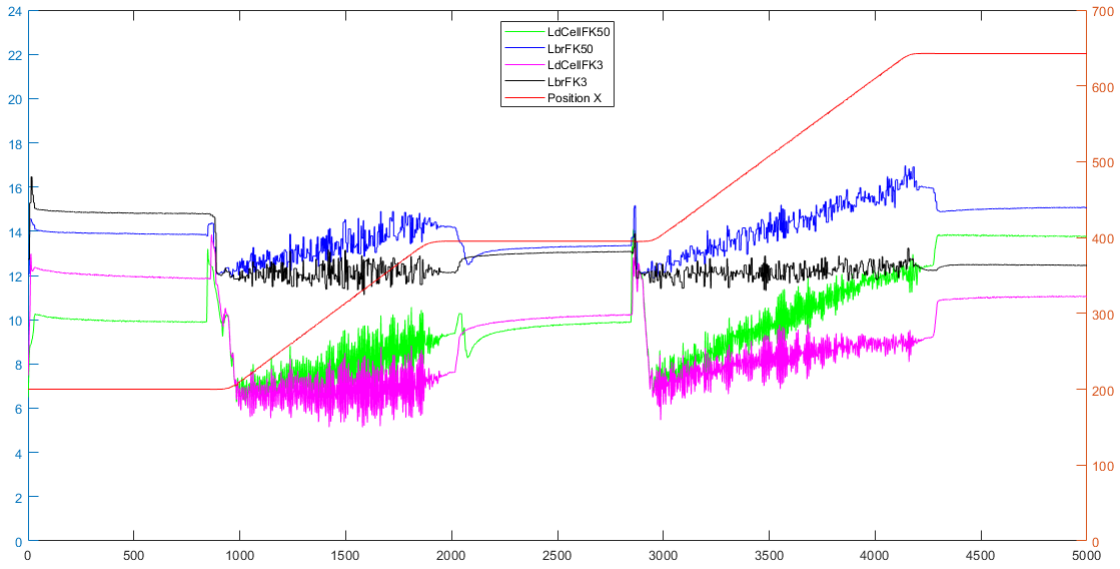


Figure 5.25: Stiffness variation example

5.9 LBR - ROS interface

The last experiment performed was aimed at integrating ROS with the robot. After validating the load cell and carrying out the experiments above, we tried to control the robot via ROS to introduce new features and give the possibility, in the future, of using the load cell measure instead of the inner torque sensors of the LBR. Unfortunately, we did not have time to implement this control with the load cell, but we could use the impedance control with parameters passed by terminal with a node running on ROS. For performing this experiment, an Application Programming Interface (API) developed by Sheffield Robotics (23) to work with ROS was used.

API architecture

The API architecture focuses on breaking out the functionality that would normally be available within the KUKA Sunrise controller run on the SmartPAD. It aims to extend the capability of the KUKA with a structure as in Figure 5.26.

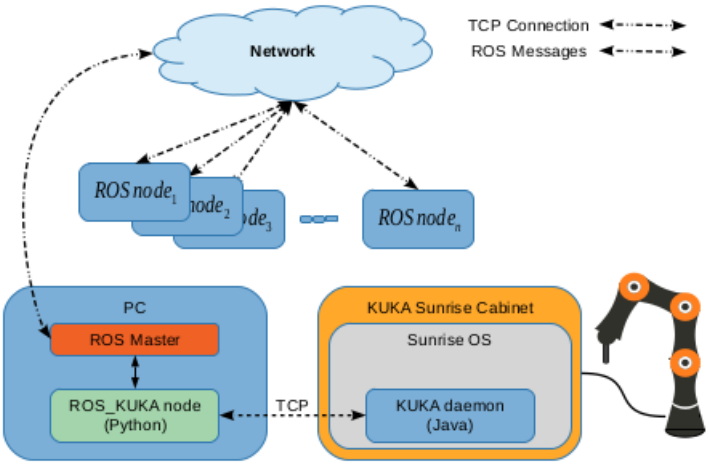


Figure 5.26: API Architecture

The KUKA daemon is a program in Java, included in the Sunrise operating system of the KUKA Sunrise Cabinet robot controller, that handles some generic and controlling tasks. This program is necessary to move the KUKA safely while using the library developed by Sheffield Robotics. It is provided by a GitHub package called **iiwa_stack** developed by IFL-CAMP.(24)
A ROS KUKA node is also implemented in Python language: it is a ROS node that plays as an intermediate between the KUKA daemon and the ROS master. It subscribes to the topics containing commands published by other ROS nodes and passes them to the KUKA daemon. In the same way, it receives status information of the KUKA from the Java application and publishes it on specific topics. By running the ROS node on an external pc, rather than KUKA Sunrise OS, it is possible to preserve robot's safety protocols, because no modification of the KUKA Sunrise Cabinet is required. Essentially more features are enhanced and added.

Designed to be very simple, interfaces to ROS to provide an easy platform for development. We used this API to make the robot able to hold freely in the workspace and test its behaviour with different values of damping and stiffness, that we passed to the robot via terminal through a program we developed to call some services of the iiwa_stack package.

The connection between the PC with ROS and the robot cabinet is established via Ethernet. We use one machine with ROS Kinetic installed, and connect an Ethernet cable to the X66 Port of KUKA cabinet. With such communication, we will be able to send and receive ROS messages to and from the aforementioned Robotic Application.

Architecture used

As users of this API, we manipulated the messages received from the robot and sent new ones as commands, developed within C++ code, taking leverage of all the ROS functionalities. To do

that we created a node called Impedance_test node to be able to call and use a service handled by the ROS_KUKA_node.

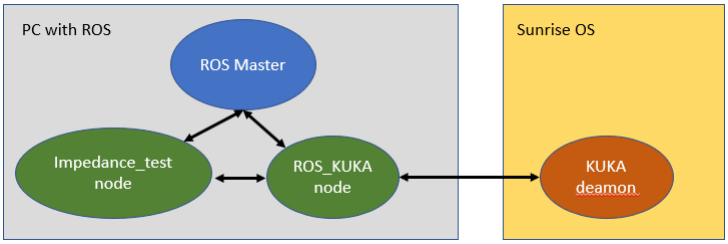


Figure 5.27: Architecture used

Before using ROS, notice that we inserted a new safety row to not exceed a certain limit speed value. After that, we tried to use the Robot in Impedance control by changing stiffness and damping values.

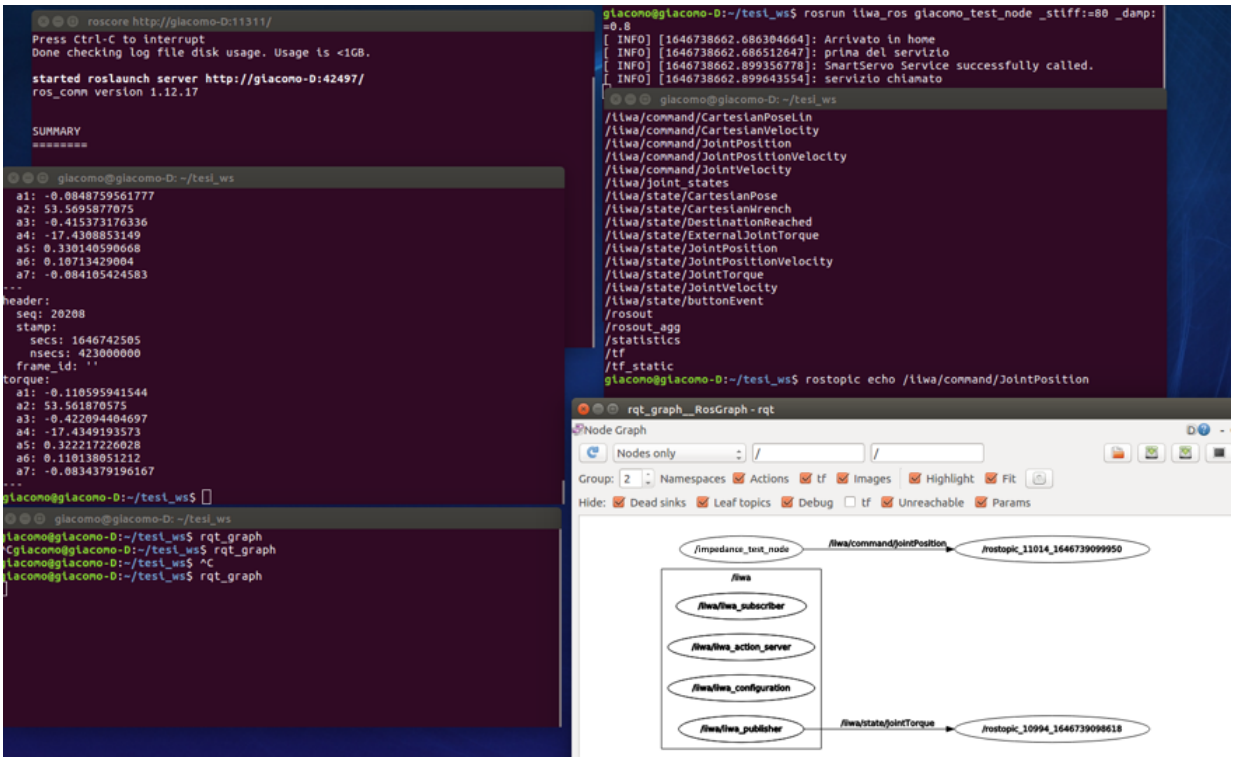


Figure 5.28: Service call for impedance control

It will be then possible to directly use the iiwa_stack topics like jointTorque or to call a service like impedance control of iiwa_stack in the impedance_test_node. Unfortunately, we did not have time to establish a connection between the load cell and ROS. This possibility is left for future developments starting from this configuration.

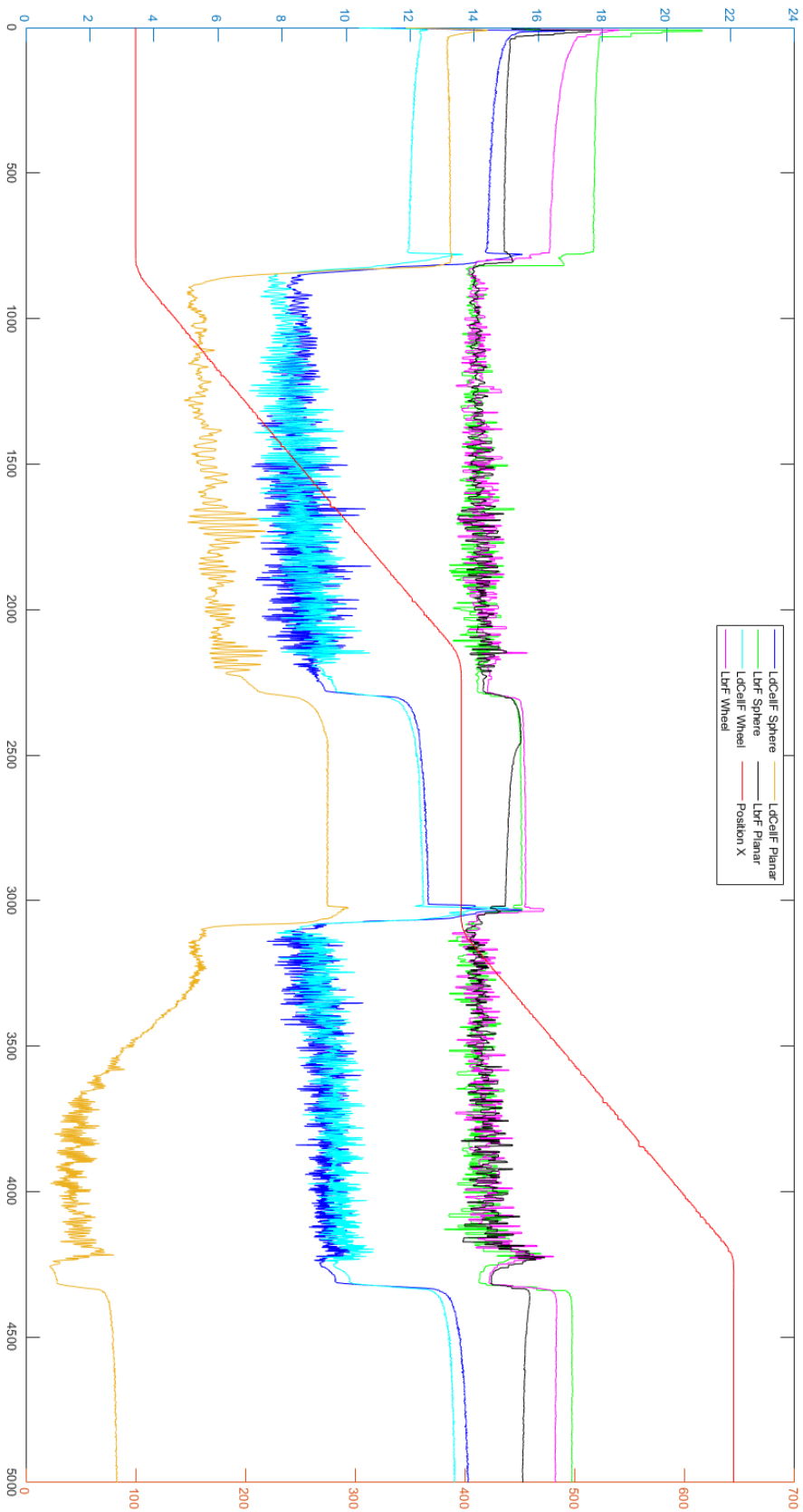


Figure 5.29: Comparison of the 3 tools with a 14N contact force

Chapter 6

Final analysis and conclusion

6.1 Results

We can conclude from the experiments performed that the LBR iiwa robot has an error limited to 1-2N and then approximately zero for the z-force measurement only in the static case, in a limited workspace region. The best region to perform the desired tasks will be on a z-height between 330mm and 465mm, and it is shown in figure 5.25.

If we want to execute a dynamic application where the robot moves along a desired trajectory in contact with a surface, the Cartesian z force measurement will be more inaccurate.
In particular:

- When the robot holds in a certain position in contact, it will have a constant force error between 3 and 4N;
- When the robot moves along the surface, it will have an average error of around 6N with some peaks of 7/8N

Notice that if we wanted to get the best results also in the dynamic case, we should have mounted the surface on the best height found in the static case. We have not been able to build a workbench at the optimal height for the limited time available, but these results could be helpful for future users if they want to create a new test bench. We have also seen how the tool used, related to the friction, affects the measurements. It is noticeable that measurement is affected by moments and transverse forces, this could mean that the measure can take these components into account.

In general, we have to say that we treated relative low forces during these experiments. Most of the tasks are well executed by the LBR with a high accuracy. In conclusion, this robot may be the right choice for many collaborative operations as it is one of the most performant

collaborative robots on the market. However, it could be interesting to rely on different solutions for very precise applications where we are mainly focused on the measurement of the force at the end effector in many different positions even close to singular configurations and near the end of workspace.

6.2 Future developments

Consequently, to what we summarized, some future developments may be interesting to treat if we would like to be further precise in force measurements. If the desired task should be further sensitive, for example if we would like to perform a medical operation (with a LBR Med very similar to LBR iiwa), we could try two different ways. We have seen that (especially with the planar tool), the moment generated by friction affects the load cell measurement a lot. There are two main further developments aimed at solving this problem:

- Firstly, it would be interesting to build a single-axis load cell model, able to show how a force perpendicular to the axis of measurement may affect the measure. The model can also be built experimentally by applying radial forces to the load cell itself. If we can model this effect, we could simply consider this phenomenon and compensate it. This could be an affordable and cheap solution
- An alternative way, more performant but more time-consuming and expensive, would be to consider a F/T Sensor measuring all the 3 Cartesian forces and the 3 Torques at the end effector. The experiments performed may be evaluated for the z axis, x and y-axis and torques. the output of a triaxial cell is certainly more similar to that provided by the LBR and would allow a comparison on all 6 force quantities.

Lastly, ROS could be an answer for integrating an external sensor when a high force accuracy is required. This could be the case of deburring, polishing, and grinding processes.

ROS could integrate an external F / t sensor more precisely than the internal force sensors of the LBR with impedance control. Using an external sensor could overcome the robot limits from the experiments, such as a lower accuracy near the singularities and at the points near the end of the workspace.

Thanks to ROS, we could solve these problems that may arise by keeping a low-level precision force control and getting a high-precision sensor at the wrist.

Bibliography

[1] G. Schuh and R. Anderl, *Industrie 4.0 Maturity Index*. acatech STUDY, 2020.

[2] S. R. Goel and P. Gupta, *Robotics and Industry 4.0*. 2019.

[3] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer.

[4] F. B. Insights, *Collaborative Robot Market with COVID-19 Impact Analysis, Component, Payload, Application, Industry And Region - Global Forecast to 2027*. NASDAQ OMX's News Release Distribution Channel, https://www.fortunebusinessinsights.com/industry-reports/collaborative-robots-market-101692, 2021.

[5] C. Melchiorri and G. Palli, *Slides of the Industrial Robotics Course*. UNIBO, 2020.

[6] P. Corke, *Base and Tool transforms*. https://robotacademy.net.au/lesson/base-and-tool-transforms/, 2017.

[7] KUKA, *KUKA Sunrise.OS 1.16, KUKA Sunrise.Workbench 1.16*. https://www.kuka.com/en-gb/products/robotics-systems/industrial-robots/lbr-iiwa.

[8] B. Siciliano and L. Sciavicco, *Robotics: modelling, planning and control*. Springer, 2009.

[9] T. Ren, Y. Dong, and K. C. Dan Wu, *Impedance Control of Collaborative Robots Based on a Joint Torque Servo with Active Disturbance Rejection*.

[10] N.Hogan, *Impedance Control: An approach to Manipulation*. Massachusetts Institute of Technology, 1985.

[11] B. Bona, *Slides of the PoliTO Robotics Course*. PoliTO,http://www.ladispe.polito.it/corsi/meccatronica/01PEEQW/2016-17/slides.html.

[12] C. Ott and Y. Nakamura, *Admittance Control using a Base Force/Torque Sensor*. 2009.

[13] A. D. Luca, *Slides of the Robotics Course*. UniRoma, https://www.diag.uniroma1.it/deluca/rob2_en/15_ImpedanceControl.pdf, 2017.

[14] H. Schildt, *Java the complete reference, seventh edition*. Mc Graw Hill, 2006.

[15] https://www.freecodecamp.org.

[16] https://www.theserverside.com.

[17] A. Alcocer, A. Robertsson, A. Valera, and R. Johansson, *Force estimation and control in robot manipulators*.

[18] *Beckhoff Manual*. https://www.beckhoff.com.

[19] C. D. Johnson and E. Edition, *Process Control Instrumentation Technology*. Pearson, 8th edition, 2005.

[20] *Load Cell and Weigh Module Handbook*. Rice Lake Weighing systems.

[21] https://www.omega.com/en-us/resources/how-to-wire-load-cell.

[22] https://www.farnell.com/datasheets/2313350.pdf.

[23] S. Mokaram, J. M. Aitken, and U. Martinez-Hernandez, *A ROS-integrated API for the KUKA LBR iiwa collaborative robot*. https://eprints.whiterose.ac.uk/115254/7/ros-integrated-api.pdf.

[24] IFL-CAMP. https://github.com/IFL-CAMP/iiwa_stack.

Appendix A

Java files of the application developed in SunriseWorkbench. Notice that some imports were omitted in order to save space.

```
1  package application; //Static force application
2  public class StaticForces extends RoboticsAPIApplication {
3      @Inject
4      private LBR lbr;
5
6      @Inject //I decide to attach the sphere tool with this
           application
7      @Named("SphereTool")
8      private Tool SphereTool;
9      private ObjectFrame toolcp;
10
11     public void robotGoHome() {
12         getLogger().info("Moving Robot in Home position");
13         lbr.move(ptp(getApplicationData().getFrame("/Home")).
               setJointVelocityRel(0.6));
14         getLogger().info("Robot in Home");}
15
16     public void robotGoStart() {
17         getLogger().info("Moving Robot to start position");
18         toolcp.move(ptp(getApplicationData().getFrame("/P1")).
               setJointVelocityRel(0.1));
19         getLogger().info("Robot in start position");}
20
21     public void run() {
22         int i, j = 0, k;
23         robotGoHome();
24         if (j == 0) robotGoStart();
25
```

```
26          while (j < 5) {  //with these nested loops I follow a
               rectangular surface for 5 different heights
27              k=0;
28              while (k < 3) {
29                  i = 0;
30                  while (i < 24) {
31                      getApplicationControl().pause();
32                      ThreadUtil.milliSleep(6000); // wait for the
                           assessment time
33                      System.out.println("POSX: "
34                              + lbr.getCurrentCartesianPosition(
                                   toolcp).getX()+ "POS Y: "
35                              + lbr.getCurrentCartesianPosition(
                                   toolcp).getY()+ "POSZ: "
36                              + lbr.getCurrentCartesianPosition(
                                   toolcp).getZ()+ "ForceZ: "
37                              + lbr.getExternalForceTorque(toolcp).
                                   getForce().getZ());
38                      if (i < 11){
39                          toolcp.move(linRel(0.0, 50.0, 0.0).
                               setJointVelocityRel(0.2));}
40                      else if ((i ==11)||(i==23)) {
41                          toolcp.move(linRel(45.0, 0.0, 0.0).
                               setJointVelocityRel(0.2));}
42                      else if (i < 23) {
43                          toolcp.move(linRel(0.0, -50.0, 0.0).
                               setJointVelocityRel(0.2));}
44                      else {  robotGoHome();
45                          getLogger().info("App finished");
46                          return;}
47                      i++;}
48                  k++;}
49              if (j < 4)
50                  toolcp.move(linRel(-270.0, 0.0, 135.0).
                       setJointVelocityRel(0.1)); //Moving to Start1
                       with shifted height
51              j++;
52          }
```

```
53          robotGoHome();
54          getLogger().info("App finished");}}
```

```
1  package application;// Point Sliding application
2  public class PointSliding extends RoboticsAPIApplication {
3      @Inject
4      private LBR lbr;
5      @Inject
6      @Named("SphereTool")
7      private Tool SphereTool;
8      private ObjectFrame toolcp;
9      @Inject
10     private PlcIoIOGroup PlcIo;
11     private static final int stiffnessZ   = 50; //soft motion
12     private static final int stiffnessY   = 2500; //stiff motion
13     private static final int stiffnessX   = 2500;
14     private static final int stiffnessROT = 300; //stiff motion
15     private ForceCondition ContactForceReached;
16     private DataRecorder logger;
17     int stop_rec;
18     private final static String informationText= "The robot, after
           the homing and the reaching of start position, moves" +
19             "in impedance control mode along the segment.\nThe
                  stiffness during motion is set to " +
20             "X="+stiffnessX+" Y="+stiffnessY +" Z="+stiffnessZ+"
                  in N/m. Rotational="+stiffnessROT+"in Nm/Rad";
21     @Override
22     public void initialize() {
23         SphereTool.attachTo(lbr.getFlange());
24         toolcp = SphereTool.getFrame("TCP");}
25     public void robotGoHome() {
26         getLogger().info("Moving Robot in Home position");
27         lbr.move(ptp(getApplicationData().getFrame("/Home")).
              setJointVelocityRel(0.3));
28         getLogger().info("Robot in Home");}
29     public void robotGoStart() {
30         getLogger().info("Moving Robot to start position");
31         toolcp.move(ptp(getApplicationData().getFrame("/StartWheel
              ")).setJointVelocityRel(0.3));
```

```
32          getLogger().info("Robot in start position");}
33      public void stopLogForce(){
34          if(logger.isRecording()){
35          logger.stopRecording();}}
36      public void run() {
37              int isCancel = getApplicationUI().displayModalDialog(
                   ApplicationDialogType.QUESTION, informationText, "
                   OK", "Cancel");
38              if (isCancel == 1){ return;}
39                  robotGoHome();
40              robotGoStart();
41              getApplicationControl().pause();
42              CartesianImpedanceControlMode ImpMode = new
                   CartesianImpedanceControlMode();
43      ImpMode.parametrize(CartDOF.X).setStiffness(stiffnessX);
44      ImpMode.parametrize(CartDOF.Y).setStiffness(stiffnessY);
45      ImpMode.parametrize(CartDOF.Z).setStiffness(stiffnessZ);
46      ImpMode.parametrize(CartDOF.ROT).setStiffness(stiffnessROT);
47      ImpMode.parametrize(CartDOF.Z).setAdditionalControlForce(10);
48      ImpMode.parametrize(CartDOF.ALL).setDamping(0.9);
49                  ContactForceReached = ForceCondition.
                       createSpatialForceCondition(toolcp,13);
50          IRisingEdgeListener ContactListener = new
                   IRisingEdgeListener()
51          { @Override
52          public void onRisingEdge(ConditionObserver
                   conditionObserver,
53          Date time, int missedEvents) {} };
54          ConditionObserver collisionObserver = getObserverManager()
                   .createConditionObserver(ContactForceReached,
                   NotificationType.MissedEvents,ContactListener);
55          collisionObserver.enable();
56          //moving to the desired path: T1, T2, T3 with planar Tool.
                P1, P2, P3 with sphere and wheel tools
57          toolcp.move(lin(getApplicationData().getFrame("/P1")).
                   setCartVelocity(2).breakWhen(ContactForceReached));
58          getLogger().info("Hold position in impedance control mode"
                   );
```

```
59          PlcIo.setStartRec(1); //Starting recordings before moving
               to P1 (for 50 seconds with 5000 samples)
60           getLogger().info("Plc Recording started");
61          startLogForce(50000);
62           getLogger().info("Logger Recording started");
63          toolcp.move((new PositionHold(ImpMode, 7, TimeUnit.SECONDS
               )));
64          getLogger().info("Moving in impedance control mode");
65          toolcp.move(lin(getApplicationData().getFrame("/P2")).
               setCartVelocity(45).setMode(ImpMode));
66          toolcp.move((new PositionHold(ImpMode, 7, TimeUnit.SECONDS
               )));
67          toolcp.move(lin(getApplicationData().getFrame("/P3")).
               setCartVelocity(45).setMode(ImpMode));
68          toolcp.move((new PositionHold(ImpMode, 7, TimeUnit.SECONDS
               )));
69          getLogger().info("App finished");
70          if (PlcIo.getInput01() == 1){
71              stopLogForce();
72              PlcIo.getInput01(); }
73          PlcIo.setStartRec(0);
74          getLogger().info("Recordings finished");
75          robotGoHome(); } }
```

```
1  //ForceLog Background application
2  package backgroundTask;
3  public class ForceLog extends RoboticsAPICyclicBackgroundTask {
4      @Inject
5      private LBR lbr;
6      @Inject
7      private PlcIoIOGroup PlcIo;
8      @Inject
9      @Named("SphereTool")
10     private Tool SphereTool;
11     private ObjectFrame toolcp;
12     @Override
13     public void initialize() {   //I attach the tool depending on
          the application I perform
14          initializeCyclic(0, 10, TimeUnit.MILLISECONDS,
```

```
                CycleBehavior.BestEffort);
15          SphereTool.attachTo(lbr.getFlange());
16          toolcp = SphereTool.getFrame("TCP");}
17      @Override public void runCyclic() {
18          PlcIo.setWriteReady(0);
19  double xPos=lbr.getCurrentCartesianPosition(toolcp).getX();
20  double yPos=lbr.getCurrentCartesianPosition(toolcp).getY();
21  double zPos=lbr.getCurrentCartesianPosition(toolcp).getZ();
22  double xForce=lbr.getExternalForceTorque(toolcp).getForce().getX
        ();
23  double yForce = lbr.getExternalForceTorque(toolcp).getForce().
        getY();
24  double zForce = lbr.getExternalForceTorque(toolcp).getForce().
        getZ();
25  double xForceInacc = lbr.getExternalForceTorque(toolcp).
        getForceInaccuracy().getX();
26  double yForceInacc = lbr.getExternalForceTorque(toolcp).
        getForceInaccuracy().getY();
27  double zForceInacc = lbr.getExternalForceTorque(toolcp).
        getForceInaccuracy().getZ();
28          //what I log on KUKA smartPAD
29  getApplicationData().getProcessData("posX").setValue(xPos);
30  getApplicationData().getProcessData("posY").setValue(yPos);
31  getApplicationData().getProcessData("posZ").setValue(zPos);
32  getApplicationData().getProcessData("forceX").setValue(xForce);
33  getApplicationData().getProcessData("forceY").setValue(yForce);
34  getApplicationData().getProcessData("forceZ").setValue(zForce);
35  getApplicationData().getProcessData("forceInaccX").setValue(
        xForceInacc);
36  getApplicationData().getProcessData("forceInaccY").setValue(
        zForceInacc);
37  getApplicationData().getProcessData("forceInaccZ").setValue(
        yForceInacc);
38          //Converting to float only the elements I want to
              communicate to PLC
39          float zf = (float)zForce;
40          byte[] zForceArray = ByteBuffer.allocate(4).putFloat(zf).
              array();
```

```
41          float xp = (float)xPos;
42          byte[] xPosArray = ByteBuffer.allocate(4).putFloat(xp).
               array();
43          float yp = (float)yPos;
44          byte[] yPosArray = ByteBuffer.allocate(4).putFloat(yp).
               array();
45          PlcIo.setWriteReady(1);
46          ThreadUtil.milliSleep(1);
47          // Converting into 4 Bytes the Float values passed to PLC
               through Ethercat Bridge
48          synchronized (this) {  //for doing the conversions
               atomically
49              PlcIo.setForceZB4(zForceArray[3] & 0xff);
50              PlcIo.setForceZB3(zForceArray[2] & 0xff);
51              PlcIo.setForceZB2(zForceArray[1] & 0xff);
52              PlcIo.setForceZB1(zForceArray[0] & 0xff);
53              PlcIo.setPosXB4(xPosArray[3] & 0xff);
54              PlcIo.setPosXB3(xPosArray[2] & 0xff);
55              PlcIo.setPosXB2(xPosArray[1] & 0xff);
56              PlcIo.setPosXB1(xPosArray[0] & 0xff);
57              PlcIo.setPosYB4(yPosArray[3] & 0xff);
58              PlcIo.setPosYB3(yPosArray[2] & 0xff);
59              PlcIo.setPosYB2(yPosArray[1] & 0xff);
60              PlcIo.setPosYB1(yPosArray[0] & 0xff);
61              PlcIo.setWriteReady(2); // passes a value to PLC when
                   conversion is made correctly on the robot side }

62  //log all Torques, Forces and Positions only if necessary because
       it slows down the cyclic task, configuration can be changed in
       Workvisual
63          ThreadUtil.milliSleep(5);} }
```

Appendix B

WritingData Function Block

Function Block aimed at saving the data recorded from load cell and LBR

```
1  FUNCTION_BLOCK  WritingData_FB
2  VAR_INPUT
3      command_enable  : BOOL; PLC_to_CSV_Array : PLC_to_CSV_Array;
4      Index_record    :UDINT; counter_cycle_test :    INT;
5  END_VAR
6
7  VAR
8      cycle_test  :   STRING(2);
9      File_Open   :   FB_FILEOPEN; File_Puts : FB_FILEPUTS;
10     File_Close  :   FB_FILECLOSE; state : INT;
11     file_csv    :   T_MAXSTRING;
12     nome_file_csv :     T_MAXSTRING     := 'C:\Users\Administrator
           \Desktop\SlidingSphere10D02';
13     sNetId      :   T_AmsNetId      := '';
14     csv_mem_buffer_writer   :   FB_CSVMEMBUFFERWRITER;
15     Line_CSV    :   T_MAXSTRING; //Linea che vado a scrivere in
           csv
16     Csv_field   :   T_MAXSTRING; // field da dare a CSV_MEMBUFFER
17     String_     :   T_MAXSTRING; // stringa che do in pasto al
           buffer
18     Tag_csv     :   T_MAXSTRING := 'Weight(Kg);LdCellF_Z;lbrF_Z;
           ForceDiff;Pos_X;Pos_Y';
19     h_file      :   UINT; n_line :  UDINT;
20 END_VAR
21
22 VAR_OUTPUT
23     command_done                :   BOOL;
```

```
END_VAR
cycle_test      := INT_TO_STRING(counter_cycle_test);
CASE state OF
0:  //INIT_RECORDING
    IF command_enable THEN
        command_done            := FALSE;
        file_open.bExecute      := FALSE;
        file_puts.bExecute      := FALSE;
        file_close.bExecute     := FALSE;
        h_file                  := 0;
        n_line                  := 0;
        state                   := 1;   //FILE_OPEN
    END_IF
1:  //FILE_OPEN
    command_done            := FALSE;
    file_open.bExecute      := TRUE;
    file_puts.bExecute      := FALSE;
    file_close.bExecute     := FALSE;
    state                   := 2;   //WAIT_OPEN
2:  //WAIT_OPEN
    Command_Done                := FALSE;
    File_Open.bExecute          := FALSE;
    File_Puts.bExecute          := FALSE;
    File_Close.bExecute         := FALSE;
    h_file                      := file_open.hFile;
    IF NOT File_Open.bBusy AND  NOT File_Open.bError THEN
            State       := 3;//WRITING_CSV;
    END_IF
3:  //WRITING_CSV
    command_done            := FALSE;
    file_open.bExecute      := FALSE;
    file_puts.bExecute      := FALSE;
    file_close.bExecute     := FALSE;
    Line_CSV                := '';
    csv_mem_buffer_writer.eCmd  := eEnumCmd_First;
    IF n_line < (Index_record + 1) AND n_line = 0 THEN
            String_ := Tag_csv; String_ := CONCAT(String_, '$N');
    ELSIF n_line < (Index_record + 1) AND n_line > 0 THEN
```

```
62              String_                 := CONCAT(REAL_TO_STRING(
                   PLC_to_CSV_Array.arrWeight[n_line]),';');
63              String_     := CONCAT(String_,REAL_TO_STRING(
                   PLC_to_CSV_Array.arrForceZ[n_line]));
64              String_     := CONCAT(String_, ';');
65              String_     := CONCAT(String_,REAL_TO_STRING(
                   PLC_to_CSV_Array.arrlbrForceZ[n_line]));
66              String_     := CONCAT(String_, ';');
67              String_     := CONCAT(String_,REAL_TO_STRING(
                   PLC_to_CSV_Array.arrForceDiff[n_line]));
68              String_     := CONCAT(String_, ';');
69              String_     := CONCAT(String_,REAL_TO_STRING(
                   PLC_to_CSV_Array.arrlbrPositionX[n_line]));
70              String_     := CONCAT(String_, ';');
71              String_     := CONCAT(String_,REAL_TO_STRING(
                   PLC_to_CSV_Array.arrlbrPositionY[n_line]));
72              String_                 := CONCAT(String_, '$N');
73      END_IF
74      IF  n_line < (Index_record + 1) THEN
75              csv_field               := STRING_TO_CSVFIELD( String_
                   , FALSE );
76              Line_csv                := csv_field;
77              n_line                  := n_line + 1;
78              State                   := 4; //WRITE_LINE
79      ELSIF n_line >= (Index_record + 1)  THEN
80              state                   := 6; //FILE_CLOSE  END_IF

81  4:  //WRITE_LINE:
82          Command_Done                := FALSE;
83          File_Open.bExecute          := FALSE;
84          File_Puts.bExecute          := TRUE;
85          File_Close.bExecute         := FALSE;
86          State                       := 5;//WAIT_PUTS;
87  5:  //WAIT_PUTS:
88          Command_Done                := FALSE;
89          File_Open.bExecute          := FALSE;
90          File_Puts.bExecute          := FALSE;
91          File_Close.bExecute         := FALSE;
```

```
92          IF NOT File_Puts.bBusy AND  NOT file_puts.bError THEN
93              State               := 3; //file write END_IF
94  6:  //FILE_CLOSE
95      command_done            := FALSE;
96      file_open.bExecute      := FALSE;
97      file_puts.bExecute      := FALSE;
98      file_close.bExecute     := TRUE;
99      State                   := 7;   //wait close
100 7:  //WAIT_CLOSE
101         Command_Done                := TRUE;
102         File_Open.bExecute          := FALSE;
103         File_Puts.bExecute          := FALSE;
104         File_Close.bExecute         := FALSE;
105         IF ( NOT file_close.bBusy ) THEN
106             h_file      := 0;
107             State       := 8; //ERROR_OR_LOGGING_END; END_IF

108 8:  //ERROR_OR_LOGGING_END
109     Command_enable := FALSE;
110     State       := 0; //init
111 END_CASE
112 file_csv := concat(nome_file_csv,'.csv');
113 File_Open(
114     sNetId      := sNetId,
115     sPathName   := file_csv,
116     nMode       := FOPEN_MODEWRITE OR FOPEN_MODETEXT,
117     ePath       := PATH_GENERIC,
118     tTimeout    := T#5S );
119 File_puts(
120     sNetId      := sNetId,
121     hFile       := h_file,
122     sLine       := Line_csv,
123     tTimeout    := T#5S );
124 file_close(
125     sNetId      := sNetId,
126     hFile       := h_file,
127     tTimeout    := T#5S );
```