

École polytechnique de Louvain

Development of an autonomous robot for construction sites



Author: **Aurèle CORRILLON**

Supervisors: **Renaud RONSSE, Cédric MOUTSCHEN**

Reader: **Renaud DETRY**

Academic year 2020–2021

Master [120] in Electro-mechanical Engineering

Abstract

Today more and more companies are focusing on robotic solutions. This is also the principal interest of the company in which this thesis is taking place.

As the safety measures to protect the workers on construction sites are tedious, they extend the duration of the task to be performed. In order to make tasks at high heights feasible and more efficient, *IMAX Pro* wants to develop a robotic solution that can work under those circumstances.

This is an ambitious project that combines many different elements and knowledge, that is why during this thesis, only a part of the whole solution can be elaborated on. Indeed, the main area of focus will be the positioning of the robot in its environment and the compensation of potential external perturbations.

After setting the context, all the different elements that constitute the solution to this part of the problem will be detailed. The main components are : the robot itself, the telehandler, the total station and its beacon and the inclination sensor. Then, the way they communicate with one another will be detailed. After that, the different tests that have been performed to validate the solution are introduced.

The first test aims at verifying that the robot can follow a trajectory set by an external device. In this case, a marker is attached to the head of the robot thanks to a designed tool, another marker is fixed to a beacon. The robot then copies the drawing made by manually moving the beacon.

The second test is designed to verify if the robot can adapt its trajectory when an external perturbation is inflicted on it. To do so the robot is programmed to draw a square on a sheet of paper. The first step of the test is to make the robot draw the square without any disturbance, this square can then be used as a reference to compare the result of a drawing with perturbations. During the first sub-test, only the orientation of the wooden board on which the paper is placed is moved in order to test if the orientation compensation works. Then a second sub-test is performed where the inclination is kept stable but the location of the wooden board is moved. Lastly, a third test combines both types of perturbations.

The results from both tests show that the implemented systems have a slight time delay but apart from that, they are capable of completing the required task. Some important limitations do have to be taken into account, indeed, the system was tested on the floor instead of at a height and multiple options to better the solution are provided.

Acknowledgement

Before the beginning of this report, I would like to warmly thank the whole team of *IMAX Pro*. They gave me the opportunity to realise my internship, as well as my thesis in the best conditions, despite the current sanitary situation. It was a great opportunity for me to learn a lot about electromechanics.

All my special thanks to Cédric Moutschen, the CEO of the *IMAX Pro* company, who was also my thesis supervisor. Despite his overbooked agenda, he supported me and took the time to discuss potential applications.

Additionally, I would also like to particularly thank two other people of the *IMAX Pro* team. Firstly, Grégory Martin, who helped me with the server application. As well as Nicolas Tiquet, who helped and supervised me at the end of my thesis, especially during all my tests on the real robot at *IMAX Pro*.

Finally, I would also like to express my thanks to Professor Renaud Ronsse, who was my academic supervisor at the Université Catholique de Louvain, for his feedback over my thesis during the academic year.



Table of contents

1	Introduction	1
1.1	Contextualization	1
1.2	Development of robotics in construction	2
1.2.1	State of the art	3
1.2.2	Today's applications	6
1.3	Overview of the project	9
1.4	Objectives of the thesis	9
2	System description	10
2.1	System overview	10
2.2	Elements of the system	11
2.2.1	Robot	11
2.2.2	Telehandler	13
2.2.3	Total station and its equipment	13
2.2.4	Inclination sensor	19
2.3	Localisation and compensation systems	20
2.3.1	Localization system	20
2.3.2	Compensation system of the position and orientation	22
3	Position and orientation controller	24
3.1	From measured data to movement instructions	24
3.2	Data transfer between the elements	25
3.2.1	Overview	26
3.2.2	Total station - Server application	27
3.2.3	Inclination sensor - Server application	30
3.2.4	Server application - Robot	33
3.3	Information sent to the robot's controller	34

3.3.1	Robtargets	34
3.3.2	Movement instructions	36
3.3.3	Quaternions	39
3.4	Limitations of the inclination sensor	40
4	Validation campaign	41
4.1	Testing setup	41
4.1.1	Follow the beacon	42
4.1.2	Square drawing	45
4.2	Software implementation	46
4.2.1	Software structure	47
4.2.2	Connecting functions	48
4.2.3	<i>SquareDrawing_Compensation</i>	49
4.2.4	<i>SquareDrawing_WithoutCompensation</i>	53
4.2.5	<i>Draw_With_Beacon</i>	54
4.2.6	Disconnecting functions	56
4.3	Test results	57
4.3.1	Follow the beacon	57
4.3.2	Square drawing with position compensation	60
4.3.3	Square drawing with orientation compensation	63
4.3.4	Square drawing with position and orientation compensation	67
4.4	Limitations of the tests	70
5	Discussion	72
5.1	Difficulties encountered	72
5.2	Improvements	73
5.3	Future researches	75
	Appendices	76
A	Quaternions - Mathematical Development	77
B	Code lines of the server application	81

1

Introduction

Self-driving cars have grown rapidly over the past several years. Nevertheless, autonomous driving technology could also revolutionize a different industry. What if the same idea of this technology is apply to the construction sector?

1.1 Contextualization

The way we build today is largely unchanged from the way we used to build 50 years ago. However, this would changed in the few decades. Indeed, the robots have progressively modified the manufacturing processes over the few last decades, as demonstrated with the production lines in the automobile industry. Now, it is the time of the construction industry to change its way of functioning.

Moreover, the construction sector represents one of the largest sectors in the global economy. However, this sector suffers from a huge amount of waste, a labor productivity declining and is also faced with a labor shortage, that the Covid-19 pandemic has further complicated [1]. In this context, the robotisation of the sector could have a positive impact on the sector.

Context of the project The presented project enrolls this development of robotics in the construction sector. Indeed, our days, the security aspect takes more and more place on the construction sites. In some cases, the time to install the security infrastructures requires more time compared to the time of the job. That is why, it would make sense to be able to overcome this security aspect. Therefore, the idea behind the project is to design an autonomous robot that can be placed in the construction sites, which requires the less human assistance.

1.2 Development of robotics in construction

To illustrate the changing context of the construction industry, the following paragraphs will present an overview of the actual developments in this sector.

Over the last few years, the range of application for mobile robot has significantly grown, as well as the application for construction robotics. Indeed, the global market for construction robotics represents a huge opportunity for developers and suppliers. Tractica predicts that this markets revenues could grow from \$22.7 million in 2018 to \$226.0 million annually by 2025. [2].

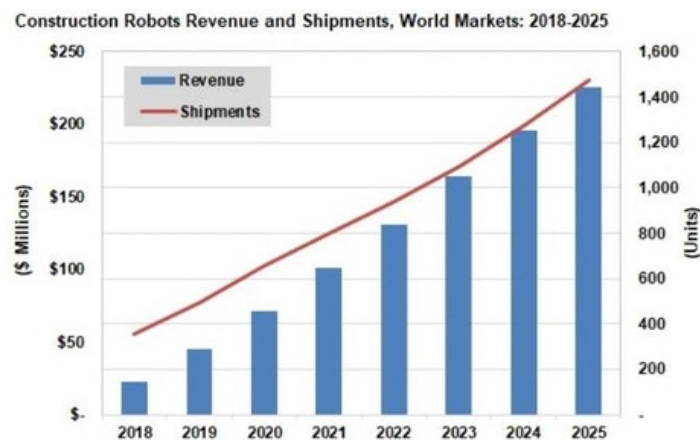


Fig. 1.1: Annual spending on construction robots

The construction industry has historically been slow to adapt the new technologies, nevertheless automation and internet of things are driving a revolution in this sector.

One of the first applications of robotic technologies to the construction industry were designed in Japan during the 70's, in order to improve the quality of pre-fabricated elements for modular residential buildings [3]. Since, the robots have begun to spread into the construction industry, moving smoothly from the factories towards the construction sites.

1.2.1 State of the art

Japan has been the first country to look into automated on-site construction processes in the late 70. Indeed, Japanese companies started to design and develop automated systems to accomplish single and repetitive work, to cope with the lack of skilled labor force for future construction. Then, this research and development has accelerated in the mid-80s. Particularly in Japan, where many construction robots were developed in the 80s and 90s based on the studies of the Japan Architectural Institute.

The first examples of the on-site construction robots were developed to assist the workers in hazardous and risky construction sites. Those single task construction robots were therefore designed for painting, concrete leveling, concrete finishing, tile setting, steel welding and so forth. The Japanese company, Shimizu, developed the first attempt of a single task construction robot in 1983, it was called SSR-1. This fireproofing spray robot was mounted on a wheeled base, in order to be movable [4].

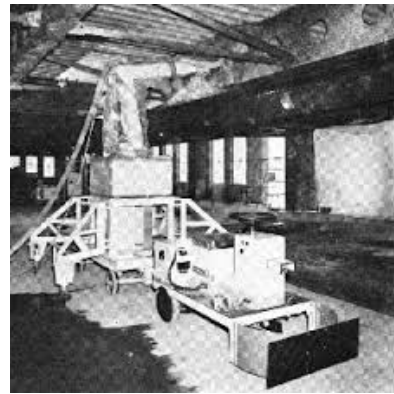


Fig. 1.2: SSR-1 [4]

Less autonomous but more collaborative robots were also designed as the 'Tai-sei's Boardman-100'. This robot was developed to ease the effort of manual labor by lifting heavy panels into place while the attachment was done by humans [5].



Fig. 1.3: ROCCO project brick assembly robot [6]

Then, automated construction systems were developed in the 90's due to the limitations of the single task construction robots. This is how, industrial robots, which were being used since the 60's for manufacturing and assembly within the automobile reached the construction industry.

Therefore, the repetitive tasks within the factories are replaced by the unique tasks in unpredictable sites. For instance, the mobile brick layer robot systems, called ROCCO,

an European project, in which a CAD model could be divided into block units and be used to calculate robots working points [5].

During the 90's, the research and development activities were mostly focused in the development on new teleoperated robotic systems and in automation of existing machines. However, the efforts in the early 2000's were dedicated more in the software integration, data acquisition and processing, safety, sensor-based process control and construction industrialization [6].

In the field of road construction, several projects have been developed during this time. For example the European project OSYRIS, which focused on the coordination of several machines in order to improve productivity. The objective was to develop a semi-autonomous guidance system of the machines based in the GPS and laser data, by controlling the speed, travelled distance, ... [7]



Fig. 1.4: OSYRIS project [7]

In the field of house building, robotics is also growing in the early 2000's. The first representative robots are mostly Japanese ones. Those robots are teleoperated and perform simple operations, as compact and control the thickness of the concrete floors, painting and steel column fire protection spraying, assembly of interior walls and ceilings, ...



Fig. 1.5: R.O.B. [5]

The earliest applications in mobile brick layer robot system, such as the ROCCO, changed the way we can build. Indeed, during the 2000's, more and more applications focus on the research in architectural assembly of similar units, such as the R.O.B. mobile robot of Gramazio and Kohler.

This research integrates sensor systems and computational processes to allow self-

calibration when needed. Then, more contemporary research focuses on more accurate sensor and vision systems to allow adaptable path planning based on the environment. For instance, to adapt to inaccuracies of the site or human interaction [5]. Today's applications research for house building include industrial robots with 3D printing.



Fig. 1.6: Prototype of aid wearable robot [8]

Furthermore, one new research topic emerged in the construction industry during the early 2000's, which is the use of robotic exoskeleton suits in the construction sites. Initially designed from a military point of view, a wearable exoskeleton robot allows the operator to overcome his natural limits and handle heavy objects during their activities without requiring large muscular power [7].

The prototype developed on this following picture 1.6, explained such an application. This research, which became a market is growing very fast. Indeed, ABI Research predicts that the robotic exoskeleton market alone will reach \$ 1.8 billion in 2025, up from \$ 68 million in 2014. They expect that 2.6 million suits could be seen on the market in 2025, against 6000 in 2019, mainly for rehabilitation [9].

The development in robotics in the last century are so various and impact so many sectors, that it is not possible to cover all of them. This previous section has tried to expose some of them, in order to have a brief idea of the evolution of which types of robots are used. The following section will be more focused on the applications currently on the market or which will be in the following years.

1.2.2 Today's applications

Nowadays, more and more robotic applications are transforming the construction sites. One example of this revolution could be the Hadrian X, which is a bricklaying machine.

Developed in Australia, this machine is capable of safely working outdoors in uncontrolled environments with speed and accuracy. The software converts 3D wall sketches into block positions, producing far less waste while drastically improving site safety [10].

Furthermore, another example of the same type is the SAM100, which is a brick laying robot designed by the company Construction Robotics. It is also designed to reduce operating costs and waste, as well as ensure safer work environments [11].



Fig. 1.7: Hadrian X [10]



Fig. 1.8: HX02 [12]

Even though a lot of effort is put into self-driving cars, autonomous vehicles could revolutionize the construction industry as well. One of the many applications coming to the market is the HX02 developed by Volvo.

This autonomous, battery-electric, load carrier is a concept machine, which incorporates shared technologies, such as a vision system that detects humans or obstacles.

This innovative concept is part of an electric site research project that predicts up to a 95 % reduction in carbon emissions [12].

Besides, another application for loading and holding is the CAT MINESTAR from the company Caterpillar. This autonomous mining truck operates safely around other equipment, like light vehicles or personnel [13].

Moreover, deep-learning and big data are also starting to make their appearance in the constructions sites. Indeed, the startup Doxel offers autonomous robots equipped with lidar. These robots scan construction sites every day to monitor progression of the sites. They are tracking what gets installed and whether it's the right thing at the right time in the right place, in order to increase labor productivity. Once the robot has scanned the site, it sends the data to the cloud, to be analysed by the 3D semantic deep-learning algorithms, to recognize all kinds of components [14].

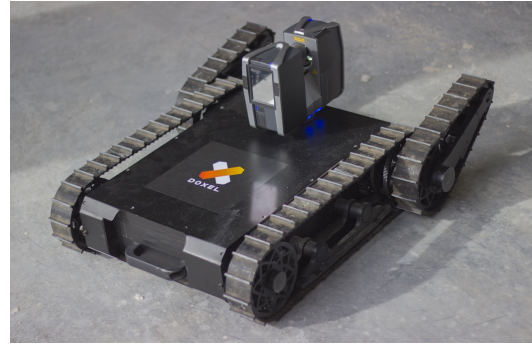


Fig. 1.9: Autonomous robot to tack construction projects [14]



Fig. 1.10: Autonomous excavator from Built Robotics [15]

The machine as showed on Fig. 1.10 is equipped with a security system to constantly check around the vehicle [15].

Self-driving, the technology mostly developed for the car, is also adapted for the construction. In fact, the company Built Robotics develops kits that retrofit existing construction equipment with lidar, GPS and Wi-Fi. Those kits give the ability to the machines to autonomously map and navigates its surroundings. The purpose of the company is not to build new dozers and excavators with their technology, but to enhance actual equipment already on the sites. This automation expands the working hours with its ability to work also during the night.

Another technology that is also arriving in construction sites are drones. Indeed, as well as the anthropomorphic robot, the drones can build walls based on the assembly of identical units. Different universities are developing new technologies in this area, like the University of Bonn, but the Catholic University of Louvain is also taking part of this improvement in the construction field. As can be seen by the research of Professor



Fig. 1.11: Autonomous drones for construction sites [16]

P. Latteur on the development of an autonomous construction system by drones [16].



Fig. 1.12: Plastering robot [17]

Industrial robots could also be used in the painting field. Indeed, the company Okibo developed an autonomous plastering robot, which can operate next to workers in a construction site. Thanks to its accurate 3D scanning combined with the accuracy of the industrial robotic arm itself, the robot can reach extreme finish quality.

Furthermore, the time spent on the construction sites is reduced and the technology prevents the exposure of human workers to potentially toxic materials. Those strong advantages make it an important competitor on the market for the following years [17].

Thanks to a collaboration between two companies ULC Robotics and SGN, an autonomous robotic system is being developed, which can be used for roadway excavation projects. This solution combines an industrial robot with an all-electric track drive system, as described on the Figure 1.13.



Fig. 1.13: Autonomous roadwork excavation [18]

Moreover, advanced data processing is also being developed to decide when and where to excavate. The artificial intelligence is also used to manage the sensors scanning the ground or localising the system. This solution still in development, will bring a more efficient method of excavation to improve the efficiency, safety, and lower the environmental impact [18].

Ready or not, advanced robotics, AI and automated hardware are all making their way into the construction industry. Through those different growing technologies, the construction 4.0 continues its development to remodel the way we build today and robotics serve as a major catalyst for it [19].

1.3 Overview of the project

After this introduction of some robotic applications that already exist in the construction sector, this section introduces the developed solution related to this thesis.

The purpose of the project is to develop a robotic solution that can reach high heights in construction sites to perform autonomous tasks that humans already perform at those heights, like machine a wooden board for example. However, the advantage of this presented autonomous solution over the workers is that the security part requires for a human at those heights can be bypassed and integrated directly in the solution.

As the solution would be placed on construction sites, a localisation system is required in order that the robot can localise in its environment. Moreover, as the solution would be placed outside on construction sites, some external perturbations would be applied on the system. Indeed, the wind would induce position and orientation disturbances on the solution. In order to perform the desired task precisely, a system has to be developed to overcome those perturbations.

1.4 Objectives of the thesis

Given the extent of the total project, only a fraction of it can be handled during this thesis. This brief section will bring out the different objectives of this thesis.

Firstly, several elements are involved to achieve to an autonomous robot for construction sites, which is the topic of the following chapter. The elements constituting the solution have been discussed and are based on the elements at the society *IMAX Pro*, at the origin of the project.

As presented in the previous Section 1.3, one important part of the project which would be covered by this thesis is the localisation and compensation part. Indeed, the localisation of the robot in its environment will be discussed. As well as how the robot will counteract the external random perturbations that are applied on the system based on external devices.

Moreover, in order to perform a good localisation and to achieve to this compensation system, a communication protocol has been established to collect the information of the position sensor and orientation sensor, which would be detailed as well as the position and orientation controller.

2

System description

The previous chapter introduced the subject of the thesis, as well as the objectives and an overview of the today's applications of robotics in construction. This chapter will focus on the system and the different elements that compose it.

2.1 System overview

To begin, it is important to have a brief overview of the system and its elements. As previously mentioned, the purpose of the system is to reach high heights in construction sites to perform jobs that the robot can actually do on the ground, thanks to the applications developed by *IMAX Pro*. That is why the robot is placed on a telehandler, as described on the schematic overview in Figure 2.1.

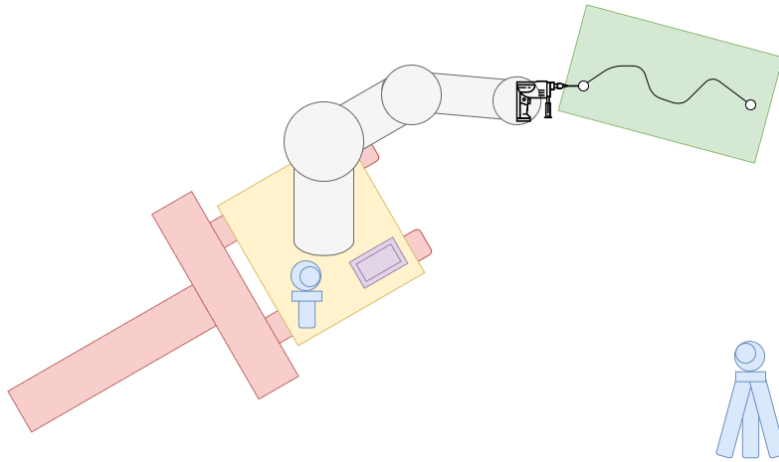


Fig. 2.1: Schematic view of the system

On this representation, the telehandler is highlighted in red. It's unique telescopic arm allows the robot to reach the working zone, shown in green. The platform connecting the two and maintaining the robot on the telehandler, is represented by the yellow square. Then, the robot itself is schematized in grey in the figure, the end-effector the tool is connected to it, enabling the robot to perform the task.

In addition, the positioning of the robot into the space is a critical aspect, which is achieved thanks to the total station and its related beacon, which are represented in blue. Finally, the orientation of the base of the robot is also an important part, because it conditions the final orientation of the end-effector, which has to be known to perform the task precisely as well as the positioning part. That is why the orientation is measured with inclination sensor, represented in purple on the schematic view.

2.2 Elements of the system

Once the different elements of the system have been introduced, a more deeper description of them is required, which is the subject of this section.

2.2.1 Robot

The first one and probably the most important element of the system is the robot. There are several companies offering robots that could perform this task in a magnitude of different sizes . However, based on the available robots at *IMAX Pro*, the chosen robot is a IRB 7600 from the company ABB, as represented on the Figure 2.2. This robot is ideal for heavy-weight applications, regardless of the industry.

Characteristics Firstly, this industrial robot has 6 rotational joints, with an anthropomorphic configuration, its base is fixed and it is commonly used in the manufacturing industry. Moreover, it has a handling capacity of 325 kg, a reachable distance of 3,1 m and the weight of the manipulator is 2440 kg. In terms of the operating system, the robot is equipped with the IRC5

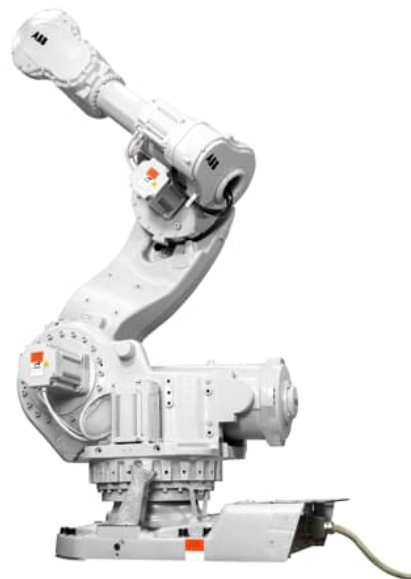


Fig. 2.2: IRB 7600 from ABB [20]

controller and robot control software, named RobotWare. This control software supports every aspect of the robot system, such as motion control, development and execution of application programs, communication etc.

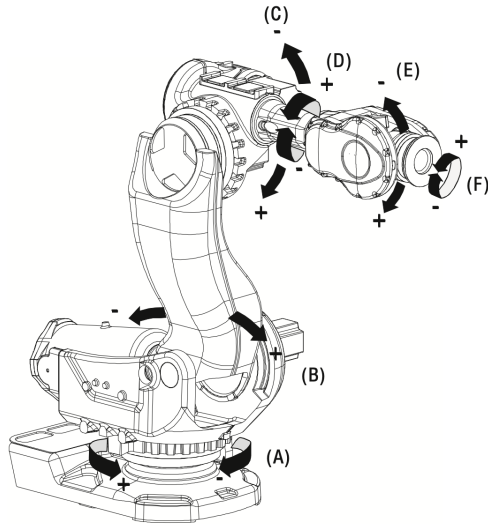


Fig. 2.3: Axis movements [21]

Axis configuration The Figure 2.3 on the left presents the layout of the 6 rotational joints that compose the robot. They are itemized from A to F. The position of all axis in this figure, also corresponds to the calibration position. At This position, the 6 axis are in the zero position. The table below Table 2.1 summarizes the robot motion. It displays the range of movement for the 6 axis, as well as the type of motion of the different axis presented on the Figure 2.3.

Axis	Type of motion	Range of movement	
		from	to
A	Rotation motion	+ 180°	- 180°
B	Arm motion	+ 85°	- 60°
C	Arm motion	+ 60°	- 180°
D	Wrist motion	+ 300°	- 300°
E	Bend motion	+ 100°	- 100°
F	Turn motion	+ 360°	- 360°

Table 2.1: Type of motion and range of movement of the IRB 7600 [21]

RAPID Instructions As the robot is controlled by a computer, a programming language is required. Indeed, to do something, a program script has to be written in a way that the robot's controller knows how to read it. That high level programming language is the RAPID. It uses some English, namely instructions, which are understandable for humans, that are translated into zeros and ones understandable by the robot's controller.

The RAPID program has the most functionality of the other high level programming languages, but it is particularly designed to control robots. It is also developed based on instructions, which enables the robot to move as the 'MoveL', which will be more detail earlier in the Section 3.3.2.

2.2.2 Telehandler

This element plays an important role in the system to reach high heights. Indeed, the arm of the telehandler deploys itself to allow the robot to reach high targets. Nevertheless, the weight of the robot, as discussed previously is very large, around 2,5 Tons. This requires significant resources, in terms of telehandler.

There are a lot of different telehandler brands, as well as a series of different characteristics. Nevertheless, considering the robot to lift, a good choice would be the 'MRT 2470 Privilege Plus' of the Manitou company.



Fig. 2.4: MRT 2470 Privilege Plus [22]

Characteristics The telescopic arm of this telehandler can reach around 25 meters. Moreover, based on its telescopic stabilisers, the telescopic arm can load 2500 kg at this height, which is the weight of the desired robot. The turret rotation is also an advantage to reach less accessible place .

2.2.3 Total station and its equipment

Once the robot is not placed on the ground anymore, a localization system is required in order that the robot know where it is in its environment. This positioning system is achieved thanks to the total station as represented in the Figure 2.5.



Fig. 2.5: Trimble SX10 [23]

The chosen total station is the one available at *IMAX Pro*, which is the Trimble SX10. This optical instrument is widely used in the topography with its scanning technology to scan up to 26,600 points per second [23].

Additionally, one of its functionalities used in the project is the Autolock mode. This mode allows the total station to track the position of the prism, represented on the Figure 2.7, while it is moving. Moreover, based on the station setup, the collected

data by the controller of the total station can be the coordinates of the prism into the space. All this will be explained more in details in the Section 2.3.

In terms of accuracy, in the studied mode, i.e the Tracking mode with a prism, it is about $2\text{mm} + 1.5\text{ppm}$. As the distances in this project are not extreme, the accuracy of the total station is very high. Indeed, as the distance between the total station and the beacon is less than 1 kilometers, the accuracy would be below 3.5 mm. As this distance would be lower than this value, the accuracy is very high.



Fig. 2.6: Controller Trimble TSC7 [24]

Data collector Moreover, the total station, as presented on the Figure 2.5 can be monitored thanks to a controller, which is the TSC7 as shown in Figure 2.6. This transportable computer, the size of a tablet, has an operating Windows 10 system, as well as the Trimble Access field software. This software allows a lot of functionalities to process the collected data from the total station. Furthermore, the controller communicates with the total station thanks to radio waves and it also has a RS232 connection port for fast collection data.

Station setup A functionality is available within this controller of the total station, which is the station setup, which can be viewed as a calibration step. It allows to define a coordinate system in the total station based on a reference line. Theoretically, three points are necessary to create a 3D Cartesian coordinate system, where three mutually perpendicular vectors and an origin are required.

Nevertheless, in this case, the determination of only one axis is required. Indeed, the 'z' axis is already defined in the total station, which is parallel to the gravity force. Then, with the total station, it is possible to determine two points in the space to form a vector, which is the second axis of the coordinate system, established as the 'x' in the controller of the total station. Where the first point of this vector is also the origin of the created coordinate system. Finally, with the right-hand rule, the third axis can be determined and those three axis form the 3D Cartesian coordinate system.

In practice, the only axis to be determined is based on two measurements of the beacon position. Therefore, the beacon has to be placed in those two desired

positions, where the first one is the origin of the created coordinate system. Thanks to the controller of the total station, the two positions are memorized, which enable to create the reference line.

As described before, a prism is required for the positioning system. This one is a 360° prism compatible with the total station Trimble SX10 as shown by the Figure 2.7.



Fig. 2.7: VXS Series 360° Prism [25]

Position measurement The system uses this prism as reflector for the total station. In practice, the total station combines two technologies to determine the position in space of the prism. On the one hand, this optical instrument uses an electronic theodolite for measuring both horizontal and vertical angles. On the other hand, it involves an electronic distance measuring device to determine the length between two points. Those technologies will be detailed further in the following paragraphs.

Those two technologies are interfaced with a data collector. This microprocessor can perform a variety of mathematical operations. Moreover, based on a station set-up, this on-board computer can combine the angles and distance measurements to determine the exact coordinate of the prism in the 3D space.

Theodolites This instrument enables to measure two angles, which are the horizontal and vertical angles. First, the horizontal angle is measured from a reference line, which is considered to be 0°. This reference line is intrinsic to the instrument and is called the meridian. The second angle to be measured is the vertical one. The latter is measured thanks to the reference line, which is the zenith, perpendicular from the horizon.

The theodolite is a telescope mounted on two protractors, one to measure the vertical angle and one to determine the horizontal angle. In the past, those angles were evaluated by reading the graduated circles. Nevertheless, today this is performed electronically. Indeed, the actual digital theodolite automatically reads and records the horizontal and vertical angles.

Then, in combination with the electronic distance measurement, described in the following paragraph, the total station can locate the prism into the 3D space, with respect to the total station.

Electronic distance measurement This is the second fundamental feature of modern surveying Total Stations. Indeed, this measurement device calculates the distance between two points by measuring the phase shift. This phase shift is calculated based on the difference between the sent wave and the one received, which is being reflected through the reflector.

This distance measurement can be performed with a reflector or not. First, the device measures the distance with a reflector based on a infrared or visible laser electromagnetic wave. Or the device measures the distance to a point without reflector based on a visible laser beam. Nevertheless, the reflectorless mode is less accurate than the reflector one.

Moreover, the wavelength λ of the electromagnetic waves are defined by the following equation :

$$\lambda = \frac{c}{f} \quad (2.1)$$

Where c corresponds to the velocity of the electromagnetic waves in its corresponding environment. The c commonly refers to the speed of the light. The speed of the light is affected by the medium through which it passes, but c can be approximated to the speed of the light in the vacuum, which is equal to 299 792,458 km/s.

As mentioned before, the distance between the total station and the reflecting point is measured thanks to the phase different. The Figure 2.8 illustrates this measurement principle. Indeed, the forward path corresponds to the sent wave, which is then reflected. Once the reflected wave is returned to the EDM, the phase shift can be determined, which will enable the calculation of the distance with the reflector.

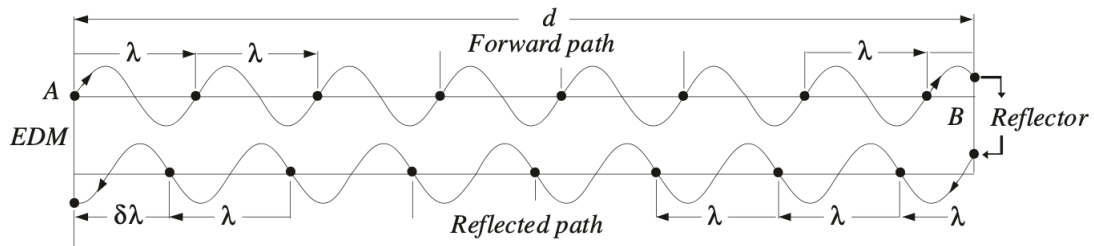


Fig. 2.8: Schematic view of the transmitted and reflected electromagnetic waves[26]

In the mathematics point of view, the two electromagnetic waves can be expressed as :

$$\begin{cases} w_1 = A \sin(\omega t) & \text{Emitted wave} \\ w_2 = A \sin(\omega(t + \delta t)) & \text{Reflected wave} \end{cases} \quad (2.2)$$

In this expression, A is the amplitude of the wave, t is the time, expressed in seconds, ω is the angular velocity of the wave, expressed in rad/s . Then, δt is the time difference between both, which is related to the phase lag $\delta\phi$. In fact, this time lag δt can be rewritten as :

$$\delta t = \frac{\delta\phi}{\omega} = \frac{\delta\phi}{2\pi f} = \frac{\delta\phi \lambda}{2\pi c} \quad (2.3)$$

Furthermore, the total path composed by the forward and reflected path of the electromagnetic wave is determined by :

$$2d = m\lambda + \delta\lambda \quad (2.4)$$

Where d is the distance between the EDM device and the reflector, λ the wavelength and $\delta\lambda$ is the part of the wavelength corresponding to the phase shift. This relation can be rewritten thanks to the previous Equation (2.3), which gives :

$$d = m\frac{\lambda}{2} + \frac{\delta\phi \lambda}{2\pi 2} \quad (2.5)$$

As described above, the distance between the EDM instrument and the reflector is composed of two terms. The first one is composed of an integral number of unit lengths, which is unknown at this stage, and the half of the wavelength. As seen on the Figure 2.8, this term depends on the frequency and can appear more than one time depending on the distance with the reflector. The second one represents the measured phase lag.

Practically, the measurement of the digital phase is highlighted nearby on the Figure 2.9. Indeed, as detail before, the digital phase measurement is based on the comparison of two sinusoidal waves, one emitted and one reflected, as represented on the two first graphs of the figure.

Those sinusoidal waves are triggered square waves, where the amplitude alternates at the same steady frequency as the sinusoidal waves, as shown on the third and fourth graphs.

Then, when square waves alternate an 'electronic gate' is triggered. As described on the fifth graph, the gate opens when the emitted wave begins a new cycle and

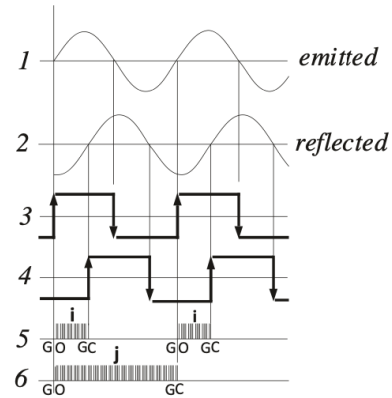


Fig. 2.9: Schematic view of the counting sequences [26]

closes when the square wave of the emitted one also begins a new cycle. Each time the gate is opened, pulses coming from a high frequency oscillator are accumulated in a counter.

Based on those two counters i and j , the second term of the Equation (2.5) can be determined as the ratio : $\frac{\delta\phi}{2\pi} \frac{\lambda}{2} = \frac{i}{j} \frac{\lambda}{2}$. Then, for a given wavelength, the only unknown to measure the distance is the integer m .

This unknown parameter can be deduced in the following way, although there exists different techniques, only one is presented here. First, by modulation of the given fundamental electromagnetic wave of the EDM device, two other electromagnetic waves, with other wavelengths can be created. This leads to three equations for the distance measurement :

$$\begin{cases} d = m_1 \frac{\lambda_1}{2} + \frac{\delta\phi_1}{2\pi} \frac{\lambda_1}{2} \\ d = m_2 \frac{\lambda_2}{2} + \frac{\delta\phi_2}{2\pi} \frac{\lambda_2}{2} \\ d = m_3 \frac{\lambda_3}{2} + \frac{\delta\phi_3}{2\pi} \frac{\lambda_3}{2} \end{cases} \quad (2.6)$$

In this equation, the generated electromagnetic wavelengths are known because they are a multiple of the fundamental unit length of the EDM device. Moreover, the digital phase for the three different wavelengths can be measured. Based on those data, a solution for the three integer parameters m_1 , m_2 and m_3 can be deduced, which yields the distance d .

Finally, the following Figure 2.10 summarizes the way the digital phase is obtained. Firstly, as described before, an electromagnetic wave is emitted in the direction of the reflector. This one is created based on an amplitude modulation. This required an carrier wave, which is modulated by a known waveform. This modulating wave enables to change the amplitude of the carrier wave in order that the modulated wave has the desirable frequency and wavelength.

Furthermore, this carrier wave in the EDM instrument of the modern total station is generally in the visible RED or in the near infrared region of the electromagnetic wavelength spectrum. Indeed, the waves are less affected by atmospheric conditions, have good dispersion properties and good reflection properties in this part of the spectrum.

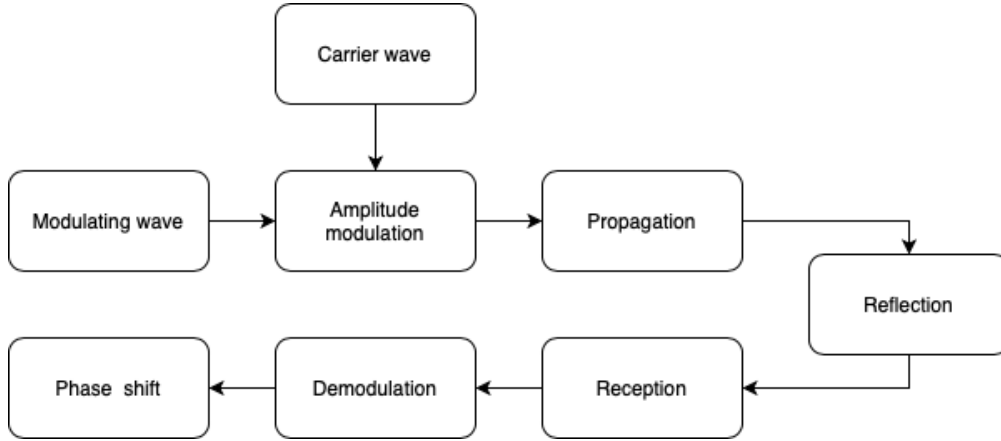


Fig. 2.10: Schematic view of the digital phase measurement for the EDM

Then, the emitted modulated wave reflects on the reflector, which is the prism in this application. This returned wave is then received by the EDM device. This wave is demodulated to extract the phase shift. All these steps are implemented for the three different wavelengths, as detailed by the Equation (2.6). Finally, this is executed to determine the distance between the EDM device, i.e the total station and the reflector, i.e the prism mounted on the basement of the robot.

2.2.4 Inclination sensor

In order to measure the orientation, a different sensor is required, because with the total station, we cannot deduce the orientation of the beacon in the space. Nevertheless, the orientation could be deduced from the technology of the total station, but this will require at least 3 beacons and more than one total station, which leads to high costs.

Different existing technologies could be used to deduce the orientation, but the chosen one is an inclination sensor, because it was available at *IMAX Pro*. This one is the JN2201 from the company 'Ifm electronics', as represented on the Figure 2.11.

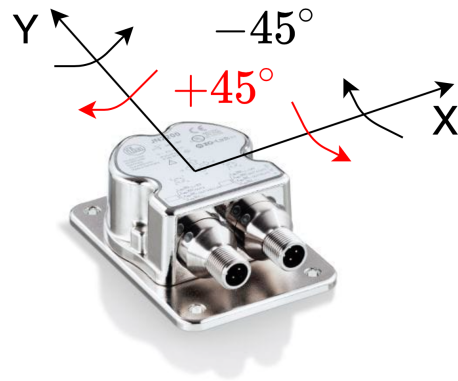


Fig. 2.11: Measurement axis of JN2201 [27]

However, one limitation of the device is that it only measures the orientation variations along two axis. As discussed in more in detail the Section 3.4, the perturbations

along the one axis that is not measured are limited in the real application.

Characteristics The two measurement axis of the sensor, as well as their positive and negative orientation directions, are shown on the Figure 2.11, with an angular range of $\pm 45^\circ$. The angular resolution of this inclination sensor used in industrial applications is 0.01° and the absolute accuracy of the sensor is $\leq 0.01^\circ$. The communication interface is based on the IO-link, which will be describes later on Section 3.2.3. Then, the measuring principle is based on the MEMS capacitive Figure 2.11.

Measuring Angles The inclination sensor measures the orientation angle of an object with respect to the force of the gravity. This measurement principle is based on MEMS capacitive. Indeed, the effect of the gravity on a tiny mass suspended in an elastic support structure is monitored thanks to an accelerometer. When the sensor tilts, the mass will move on account of the gravity, generating a capacitive change between the mass and the supporting structure. Based on this modification, the tilt angle can be measured.

2.3 Localisation and compensation systems

As the robot mounted on the telescopic arm of the telehandler will reach high heights, it has to localise in its environment. Furthermore, some perturbations like the wind or the non-rigidity of the system at those heights can degrade the final accuracy of the robot. Those perturbations are generally difficult to quantify. That is why, a compensation system is also required to counteract those perturbations.

Before looking to the compensation system, the robot has first to localise in its environment. This is the first topic describes in the following Section 2.3.1.

2.3.1 Localization system

In order to perform a task at high heights, the robot has to localise itself in its environment, in order to know where it has to for example machine a wooden plate. This can be achieved thanks to the total station available at *IMAX Pro*, as presented in the Section 2.2.3. As mentioned in the Figure 2.12, the localisation of the robot in its environment is divided into two steps.

As shown by the Figure 2.12, the self-localization is done in two steps. First, the 360° prism is used to initiate a station setup, to create the desired coordinate system, which also corresponds to calibration step. Then, this 360° prism can be

placed on the base of the robot to determine the position of the robot's base with respect to the working zone.

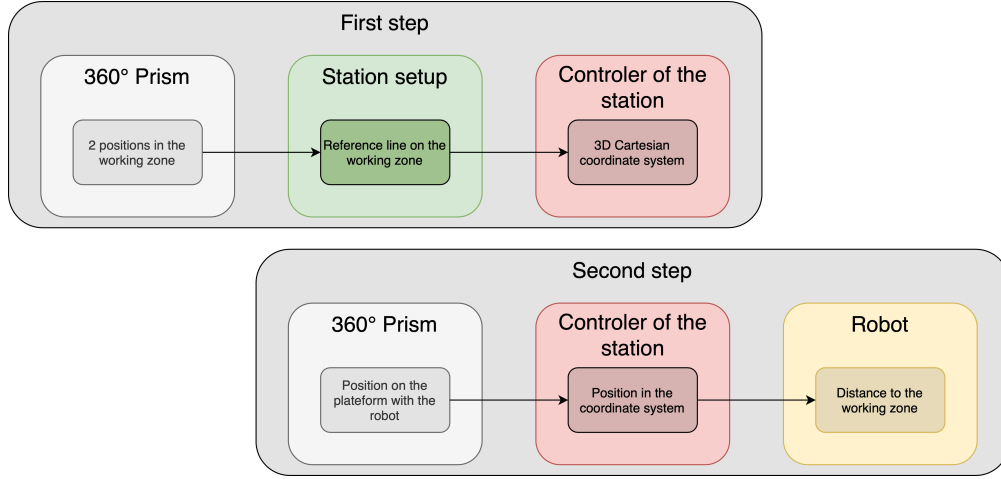


Fig. 2.12: Schematic view of the localisation process

First step As presented in Section 2.2.3, the station setup enables to create a 3D Cartesian coordinate system based on reference line. Therefore, to create this reference line, two positions of the 360° prism are needed. Where the first one is considered as the origin of the created coordinate system. In the real case, the prism is placed for the first measurement at the corner of a edge for example of the wooden plate to machine. While for the second one, the prism is placed at the following corner on the same edge, to create a reference line parallel to the wooden plate to machine.

Thanks to those two measurements on the working zone, a reference line is created, which is the x axis of the created coordinate system. It enables the creation of a 3D Cartesian coordinate system, as the determination of the other axis are deduced from the first one, as explained in the Section 2.2.3.

Second step As previously described, a coordinate system can be created thanks to the controller of the total station and then the communicated data can be the coordinate positions of the prism into this system. But at this level, the robot can not self-localise in its environment. Indeed, its reference system has to be coordinated with the created 3D Cartesian coordinate system.

Therefore, once the coordinate system is created, the 360° prism is then fixed on the platform. Where, the distance between this prism and the base of the robot

is known. Then, based on this distance combined with the coordinate position of the prism with respect to the working zone, the distance between the base of the robot and the working zone can be calculated. Therefore, the robot knows the distance between its end-effector and the origin of the created 3D Cartesian coordinate system. Therefore, the robot can localise it, with respect to the working zone.

Orientation The previous part of the section explains how the localisation in terms of position is executed. In terms of the orientation, this is due thanks to the calibration position of the robot. Indeed, before the beginning of the task, the different axis of the robot are placed in their reference positions with is the one represented in the Figure 2.3. Therefore, the system is calibrated in orientation, because each axis refers to the zero position.

2.3.2 Compensation system of the position and orientation

In terms of the compensation system used to counteract the perturbations, this one is represented in the Figure 2.13 below.

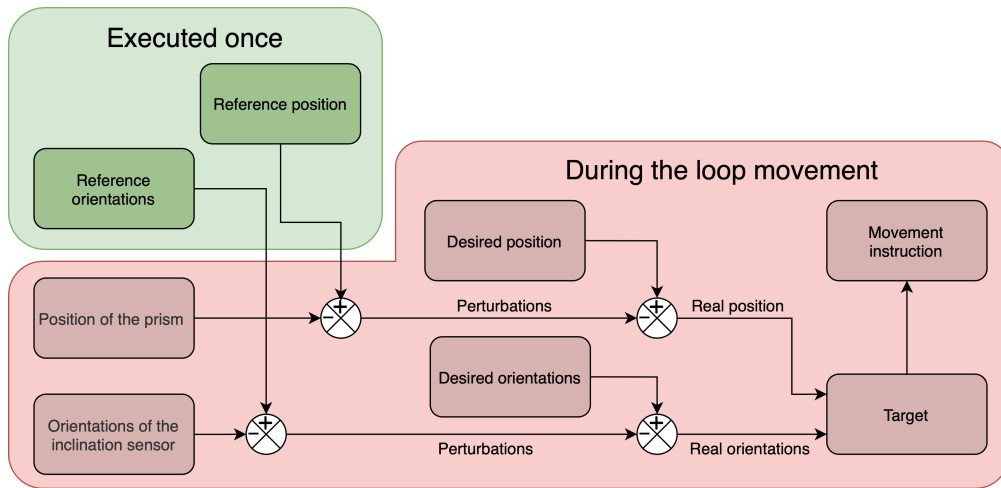


Fig. 2.13: Schematic view of the compensation system

As described in Figure 2.13, the compensation system is split into two parts. There is the orientation part and the position part. Indeed, as the perturbations can be acting according to the 3 degrees of translation and the 3 degrees of rotation, two devices are required. One is collecting the data for the position compensation, when the other is measuring the orientation angles. Moreover, there is also a part which is executed once, while the other is executed at each step during the loop movement.

Executed once In order to compensate the perturbations, the measured data has to be compared to a reference. For the position part, it is assumed that a station setup has been performed, which as discussed in Section 2.2.3 enables to create a 3D Cartesian coordinate system. Then, before the beginning of the movement or the task of the robot, the coordinate of the position of the prism is taken as a reference.

For the orientation part, it is the same, the measured data of the inclination sensor are compared to the reference angles. In this case, the reference angles would usually be zero, because for a good stability, the platform has to be parallel to the ground.

During the loop movement Once, the loop movement begins the collected data from the controller of the total station and the inclination sensor are compared to the reference one. For the position, the current position of the prism is compared to the reference coordinate, which determines the perturbations. For the orientation, the current orientations of the inclinations sensor are collected and compared with the reference ones.

After the determination of the perturbations, those ones are compared with the desired position and orientations of the end-effector, which lead to the real position and orientations of the robot. Indeed, if for example a perturbation of 10 cm is applied in the right direction, then to counteract those ones, the position of the robot will shift from 10 cm to the left.

Then, the real position and orientations of the end-effector take part of a 'robtarguet', i.e. the desired target the robot has to reach. Then, this target is reached thanks to a movement instruction, which will be described more in detail in the Section 3.3.

Finally, the entire movement is discretized in order to achieve to a correction of the position and orientation between each incremental movement, as describes before.

Summary of the chapter

At high heights, the robot placed on the telescopic arm of the telehandler is submitted to different perturbations. Those ones can be measured thanks to the total station and the inclination sensor. Then, based on those data, the position and orientations of the end-effector can be adjusted to counter the perturbations.

3

Position and orientation controller

This chapter will be dedicated to the position and orientation control to counteract the perturbations. It will begin with a schematic view of the compensation system. Then, a communication protocol has been developed to received the information from the inclination sensor and the controller of the total station. Followed by a description of the movement instructions sent to the robot's controller. To conclude with a brief word about the fact that the inclination sensor only returns two angles.

3.1 From measured data to movement instructions

Once the communication has been established with the inclination sensor and the controller of the total station, which will be the topic of the following Section 3.2, information is collected by the server application. As described earlier, the base of the robot suffers from external perturbations which have to be compensated in order to have a good accuracy of the end-effector.

In practice, based on the setup of the total station, the information given by the controller of the total station can be in the form of coordinate (X, Y and Z). Then, with respect to the first data, which serves as a reference position, the position perturbations along the three axis can be measured, as described in the Section 2.3.2.

Then, those measured perturbations are subtracted from the desired positions along the three axis of the robot. If the coordinate system defined by the station setup and the coordinate system of the robot are aligned. Otherwise, a transformation matrix has to be determined to have the link between the two.

In the real case, the controller of the total station sends the value of the position

of the prism, which is compared to the reference one to evaluate the perturbations in this case along the Y axis, as shown by Figure 3.1. Once the perturbations are known, they are subtracted from the desired position of the end-effector, if the coordinate system of the robot is aligned with the one of the total station. If not, a rotation matrix has to be applied.

The same process is used to compensate the perturbations along the orientation. However, the data sent by the inclination sensor are Euler angles and the robot uses quaternion for the orientation. Therefore, a conversion between both has to be performed, which is detailed in the Section 3.3.3.

Once the desired position and orientations have been compensated, they are sent to the robot through a movement instruction. This enables the robot to execute a movement to reach the desired position and orientations, which will be explained in the Section 3.3.

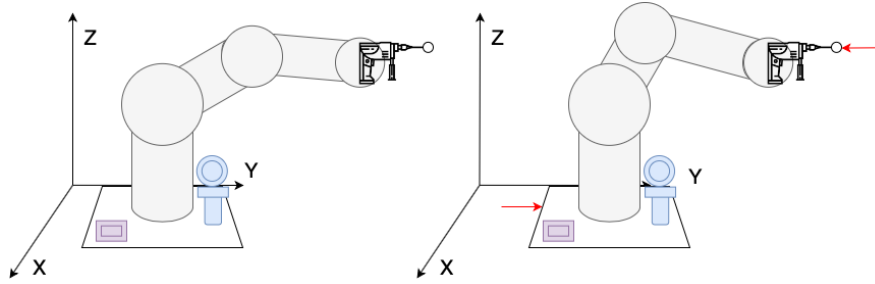


Fig. 3.1: Schematic view of the compensation of the position along one axis

3.2 Data transfer between the elements

In order to collect the data to localise and compensate the positions and the orientations of the end-effector, a communication protocol has been developed. This one enables the different devices to communicate between them.

The first subsection will briefly introduce all the elements and how they are connected. Then, the second subsection will detail the TCP/IP communication protocol to receive the data from the total station. While the third subsection will focus on the Modbus communication protocol to collect the information coming from the inclination sensor. Finally, a brief introduction of the following section on the communication with the robot's controller will be made.

3.2.1 Overview

As explained previously, the Figure 3.2 details the related devices and how they are connected to achieve to the communication of the data required for the localisation and compensation systems.

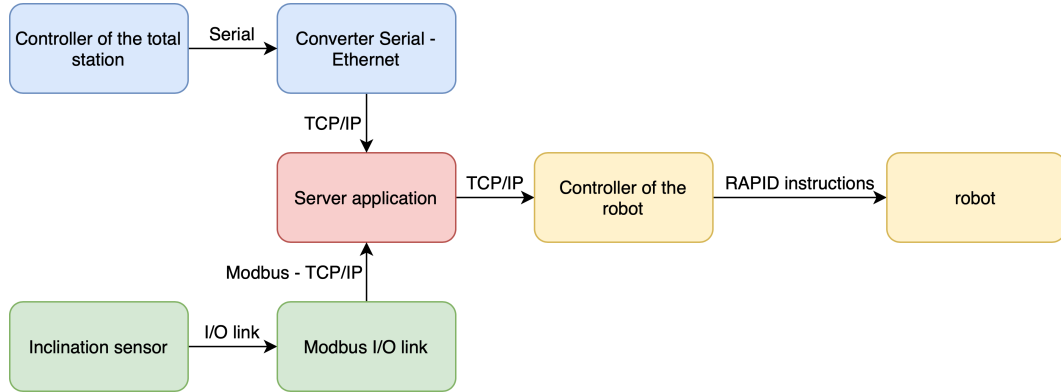


Fig. 3.2: Overview of the communication between the different devices

First, the blue part enables the server application to receive the position information of the prism coming from the controller of the total station. This is done through a converter that transforms the Serial Communication Protocol into a TCP/IP communication protocol. This converter allows all the devices to communicate with the same protocol and the same port, the Ethernet port, which is easier.

Secondly, the green part represents the communication of the information delivered from the inclination sensor. This is performed through another device, which is a I/O link master with a Modbus TCP/IP interface. It enables to receive the data from the inclination sensor, which has a I/O link communication interface. Then, this I/O link master communicates with the server application through the Modbus TCP/IP communication, which enables both to communicate via the Ethernet network, as for the position information part.

Once the different data are received by the server application, it processes them. Then, the modified position is sent to the controller of the robot as a movement instruction, which contains a 'robtarg', with the desired position. More details are provided in the Section 3.3.1. That information is then sent to the controller via the TCP/IP protocol.

Finally, the robot receives the data via the RAPID instructions, which is a high level programming language understood by the robot's controller. Those instructions can be executed to reach the desired position with the end-effector.

3.2.2 Total station - Server application

As presented in the introduction's chapter, a communication protocol has been developed in order to recuperate the data from the controller of the total station, which is developed through this subsection. This section will describe the communication protocol, as well as its implementation in the real case and the interpretation in lines of code.

TCP/IP Description Let's begin with a description of how the TCP/IP communication protocol, which is used to receive the information coming from the converter, works. Firstly, TCP stands for Transmission Control Protocol. It is a communication standard that allows application programs and computing devices to exchange messages over a network. It is largely used in the digital network communications and ensures end-to-end data delivery.

This communication protocol organises the data so that the information can be transmitted between a server and a client. To guarantee the integrity of the data communicated on the network, TCP/IP establishes a connection between the source and the destination before the transfer begins. Indeed, each device has its own IP address, which enables it to communicate and exchange data with other devices.

TCP and IP are two separate protocols that work together to ensure data transmission to the desired destination within a network. IP obtains and defines the address of the device to which the data has to be sent. While the TCP protocol is then responsible to transport the data through the network and ensuring that it is delivered to the destination with the desired IP address [28].

Connection and data stream Figure 3.3 summarises how the connection between the server and the desired client is realised in practice, as well as how the data is transferred between both.

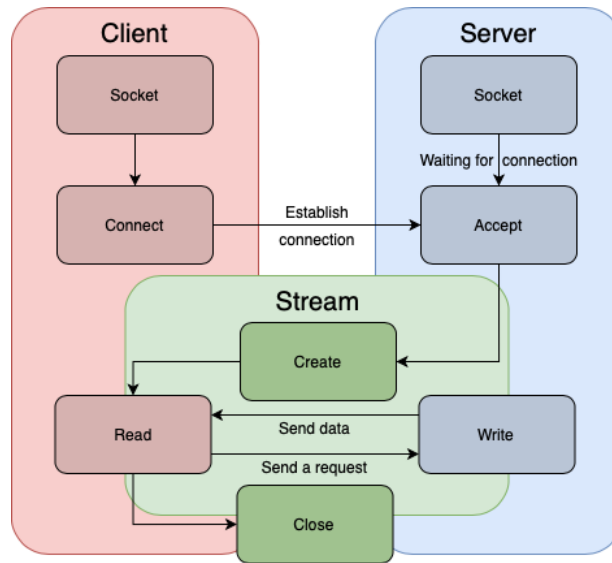


Fig. 3.3: Schematic view of the communication between the controller of the total station and the server application thanks Serial to Ethernet converter

Firstly, the server creates a socket, which is a network structure that serves as an endpoint for receiving and sending data across the network. This socket is identified to others with its socket address, which is based on its IP address and its port number. Once it is created, the server is waiting for a connection with another device.

After that, the client also creates a socket in the same way as the server, with its own IP address and port number. Once it is performed, the client establishes a connection with the server, allowing to create a network stream for the data transmission. Thanks to this network stream, the client can send a request to the server and in return, the server sends the data to the client. Once the data transmission is finished, the stream is closed.

Implementation in lines of code The TCP/IP protocol enables to receive the information from the controller of the total station, but for it to work, it has to be implemented in lines of code into the server application.

In the case of the project, the server application is not directly connected to the controller of the total station, as discussed in the 3.2.1. Indeed, a Serial to Ethernet convert is used, because the controller sends the data over the Serial Communication Protocol and to receive it in a TCP/IP protocol a converter is needed. Then, in this layout, the server is the converter, which recuperates the

data of the controller and sends it to the client, which is the server application. In terms of lines of code, only the implementation of the client is performed, because the converter manages itself the server part.

This begins with the initialization of the TCP client in the server application with the IP address and the port number of the server, i.e. the converter, to establish the connection, as shown by Figure 3.3.

```
TcpClient client = new TcpClient(ip, port); // Create a new instance of the class TcpClient with its Ip address and port
```

Fig. 3.4: Initialization of the TCP client

After, the creation of this TCP client, a network stream is required to transfer the data, as shown on the Figure 3.3. This is realised thanks to the function 'GetStream', based on the TCP client class just initiated. This network stream is created based on the IP address and the port number of the server to communicate with it.

```
NetworkStream stream = client.GetStream(); // Create a stream with the TCP communication protocol to receive and send data
```

Fig. 3.5: Initialization of the network stream

Once the network stream is created, the data is sent by the server and the client can read them via the function 'Read'. After, the bytes collected by this function are changed into the ASCII representation in order to have the right representation of the information.

```
Byte[] data = new Byte[256];  
String responseData = String.Empty;  
  
Int32 bytes = stream.Read(data, 0, data.Length); // Collect the data on the stream  
responseData = System.Text.Encoding.ASCII.GetString(data, 0, bytes); // Change the bytes collected on the stream into ASCII representation  
Console.WriteLine("Received: {0}", responseData);
```

Fig. 3.6: Data received from the server

When the server application no longer uses the data coming from the controller of the total station, the stream with the convert is closed simply by the function 'Close'.

Configuration of the output data In the side of the server, it is the converter that managed the sent data. Nevertheless, the parameters that characterize its communication protocol, like the baud rate, data size, the parity or

the stop bits can be modified. The output data coming from the controller of the total station can also be managed, as the communication format. Moreover, another functionality is also available on the controller of the total station, which is the configuration of the output data, which is describes in this paragraph.

Indeed, the table on the right represents all the possible data that the controller of the total station can send, through its RS232 connector in this case. The labels from 7 to 11 represent the angles and distances computed by the total station. Then, the labels from 37 to 39 are related respectively to the coordinate X, Y and Z of the prism position with respect to the total station. Before the system can output those values, a station setup is required, as described in Section 2.3.1. Finally, the two last labels are not useful for the application.

Label	Text	Description
7	HA	Horizontal Angle
8	VA	Vertical Angle
9	SD	Slope Distance
10	VD	Verticla Distance
11	HD	Horizontal Distance
37	N	Northing
38	E	Easting
39	ELE	Elevation
51	Date	Date
52	Time	Time

Table 3.1: Available output data [29]

Moreover, one thing also to mention is that with the Autolock mode, as described before in the Section 2.2.3, the total station has to be locked on a prism, in order for the data to be sent. Otherwise, no data are sent by the controller, what blocks the server application.

Furthermore, it also possible to configure the bandwidth of the transmission via the functionality of the controller. Indeed, the baud rate, the parity, the stop bit, as well as the data bits can be modify, as for the converter. Nevertheless, these modifications have to match with the ones of the Serial to Ethernet convert, which can also change this information. Otherwise, the data are not collected correctly in the side of the server application.

3.2.3 Inclination sensor - Server application

This section will deal with the orientation part and how the angles of the inclination sensor are collected by the server application to process them. Indeed, as for the controller of the total station, a communication protocol has been established to recuperate the data from this sensor.

Modbus TCP/IP Description As detailed in Section 3.2.1, Modbus TCP/IP is the protocol used to received the information from the inclination sensor. It is a messaging structure developed to establish communication between devices. Modbus TCP/IP is, as the TCP/IP protocol, configured as a Master-Slave or Client-Server communication.

Modbus is a widely used industrial network protocol in the industrial manufacturing environment. It has been implemented to transfer discrete/analog I/O and record data between control devices. A communication with the protocol Modbus is always initiated by the master node to the slave node. The latter one will never transmit data without receiving a request from the master node [30].

Connection and data stream The Figure 3.7 summarizes how the communication between the server application and the I/O link master is established and how the information of the inclination sensor can be collected by the server application.

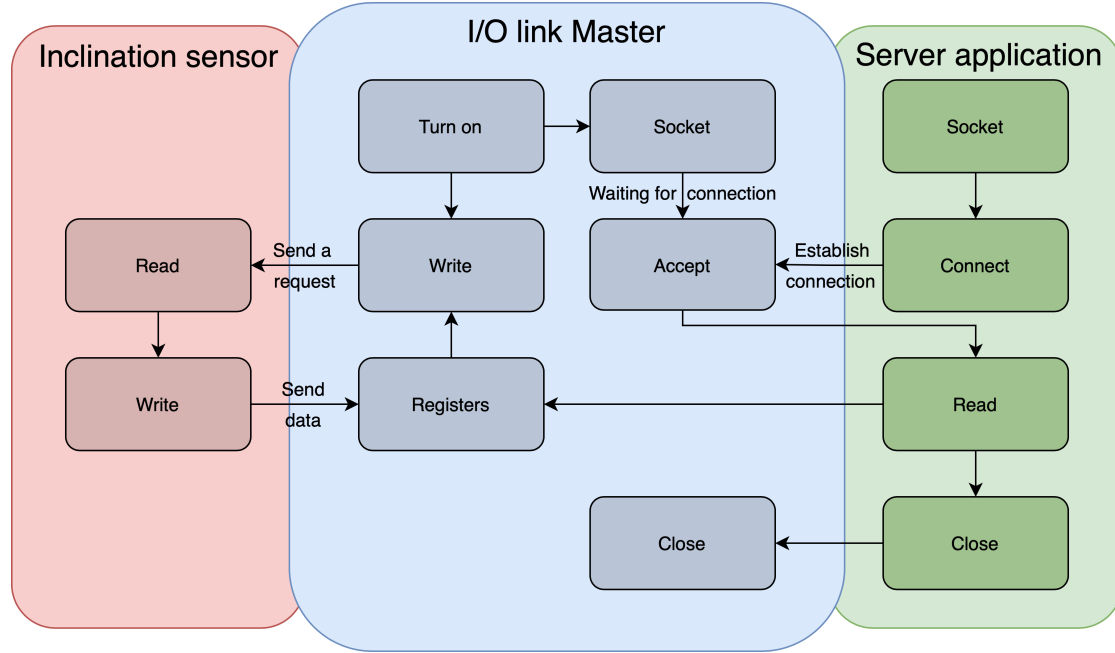


Fig. 3.7: Schematic view of the communication between the inclination sensor and the server application thanks to the I/O link master

Firstly, there is the I/O link master, which allows the transition of the data between the inclination sensor and the server application. The data coming from the inclination sensor is as the form of digital output, while the output data sent to the server application are based on the Modbus TCP/IP protocol.

Once the I/O link is turned on, it creates a socket and waits until another device wants to connect. It then sends a request to the inclination sensor and in return the latter one sends the values of its inclination angles through its digital output. Collected by the I/O link master, those values are written into its register, which store the information.

Then, in the side of the server application, it also creates a socket based on the IP address of the I/O link master and its port number. Indeed, in the Modbus TCP/IP protocol, the communication of the data is realised in the same way as the one with the controller of the total station. The TCP, or Transport Communication Protocol, ensures that data is received and sent correctly using an Ethernet network. While the IP address is the delivered address, as evoked in the previous Section 3.2.2.

After that, a connection request is sent to establish a connection between the server application and the I/O link master. Once it is performed, the server application can read the values of the angles into the corresponding registers, with the appropriated number. Finally, the communication is closed, which closes the link between both.

Implementation in lines of code This paragraph will cover how this is translated in terms of lines of code. First, a Modbus client is created, as described in Figure 3.8, based on the IP address of the I/O link master and its port number. Then, this client is connected to I/O link master with the function 'Connect'. One thing to mention is that this function only gives the hand back to the code if the connection has been established. Thereby, if the code is running a connection has been established with the I/O link master and then the data can be recuperated.

```
ModbusClient modbusClient = new ModbusClient(ip, port); // Create a new instance of the class ModbusClient with its Ip address and port
modbusClient.Connect(); // Connection to the Modbus
```

Fig. 3.8: Initialization of the Modbus client

Then, as evoked earlier in the subsection, the angles measured by the inclination sensor are written into the registers of the I/O link master. Thanks to the function 'ReadInputRegisters' available with the Modbus client class, the server application can read the values of the desired registers, passing its referencing value into the arguments of the function. In practice, the value of the X orientation of the inclination sensor is registered into the number 200 and the Y value is written into the number 201.

```
int[] readInputRegisterX = modbusClient.ReadInputRegisters(281, 1); // Read the Input Register number 281, where the value of the X angle of the inclination sensor is
int[] readInputRegisterY = modbusClient.ReadInputRegisters(280, 1); // Read the Input Register number 280, where the value of the Y angle of the inclination sensor is
```

Fig. 3.9: Data reads on the I/O link master

Finally, the communication between the I/O link master and the server application is simply closed via the function 'Close' of the client.

```
modbusClient.Disconnect(); //Disconnect from the Modbus communication
```

Fig. 3.10: Disconnection with the I/O link master

3.2.4 Server application - Robot

The data collected from the inclination sensor and the controller of the total station are processed into the server application and then sent to the robot. This data is sent to the robot as a movement instruction, which contains the targets that the robot has to reach. One target contains the final position and orientation of the end-effector. This will be further elaborated in Section 3.3.1.

As shown in Figure 3.11, first, the server application communicates with the controller of the robot via a TCP/IP protocol as described in the previous subsections. Indeed, based on the IP address and the port number, the server application can send the data to the controller. This data is sent as movement instructions into a string format.

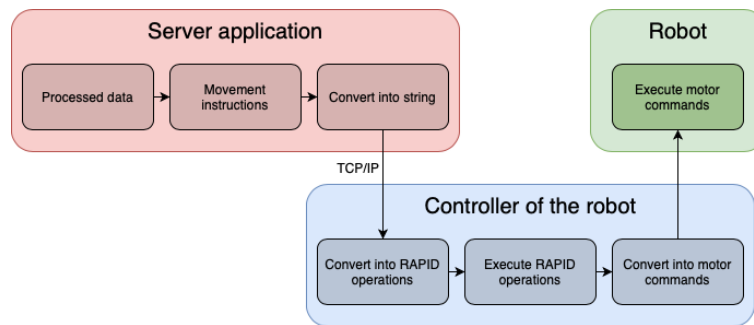


Fig. 3.11: Schematic view from the processed data until the motor joint executions

Then, the robot's controller receives it and changes it into RAPID instructions, which can be interpreted by the controller. Based on those instructions, the controller can send the correct joint motor commands to the robot, to reach the desired position and orientation for the end-effector.

3.3 Information sent to the robot's controller

As shown in the Figure 3.11, the data is processed in the server application and then sent to the controller of the robot. The information of position and orientation is sent into a movement instruction, which contains the desired position and orientation of the end-effector, described as a 'robtarger'.

The two following subsections will describe, how this 'robtarger' and the movement instruction are implemented into the RAPID instructions. Finally, a subsection will provide the definition of a quaternion, which is used to represent the spatial orientations of the end-effector.

3.3.1 Robtargets

'Robtargets' or robot targets are data types used to define the position of the robot and external axis in the RAPID instructions. They are used to define the position and orientation in the movement instructions to which the robot and external axis have to move. The different components that compose the 'robtarger' are the translation, the rotation, the robot configuration and the external axis, which are described below.

Translation Robot targets have a position in space, which is represented by the translation. This refers to the position (X, Y and Z) of the tool center point, named as 'TCP'. Its data type is 'pos', which contains the 3 values of the position expressed in numeric values. Another thing to mention is that this position is specified in relation to the current object coordinate system. If no work object is specified then this is the world coordinate system [31].

Rotation The robot target also has an orientation in space. This is described by the rotation, which contains the orientation of the tool, expressed in the form of a quaternion. The latter one is composed of 4 elements, q_1 , q_2 , q_3 and q_4 , which are expressed in numerical values. As well as the translation, this quaternion is specified in relation to the current object coordinate system [31]. Its mathematical description will be explained more in detail in the Section 3.3.3.

Robot configuration When calculating the corresponding axis positions, there will often be two or more possible configurations. Indeed, for the same position and orientation of the tool, the robot can have several configurations of the robot's axis.

To distinguish those possible configurations, the robot configuration is specified using four axis values. For a rotating axis, which is the case for all the joints of the robot, the value defines the current quadrant of the robot axis. Those quadrants are numbered 0, 1, 2 and 3 and they can also be negative from -4 to -1, as modeled by the Figure 3.12. Each number is connected to the current joint angle of the axis. For example, the quadrant 0 is the first quarter revolution, 0 to 90°, in a positive direction from the zero position etc.

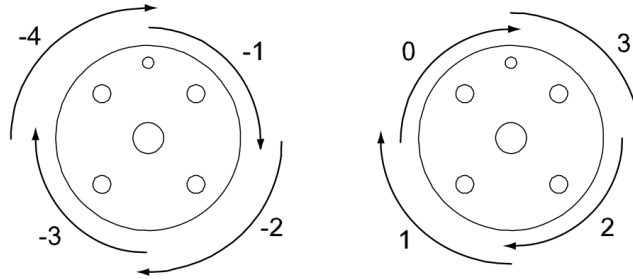


Fig. 3.12: Example of the possible configuration quadrants for the axis 6 [31]

The robot configuration is expressed in terms of 4 values, which are cf1, cf4, cf6 and cfx. For the first three, it corresponds to the current quadrant of the joint axis 1, 4 or 6, expressed as a positive or negative integer, as defined in ???. The value cfx differs from one type of robot to another, in our case, the current quadrant of axis 2 is used [31]. In summary, those four numbers allow to chose from one configuration to another.

Example The Fig. 3.13 shows a basic example of a 'robtargt' implementation in the RAPID operations for the real case.

```
CONST robtarget p15 := [ [600, 500, 225.3], [1, 0, 0, 0], [1, 1,
0, 0];
```

Fig. 3.13: Example of the data type 'robtargt' [31]

The first three numbers are the position of the tool, the translation part, which will be used to modify the position of the TCP in the space. Then, the following four numbers are the quaternions, which describe the orientation of the tool. The last four numbers into brackets represent the robot configuration.

3.3.2 Movement instructions

As described before, the 'robtargets' are used to set the position and orientation of the desired movement. This section will detail more on how those movements instructions are composed.

There are different types of movement. For instance, the joint movement or 'MoveJ', which describes a movement of the robot from one point to another, without forming a straight line. In this movement instruction, all axis reach the destination position at the same time.

Another type is the linear movement instruction or 'MoveL', which describes a linear movement of the tool center point (TCP) to a given destination. When the TCP is to remain stationary then this instruction can also be used to reorient the tool [31].

In fact, this movement is composed by five principal arguments, which are the robtarget, the speed, the zone, the tool and finally the work object, as detailed hereafter.

Robtarget The 'robtargets' or robot target has been explained in the previous Section 3.3.1. It is used to define the position and orientation of the robot and its configuration.

Speed The speed is represented by the data type 'speeddata', which sets the velocity of the tool center point, the tool reorientation, and the possible external axis. It is expressed in mm/s for the speed of the TCP and for linear external axis and in degrees/s for reorientation of the tool and rotating external axis [31].

Zone Then, the zone is used to specify how close to the programmed position the axis must be before moving towards to the next position. Indeed, the programmed position can be terminated either in the form of a stop point or a fly-by point.

On the one hand, the stop point signifies that the robot and the external axis must reach the specified position before that the program execution continues with the next instruction.

On the other hand, the fly-by point denotes that the programmed position is never attained. Instead, the direction of motion is changed before the position is reached. Thereby, the data type 'zonedata' is used to describes the size of the generated corner path, as shown on the Figure 3.14. Other parameters can be used

to characterize this generated corner path but they will not be used during this project [31].

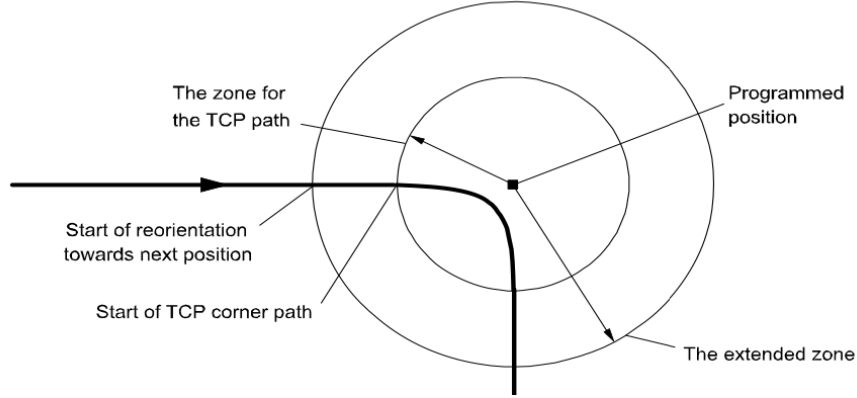


Fig. 3.14: Schematic view of the zone for the TCP path [31]

Tool Moreover, the 'tooldata' is used to specify the characteristics of a tool, which can be mounted on the end-effector of the robot. Different parameters can be used to describe this tool.

First, the 'tool frame', which refers to the tool coordinate system. This data type is composed by the position of the TCP (X , Y and Z) expressed in the wrist coordinate system, as shown on the Figure 3.15. As well as, the orientation of the tool coordinate system, expressed in the wrist coordinate system as a quaternion.

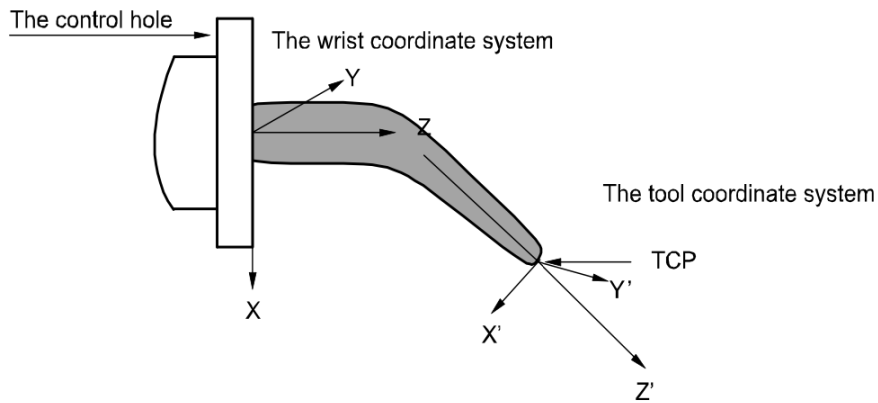


Fig. 3.15: Schematic view of the tool coordinate system [31]

Moreover, the 'tool load' is also specified, which is composed of the weight of the tool, the center of gravity of the tool load and the moments of inertia of the tool relative to its center of mass[31].

In the real application, a tool has been developed, as it would be presented more in detail in the Section 4.1. This tool has been characterise by a translation, which enables the robot to know that the position of the TCP has moved from the measured distance of the tool.

Work object Another useful parameter in the 'MoveL' instruction is the work object. This is used to describe the work object that the robot welds, processes etc. Therefore, some data are related to this parameter to characterise this work object.

For instance, the user coordinate system can be specified, which refers to the position of the current work surface or fixture. As displayed in the Figure 3.16, the object coordinate system can be indicated, which refers to the position of the current work object. It is defined in the user coordinate system, while the latter is defined in the world coordinate system.

Both of them are composed of the position of the origin of the coordinate system and likewise the orientation of the coordinate system is expressed as a quaternion.

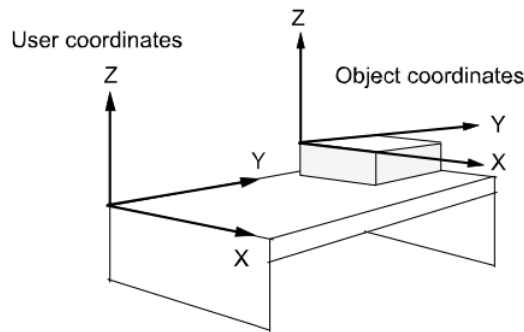


Fig. 3.16: Schematic view of the object coordinate system defined in the user coordinate system[31]

In the real application, as also developed more in detail in the Section 4.1, the work object is used to define the wooden board where the robot will execute the test. Moreover, the object coordinates is also used to reference the sheet of paper involved in the testing part.

Example The Figure 3.17 shows a basic example of a linear movement with the instruction 'MoveL'. Here, 'p1' refers to the position to reach, thus the 'robtaraget'. Then, the 'v1000' is the speed at which the target is reached, with the TCP tool 'tool2'. Here, no work object are specified, then the position of the 'robtaraget' is related to the world coordinate system. Finally, 'z30' refers to the zone data, which describes the size of the generated corner path, in case of a continuous movement with movement instructions.

```
MoveL p1, v1000, z30, tool2;
```

Fig. 3.17: Example of the instruction 'MoveL' [31]

3.3.3 Quaternions

As mentioned before, quaternions are used instead of Euler angles in the RAPID instructions, which are used by ABB to define the tool workplane orientation, in the 'robtaraget', as presented in Section 3.3.1. Like the Euler angles, the quaternions are representing spatial orientations of elements in three dimensional space. Therefore, a conversion is required to change the Euler angles coming from the inclination sensor into the appropriate quaternion.

In this subsection only the matrix allowing to create the quaternion representation based on the Euler angles is provided. All the mathematical notations and developments are available in the Appendix A.

Therefore, the matrix in the Equation (A.11) represents a quaternion, which is represented by fourth elements, which correspond to the fourth lines of the matrix :

$$q_E = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right) \cos\left(\frac{\beta}{2}\right) \cos\left(\frac{\gamma}{2}\right) + \sin\left(\frac{\alpha}{2}\right) \sin\left(\frac{\beta}{2}\right) \sin\left(\frac{\gamma}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right) \cos\left(\frac{\beta}{2}\right) \cos\left(\frac{\gamma}{2}\right) - \cos\left(\frac{\alpha}{2}\right) \sin\left(\frac{\beta}{2}\right) \sin\left(\frac{\gamma}{2}\right) \\ \cos\left(\frac{\alpha}{2}\right) \sin\left(\frac{\beta}{2}\right) \cos\left(\frac{\gamma}{2}\right) + \sin\left(\frac{\alpha}{2}\right) \cos\left(\frac{\beta}{2}\right) \sin\left(\frac{\gamma}{2}\right) \\ \cos\left(\frac{\alpha}{2}\right) \cos\left(\frac{\beta}{2}\right) \sin\left(\frac{\gamma}{2}\right) - \sin\left(\frac{\alpha}{2}\right) \sin\left(\frac{\beta}{2}\right) \cos\left(\frac{\gamma}{2}\right) \end{bmatrix} \quad (3.1)$$

This relation allows to calculate any quaternions based on the three Euler angles, which can represented any orientations in the 3D space. This is used in particular to convert the angles received by the inclination sensor into the appropriate quaternion, this will be discussed later on the Section 4.1.

3.4 Limitations of the inclination sensor

Unfortunately, as presented in the Section 2.2.4, the inclination sensor returns only two angles, which are the orientation angles of the axis coplanar of the device. In addition, angular range of those measurements are limited to $+ 45^\circ$ to $- 45^\circ$. Therefore, those two factors are restricting the orientation perturbation measurements.

Those two limitations due to the constraints of the device are in practice not an issue. Indeed, in the real application, the inclination sensor would be placed on the base of the robot to measure the orientation perturbations applied on it. In this configuration, those perturbations would not exceed the limitation range of the inclination sensor. Otherwise, the whole system would break down.

Furthermore, the third and not measured axis is the one perpendicular to the robot base, i.e. in the opposite direction of the gravity. In the real application, the orientation perturbations along this axis are limited. Nevertheless, to be as complete as possible, the perturbations around this third should be measured in order to perform a complete position and orientation compensation of the disturbances.

Summary of the chapter

The communication protocol takes an important part of the compensation process. Indeed, it enables all the devices to communicate with each other. This allows the server application to receive the orientation of the inclination sensor and the position of the beacon.

Those combined data are used to modify the target of the desired position and orientation of the robot, in order to counteract the possible disturbances applied on the system. Once the target has been modified, it can be sent to the robot through a movement instruction to reach the desired position and orientation of the tool center point.

4

Validation campaign

This section will be dedicated to the validation campaign of the application. As the system has not been tested with a telehandler, the testing setup has been redesigned, which is the topic of the first section. Then, the related software implementation will be detailed, with the corresponding implemented functions. Finally, the different results of executed tests will be presented.

4.1 Testing setup

As evoked earlier, a testing setup has been redesigned in order to validate the compensation system of the position and orientation. This section will describe how this setup is composed, as well as the different tests that will be performed. Finally, this will bring to the following section dedicated to the software implementation of those tests.

The testing validation will be split into two different tests. The first one allows the robot to follow the trajectory of the beacon. Thanks to the pen mounted on the end of the robot, the latter one will draw on a sheet of paper the trajectory of the beacon.

For the second test, a pen will always be mounted at the end of the robot, but the latter one will not draw the trajectory of the beacon but a square. The robot will not simply draw a square, because random perturbations will be applied on the wooden board where the sheet of paper is located.

Moreover, this second test will be divided into three part. For the first one, only position perturbations will be applied on the system. Then, for the second one only orientation perturbations will be generated. While for the last one, the

wooden board will suffer from position and orientation perturbations. All those tests will be described in the rest of this section.

4.1.1 Follow the beacon

As introduced, the first test will examine the capacity of the robot to follow a desired trajectory created with the 360° prism, as presented by the Figure 2.7. Indeed, thanks to the pen mounted at the end of the robot, the robot will draw the trajectory of the beacon on a sheet of paper attached to a wooden board. While the trajectory of the beacon will also be draw on a sheet of paper thanks to the pen located on the bottom of the beacon. All the components involved in this test are represented in the Figure 4.1.

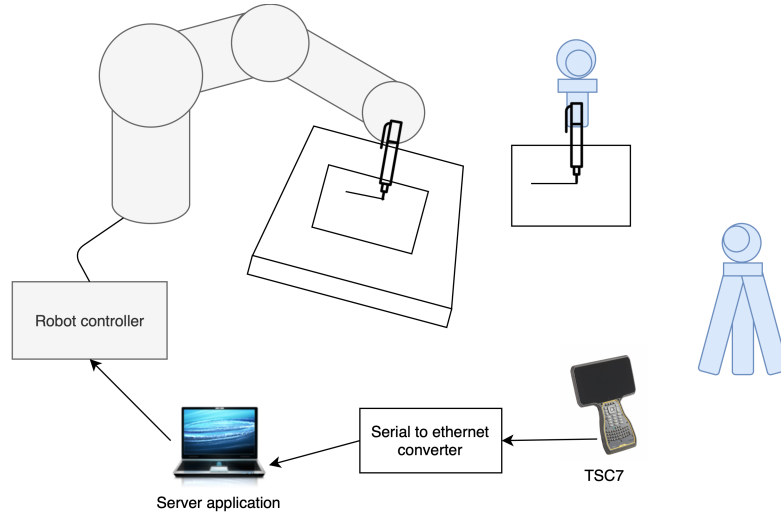


Fig. 4.1: Overview of the different elements constituting the drawing trajectory test

Thanks to the total station, the position of the beacon can be deduced. Indeed, if a station setup has been performed before the beginning of the drawing movement, the controller of the total station can return the coordinates position of the beacon. As explained in the Section 2.2.3, the station setup enables to create a 3D Cartesian coordinate system. Then, the returned values of the controller of the total station are the coordinates position of the beacon in this system.

Therefore, just before the beginning of the drawing movement, the coordinates of the beacon are taken as the reference. Then, the following coordinates sent by the controller of the total station are compared to the reference ones. The result is then sent to the desired position of the robot, which is fixed and is the center of the robot.

After, a movement instruction is executed to reach the position. Once this position is reached, the data coming from the controller are collected again and compared to the reference coordinates and an second movement instruction is created to reach the second desired position. This continues until the end of the trajectory established by the beacon. It results to the same trajectory on the sheet of paper as the one made by the beacon.

The rest of this subsection will be dedicated to the description of the real elements used for the test. As well as a view of those different elements.

Designed tool First and foremost, to be able to draw, a tool to maintain the pen has been developed. This tool as shown by the Figure 4.2 is composed of two copper tubes nested one inside the other. The external one is fixed. While the other can move in the first one and is retained by a stopper.

Moreover, the space at the top of the two cylinders allows a safety margin of a few centimeters. Indeed, if the robot presses on the sheet of paper too much, the internal cylinder goes up, which avoids breaking the marker.

In terms of the marker, this one is fixed in the internal cylinder. This presented tool will be used for both tests. Section 4.1.2.



Fig. 4.2: Designed tool to maintain the pen

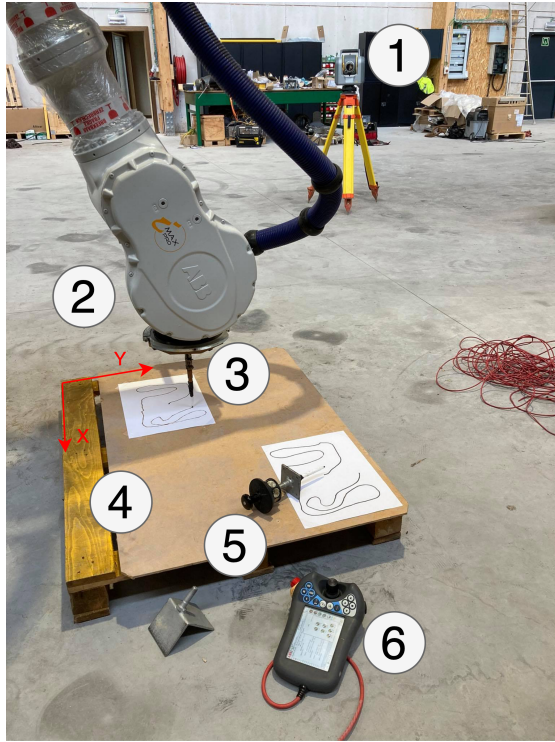


Fig. 4.3: View of the testing setup for the trajectory following

Implemented solution The Figure 4.3 on the left illustrated the setup of the test just introduced previously in this subsection. Then, in the first realised test, the robot (2) has to follow the trajectory defined by the beacon (5). The marker in the bottom of the beacon enables to draw on the sheet of paper, as shown on the picture, as well as the marker maintained by the tool of the robot (3).

The total station (1) in the back of the picture is tracking the current position of the beacon. Connected to its controller, that enables the latter one to send the coordinates position of the beacon to the server application. Nevertheless, a station setup has to be performed before the beginning of the test, otherwise, no coordinates can be sent.

Indeed, as explained in the Section 2.2.3, this allows to create a 3D Cartesian coordinate system. In this case this coordinate system is created with a reference line on the edge of the wooden board (4). Therefore, the coordinates of the beacon correspond to the position on this wooden board. The latter one corresponds to the 'work object', such as defined in the ABB language. This work object, as many things, can be created directly thanks to the 'FlexPendant' (6), which enables to manage the controller of the robot and also as emergency button.

Based on those coordinates compared to the first one before the beginning of the movement, a 'robtargt' inside a movement instruction can be sent to the controller of the robot.

Once this instructions has been executed, the robot has reached the current position of the beacon. Therefore, another measurement of the beacon is collected and the loop restarts until the end of the drawing movement.

Finally, this enables the robot to draw the same trajectory as the one of the beacon, as represented on the Figure 4.3. The Section 4.3.1 will be dedicated to the results of the different tests realised with this setup.

4.1.2 Square drawing

As presented in the beginning of this section, the second test to be performed is a square drawing movement. Moreover, this testing part will be separated into three different tests, where the square drawing movement would be submitted to different perturbations. In the first one, only position perturbations are applied, while during the second one only orientation perturbations are effected the system. Lastly, both types of perturbations are taking place in the last one. Here, the setup for the last one is presented and the different elements involved in this test are presented in the Figure 4.4.

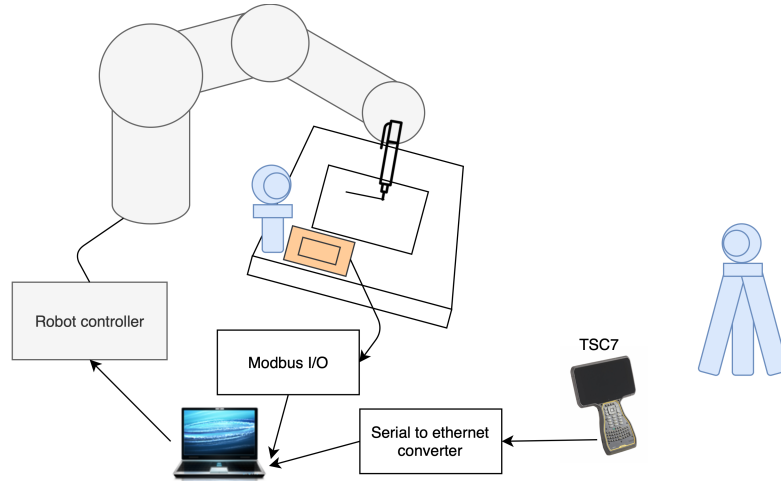


Fig. 4.4: Overview of the different elements constituting the drawing square test

First, the robot draws a shape, in this case a square, thanks to the tool that maintains the marker. Once the robot is drawing, position and orientation disturbances are applied on the wooden board, where the sheet of paper is located.

Those orientation perturbations can be measured thanks to the inclination sensor, they are then transmitted to the server application through the I/O link master as defined in the Section 3.2.3. In terms of the position perturbations, they are calculated thanks to the controller of the total station as described in the Section 3.1.

In order to be able to counteract the disturbances applied on the system, the square movement drawing is discretised. This enables to correct the position and orientation of the following incremental movement based on the collected data. Therefore, between each incremental movement, the data from the inclination sensor and the controller of the total station are collected and to modify the following 'robtargt' as explained in the Section 3.1.

Implemented solution The Figure 4.5 shows the robot (1) executing a square drawing movement while random position and orientation perturbations (6) are affecting the working zone.

Those perturbations applied during the movement can be estimated through the angles of the inclination sensor (2) and the coordinate of the beacon (3). As for the first test, a station setup has to be performed before the drawing movement. This enables to create a 3D Cartesian coordinate system on the wooden board (5). In practice, the reference line, as described in the Section 2.2.3 is established based on two measured points on the edge of the wooden board in the back of the figure.

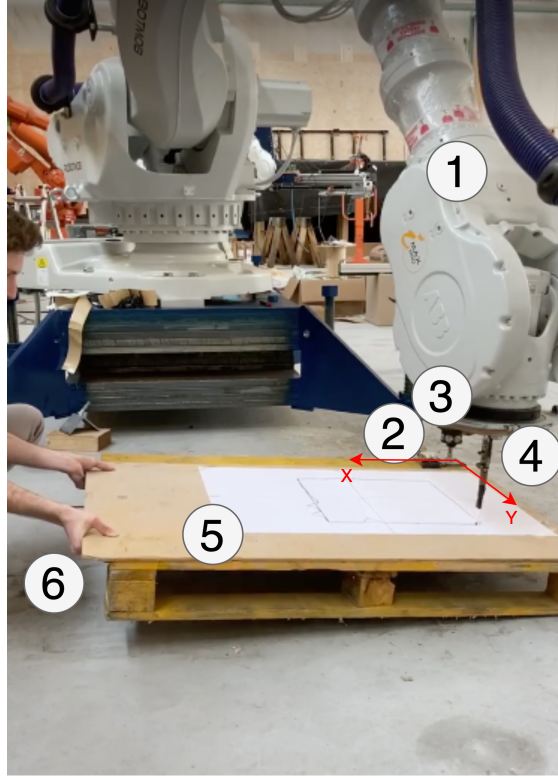


Fig. 4.5: View of the testing setup for the square drawing

The compensation of the position and orientation during the drawing movement allows the marker held by the designed tool (4) to draw a shape as closely resembling a square as possible.

4.2 Software implementation

This chapter will be dedicated to the software implementation as it will explain the different implemented functions, both the ones developed for the project as well as the ones that have been developed by *IMAX Pro* and were useful for the project.

Additionally, this section will also highlighted how those functions are arranged to achieve to the correct movement instructions sent to the robot. Lastly, this section will show the implementation in lines of codes of the different tests explained before.

4.2.1 Software structure

As evoked before, due to the high resources needed to develop the entire solution, this one has not been implemented. Instead, a testing setup has been developed in order to validate the compensation system. This section will develop how it has been implemented through the software application.

In terms of the software structure, it is divided into three parts. The first one refers to the connection part on the different devices. Then, the main functions developed to execute the different drawing movements. Finally, the last part refers to the disconnection with all the devices connected in the first part, as displayed on the Figure 4.6.

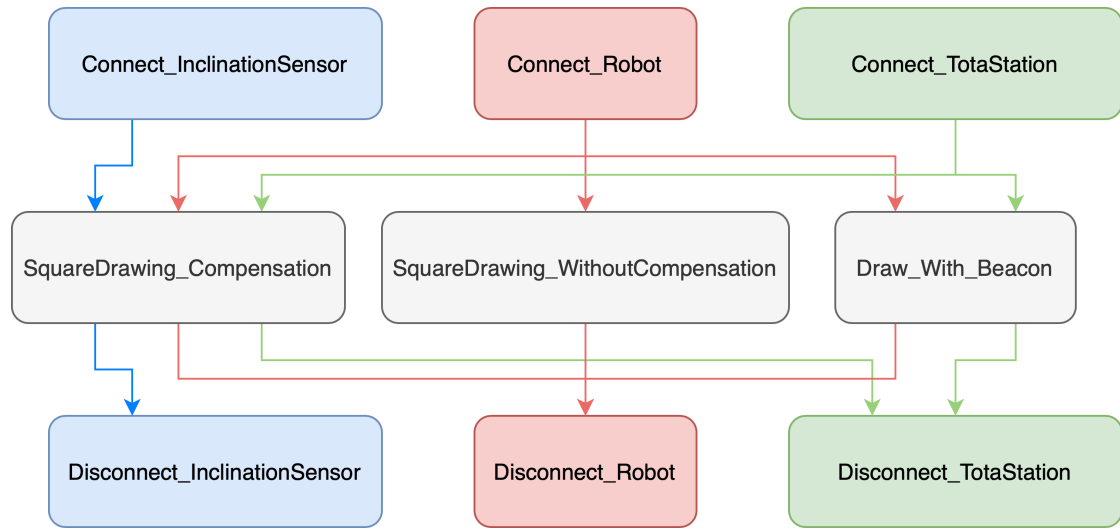


Fig. 4.6: Schematic view of the software implementation

In this Figure 4.6, as their names suggest, the three first functions allow the connection part. They are described by '`Connect_InclinationSensor`', '`Connect_Robot`' and '`Connect_TotaStation`'.

Moreover, in the middle of this Figure 4.6, the three main functions are represented by '`SquareDrawing_Compensation`', '`SquareDrawing_WithoutCompensation`',

'*Draw_With_Beacon*'. The first one allows the robot to draw a square, while perturbations are applied on the system. Those perturbations are measured thanks to the inclination sensor for the orientation part and the total station for the position part. The second one is dedicated to the reference square drawing movement executed by the robot, allowing a comparison between the two square drawing movements. The final one enables the robot to follow the trajectory described by the beacon.

Now that the different functions that compose the software application have been introduced, a more detailed look at each one is required to understand in which way the testing part has been implemented. All the lines of codes of each detailed functions below are available in the Appendix B.

4.2.2 Connecting functions

The first step of the software application is to connect all the devices to each other. This task is performed by the three following functions.

Connect_Robot The first one enables the connection between the software application and the robot. This connection is realised based on the own ID of the robot's controller, as well as their user name and its password. Moreover, a distinction is also made if the controller is real or virtual. Indeed, it is also possible to test the drawing movements on RobotStudio application, thanks to a virtual robot and controller. As well as in the real case as presented by the testing setup.

Based on this information, a connection with the robot's controller can be established and a new instance of the class 'RoboMgr' is created.

This class 'RoboMgr' developed by *IMAX Pro* contains a lot of functions related to the robot's utilisation. It also manages the transmission information to the robot's controller, as well as the format changes into RAPID Instructions.

Connect_TotalStation Then, the connection with the controller of the total station has also to be established. As highlighted in the Section 3.2.2, it communicates with the TCP/IP protocol through the Serial to Ethernet converter. Therefore, the connection is performed based on the IP address and the port number of this converter.

In the software application, a new instance of the class 'TcpClient' is created to connect the specified port on the specified host based on this data. Then, the

underlying stream of data is created to access the network, thanks to the class 'NetworkStream', as shown in the Figure 3.3.

Connect_InclinationSensor The final connection to establish is the one with the inclination sensor. As explained in the Section 3.2.3, the Modbus TCP/IP protocol is used for the data communication. Therefore, a new instance of the class 'ModbusClient' is created based on the IP address and the port number of the I/O link master. Once it is connected, the information can be read on the appropriated registers of the I/O link, which processes the data from the inclination sensor, as shown in the Figure 3.7.

4.2.3 *SquareDrawing_Compensation*

As presented before, this implemented function is the most important one. Indeed, it executes the drawing movement of a square, but this could be any figure, while compensating the random perturbations applied on the wooden board. This perturbations can be both on the position but also according to the orientation. The Figure 4.7 summarizes how this drawing movement is executed.

First, before calling this function in the application software, the connecting functions are called, as developed in the previous Section 4.2.2. Those functions enable the connection between the software application and the controller of the robot, as well as the connection with the measuring devices, which are the controller of the total station and the inclination sensor.

Then, the function '*SquareDrawing_Compensation*' is organized in the following way, as shown in the Figure 4.7. This one is separated by two parts. The first one is executed once during the square drawing movement. While the second part is executed at each step during the loop movement. The different functions involved in those two parts are detailed below.

ParseWobjData In the beginning, the implemented function '*ParseWobjData*' recuperates the work object created directly on the robot. This work object corresponds to the wooden board, where the robot is drawing the square, as detailed before in the Section 4.1. This created work object is used as the reference coordinates system for the position of the targets sent to the controller of the robot.

Indeed, based on three saved positions of the robot, a work object can be created, with its own coordinate system. It has been realised thanks to the 'FlexPendant', that is why, this one is saved into the robot's controller as a RAPID instruction.

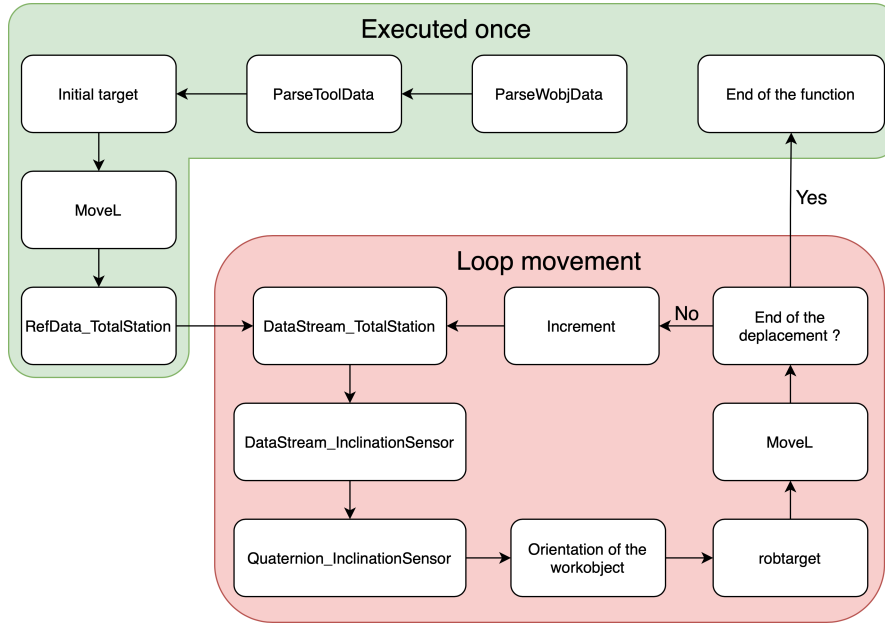


Fig. 4.7: Schematic view of the implemented function for the drawing movement with compensation

Nevertheless, as it is used in the software application, it has to be recuperated from the RAPID Instructions, this is what is done by this function '*ParseWobjData*'.

ParseToolData Like the work object, a tool has also been defined by the 'Flex-Pendant' directly on the robot's controller. This tool resumes the characteristics of the pen, which is attached on the end of the robot.

The tool data is used to describe the characteristics of the tool mounted on the end of the robot, which in this case, refers to a translation of the TCP position. This function '*ParseToolData*' enables as the previous one, to recuperate this created 'tool data' from the robot's controller in the form of a RAPID instruction, in order to use it in the software application.

Initial target and MoveL Once this data are recovered, the robot is sent to the initial position. It corresponds to the first point of the square, where the square drawing movement begins. To reach this initial target, a linear movement '*MoveL*' is sent to the robot with the desired position to reach.

RefData_TotalStation When the robot is positioned in the first edge of the square, the function '*RefData_TotalStation*' is executed. This one returns the

first position coordinates of the beacon sent by the controller of the total station. Based on that, the following data sent will be referenced with respect to this position.

In practice, once the function is called, the 20 first data are excluded, to avoid a format error. Then, the following package of data sent by the controller of the total station is collected on the stream. Those collected bytes are then changed into the ASCII representation. After, this one string format is splitted into three separated doubles, to form the three reference X, Y and Z coordinates for the beacon position.

DataStream_TotalStation The first thing to do in this loop movement is to collect the data from the connected devices. This begins with the total station. In fact, the implemented function '*DataStream_TotalStation*' recuperates the data from the controller of the total station and returns the position displacements along the three axis defined by the setup of the total station. Those displacements are referenced with respect to the initial position returned previously by the function '*RefData_TotalStation*'.

DataStream_InclinationSensor Orientation perturbations can also be induced on the system. Those ones are measured thanks to the inclination sensor. As the latter is connected to the server application through a I/O link device, the proper registers have to be read.

In practice, thanks to the class '*ModbusClient*', the function '*ReadInputRegisters*' can be used. Only the desired address should be written in the arguments. This one can be found in the datasheet of the I/O link device. It enables to collect the X and Y angle values of the inclination sensor.

Quaternion_InclinationSensor The angles recuperate by the function '*DataStream_InclinationSensor*' are in the form of Euler angles. Nevertheless, to modify the orientation of the work object with respect to the measured angles, a quaternion is required. Therefore, the purpose of this function is to convert the Euler angles given by the sensor into the corresponding quaternion, as explained in the Section 3.3.3.

Orientation of the work object Based on this quaternion, the orientation of the work object can be adjusted to correspond to the angles measured by the inclination sensor. Indeed, the device measures the orientation angles, which are

then converted into quaternion. The latter one is used to modify the orientation of the work object, i.e the wooden board.

Robtarget Once the orientation of the work object has been modified, a 'robtarget' is created with the desired position and orientation of the tool center point, i.e the tip of the pen. Its position depends on the increment, the referenced corner and the possible correction applied based on the total station. Moreover, this target has also an orientation, which is constant in this case because the orientation of the work object already compensate the disturbances measured by the inclination sensor.

MoveL Then, based on this 'robtarget', a linear movement instruction 'MoveL' is performed. Therefore, this instruction executed a linear movement to reach the desired position and orientation defined in the 'robtarget'. Moreover, this movement is executed with respect to a coordinate reference system, which is here defined by the work object presented before.

As described in the Section 3.3.2, this instruction also takes as arguments the 'Tool', which is here the pen, as presented by the Figure 4.2 and defined thanks to the function '*ParseToolData*'.

End of the movement Finally, once the movement instruction is executed, the condition of the movement loop is verified. If it is the end of the square, then the function is simply ending and the disconnecting functions take the lead. Otherwise, the loop movement restarts and the length of one incremental movement is added to the increment.

Increment Indeed, as the instruction 'MoveL' is a blocking instruction, the path executed by the robot has to be discretise in order to compensate the external perturbations applied on the system between two incremental movements. Therefore, the square drawing movement is discretise in a series of little incremental movement between which the compensation of the position and orientation perturbations are realised.

End of the function Once the final point on the square is reaching, the function ends and then the functions constituting the disconnecting part are called.

4.2.4 *SquareDrawing_WithoutCompensation*

As presented earlier, the square drawing movement with disturbances is compared to a reference one. Therefore, the purpose of this function is to simply to execute a square drawing movement with compensation of the perturbations. As the robot is placed on the ground in the hall of *IMAX Pro*, the perturbations applied on the system are significant.

Based on this reference square drawn on the sheet of paper, the one drawn under perturbations can be compared. This will be shown in the

In terms of functions involved in this square drawing reference movement, they are similar to the ones used in the previous Section 4.2.3. Excepting that some of them are not longer useful, as explained below. Moreover, in this case, only the connecting function to the controller of the robot is required. Indeed, as the external perturbations are not measured, no connections to the inclination sensor and the controller of the total station are performed.

As in the Section 4.2.3 before, the square drawing reference movement is separated into two parts. The first one is executed once during the drawing movement, while the second part is executed for each incremental movement. As described by the Figure 4.8 below.

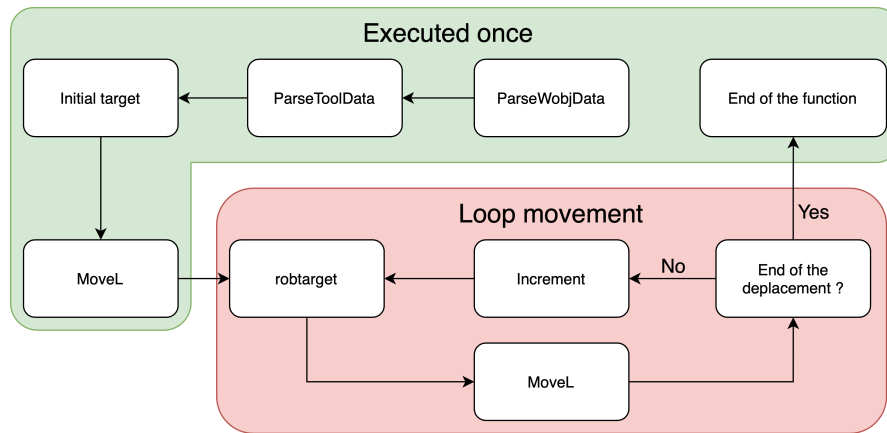


Fig. 4.8: Schematic view of the implemented function for the drawing movement without compensation

The function begins with the implemented function '*ParseWobjData*', which as presented on the Figure 4.7, enables the server application to recuperate the work object, created on the 'FlexPendant' directly on the robot's controller. This work object is useful to create a new 3D Cartesian coordinate system on the wooden

board.

Then the function '*ParseToolData*' is used in the same way as presented on the Figure 4.7 before. It enables to recuperate the tool data create like the work object. This tool data describes the designed tool, with its characteristics.

Then, the initial position of the movement, which is the first edge of the square is reached thanks to a linear movement instruction '*MoveL*'.

Once this position is reached, the loop movement begins. Therefore, a new '*robtarg*' is created with the following position to attain. This '*robtarg*' included the position, as well as the desired orientation of the tool center point. However, this orientation does not change, because no perturbations are applied on the wooden board. Only the position of the targets change between to incremental movement to form a square.

When the appropriated '*robtarg*' is created, a linear movement instruction is effectuated to reach the value defined in this '*robtarg*'. As the movement instruction takes the wooden board as a reference coordinate system, the position of the target has also to be referenced with respect to this system.

After, the condition of the while loop is verified, if the final position of the square is not reached. Then, the incremental movement continues and the length of one incremental movement is added to the increment. Otherwise, it is the end of the function and the server application is disconnected from the robot's controller.

4.2.5 *Draw_With_Beacon*

Finally, the third main function that also correspond to a testing movement is the one that enables the robot to draw the same trajectory as the beacon. This function allows to test the capacity of the robot to react on the information sent by the controller of the total station. This last presented main function would be the one tested in the testing part in the Section 4.3.1.

In practice, based on the information of the coordinate position of the beacon, the robot is able to draw the same trajectory as the one realised by the marker placed on the bottom of the beacon as shown before by the Figure 4.3.

In terms of software implementation, which is the topic of this section, the main function presented here is divided into two parts as the two other main functions.

As shown in the Figure 4.9, first the part executed once is realised then the loop movement is performed.

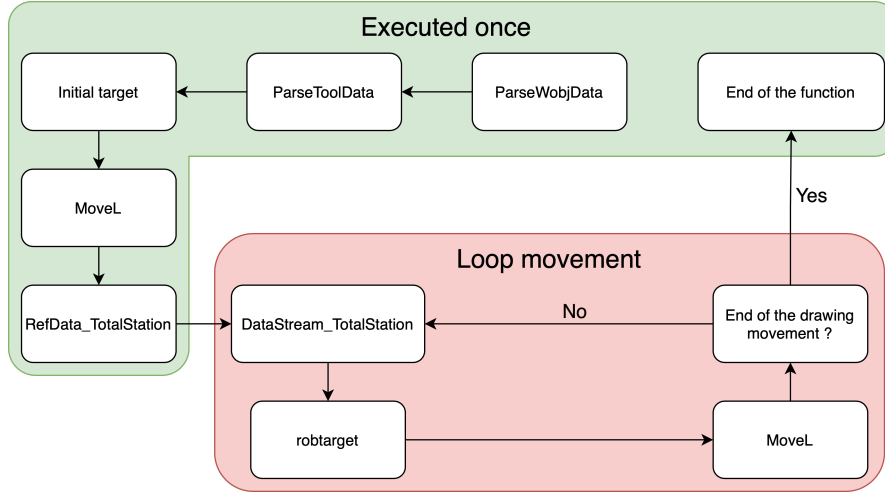


Fig. 4.9: Schematic view of the implemented function for the drawing movement with the beacon

The drawing trajectory of the beacon starts with the two functions '*ParseWobjData*' and '*ParseToolData*', as presented before, those functions enable to recuperate the work object and the tool data created directly on the controller of the total station.

Then, the robot reaches the first position of the drawing movement, which corresponds to the center of the sheet of paper. This is due thanks to a '*robtarget*', which contains the coordinates of the position. Afterwards, a movement instruction is performed to reach this desired position.

Once, the marker of the tool is on the initial position, the current coordinates of the beacon position are taken as a reference, thanks to the function '*RefData_TotalStation*'.

After, the position of the beacon measured by the controller of the total station is collected by the server application. Those collected data are then subtracted from the reference coordinates, to evaluate the displacement of the beacon from the first position.

This displacement along the three axis is then added to the first coordinates of the robot position. This creates the desired position for the marker at the end of the robot, which corresponds to the '*robtarget*'. It will enable the robot to follow

the trajectory established by the beacon, thanks to a linear movement instruction executed by the robot as a 'MoveL'.

Finally, the condition of the movement loop is verified, as in the two other case. If the movement is finished, then it is the end of the function and a disconnection with the robot and the controller of the total station is performed. Otherwise, the drawing movement of the trajectory continues and a new value of the coordinates position of the beacon is collected.

4.2.6 Disconnecting functions

The last step of the software application is to disconnect the devices to the server application. This task is performed by the three following functions. It corresponds to the last part, as presented on the Figure 4.6.

Disconnect_Robot As the name suggests, this function disconnect the server application from the robot's controller. This simply interrupts the communication between the application and the robot's controller. This function is used after the three main functions presented before.

Disconnect_TotalStation Furthermore, this function also enables the disconnecting with the controller of the total station. This function is used after the two main functions '*SquareDrawing_Compensation*' and '*Draw_With_Beacon*'. Indeed, those two functions are connected to the controller of the total station to collect its data.

Disconnect_InclinationSensor Finally, the connection with the inclination sensor also has to be ended. This is due thanks to this function. This one is executed after the main function '*SquareDrawing_Compensation*', which is the only one to require a connection with this sensor.

4.3 Test results

In this chapter, the setup of the different realised tests have been presented in the Section 4.1, as well as the different elements involved in those ones. Then, the software implementation has been detail for the main functions that describes the different drawing movement. Therefore, the purpose of this section is to highlight the different results obtained thanks to the setup and the implemented functions detailed before.

4.3.1 Follow the beacon

The first test to be realised is the one describing in the Section 4.1.1. The implemented function in the server application is named '*Draw_With_Beacon*', which has been explained in the Section 4.2.5.

Here, the capability of following a random trajectory is examined. Indeed, thanks to the marker at the bottom of the beacon, a random trajectory can be draw on a sheet of paper. The purpose of the robot is to draw the same trajectory on its sheet of paper, as as precisely as possible. Two tests would be performed with different velocity as explained below.

Speed of 250 mm/s The first test of this drawing movement is executed with a speed of 250 mm/s. This corresponds to the speed of the linear movement instruction. Based on this speed, the result of two drawing movements are present in the Figure 4.10.

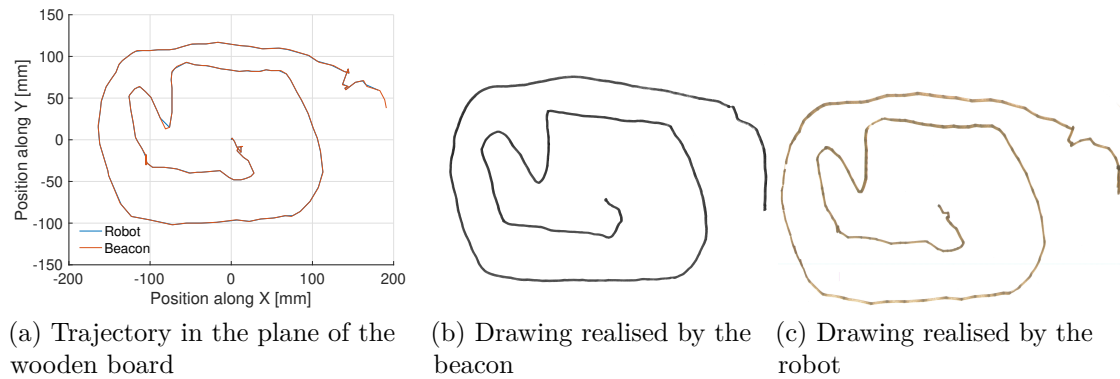


Fig. 4.10: Drawing movement of the trajectory following with a speed of 250 mm/s

Based on the two trajectories plotted in the Figure 4.10a, it can be concluded

that the tool center point of the robot has correctly followed the trajectory defined by the beacon on the wooden board plane.

Moreover, the drawing realised thanks to the marker placed on the designed tool, as represented by the Figure 4.2 is shown in the Figure 4.10c. This one can be compared to the one realised by the marker located on the bottom of the beacon. The drawing accomplished by this marker is shown on the Figure 4.10b.

In the real drawings, there are some imprecisions due to the fact that the marker at the bottom of the beacon is attached only with adhesive tape. Therefore, it was moving and not always perfectly perpendicular to the bottom of the beacon, which induce a lot of imprecisions on the drawing.

The following three graphs displayed in the Figure 4.11 illustrate that a position error occurs between the current position of the robot and the position of the beacon for the same time increment. Indeed, as shown on the Figure 4.11a, this position error along the X or Y axis, as defined in the Figure 4.3 can be up to 5 centimeters. This position error is also highlighted in the two Figure 4.11b and Figure 4.11c, with the shift of the robot position with respect to the beacon position at the same time increment.

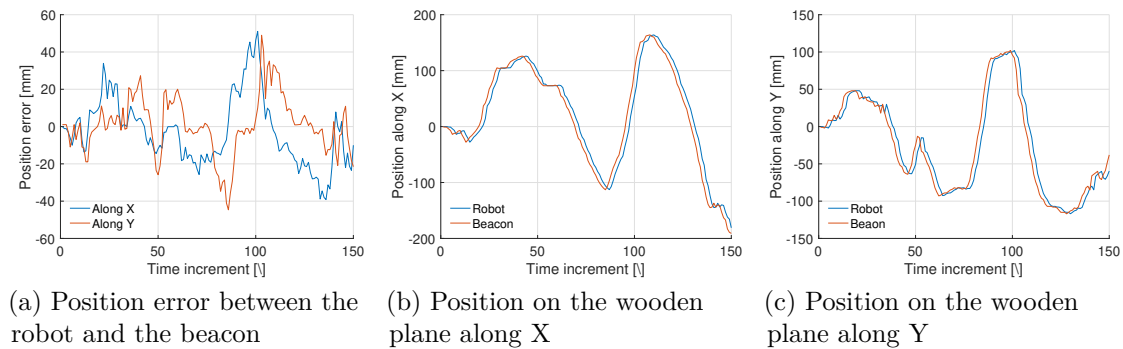


Fig. 4.11: Position error between the robot position and the beacon position, position along X and position along Y at each time increment with a speed of 250 mm/s

This is due to the fact that there is a time delay between the position measurement of the beacon and the real movement instruction to reach this appropriate position. This time delay is inherent to the system, because the position of the beacon is not sent all the time. Indeed, a new position is only available all the 100 ms thanks to the controller of the total station.

Moreover, the linear movement instruction is a blocking instruction. This means that as long as the position and the orientation inside the instruction are not reached, the following lines of code are not executed. As discussed in the Section 5.2, some improvements are possible to decrease this delay.

Speed of 500 mm/s Another test has been realised with an increased speed, to bring out the if the speed has a significant impact to decrease the position error. Therefore, during this test, the speed used before has been doubled. Its value is now 500 mm/s.

As in the previous test, the trajectory of the beacon and the robot in the plane of the wooden board plotted in the Figure 4.12a are comparable. In terms of the real drawing, as illustrated on the Figure 4.12b and Figure 4.12c, some differences occur but are related to the imprecisions of the test, as discussed for the previous test.

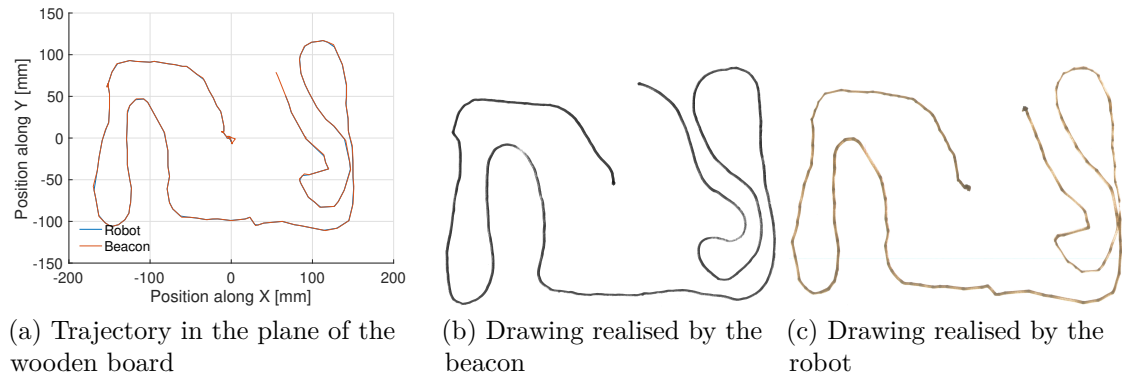


Fig. 4.12: Drawing movement of the trajectory following with a speed of 500 mm/s

The following presented graphs bring out the position error that occurs between the robot position and the beacon position at the same time increment. Indeed, as evoked earlier, the drawing movement is divided into a series of incremental movement, between which the coordinates of the beacon are collected. However, this is not instantaneous and a time delay of around one second is observed between the moment of the beacon position and the moment at which the robot reaches this position.

Nevertheless, the graphs plotted in the Figure 4.13a show that the position errors along the X and Y axis decrease, as the speed increases. However, a position error would always subsist because of the time delay present due to the measuring

devices and due to the time between the received information by the controller and the time to reach the desired position.

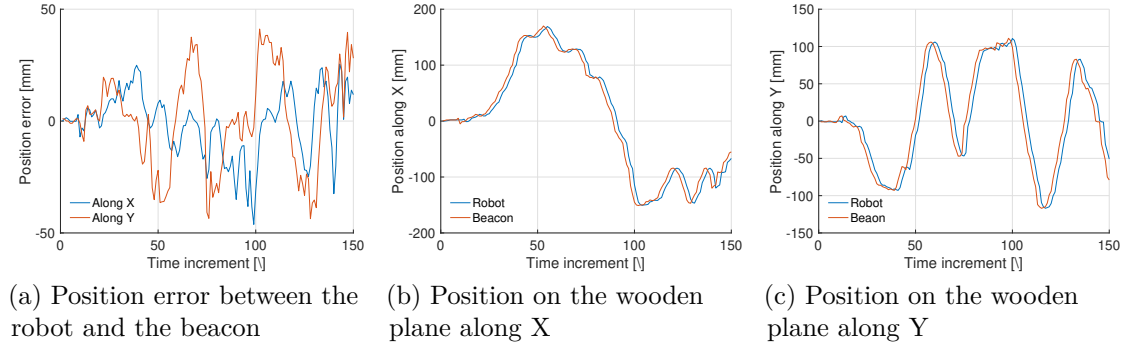


Fig. 4.13: Position error between the robot position and the beacon position, position along X and position along Y at each time increment with a speed of 500 mm/s

4.3.2 Square drawing with position compensation

The second test to be achieved is related to the square drawing movement. During this test, only the positions perturbations would be applied to the wooden board. Therefore, only the data from the controller of the total station would be used to compensate those disturbances.

This test is realised in the configuration setup presented in the Section 4.1.2. In terms of the implementation this testing part used the function '*Square-Drawing_WithoutCompensation*' to create the reference square drawing movement. While the part with the perturbations used the function '*SquareDrawing_Compensation*' to compensation the position disturbances incurred by the system.

Moreover, this testing part shows the capability of the robot to modify its position during a predefined movement in order to counteract the random perturbations applied on the working zone, i.e. the wooden board in this case.

Speed of 250 mm/s The two Figure 4.14a and Figure 4.14b show the results of the square drawing movement on the wooden board plane with a tool center point speed of 250 mm/s. On the left Figure 4.14a, the blue trajectory is the reference square, while the blue one is the square obtained thanks to the compensation system. Whereas, the red trajectory is what would be obtained if no compensation

system would be used. On the right Figure 4.14b, the two real square drawings are displayed.

The left Figure 4.14b shows that due to the imprecisions of the setup, the square drawing of the robot does not perfectly correspond to the reference one. Indeed, the errors in the designed tool induce errors in the drawing movement as represented on the figure.

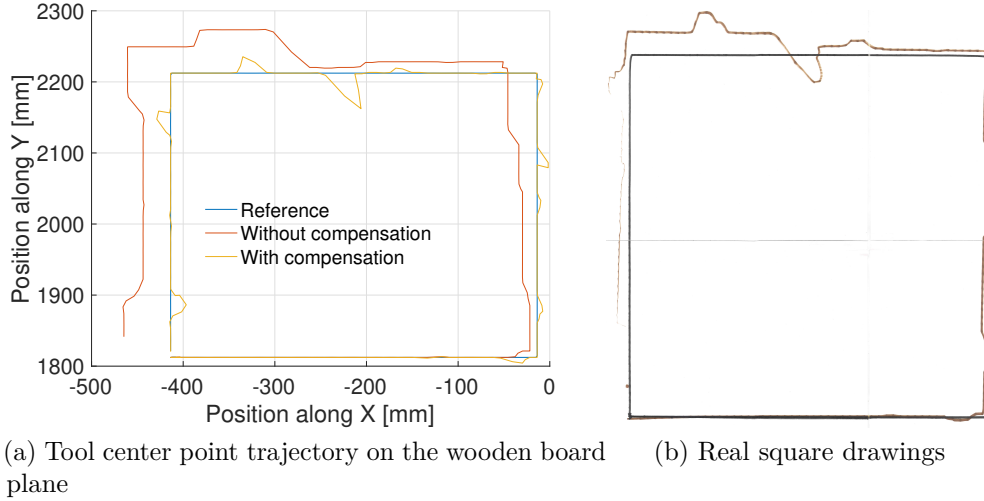


Fig. 4.14: Trajectory on the wooden board plane along the X and Y axis as defined on the Figure 4.5 with a speed of 250 mm/s

Then, the Figure 4.15a shows the position errors between the reference square and the one draw during the disturbances along the X axis of the wooden board in blue and along the Y axis of the wooden board in red, as defined in the Figure 4.5.

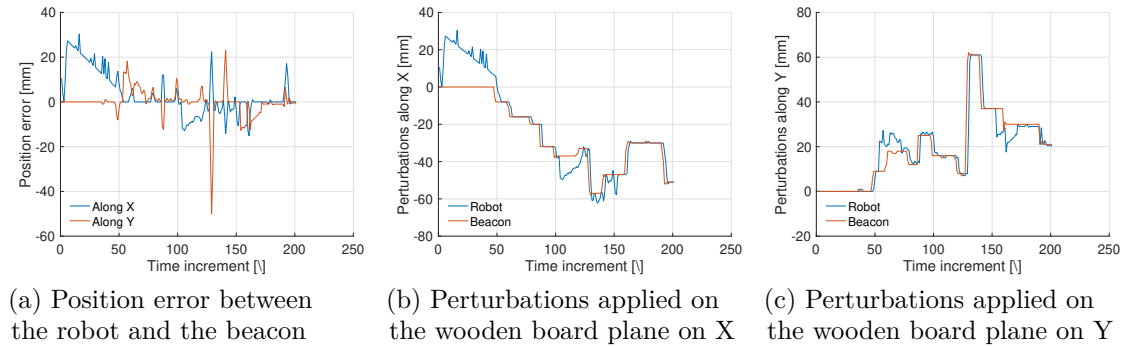


Fig. 4.15: Position error and sum of the perturbations applied on each time increment

Moreover, the two Figure 4.15b and Figure 4.15c display the perturbations applied on the system. Indeed, the beacon position along X and Y is denoted in red, while the corresponding robot position to counteract those position variations is denoted in blue.

Despite the error peak along the Y axis of around 5 centimeters, the position errors does not exceed the 2 centimeters. This signifies that if the positions perturbations are limited in speed, the robot can greatly counteract those disturbances. In practice, if the robot would be mounted at the end of the telescopic arm of the telehandler, the perturbations applied on the system would be limited in speed. Therefore, the robot would be able to correct them better than in the case of the test.

Speed of 500 mm/s A second test has been realised with an increased speed. This one enables to bring out if an increased speed can have a positive impact on the position errors. Therefore, during this second test, the speed value of the tool center point has been doubled from the previous one, like in the two tests presented before to follow a trajectory.

As for the previous test, the two Figure 4.16a and Figure 4.16b show the results of the square drawing movement in the plane of the wooden board. In this case, the real square drawings are more accurate as shown in the right figure.

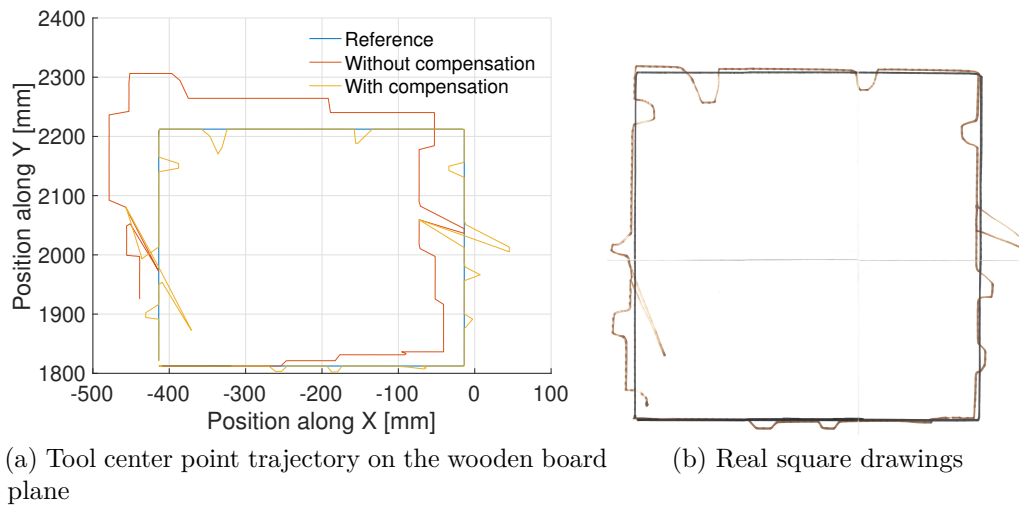


Fig. 4.16: Trajectory on the wooden board plane along the X and Y axis as defined on the Figure 4.5 with a speed of 500 mm/s

Furthermore, the Figure 4.17a displays the position errors along X and Y at each time increment between the reference square and the one drawing under position disturbances. Those axis are referenced as in the Figure 4.5.

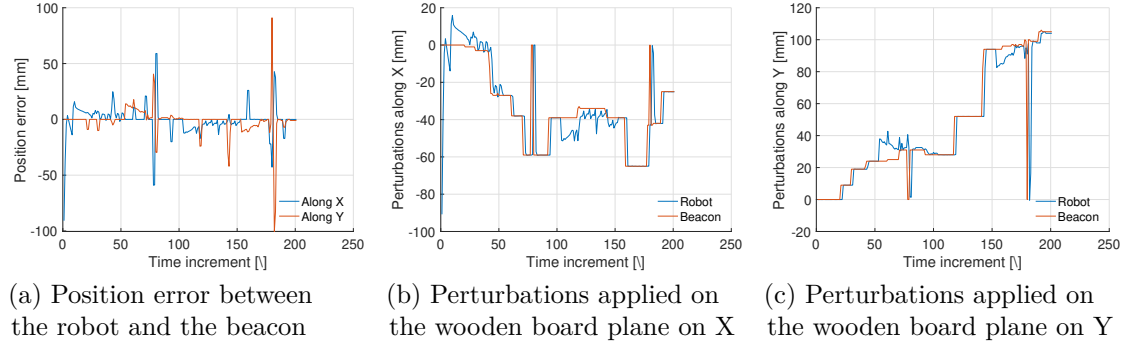


Fig. 4.17: Position error and sum of the perturbations applied on each time increment

Then, the two Figure 4.17b and Figure 4.17c express the position perturbations applied on the wooden board. Those disturbances are measured thanks to the modified position of the beacon, which is displayed in red on those figures. Then, the modified position of the tool center point to counteract those disturbances is manifested in blue on the same figures.

In this test, despite some wrong coordinate positions sent by the controller of the total station, represented by the peaks on the Figure 4.17, the increased speed seems to reduce the position error induce by the perturbations.

4.3.3 Square drawing with orientation compensation

To follow the continuity of the previous test, the third to be achieved is also related to the square drawing movement. Nevertheless, during this test only orientation perturbations would be applied on the wooden board. That is why only the data coming from the inclination sensor are used, no connection with the controller of the total station is established in this case.

This test realised in the same way as the previous one dedicated on the position perturbations part is in the configuration describes by the Section 4.1.2. In this test, as described in the previous Section 4.3.2, two squares are drawing thanks to the marker mounted on the end of the robot. One is the reference one, while the second is drawing under orientations disturbances.

The results of the two tests developed below show the capability of the robot to modify its orientation during a predefined movement, in this case a square drawing movement, in order to counteract the disturbance applied on the working zone.

During those two tests, the orientation perturbations are applied on the system in order to rotate the wooden board along its X and Y axis as defined by the Figure 4.5. As explained in the Section 4.2.3, the angles measured by the inclination sensor enables to modify the orientation of the work object, i.e. the wooden board.

Speed of 250 mm/s The first test has a speed value of the tool center point sets to 250 mm/s. The results of the square drawing movement under this speed on the wooden board plane is displayed in the Figure 4.18. On the Figure 4.18a, the two trajectory of the drawing movement are exactly the same.

Although, the orientation perturbations induce position perturbations at the tool center point position, they are small enough along Y and X to be counteract precisely. Indeed, due to the distance between the end of the robot and the rotation point, position variations are also induced on the wooden board where the marker of the robot is.

In the real drawings, some imprecisions are mainly due to the fact that the tool designed to draw the square is not perfectly centered, which induce the deviation between the two square drawing movements.

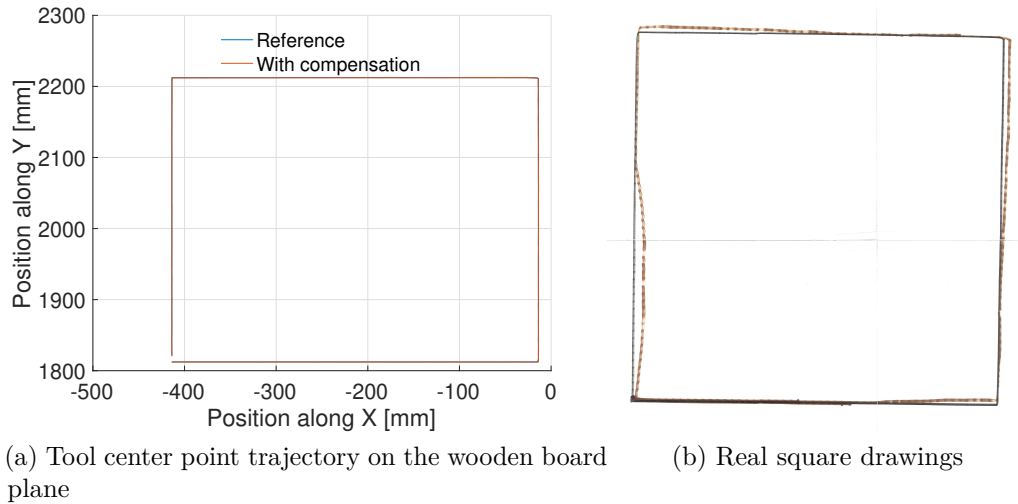


Fig. 4.18: Trajectory on the wooden board plane along the X and Y axis as defined on the Figure 4.5 with a speed of 250 mm/s

Then, the Figure 4.19a shows the angle variations applied on the wooden board. In this figure, the red line corresponds to the a combination of the angles measured by the inclination sensor, while the blue one corresponds to the orientation variations of the robot in reaction of the orientation perturbations. In this graph, a positive variation is linked to an orientation perturbation along the X axis, while a negative one is linked to the Y axis.

As discussed earlier, the orientation disturbances also induce position variation of the wooden board. The Figure 4.19b shows the elevation variations at the position of the robot on the wooden board thanks to the angles measured by the inclination sensor in red. In the same graph, the blue line represents the corresponding elevation of the robot in reaction to the position variations induced by the orientation modifications.

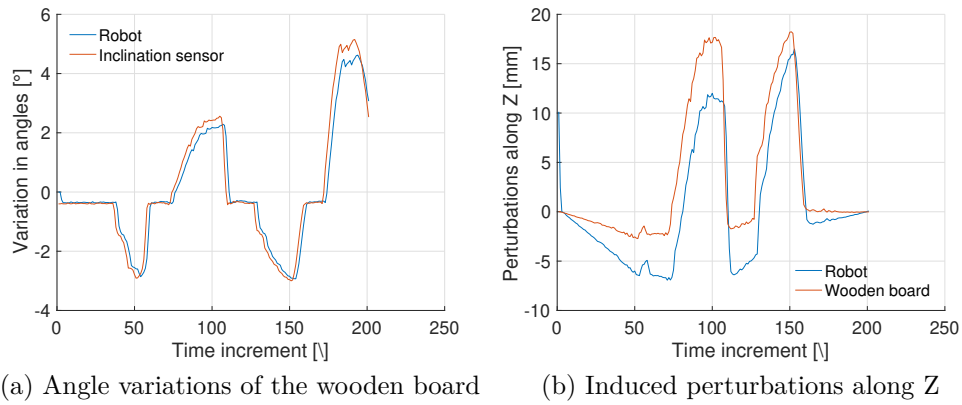


Fig. 4.19: Visualisation of the orientation perturbations applied on the wooden board with a speed of 250 mm/s

Therefore, as presented on the the two Figure 4.18 and Figure 4.18, the robot changes its orientation, when the system is encountered to orientations disturbances, to be always parallel to the working zone. Moreover, due to the distance with the rotation axis where those perturbations are applied, the robot also change its position to keep the same distance with the working zone.

Speed of 500 mm/s As in the other case, a comparison is performed between two speed values of the robot. Therefore, this test is the same as the previous one, excepting that the speed value is now fixed to 500 mm/s.

As for the previous test, the two following Figure 4.20a and Figure 4.20b illustrate the drawing movement into the plane of the wooden board. For this case,

the same results are obtained as before. The differences noticed between the two real drawings come from the approximated centering position of the marker on the tool of the robot.

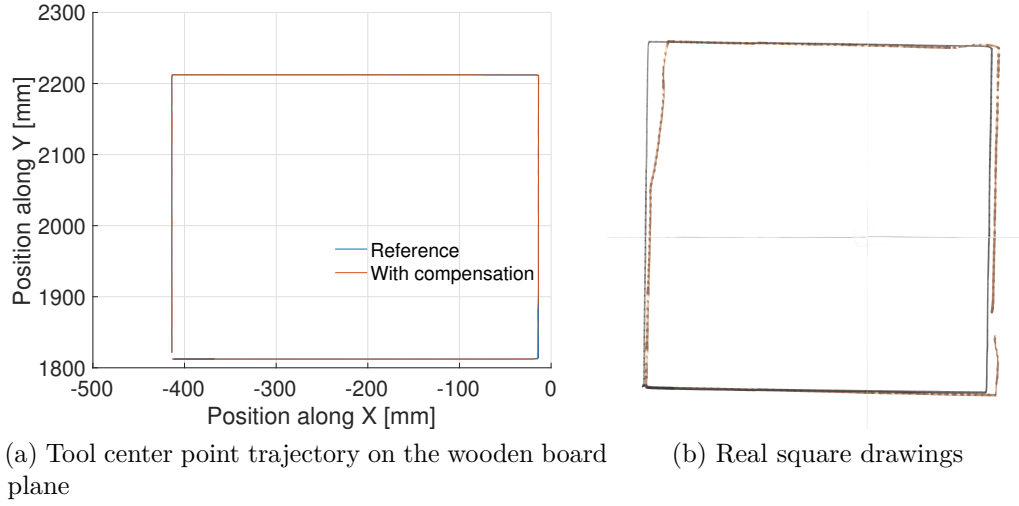


Fig. 4.20: Trajectory on the wooden board plane along the X and Y axis as defined on the Figure 4.5 with a speed of 500 mm/s

The graph, on the left in the Figure 4.21a shows that despite the delay due to the measuring time between two incremental movements, the orientation change of the robot follows the measured orientation thanks to the inclination sensor. As well as the position perturbations induce by the orientation disturbances, which are also compensate by the robot.

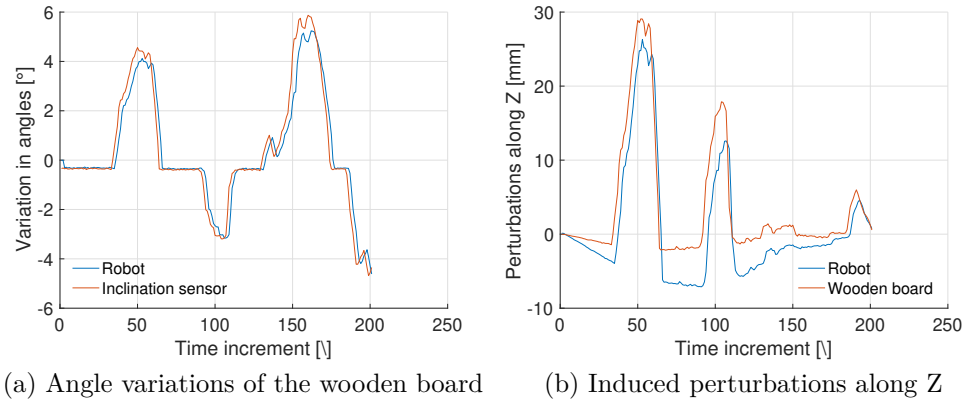


Fig. 4.21: Visualisation of the orientation perturbations applied on the wooden board with a speed of 500 mm/s

4.3.4 Square drawing with position and orientation compensation

Finally, to conclude this test results section, the fourth testing part is also related to the square drawing movement. Nevertheless, in this part, position and orientation perturbations would be applied on the wooden board. Therefore, a connection has to be established with the controller of the total station and the inclination sensor to have all the data necessary to compensate those disturbances.

The configuration is the same as the two previous testing part, a reminder of the testing setup is available at the Section 4.1.2. In this testing part, two square drawing movements have been realised thanks to the marker located on the tool of the robot. Indeed, the first movement is a reference one for the second, which is affecting by position and orientation.

During this test, position disturbances would be apply along the X and Y axis of the wooden board, as defined by the Section 4.1.2. As well as orientation disturbances along this two axis. The purpose of this test is then to highlight the compensation system, during the square drawing movement.

Speed of 250 mm/s The first of the two realised tests has been performed with a tool center point speed of 250 mm/s. The results of the trajectory of the robot on the wooden board plane is shown on the Figure 4.22a. The yellow trajectory indicates that the position of the robot has been modified to compensate the position disturbances measured thanks to the beacon coordinates.

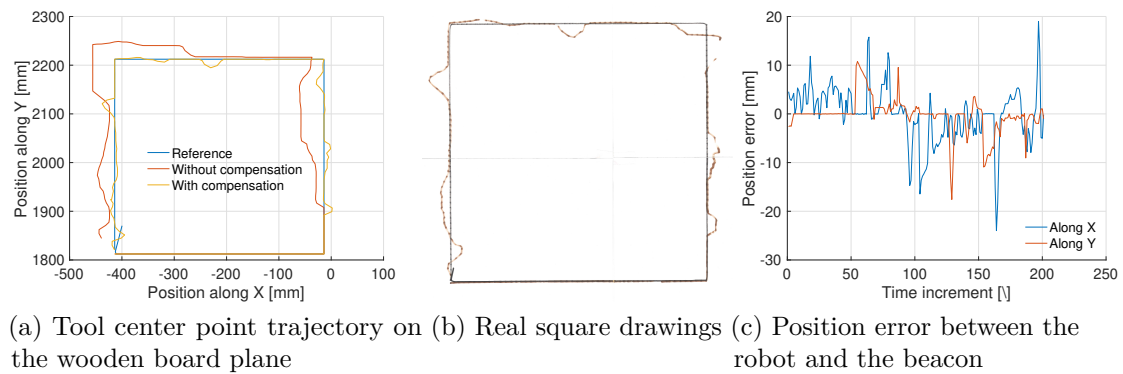


Fig. 4.22: Trajectory on the wooden board plane along the X and Y axis as defined on the Figure 4.5 with a speed of 250 mm/s

This compensation is also proved by the real square drawings on the Figure 4.22b. Despite the imprecisions of the system, the square drawing effected under the perturbations is actually closed to the reference one. Moreover, the Figure 4.23c demonstrates that the position error along the X and the Y of the wooden board plane does not exceed 2 centimeters, except one time.

Despite a delay, the robot is able to counteract the perturbations applied on the system on the X and Y axis of the wooden board as displayed by the Figure 4.23a. In this figure, a shift of 20 millimeters has been practiced on the This has already been proved in the other testing part that compensate the position perturbations.

Finally, in this testing part, orientation disturbances are also affecting the working zone. Therefore, the robot has also to counteract those perturbations. The Figure 4.23b shows the angles variations applied on the system. Those variations are a combination of the two angles measured by the inclination sensor. This graph demonstrates that the robot changes its orientation to be parallel with the working zone.

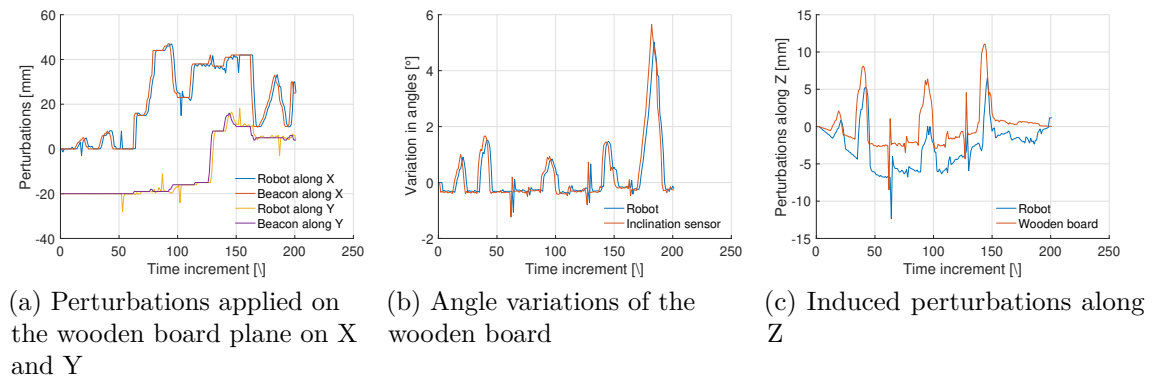


Fig. 4.23: Visualisation of the position and orientation perturbations applied on the wooden board with a speed of 250 mm/s

As already discussed in the test with only the orientation perturbations, the orientations disturbances also induced positions perturbations. Those ones are illustrated on the Figure 4.24c. This graph illustrates that the robot changes its position to also compensate this induced perturbations to maintain the same distance with the working zone.

Speed of 500 mm/s A final test has been performed, which is the same as the previous, with only a doubled speed value. Indeed, in this case, the speed value to

execute the movement instruction is set to 500 mm/s.

The two Figure 4.24a and Figure 4.24b show the results for the square drawing movement. In this test, the random external perturbations applied on the wooden board are more important than in the previous test. Nevertheless, thanks to the increased speed, the accuracy of the two square drawing movements are relatively equivalent. Indeed, despite the few peaks over 2 centimeters, the position error does not exceed 2 centimeters as in the previous case.

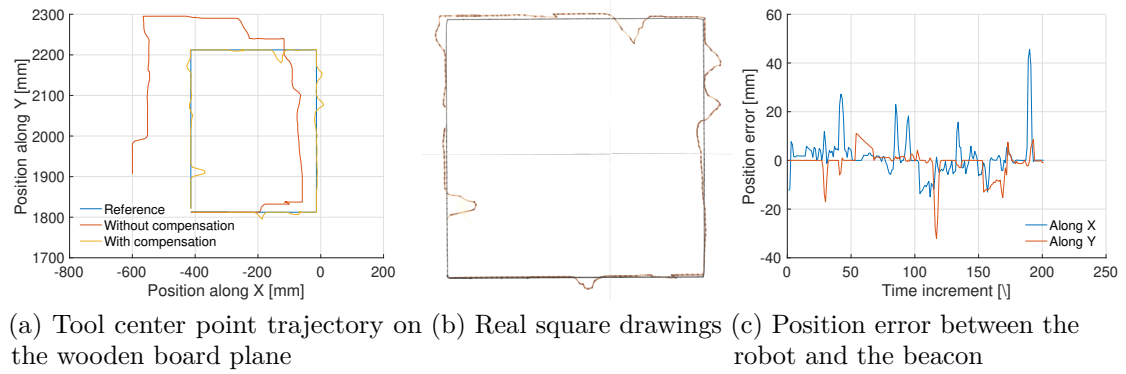


Fig. 4.24: Trajectory on the wooden board plane along the X and Y axis as defined on the Figure 4.5 with a speed of 500 mm/s

Furthermore, the capability of the robot to compensate the external position disturbances allows to correct its trajectory during a movement, in this case a square drawing movement, as displayed by the Figure 4.25a.

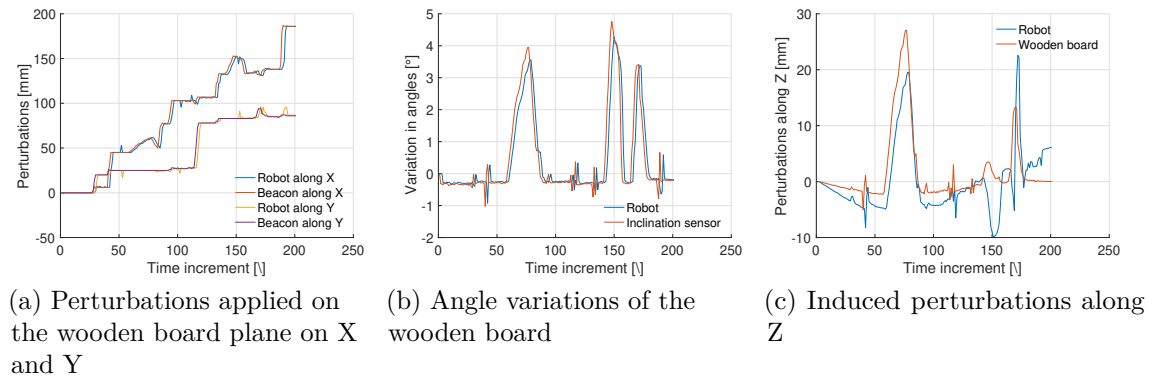


Fig. 4.25: Visualisation of the position and orientation perturbations applied on the wooden board with a speed of 250 mm/s

Moreover, the robot not only counteract the position perturbations, but also the ones occurring along the orientation. As demonstrated by the Figure 4.25b, the robot changes its orientation, thanks to the inclination sensor measurements to be parallel the wooden board.

Finally, as demonstrated through all the results, the robot is able to modify its position and orientation to counteract the external and random perturbations applied on the system. Nevertheless, a delay is inherent to the system due to the measurement devices and the time to realise the incremental movement during which no measurements can be collected.

4.4 Limitations of the tests

The purpose of those tests were to draw a square or the trajectory created thanks to the beacon. Nevertheless, to correctly draw on the sheet of paper, perturbations along the axis normal to the sheet of paper have to be limited. Otherwise, the marker will no touch the sheet of paper anymore. This restricts the testing part because the compensation of the position can only be tested along two axis.

However, this is not an issue, because if it is working along two axis, as it is the same principle along the third one, this compensation system will also correctly counteract the perturbations along this third axis.

Moreover, another limitation of the testing setup for the square drawing movement is that the beacon has to be placed as indicated in the Figure 4.5. Indeed, in order to not compensate twice the perturbations along an axis, the beacon has to be positioned in the origin of the work object, as shown by Figure 4.5.

Otherwise, if an orientation perturbation is applied on the system around another axis of the origin of the work as shown by the Figure 4.26, this will induce a movement of the beacon, describes by the red arrow.

Therefore, the compensation system will compensate the position through the measurement of the beacon position, but this orientation perturbation will also induce a position perturbation. Indeed, the inclination sensor will mea-

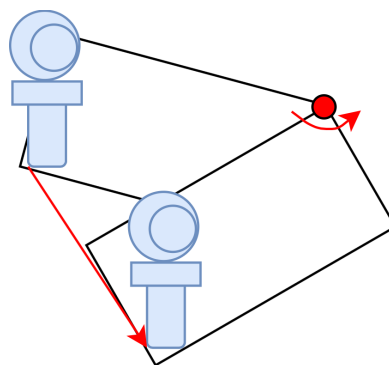


Fig. 4.26: Limitation of the system

sure the rotation and the orientation of the work object will be changed. This will induced another position variation. Finally, the position would be too much compensate and that will lead to imprecisions of the system.

Summary of the chapter

Through this chapter, the different realised tests have been presented. Thanks to the two setup presented in the first section fourth different tests have been performed, with at each case two speed values for the movement instructions.

Those tests bring out the capability of the robot to follow the random trajectory defined by the beacon. As well as the capability of the implemented compensation system to counteract the position and orientations perturbations applied in the working zone, i.e. the wooden board.

Despite the timing delay inherent to the system and the imprecisions of the real drawing movement, the results are conclusive that the compensation system improve the trajectory movement in case of position and orientation perturbations.

This final chapter is the opportunity to discuss the different difficulties encountered during this project, as well as the solutions implemented to overcome them and the improvements that could still be added to the system.

5.1 Difficulties encountered

Some difficulties appeared during the period of the project, other ways have been found in order to be able to implement the solution anyway. Due to these adaptations, the current solution is not the most elegant way to tackle the problem but the best option given the circumstances and available resources has always been aimed for.

First of all, after some research, the simplest way to correct the robot's trajectory seemed to be by using the EGM functionality on the robot which uses an external sensor. This option had been found by doing only some theoretical research. Later on, it appeared that the implementation was going to be more difficult. Indeed the available robot at *IMAX Pro* has an older software that the needed version to be able to apply this solution.

After negotiations with the ABB company, which has developed this functionality, it was estimated that upgrading the robot would cost approximately 10 000 €. *IMAX Pro* was not willing to make such an investment. Therefore, another, more manual solution had to be found. The solution was to discretise the trajectory realised by the robot in order to be able to take in to account the measured data of the external perturbations.

Furthermore, another difficulty was to test the system in its wholeness. Indeed, as the available robot at *IMAX Pro* have important dimensions, as presented in the Section 2.2.1, a massive telehandler was required to support it. However, the latter one was not available at *IMAX Pro* and therefore high costs would be required to test the system in its wholeness.

Additionally, as this thesis process was conducted during the pandemic, working from home is the norm and employees at *IMAX Pro* try to physically come to the office as little as possible. Placing the robot on a telehandler would have required the presence of multiple employees which the company considered too risky considering that my thesis is solely research and not a payed project.

In this way, the testing setup was redesigned in order that the compensation system can be tested without the telehandler. In this testing setup, as detailed in the Section 4.1, the robot was fixed on the ground and it had to follow a random trajectory or effectuated a drawing movement under perturbations. Finally, those missing element was not an issue to bring out the capability of the robot to counteract the external random perturbations applied on the system.

5.2 Improvements

In terms of the improvements, the following paragraphs itemize the different elements that degrade the timing delay of the solution. As well as the different elements that can be added or changed in order to improve the timing reaction between the measured data on the executed movements as presented in the Section 4.3. This list is non exhaustive and other elements of modifications can be appended to the system to improve it.

Wind prediction First, in order to better predict the external perturbations that can be applied on the system, another external device could be added to the system. This one is an anemometer, which enables to measure the wind speed, wind pressure, as well as the direction of the wind. This one could be used to predict the direction and the intensity of the wind, in order to predict the position and orientation changes due to the wind.

Indeed, outside, at important heights on constructions sites, one major elements that can lead to disturbances is the wind. One better way to counteract them, is not just to only measured the perturbations thanks to the external devices, but also to predict them in order to be more reactive to those disturbances. Moreover,

as the wind is more or less affecting the same direction, those predictions could improve the system.

Orientation measurements The chosen element to measure the orientation variations applied on the system is the inclination sensor, as presented in the Section 2.2.4. It was a practical choice because this sensor was available at *IMAX Pro* and the range of its measurements corresponded to the application. Nevertheless, as mentioned before in the Section 3.4, this device measures only the rotation about two axis.

Therefore, to improve the system, others sensor could be used. As an example an 'Inertial Measurement Unit' sensor, denoted as 'IMU', could be employed. This electronic device measures the acceleration, orientation and magnetic forces based on its 3 accelerometers, 3 gyroscopes and 3 magnetometers. One per axis for each axis of the 3D space. That is why this device can be used to measure the orientation variations that can occur due to the perturbations along the three axis.

Timing delay One major drawback of the compensation system is its timing delay, which leads to position error. Indeed, there a delay between the moment of measured information collected by the server application and the moment where the robot has finally reach the desired position counteracting the external perturbations.

Although, the information sent by controller of the total station is not available in continuous, a new data packet is sent every 100 ms. This is not the part of the system that massively increases the timing delay. Indeed, the part that increases this delay is due to the fact that the server application has to send a new movement instruction with the corrected position and orientation between each incremental movement. As this is a blocking instruction, the system is blocking until the movement has reached the desired target.

Imprecisions of the designed tool As already discussed in the Section 4.3, the designed tool induce imprecisions in the real drawings. Those ones are due to the mechanical slack in the tool, as well as the measurement imprecisions or the non centering on the robot.

To be more precise in the tool definition, a metrology device can be used. For example, the society 'FARO' provides this type of extremely 3D measurement devices. This one could be used to precisely determine the position of the tip of the pen, with respect to the end of the robot. This will minimize the error induce in the drawing movement, especially for the orientation perturbations.

5.3 Future researches

This thesis provides a solution to the location and compensation part of the problem. Further research has to be conducted in order to find an elegant solution to the whole problem. Additionally, as developed in the Section 5.2, multiple options are available to improve the presented solution.

The result from this study can also be applied in the context of real-time machining where the different trajectories would be defined directly by a beacon.

Appendices

A

Quaternions - Mathematical Development

There are different ways to represent an arbitrary rotation in 3D based on at least 3 numbers. Obviously, there is the 3-number representations as the Euler angles or the rotation matrices. Nevertheless, the 4-number representations is also used through the quaternion. This notation is preferred to use instead of rotation matrices, because they are more compact, efficient and numerically stable. Additionally, they can avoid the problem of gimbal lock. Nevertheless, they are not as intuitive and easy to understand as Euler angles. Therefore, this section will be dedicated to a brief description of it.

Algebra The quaternion is a generalisation of the complex numbers with three imaginary parts. It is written formally as $q_0 + q_1i + q_2j + q_3k$, where q_i are the real numbers and the imaginary parts are denoted by i, j and k . Moreover, those imaginary parts are following the identities :

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ ij = k, ji = -k \\ jk = i, kj = -i \\ ki = j, ik = -j \end{cases} \quad (\text{A.1})$$

Furthermore, the multiplication of the imaginary numbers are not commutative. The Fig. A.1 helps to find the sign of the multiplication. Indeed, considering that the symbols are been arranged cyclically. Then, if two symbols to be multiplied are consecutive clockwise, the sign is plus. However, if they are counterclockwise then the sign is minus.

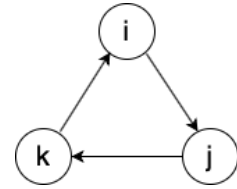


Fig. A.1: Sign of multiplication

In addition, one arithmetic operation, which will be used later in this subsection is the multiplication of two different quaternions. This operation is defined by [32] :

$$\begin{aligned}
& (q_0 + q_1 i + q_2 j + q_3 k) \cdot (p_0 + p_1 i + p_2 j + p_3 k) \\
&= (q_0 p_0 - p_1 q_1 - q_2 p_2 - p_3 q_3) + \\
&\quad (q_0 p_1 + q_1 p_0 + q_2 p_3 - q_3 p_2) i + \\
&\quad (q_0 p_2 + q_2 p_0 - q_1 p_3 + q_3 p_1) j + \\
&\quad (q_0 p_3 + q_3 p_0 + q_1 p_2 - q_2 p_1) k
\end{aligned} \tag{A.2}$$

Quaternion for rotation In the 3D space, a vector can be expressed as a pure quaternion, which is a quaternion with no real part. The real part being the q_0 in the general form. Moreover, in the 3D space, a rotation is represented by a quaternion q_R with the additional requirement that its norm $|q_R|$ has to be equal to 1. Then, a rotation from one coordinate system A to another coordinate system B is given by the conjugation operation :

$$q_B = q_R q_A q_R^* \tag{A.3}$$

In this equation, q_R^* corresponds to the conjugate of the q_R quaternion. Then, q_B represents the 3D vector, which is the result from the rotation q_R of a 3D vector defined in the A coordinate system. The result computed below is a vector since it has no real part :

$$\left\{ \begin{aligned} & q_B = (q_0 + q_1 i + q_2 j + q_3 k) (x i + y j + z k) (q_0 - q_1 i - q_2 j - q_3 k) \\ \iff & q_B = (x(q_0^2 + q_1^2 - q_2^2 - q_3^2) + 2y(q_1 q_2 - q_0 q_3) + 2z(q_0 q_2 + q_1 q_3)) i + \\ & (2x(q_0 q_3 + q_1 q_2) + y(q_0^2 - q_1^2 + q_2^2 - q_3^2) + 2z(q_2 q_3 - q_0 q_1)) j + \\ & (2x(q_1 q_3 - q_0 q_2) + 2y(q_0 q_1 + q_2 q_3) + z(q_0^2 - q_1^2 - q_2^2 + q_3^2)) k \end{aligned} \right. \tag{A.4}$$

Then, the quaternion q_B can be simplified since the norm of a rotation quaternion is 1. This gives the following matrix, denoted as M [32] :

$$M = 2 \cdot \begin{bmatrix} q_0^2 + q_1^2 - 0.5 & q_1 q_2 - q_0 q_3 & q_0 q_2 + q_1 q_3 \\ q_0 q_3 + q_1 q_2 & q_0^2 + q_2^2 - 0.5 & q_2 q_3 - q_0 q_1 \\ q_1 q_3 - q_0 q_2 & q_0 q_1 + q_2 q_3 & q_0^2 + q_3^2 - 0.5 \end{bmatrix} \tag{A.5}$$

From rotation matrix to quaternion The previously calculated matrix M represents a rotation matrix. However, as describes before, the quaternion also characterized a rotation. Therefore, based on a given rotation matrix M , the quaternion that represents the same rotation can be computed. First, the trace of

the matrix M is calculated as the sum of the elements of the diagonal :

$$\begin{aligned} \text{Trace}(M) &= M_{11} + M_{22} + M_{33} \\ &= 4q_0^2 - 1 \end{aligned} \quad (\text{A.6})$$

Then, the first element of the quaternion q_0 is solved based on the previous Equation (A.6). The second element q_1 is found from the M_{11} element of the matrix M . Then, the third and fourth elements q_2 and q_3 are similarly resolved based on the matrix elements M_{22} and M_{33} , which give for the fourth elements of the quaternion [32] :

$$\begin{aligned} q_0 &= \sqrt{\frac{\text{Trace}(M) + 1}{4}} \\ q_1 &= \sqrt{\frac{M_{11}}{2} + \frac{1 - \text{Trace}(M)}{4}} \\ q_2 &= \sqrt{\frac{M_{22}}{2} + \frac{1 - \text{Trace}(M)}{4}} \\ q_3 &= \sqrt{\frac{M_{33}}{2} + \frac{1 - \text{Trace}(M)}{4}} \end{aligned} \quad (\text{A.7})$$

Conversion between Euler angles and Quaternion The data received by the server application are not at the form of a quaternion. Therefore, a conversion from a given rotation angle to the corresponding quaternion has to be computed.

In addition, any arbitrary rotation in the 3D space can be characterized by a combination of three angles. Those angles are denoted as the Euler angles. Each one can be represented by a standard rotation matrix about one of the three principles axis, as on the following Equation (A.8). The first matrix represents a rotation of an angle α around the X axis, while the second one is about the Y axis and the last one is about the Z axis :

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}, \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}, \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{A.8})$$

Based on the previous Equation (A.7), a quaternion can be calculated for each of the elementary rotation matrix. This gives the following quaternions :

$$q_X = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right) \\ 0 \\ 0 \end{bmatrix}, \quad q_Y = \begin{bmatrix} \cos\left(\frac{\beta}{2}\right) \\ 0 \\ \sin\left(\frac{\beta}{2}\right) \\ 0 \end{bmatrix}, \quad q_Z = \begin{bmatrix} \cos\left(\frac{\gamma}{2}\right) \\ 0 \\ 0 \\ \sin\left(\frac{\gamma}{2}\right) \end{bmatrix} \quad (\text{A.9})$$

Then, based on those three standard rotation matrices, presented in the Equation (A.8), a general rotation matrix can be calculated. In the same way, based on those three previous quaternion in the Equation (A.9), a general expression of a quaternion can be developed. This general representation is expressed from the three Euler angles, which make the link between the two different representations. This general representation is based on the multiplication of the three quaternions presented in the Equation (A.9) :

$$q_E = q_Z \cdot q_Y \cdot q_X \quad (\text{A.10})$$

This equation can be calculated based on the arithmetic operation described in the Equation (A.2), which gives [32] :

$$q_E = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right) \cos\left(\frac{\beta}{2}\right) \cos\left(\frac{\gamma}{2}\right) + \sin\left(\frac{\alpha}{2}\right) \sin\left(\frac{\beta}{2}\right) \sin\left(\frac{\gamma}{2}\right) \\ \sin\left(\frac{\alpha}{2}\right) \cos\left(\frac{\beta}{2}\right) \cos\left(\frac{\gamma}{2}\right) - \cos\left(\frac{\alpha}{2}\right) \sin\left(\frac{\beta}{2}\right) \sin\left(\frac{\gamma}{2}\right) \\ \cos\left(\frac{\alpha}{2}\right) \sin\left(\frac{\beta}{2}\right) \cos\left(\frac{\gamma}{2}\right) + \sin\left(\frac{\alpha}{2}\right) \cos\left(\frac{\beta}{2}\right) \sin\left(\frac{\gamma}{2}\right) \\ \cos\left(\frac{\alpha}{2}\right) \cos\left(\frac{\beta}{2}\right) \sin\left(\frac{\gamma}{2}\right) - \sin\left(\frac{\alpha}{2}\right) \sin\left(\frac{\beta}{2}\right) \cos\left(\frac{\gamma}{2}\right) \end{bmatrix} \quad (\text{A.11})$$

This relation allows to calculate any quaternions based on the three Euler angles, which can represented any orientations in the 3D space.

B

Code lines of the server application

Function B.1 Connection with the robot's controller

```
1 // Connection to the controller of the robot, based on the Id,
  the user and the password, and a boolean for the real or the
  virtual case
2 // Returns a RoboMgr to have access to all the functions for
  the control of the robot
3 public RobotMgr.RobotMgr Connect_Robot(string Id, string user,
  string password, bool isVirtual)
4 {
5     ABBControllerMgr CtrlMgr = new ABBControllerMgr(isVirtual);
6     CtrlMgr.Instance();
7     if (CtrlMgr.Connect(Id, user, password))
8     {
9         CtrlMgr.StartMainRapidTask();
10        //RobMgr = new RobotMgr.RobotMgr(Connection.RobDTO.Id,
            (RobotMgr.RobotMgr.CONNECTION_MODE)Connection.CMode,
            CtrlMgr, new ToolChanger());
11    }
12    else
13    {
14        Console.WriteLine("Error, Robot not found");
15    }
16    RobotMgr.RobotMgr RobMgr = new RobotMgr.RobotMgr(Guid.
        NewGuid(), RobotMgr.RobotMgr.CONNECTION_MODE.
        VirtualNative, CtrlMgr, new ToolChanger());
17    return RobMgr;
18 }
```

Function B.2 Disconnection with the robot's controller

```
1 // Disconnection to the controller of the robot
2 // Take into arguments the robot controller of the robot
3 public void Disconnect_Robot(RobotMgr.RobotMgr RobMgr)
4 {
5     RobMgr.Disconnect();
6 }
```

Function B.3 Drawing movement of a square with compensation

```
1 // The robot draws a square, with the compensation system based
   // on the values of the total station and the inclination
   // sensor
2 // The function takes in arguments, the side of the square, the
   // number of increment of the movement (a multiple of 4), the
   // RobMgr, which enables the connection with the controller,
3 // the NetworkStream, which is the stream connected with the
   // controller of the total station and finally the modbusclient
   // , which allows the connection with the inclinasion sensor
4 public void SquareDrawing_Compensation(int side, int step,
   RobotMgr.RobotMgr RobMgr, ModbusClient modbusclient,
   NetworkStream stream)
5 {
6     List<RapidOperation> ROp = new List<RapidOperation>();
   // List of operations
7     UsualMoveOperation MoveL; // Linear movement
8     int displacement = side * 4; // Length of the total
   displacement
9     int increment = displacement / step; // The length
   of one incremental movement
10    int i = 0; // Sum of the incremental movement for the
   while loop
11
12    WobjData wobjdata = ParseWobjData(RobMgr); ; //
   Define the work object which is the wood board
13    ToolData currentTool = ParseToolData(RobMgr, "tPen");
14    wobjdata.oFrame.Trans = new Pos(100, 0, 0); //
   Translation vector to reach initial position of the
   square
15
16    //Orient orient = new Orient(0, -1, 0, 0); //
   Initial orientation of the robot
17    RobTarget Robtarget_5 = new RobTarget(new Pos(), new Orient
   (0, 0.707107, -0.707107, 0), new ConfData(-1, 0, 0, 0),
   new ExtJoint(500)); // Initial target of the robot
18    MoveL = new UsualMoveOperation(TYPE_USUAL_MOVE.MoveL,
   Robtarget_5, RobotMgrConstant.v1000, currentTool,
   RobotMgrConstant.z0, wobjdata);
```

```

19  ROp.Add(new RapidOperation(OPERATION_TYPE.UMove, Move1));
20  RobMgr.DoSequence(ROp);           // Do the linear movement
    to the initial target
21  Thread.Sleep(2000);               // Wait to reach the initial
    position
22
23  double[] positions_ref = new double[3];    // Initial
    position of the beacon of the total station, which is
    taken as the reference position before the movement
24  positions_ref = RefData_TotalStation(stream);    //
    Return the first position send by the total station,
    which is the reference
25  double[] error_pos = new double[3];
26  double[] angles = new double[3];
27
28  UsualActionOperation accset = new UsualActionOperation(
    TYPE_USUAL_ACTION.AccSet, new AccelerationOperation(10,
    10));    // Smooth the movement of the robot
29  ROp.Add(new RapidOperation(OPERATION_TYPE.UAction, accset))
    ;
30
31  StreamWriter sw_beacon = new StreamWriter("C:\\Tests\\
    Error_Pos_Beacon.txt");    // Pass the filepath and
    filename to the StreamWriter Constructor for the error
    of position of the beacon
32  sw_beacon.WriteLine(positions_ref[0].ToString() + ", " +
    positions_ref[1].ToString() + ", " + positions_ref[2].
    ToString());    // Write the reference position of
    the beacon in a file
33  StreamWriter sw_pos_robot = new StreamWriter("C:\\Tests\\
    Pos_Robot.txt");    // Pass the filepath and filename
    to the StreamWriter Constructor for the current
    position of the robot
34
35  StreamWriter sw_inclinaison_sensor = new StreamWriter("C:\\
    Tests\\Angles_Inclinaison_sensor.txt");    // Pass
    the filepath and filename to the StreamWriter
    Constructor for the angles given by the inclinaison
    sensor
36  StreamWriter sw_angles_robot = new StreamWriter("C:\\Tests
    \\Angles_Robot.txt");    // Pass the filepath and
    filename to the StreamWriter Constructor for the current
    orientation of the robot
37
38  while (i <= displacement)           // Loop for the movement to
    draw the square
39  {
40      error_pos = DataStream_TotalStation(positions_ref[0],
        positions_ref[1], positions_ref[2], stream);

```

```

41
42     angles = DataStream_InclinationSensor(modbusclient);
43     System.Numerics.Quaternion quaternions = new System.
44         Numerics.Quaternion();
45     quaternions = Quaternions_InclinationSensor(wobjdata,
46         angles);
47
48     //Console.WriteLine("Angle : X : " + angles[0]);
49     //Console.WriteLine("Angle : Y : " + angles[1]);
50
51     Orient orient = new Orient((double)quaternions.W, (
52         double)quaternions.X, (double)quaternions.Y, (double
53         )quaternions.Z);
54     //Console.WriteLine("Quaternion : " + orient);
55     wobjdata.oFrame.Rot = orient;
56
57     RobTarget RobT = RobMgr.GetLinearRobotPosition(
58         ABBCoordinatorMgr.ABBCoordinateSystemType.World);
59     // Return the current position of the robot
60     sw_pos_robot.WriteLine(RobT.Trans.X.ToString() + ", " +
61         RobT.Trans.Y.ToString() + ", " + RobT.Trans.Z.
62         ToString()); // Write the current position of
63         the robot in to the corresponding file
64
65     sw_beacon.WriteLine(error_pos[0].ToString() + ", " +
66         error_pos[1].ToString() + ", " + error_pos[2].
67         ToString()); // Write the current position of
68         the beacon in to the corresponding file
69
70     System.Numerics.Quaternion quaternion_robot = new
71         System.Numerics.Quaternion((float)RobT.Rot.Q1, (
72         float)RobT.Rot.Q2, (float)RobT.Rot.Q3, (float)RobT.
73         Rot.Q4);
74
75     System.Numerics.Vector3 angles_robot =
76         RemoteObjectQCtrlRobot.ConverterToolBox.
77         QuaternionToEuler(quaternion_robot); // Convert
78         the current quaternion of the robot in to the
79         appropriate Euler angles
80
81     sw_angles_robot.WriteLine(angles_robot.X.ToString() + "
82         , " + angles_robot.Y.ToString() + ", " +
83         angles_robot.Z.ToString()); // Write the current
84         angle orientation of the robot in to the
85         corresponding file
86
87     sw_inclinaison_sensor.WriteLine(angles[0].ToString() +
88         ", " + angles[1].ToString() + ", " + "0"); //
89         Write the current angle orientation of the
90         inclination sensor in to the corresponding file
91     Console.WriteLine(angles_robot);
92
93

```

```

64         if (i <= displacement / 4)           // First side of the
        square
65     {
66         if (i == displacement / 4)
67         {
68
69             Robtarget_5 = new RobTarget(new Pos(
                displacement / 4, 0, 0), new Orient(0,
                0.707107, -0.707107, 0), new ConfData(-1, 0,
                0, 0), new ExtJoint(500));
70         }
71         else
72         {
73             RobTarget Robtarget_2 = new RobTarget(new Pos(
                Robtarget_5.Trans.X + i + increment -
                error_pos[0], Robtarget_5.Trans.Y +
                error_pos[1], Robtarget_5.Trans.Z -
                error_pos[2]), new Orient(0, 0.707107,
                -0.707107, 0), Robtarget_5.Robconf, new
                ExtJoint(500));
74             Move1 = new UsualMoveOperation(TYPE_USUAL_MOVE.
                MoveL, Robtarget_2, RobotMgrConstant.v1000,
                currentTool, RobotMgrConstant.z0, wobjdata);
75             ROp.Add(new RapidOperation(OPERATION_TYPE.UMove
                , Move1));
76         }
77     }
78     if (i > displacement / 4 && i <= displacement / 2)
        // Second side of the square
79     {
80         if (i == displacement / 2)
81         {
82             Robtarget_5 = new RobTarget(new Pos(
                displacement / 4, displacement / 4, 0), new
                Orient(0, 0.707107, -0.707107, 0), new
                ConfData(-1, 0, 0, 0), new ExtJoint(500));
83         }
84         else
85         {
86             RobTarget Robtarget_2 = new RobTarget(new Pos(
                Robtarget_5.Trans.X - error_pos[0],
                Robtarget_5.Trans.Y + i + increment -
                displacement / 4 + error_pos[1], Robtarget_5
                .Trans.Z - error_pos[2]), new Orient(0,
                0.707107, -0.707107, 0), Robtarget_5.Robconf
                , new ExtJoint(500));
87             Move1 = new UsualMoveOperation(TYPE_USUAL_MOVE.
                MoveL, Robtarget_2, RobotMgrConstant.v1000,
                currentTool, RobotMgrConstant.z0, wobjdata);

```



```

88         ROp.Add(new RapidOperation(OPERATION_TYPE.UMove
89             , Move1));
89     }
90 }
91 if (i > displacement / 2 && i <= 3 * displacement / 4)
92     // Third side of the square
93 {
94     if (i == 3 * displacement / 4)
95     {
96         Robtarget_5 = new RobTarget(new Pos(0,
97             displacement / 4, 0), new Orient(0,
98             0.707107, -0.707107, 0), new ConfData(-1, 0,
99             0, 0), new ExtJoint(500));
100     }
101     else
102     {
103         Robtarget_2 = new RobTarget(new Pos(
104             Robtarget_5.Trans.X - i - increment +
105             displacement / 2 - error_pos[0], Robtarget_5
106             .Trans.Y + error_pos[1], Robtarget_5.Trans.Z
107             - error_pos[2]), new Orient(0, 0.707107,
108             -0.707107, 0), Robtarget_5.Robconf, new
109             ExtJoint(500));
110         Move1 = new UsualMoveOperation(TYPE_USUAL_MOVE.
111             MoveL, Robtarget_2, RobotMgrConstant.v1000,
112             currentTool, RobotMgrConstant.z0, wobjdata);
113         ROp.Add(new RapidOperation(OPERATION_TYPE.UMove
114             , Move1));
115     }
116 }
117 if (i > 3 * displacement / 4 && i < displacement)
118     // Final side of the square
119 {
120     Robtarget_2 = new RobTarget(new Pos(
121         Robtarget_5.Trans.X - error_pos[0], Robtarget_5.
122         Trans.Y - i - increment + 3 * displacement / 4 +
123         error_pos[1], Robtarget_5.Trans.Z - error_pos
124         [2]), new Orient(0, 0.707107, -0.707107, 0),
125         Robtarget_5.Robconf, new ExtJoint(500));
126     Move1 = new UsualMoveOperation(TYPE_USUAL_MOVE.
127         MoveL, Robtarget_2, RobotMgrConstant.v1000,
128         currentTool, RobotMgrConstant.z0, wobjdata);
129     ROp.Add(new RapidOperation(OPERATION_TYPE.UMove,
130         Move1));
131 }
132 RobMgr.DoSequence(ROp);
133 i = i + increment;
134 //Thread.Sleep(100);
135 }

```

```

114     sw_beacon.Close(); // Close the file
115     sw_pos_robot.Close(); // Close the file
116     sw_inclinaison_sensor.Close(); // Close the file
117     sw_angles_robot.Close(); // Close the file
118 }

```

Function B.4 Drawing movement of a square without compensation

```

1 // The robot draws a square, without the compensation system
  based on the values of the total station and the inclination
  sensor
2 // The function takes in arguments, the side of the square, the
  number of increment of the movement (a multiple of 4) and
  the RobMgr, that which the connection with the controller
3 public void SquareDrawing_WithoutCompensation(int side, int
  step, RobotMgr.RobotMgr RobMgr)
4 {
5     List<RapidOperation> ROp = new List<RapidOperation>();
      // List of operations
6     UsualMoveOperation Move1; // Linear movement
7     int displacement = side * 4; // Length of the total
      displacement
8     int increment = displacement / step; // The length
      of one incremental movement
9     int i = 0;
10
11     WobjData wobjdata = ParseWobjData(RobMgr); //
      Define the work object which is the wood board
12     ToolData currentTool = ParseToolData(RobMgr, "tPen");
      // Define the current tool object of the robot
13     wobjdata.oFrame.Trans = new Pos(100, 0, 0); //
      Translation vector to reach initial position of the
      square
14
15     StreamWriter sw_pos_robot_ref = new StreamWriter("C:\\Tests
      \\Pos_Robot_Ref.txt"); // Pass the filepath and
      filename to the StreamWriter Constructor for the
      reference position of the robot
16     StreamWriter sw_angles_robot_ref = new StreamWriter("C:\\
      Tests\\Angles_Robot_Ref.txt"); // Pass the
      filepath and filename to the StreamWriter Constructor
      for the reference orientation of the robot
17
18     RobTarget Robtarget_5 = new RobTarget(new Pos(), new Orient
      (0, 0.707107, -0.707107, 0), new ConfData(-1, 0, 0, 0),
      new ExtJoint(500)); // Initial target of the robot
19     Move1 = new UsualMoveOperation(TYPE_USUAL_MOVE.MoveL,
      Robtarget_5, RobotMgrConstant.v1000, currentTool,
      RobotMgrConstant.z0, wobjdata);

```

```

20  ROp.Add(new RapidOperation(OPERATION_TYPE.UMove, Move1));
21  RobMgr.DoSequence(ROp);           // Do the linear movement
    to the initial target
22  Thread.Sleep(2000);               // Wait to reach the initial
    position
23  UsualActionOperation accset = new UsualActionOperation(
    TYPE_USUAL_ACTION.AccSet, new AccelerationOperation(10,
    10));           // Smooth the movement of the robot
24  ROp.Add(new RapidOperation(OPERATION_TYPE.UAction, accset))
    ;
25
26  while (i <= displacement)         // Loop for the movement to
    draw the square
27  {
28      RobTarget RobT = RobMgr.GetLinearRobotPosition(
        ABBCoordinatorMgr.ABBCoordinateSystemType.World);
        // Return the current position of the robot
29      sw_pos_robot_ref.WriteLine(RobT.Trans.X.ToString() + ",
        " + RobT.Trans.Y.ToString() + ", " + RobT.Trans.Z.
        ToString());           // Write the current position of
        the robot in to the corresponding file
30      System.Numerics.Quaternion quaternion_robot = new
        System.Numerics.Quaternion((float)RobT.Rot.Q1, (
        float)RobT.Rot.Q2, (float)RobT.Rot.Q3, (float)RobT.
        Rot.Q4);
31      System.Numerics.Vector3 angles_robot_ref =
        RemoteObjectQCtrlRobot.ConverterToolBox.
        QuaternionToEuler(quaternion_robot);           // Convert
        the current quaternion of the robot in to the
        appropriate Euler angles
32      sw_angles_robot_ref.WriteLine(angles_robot_ref.X.
        ToString() + ", " + angles_robot_ref.Y.ToString() +
        ", " + angles_robot_ref.Z.ToString());           // Write
        the current angle orientation of the robot in to the
        corresponding file
33
34      if (i <= displacement / 4)     // First side of the
        square
35      {
36          if (i == displacement / 4)
37          {
38              Robtarget_5 = new RobTarget(new Pos(
                displacement / 4, 0, 0), new Orient(0,
                0.707107, -0.707107, 0), new ConfData(-1, 0,
                0, 0), new ExtJoint(500));
39          }
40          else
41          {
42              RobTarget Robtarget_2 = new RobTarget(new Pos(

```

```

        Robtarget_5.Trans.X + i + increment,
        Robtarget_5.Trans.Y, Robtarget_5.Trans.Z),
        Robtarget_5.Rot, Robtarget_5.Robconf, new
43      ExtJoint(500));
        Move1 = new UsualMoveOperation(TYPE_USUAL_MOVE.
        MoveL, Robtarget_2, RobotMgrConstant.v1000,
        currentTool, RobotMgrConstant.z0, wobjdata);
44      ROp.Add(new RapidOperation(OPERATION_TYPE.UMove
        , Move1));
45    }
46  }
47  if (i > displacement / 4 && i <= displacement / 2)
    // Second side of the square
48  {
49    if (i == displacement / 2)
50    {
51      Robtarget_5 = new RobTarget(new Pos(
        displacement / 4, displacement / 4, 0), new
        Orient(0, 0.707107, -0.707107, 0), new
        ConfData(-1, 0, 0, 0), new ExtJoint(500));
52    }
53    else
54    {
55      RobTarget Robtarget_2 = new RobTarget(new Pos(
        Robtarget_5.Trans.X, Robtarget_5.Trans.Y + i
        + increment - displacement / 4, Robtarget_5
        .Trans.Z), Robtarget_5.Rot, Robtarget_5.
        Robconf, new ExtJoint(500));
56      Move1 = new UsualMoveOperation(TYPE_USUAL_MOVE.
        MoveL, Robtarget_2, RobotMgrConstant.v1000,
        currentTool, RobotMgrConstant.z0, wobjdata);
57      ROp.Add(new RapidOperation(OPERATION_TYPE.UMove
        , Move1));
58    }
59  }
60  if (i > displacement / 2 && i <= 3 * displacement / 4)
    // Third side of the square
61  {
62    if (i == 3 * displacement / 4)
63    {
64      Robtarget_5 = new RobTarget(new Pos(0,
        displacement / 4, 0), new Orient(0,
        0.707107, -0.707107, 0), new ConfData(-1, 0,
        0, 0), new ExtJoint(500));
65    }
66    else
67    {
68      RobTarget Robtarget_2 = new RobTarget(new Pos(
        Robtarget_5.Trans.X - i - increment +

```

```

        displacement / 2, Robtarget_5.Trans.Y,
        Robtarget_5.Trans.Z), Robtarget_5.Rot,
        Robtarget_5.Robconf, new ExtJoint(500));
69     Move1 = new UsualMoveOperation(TYPE_USUAL_MOVE.
        MoveL, Robtarget_2, RobotMgrConstant.v1000,
        currentTool, RobotMgrConstant.z0, wobjdata);
70     ROp.Add(new RapidOperation(OPERATION_TYPE.UMove
        , Move1));
71     }
72 }
73 if (i > 3 * displacement / 4 && i < displacement)
    // Final side of the square
74 {
75     RobTarget Robtarget_2 = new RobTarget(new Pos(
        Robtarget_5.Trans.X, Robtarget_5.Trans.Y - i -
        increment + 3 * displacement / 4, Robtarget_5.
        Trans.Z), Robtarget_5.Rot, Robtarget_5.Robconf,
        new ExtJoint(500));
76     Move1 = new UsualMoveOperation(TYPE_USUAL_MOVE.
        MoveL, Robtarget_2, RobotMgrConstant.v1000,
        currentTool, RobotMgrConstant.z0, wobjdata);
77     ROp.Add(new RapidOperation(OPERATION_TYPE.UMove,
        Move1));
78     }
79     RobMgr.DoSequence(ROp);
80     i = i + increment;
81     //Thread.Sleep(100);
82 }
83 sw_pos_robot_ref.Close();
84 sw_angles_robot_ref.Close();
85 }

```

Function B.5 Drawing movement following the beacon

```

1 // Move the robot in the same way as the beacon of the total
  station. The robot can reproduce the path of the beacon
2 // The function takes in arguments, the initial position of the
  robot and a RobMagr, the connection with the controller of
  the robot
3 public void Draw_With_Beacon(RobotMgr.RobotMgr RobMgr,
  NetworkStream stream)
4 {
5     List<RapidOperation> ROp = new List<RapidOperation>();
        // List of operations
6     UsualMoveOperation Move1;           // Linear movement
7     int i = 0;
8
9     //Thread.Sleep(20000);           // Wait to be in position
10

```

```

11 StreamWriter sw_pos_robot = new StreamWriter("C:\\Tests\\
    Pos_Robot_Beacon.txt"); // Pass the filepath and
    filename to the StreamWriter Constructor for the
    position of the robot
12 StreamWriter sw_pos_beacon = new StreamWriter("C:\\Tests\\
    Pos_Beacon.txt"); // Pass the filepath and
    filename to the StreamWriter Constructor for the
    position of the beacon
13
14 WobjData wobjdata = ParseWobjData(RobMgr); ; //
    Define the work object which is the wood board
15 ToolData currentTool = ParseToolData(RobMgr, "tPen");
16 wobjdata.oFrame.Trans = new Pos(100, 0, 0); //
    Translation vector to reach initial position on the
    paper
17
18 RobTarget Robtarget_5 = new RobTarget(new Pos(), new Orient
    (0, 0.707107, -0.707107, 0), new ConfData(-1, 0, 0, 0),
    new ExtJoint()); // Initial target of the robot
19 Move1 = new UsualMoveOperation(TYPE_USUAL_MOVE.MoveL,
    Robtarget_5, RobotMgrConstant.v500, currentTool,
    RobotMgrConstant.z0, wobjdata);
20 ROp.Add(new RapidOperation(OPERATION_TYPE.UMove, Move1));
21 RobMgr.DoSequence(ROp); // Do the linear movement
    to the initial target
22 Thread.Sleep(2000); // Wait to reach the initial
    position
23
24 double[] positions_ref = new double[3]; // Initial
    position of the beacon of the total station, which is
    taken as the reference position before the movement
25 positions_ref = RefData_TotalStation(stream); //
    Return the first position send by the total station,
    which is the reference
26 double[] positions = new double[3];
27 double[] angles = new double[3];
28 sw_pos_beacon.WriteLine(positions_ref[0].ToString() + ", "
    + positions_ref[1].ToString() + ", " + positions_ref[2].
    ToString()); // Write the reference position of the
    robot in to the corresponding file
29
30 UsualActionOperation accset = new UsualActionOperation(
    TYPE_USUAL_ACTION.AccSet, new AccelerationOperation(10,
    10)); // Smooth the movement of the robot
31 ROp.Add(new RapidOperation(OPERATION_TYPE.UAction, accset))
    ;
32
33 while (i < 150) // Loop for the drawing movement
34 {

```

```

35     positions = DataStream_TotalStation(positions_ref[0],
36                                         positions_ref[1], positions_ref[2], stream);
37
38     RobTarget RobT = RobMgr.GetLinearRobotPosition(
39         ABBControllerMgr.ABBCoordinateSystemType.World);
40         // Return the current position of the robot
41     sw_pos_robot.WriteLine(RobT.Trans.X.ToString() + ", " +
42         RobT.Trans.Y.ToString() + ", " + RobT.Trans.Z.
43         ToString()); // Write the current position of
44         the robot in to the corresponding file
45     sw_pos_beacon.WriteLine(positions[0].ToString() + ", "
46         + positions[1].ToString() + ", " + positions[2].
47         ToString()); // Write the current position of
48         the beacon in to the corresponding file
49
50     RobTarget Robtarget_2 = new RobTarget(new Pos(
51         Robtarget_5.Trans.X - positions[0], Robtarget_5.
52         Trans.Y + positions[1], Robtarget_5.Trans.Z -
53         positions[2]), Robtarget_5.Rot, Robtarget_5.Robconf,
54         new ExtJoint());
55     Move1 = new UsualMoveOperation(TYPE_USUAL_MOVE.MoveL,
56         Robtarget_2, RobotMgrConstant.v500, currentTool,
57         RobotMgrConstant.z0, wobjdata);
58     ROp.Add(new RapidOperation(OPERATION_TYPE.UMove, Move1)
59         );
60     RobMgr.DoSequence(ROp);
61     //Thread.Sleep(100);
62     i++;
63     Console.WriteLine(i);
64 }
65 sw_pos_beacon.Close();
66 sw_pos_robot.Close();
67 }

```

Function B.6 Connection with the controller of the total station

```
1 // Connection to the controller of the total station, with its
  Ip address and its port
2 // Return a NetworkStream which is the stream to receive and
  send data to the controller of the total station
3 public NetworkStream Connect_TotalStation(string Ip, int port)
4 {
5     TcpClient client = new TcpClient(Ip, port);          //
      Create a new instance of the class TcpClient with its Ip
      address and port
6     NetworkStream stream = client.GetStream(); // Create a
      stream with the TCP communication protocol to receive
      and send data
7     return stream;
8 }
```

Function B.7 Reference position of the beacon

```
1 // Once the controller is connected, the 20 first data are
  collected, and that defines the reference position of the
  beacon seen by the total station
2 // The returned data are the X, Y and Z reference position with
  the installation of the total station as reference
3 public double[] RefData_TotalStation(NetworkStream stream)
4 {
5     int m = 0;
6
7     while (m < 20)          // Loop to exclude the 20 first data,
      which could lead to a format error
8     {
9         Byte[] data = new Byte[256];
10        String responseData = String.Empty;
11
12        Int32 bytes = stream.Read(data, 0, data.Length);
13        // Collect the data on the stream
14        responseData = System.Text.Encoding.ASCII.GetString(
15            data, 0, bytes);          // Change the bytes
16        // collected on the stream into ASCII representation
17        Console.WriteLine("Received: {0}", responseData);
18        m++;
19    }
20
21    Byte[] data_ref = new Byte[128];
22    String responseData_ref = String.Empty;
23    Int32 bytes_ref = stream.Read(data_ref, 0, data_ref.Length)
24        ;          // Collect the data on the stream
25    responseData_ref = System.Text.Encoding.ASCII.GetString(
26        data_ref, 0, bytes_ref);          // Change the bytes
```



```

        collected on the stream into ASCII representation
22
23 double[] positions_ref = new double[3];          //
        Determine the reference position based on the values
        send by the controller of the total station
24 string[] split_ref = responseData_ref.Split('\n');
        // Split the collected data into separate
        doubles
25 double X_ref = double.Parse(split_ref[1].Remove(0, 3),
        CultureInfo.InvariantCulture);          // X position
        returned by the controller of the total station
26 double Y_ref = double.Parse(split_ref[2].Remove(0, 3),
        CultureInfo.InvariantCulture);          // Y position
        returned by the controller of the total station
27 double Z_ref = double.Parse(split_ref[3].Remove(0, 3),
        CultureInfo.InvariantCulture);          // Z position
        returned by the controller of the total station
28
29 positions_ref[0] = X_ref * 1000;          // X reference
        position
30 positions_ref[1] = Y_ref * 1000;          // Y reference
        position
31 positions_ref[2] = Z_ref * 1000;          // Z reference
        position
32
33 Console.WriteLine("Reference in X: {0}", positions_ref[0]);
34 Console.WriteLine("Reference in Y: {0}", positions_ref[1]);
35 Console.WriteLine("Reference in Z: {0}", positions_ref[2]);
36
37 return positions_ref;
38 }

```

Function B.8 Recuperation of the data coming from the controller of the total station

```

1 // Allow the recuperation of the data on the controller of the
  total station
2 // Return the displacement in X, Y and Z, with respect to the
  reference position
3 public double[] DataStream_TotalStation(double X_ref, double
  Y_ref, double Z_ref, NetworkStream stream)
4 {
5     double[] error_pos = new double[3];          // Returned data
6     try
7     {
8         Byte[] data = new Byte[1024];
9         String responseData = String.Empty;
10
11         Int32 bytes = stream.Read(data, 0, data.Length);
            // Collect the data on the stream

```

```

12     responseData = System.Text.Encoding.ASCII.GetString(
13         data, 0, bytes);           // Change the bytes
14                                     // collected on the stream into ASCII representation
15     Console.WriteLine("Received: {0}", responseData);
16
17     string[] split = responseData.Split('\n');           //
18                                     // Split the collected data into separate doubles
19     double X = double.Parse(split[1].Remove(0, 3),
20         CultureInfo.InvariantCulture);           // X position
21     double Y = double.Parse(split[2].Remove(0, 3),
22         CultureInfo.InvariantCulture);           // Y position
23     double Z = double.Parse(split[3].Remove(0, 3),
24         CultureInfo.InvariantCulture);           // Z position
25
26     error_pos[0] = X_ref - X * 1000;           // Error in the
27                                     // X position with respect to the X reference position
28     error_pos[1] = Y_ref - Y * 1000;           // Error in the
29                                     // Y position with respect to the X reference position
30     error_pos[2] = Z_ref - Z * 1000;           // Error in the
31                                     // Z position with respect to the X reference position
32
33     Console.WriteLine("Error in X : {0}", error_pos[0]);
34     Console.WriteLine("Error in Y : {0}", error_pos[1]);
35     Console.WriteLine("Error in Z : {0}", error_pos[2]);
36 }
37
38 catch (ArgumentOutOfRangeException)           // Catch the
39     possible error in the data transfer
40 {
41     Console.WriteLine("Not good format");
42 }
43
44 catch (FormatException)           // Catch the
45     possible error in the data transfer
46 {
47     Console.WriteLine("Not good format");
48 }
49
50 return error_pos;           // Displacement in X, Y and Z, with
51                             // respect to the reference position
52 }

```

Function B.9 Disconnection with the controller of the total station

```

1 // Disconnect the total station controller
2 public void Disconnect_TotalStation(NetworkStream stream)
3 {
4     stream.Close();           // Close the communication with
5                               // the controller of the total station
6 }

```

Function B.10 Connection with the inclination sensor

```
1 // Connection to inclination sensor, with its Ip address and
  its port number
2 // Return a ModbusClient to have access to the registers of the
  inclination sensor, where the data is located
3 public ModbusClient Connect_InclinationSensor(string Ip, int
  port)
4 {
5     ModbusClient modbusClient = new ModbusClient(Ip, port);
        // Create a new instance of the class
        ModbusClient with its Ip address and port
6     modbusClient.Connect(); // Connection to the Modbus
7     return modbusClient;
8 }
```

Function B.11 Recuperation of the data coming from the inclination sensor

```
1 // Allow the recuperation of the data on the InputRegisters of
  the inclination sensor based on the Modbus communication
  protocol
2 // Return the angles in degrees given by the inclination sensor
3 public double[] DataStream_InclinationSensor(ModbusClient
  modbusClient)
4 {
5     double[] angles = new double[2]; // Returned data
6     try
7     {
8         if (modbusClient == null)
9         {
10             angles[0] = angles[1] = 0;
11             return angles;
12         }
13         if (modbusClient.Connected == false)
14             modbusClient.Connect();
15
16         if (modbusClient.Connected)
17         {
18             try
19             {
20                 List<int> readInputRegisterX = null;
21                 List<int> readInputRegisterY = null;
22
23                 readInputRegisterX = modbusClient.
                    ReadInputRegisters(200, 1).ToList(); //
                    Read the Input Register number 200, where
                    the value of the X angle of the inclination
                    sensor is
24                 readInputRegisterY = modbusClient.
```

```

        ReadInputRegisters(201, 1).ToList(); //
        Read the Input Register number 201, where
        the value of the Y angle of the inclination
        sensor is

25
26         if (readInputRegisterX != null &&
            readInputRegisterX.Count > 0)
27             angles[0] = (double)readInputRegisterX[0] *
                1 / 100; // The collected data is
                        an integer with the precision of 0.01,
                        the number is divided by 100 to have the
                        good value
28         else
29             Console.WriteLine("Input Register X is
                empty");

30
31         if (readInputRegisterY != null &&
            readInputRegisterY.Count > 0)
32             angles[1] = (double)readInputRegisterY[0] *
                1 / 100; // The collected data is
                        an integer with the precision of 0.01,
                        the number is divided by 100 to have the
                        good value
33         else
34             Console.WriteLine("Input Register Y is
                empty");

35
36         if (angles.Where(val => val == 0).Count() != 0)
37             modbusClient.Disconnect();
38     }
39     catch (Exception ex) // Catch the possible
        error in the data transfer
40     {
41         Console.WriteLine("Read Input Register
            Exception : " + ex.Message);
42         modbusClient.Disconnect();
43     }
44 }
45 Console.WriteLine("Value of the two InputRegisters " +
    "X : " + angles[0].ToString() + " and Y : " +
    angles[1].ToString());
46 }
47 catch (ArgumentOutOfRangeException) // Catch the
    possible error in the data transfer
48 {
49     Console.WriteLine("Not good format");
50 }
51 catch (FormatException) // Catch the
    possible error in the data transfer

```

```

52     {
53         Console.WriteLine("Not good format");
54     }
55     return angles;
56 }

```

Function B.12 Disconnection with the inclination sensor

```

1 // Disconnect the inclination sensor
2 public void Disconnect_InclinationSensor(ModbusClient
    modbusClient)
3 {
4     modbusClient.Disconnect();           // Disconnect from the
        Modbus communication
5 }

```

Function B.13 Convert Euler angles into a quaternion

```

1 // Convert the angles given by the inclination sensor in to
    quaternions
2 // Return a quaternion, based on the given angles
3 public System.Numerics.Quaternion Quaternions_InclinationSensor
    (WobjData wobjdata, double[] angles_incl_sensor)
4 {
5     System.Numerics.Quaternion quaternion_sensor = new System.
        Numerics.Quaternion();
6     System.Numerics.Vector3 angles_sensor = new System.Numerics
        .Vector3((float)angles_incl_sensor[0], (float)
        angles_incl_sensor[1], 0);           // angles from the sensor
7
8     quaternion_sensor = RemoteObjectQCtrlRobot.ConverterToolBox
        .EulerToQuaternion(angles_sensor);   // Function to
        convert Euler angles into quaternions
9
10    Console.WriteLine("Sensor : " + angles_sensor);
11
12    return quaternion_sensor;
13 }

```

Function B.14 Recuperation of the appropriated work object

```

1 // Retrieve the WobjData defined in the RAPID instructions,
    which is the wood board in this application
2 // Return a WobjData
3 public WobjData ParseWobjData(RobotMgr.RobotMgr RobMgr)
4 {
5     WobjData RapidWobj = new WobjData();

```

```

6     string[] strWobj = RobMgr.CtrlMgr.GetABBCtrlRapidData("
          T_ROB1", "MainModule", "palette");
7     string[] split = strWobj[0].Split(',');           // Split
          the collected data
8     {
9         RapidWobj.Robhold = bool.Parse(split[0].Remove(0, 1));
10        RapidWobj.ufProg = bool.Parse(split[1]);
11        RapidWobj.ufMec = split[2];
12        RapidWobj.uFrame.Trans = new Pos(double.Parse(split[3].
          Remove(0, 2), CultureInfo.InvariantCulture), double.
          Parse(split[4], CultureInfo.InvariantCulture),
          double.Parse(split[5].Remove(8), CultureInfo.
          InvariantCulture));
13        RapidWobj.uFrame.Rot = new Orient(double.Parse(split
          [6].Remove(0, 1), CultureInfo.InvariantCulture),
          double.Parse(split[7], CultureInfo.InvariantCulture)
          , double.Parse(split[8], CultureInfo.
          InvariantCulture), double.Parse(split[9].Remove(9),
          CultureInfo.InvariantCulture));
14        RapidWobj.oFrame.Trans = new Pos(double.Parse(split
          [10].Remove(0, 2), CultureInfo.InvariantCulture),
          double.Parse(split[11], CultureInfo.InvariantCulture
          ), double.Parse(split[12].Remove(1), CultureInfo.
          InvariantCulture));
15        RapidWobj.oFrame.Rot = new Orient(double.Parse(split
          [13].Remove(0, 1), CultureInfo.InvariantCulture),
          double.Parse(split[14], CultureInfo.InvariantCulture
          ), double.Parse(split[15], CultureInfo.
          InvariantCulture), double.Parse(split[16].Remove(1),
          CultureInfo.InvariantCulture));
16    }
17    return RapidWobj;
18 }

```

Function B.15 Recuperation of the appropriated tool

```

1 // Retrieve the ToolData defined in the RAPID instructions,
          which is the tool with the pen attached on it
2 // Return a ToolData
3 public ToolData ParseToolData(RobotMgr.RobotMgr RobMgr, string
          toolName)
4 {
5     ToolData rapidTool = new ToolData();
6     string[] strTool = RobMgr.CtrlMgr.GetABBCtrlRapidData("
          T_ROB1", "MainModule", toolName);
7     string[] split = strTool[0].Split(',');           // Split
          the collected data
8     {
9         rapidTool.Robhold = bool.Parse(split[0].Remove(0, 1));

```

```

10         rapidTool.tFrame.Trans = new Pos(double.Parse(split[1].
            Remove(0, 2), CultureInfo.InvariantCulture), double.
            Parse(split[2], CultureInfo.InvariantCulture),
            double.Parse(split[3].Remove(3, 1), CultureInfo.
            InvariantCulture));
11         rapidTool.tFrame.Rot = new Orient(double.Parse(split
            [4].Remove(0, 1), CultureInfo.InvariantCulture),
            double.Parse(split[5], CultureInfo.InvariantCulture)
            , double.Parse(split[6], CultureInfo.
            InvariantCulture), double.Parse(split[7].Remove(1,
            2), CultureInfo.InvariantCulture));
12         rapidTool.tLoad = new LoadData();
13     }
14     return rapidTool;
15 }

```

Function B.16 Polynomial regression of order 3

```

1 public Vector<double> Poly_Regression(double[] X, double[] Y)
2 {
3     int order = 3;
4     double sum_X_1 = 0.0;
5     double sum_X_2 = 0.0;
6     double sum_X_3 = 0.0;
7     double sum_X_4 = 0.0;
8     double sum_Y_1 = 0.0;
9     double sum_Y_2 = 0.0;
10    double sum_Y_3 = 0.0;
11    //double [,] matrix = new double[order, order]; //
        polynomial of order 2
12    //double[] vector = new double[order];
13    Matrix<double> matrix = Matrix.Create<double>(order, order)
        ;
14    Vector<double> vector = Vector.Create<double>(order);
15    Vector<double> sol = Vector.Create<double>(order);
16
17    for (int i = 0; i < X.GetLength(0); i++)
18    {
19        sum_X_1 = sum_X_1 + X[i];
20        sum_X_2 = sum_X_2 + Math.Pow(X[i], 2);
21        sum_X_3 = sum_X_3 + Math.Pow(X[i], 3);
22        sum_X_4 = sum_X_4 + Math.Pow(X[i], 4);
23        sum_Y_1 = sum_Y_1 + Y[i];
24        sum_Y_2 = sum_Y_2 + X[i] * Y[i];
25        sum_Y_3 = sum_Y_3 + Math.Pow(X[i], 2) * Y[i];
26    }
27    matrix[0, 0] = X.GetLength(0);
28    matrix[0, 1] = sum_X_1;
29    matrix[0, 2] = sum_X_2;

```

```
30     matrix[1, 0] = sum_X_1;
31     matrix[1, 1] = sum_X_2;
32     matrix[1, 2] = sum_X_3;
33     matrix[2, 0] = sum_X_2;
34     matrix[2, 1] = sum_X_3;
35     matrix[2, 2] = sum_X_4;
36     vector[0] = sum_Y_1;
37     vector[1] = sum_Y_2;
38     vector[2] = sum_Y_3;
39     sol = matrix.Solve(vector);
40     Console.WriteLine(sol);
41     return sol;
42 }
```

Bibliography

- [1] CNBC, “How autonomous robots are changing construction.” <https://www.youtube.com/watch?v=zBvvb0Lq3t0>, November 30 , 2020.
- [2] Tractica, “Construction robotics market forecast to reach \$ 226m by 2025.” <https://www.forconstructionpros.com/profit-matters/news/21069111/construction-robotics-market-forecast-to-reach-226m-by-2025>, May 16, 2019.
- [3] A. B. M. N. M. R. G. Carra, A. Argiolas, “Robotics in the construction industry: state of the art and future opportunities,” 2018.
- [4] T. Yoshida, “A short history of construction robots research development in a japanese company,” 2006.
- [5] M. Mason, “Crafting decisions,” May 2020.
- [6] C. Balaguer, “Nowadays trend in robotics and automation in construction industry : Transistion from hard to soft roobotics,” 2004.
- [7] C. Balaguer and M. Abderrahim, “Trend in robotics and automation in construction,” 2008.
- [8] A. Simpson, “Wearable robotics aid construction workers.” <https://phys.org/news/2008-02-wearable-robotics-aid-workers.html>, 18/2/2008.
- [9] J. Thilmany, “Exoskeletons for construction workers are marching on-site.” <https://constructible.trimble.com/construction-industry/exoskeletons-for-construction-workers-are-marching-on-site>, 27/2/2019.
- [10] F. Robotics, “Fbr website.” <https://www.fbr.com.au/view/hadrian-x>, 2018.

- [11] C. Robotics, “Construction robotics website.” <https://www.construction-robotics.com/sam100/>, 2019.
- [12] P. Information, “Volvo ce unveils the next generation of its electric load carrier concept.” <https://www.volvoce.com/global/en/news-and-events/press-releases/2017/conexpo-vegas-2017/volvo-ce-unveils-the-next-generation-of-its-electric-load-carrier-concept/>, 3/9/2017.
- [13] Caterpillar, “Cat minestar solutions for loading and hauling.” [https://www.cat.com/en\\$\\$_US/by-industry/mining/surface-mining/surface-technology/loading-hauling.html](https://www.cat.com/en$$_US/by-industry/mining/surface-mining/surface-technology/loading-hauling.html), 2021.
- [14] E. Ackerman, “Ai startup using robots and lidar to boost productivity on construction sites.” <https://spectrum.ieee.org/autaton/robotics/industrial-robots/doxel-ai-startup-using-lidar-equipped-robots-on-construction-sites>, 24/01/2018.
- [15] G. Kumarak, “Built robotics’ massive construction excavator drives itself.” <https://techcrunch.com/2019/03/29/built-robotics-massive-construction-excavator-drives-itself/>, 29/03/2019.
- [16] M. Monnart, “Développement et validation d’un système de guidage autonome pour l’assemblage d’éléments drone-compatibles,” 2019.
- [17] Okibo, “Okibo website.” <https://okibo.com/our-robot/>, 2021.
- [18] K. Shaw, “Ulc robotics, sgn to develop all-electric excavation robot.” <https://www.roboticsbusinessreview.com/energy-mining/ulc-robotics-sgn-to-develop-all-electric-excavation-robot/>, January 21, 2020.
- [19] K. Matthews, “How robotics and automation are changing the construction industry.” <https://www.roboticstomorrow.com/article/2019/04/how-robotics-and-automation-are-changing-the-construction-industry/13527>, 16/4/2019.
- [20] ABB, “Abb website - industrial robots - irb 7600.” <https://new.abb.com/products/robotics/industrial-robots/irb-7600>, 2021.
- [21] A. Robotics, “Product specification : Irb 7600,” 2004-2021.

- [22] Manitou, “Technical sheet : Mrt-x 2470 privilege plus,” 2021.
- [23] T. Inc., “Trimble website - total station sx10.” <https://fr.geospatial.trimble.com/products-and-solutions/trimble-sx10>, 2021.
- [24] T. Inc., “Trimble website - trimble tsc7 controller.” <https://geospatial.trimble.com/products-and-solutions/trimble-tsc7>, 2021.
- [25] T. Inc., “Trimble website - prisms targets.” <https://geospatial.trimble.com/Optical-Accessories>, 2021.
- [26] R. Deakin, “Edm: Notes on electronic distance measurement,” 17-May-2016.
- [27] I. electronic, “Ifm website - inclination sensor jn2201.” <https://www.ifm.com/fr/en/product/JN2201>, 2021.
- [28] Fortinet, “What is a transmission control protocol tcp/ip model?.” <https://www.fortinet.com/resources/cyberglossary/tcp-ip>, 2021.
- [29] Trimble, “Trimble access, help portal, data output.” https://help.trimblegeospatial.com/TrimbleAccess/2019.00/en/GDM_Data_Output.htm?tocpath=Survey%20setup%7CConventional%20surveys%7CInstrument%20functions%20and%20settings%7C_____13, 2018-2019.
- [30] I. Shields, “Communications and protocols used in industrial automation.” <https://www.industrialshields.com/blog/arduino-industrial-1/post/communications-and-protocols-used-in-industrial-automation-231>, 2021.
- [31] A. Robotics, “Technical reference manual : Rapid instructions, functions and data types,” 2004.
- [32] M. Ben-Ari, “A tutorial on euler angles and quaternions,” 2014-17.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl