UNIVERSIDADE D
COIMBRA

Pedro Manuel Videira Marques

RAPID DEVELOPMENT OF A HIGH-LEVEL ROBOT CONTROLLER FOR AUTOMATED TOUCH-SCREEN TESTING

Dissertação no âmbito do Mestrado Integrado em Engenharia Mecânica, ramo de Produção e Projeto, orientada pelo Professor Doutor Joaquim Norberto Cardoso Pires da Silva e apresentada ao Departamento de Engenharia Mecânica da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Outubro de 2021

1 2 9 0

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

Rapid Development of a High-Level Robot Controller for Automated Touch-Screen Testing

Submitted in Partial Fulfilment of the Requirements for the Master's Degree in Mechanical Engineering in the speciality of Production and Project

Desenvolvimento Rápido de um Controlador Robótico de Alto Nível para Teste Automatizados de Ecrãs Tácteis

Author

Pedro Manuel Videira Marques

Advisors

Professor Doutor Joaquim Norberto Cardoso Pires da Silva

Professor Doutor Ricardo Nuno Madeira Soares Branco

Jury

President	**Professora Doutor Ana Paula Bettencourt Martins Amaro** **Professora Auxiliar da Universidade de Coimbra**
Vowels	**Professora Doutor Trayana Tankova** **Professora Auxiliar da Universidade de Coimbra**
	Mestre Carlos Zhu **Investigador da Universidade de Coimbra**
Advisor	**Professor Doutor Ricardo Nuno Madeira Soares Branco** **Professor Auxiliar da Universidade de Coimbra**

Institutional Collaboration

speedgoat **Speedgoat**

Coimbra, September, 2021

“Intelligence is the ability to adapt to change.”

Stephen Hawking

In memory of Joaquim Norberto Pires

ACKNOWLEDGEMENTS

Before anything else, I want to thank Professor Norberto Pires, the supervisor of this thesis. Although he was a very busy man, he always found the time to explained to me, in a way that only a person of his genius would, how simple a solution can be.

Also, I want to write down a huge thanks to the people at Speedgoat, Daniel Hediger, Daniel Fonte, and Manuel Fedou, who gave me the opportunity to discover my new passion and for sharing with me all their knowledge and enthusiasm.

Advice given by Carlos Zhu and João Costa, two investigators that shared their time in the lab with me. They were with me most of the time, in person or not, and allowed me to do, what was necessary to advance, experimentally, in my research. In every doubt I had, they were always available to clarify me.

I must extend this acknowledgment list to my father, José, for his advice and for showing me that anything is possible with a little bit of work and commitment. To my mother, brother and girlfriend, for always being there, giving me the strength to lead this dissertation till the end.

Finally, to all my friends and colleagues, for all the adventures we lived and for all the helped you provided me during this last five years. Without these amazing people, I could never understand nor see the beauty of this city.

Abstract

This dissertation addresses the automation problem for a robot arm utilized for touch screen testing by following the model-based design methodology for the development of a high-level controller.

With recent studies suggesting a substantial growth in the usage of touch screen technology in the next decade, this can put at risk the production of touch screens utilized in a diversify field of sectors. In this research, it is presented a new alternative that will allow to support this increase in product demand, while considering other factors such as sustainability and efficiency.

The manipulator controller here presented was developed in two stages. The first one, also known as model-in-the-loop, consisted in the development and usage of a digital twin model utilized in a virtual environment in order to simulate the robot behavior. The simulated results, allowed to observe and correct the controller's behavior, preventing from programming flaws. In the second one, it was utilized a dedicated hardware device, provided by Speedgoat, that allowed us to verify the quality of our manipulator and its capacity to operate in a real-time environment. This second stage, called rapid control prototyping, was accomplish by connecting all the individual components to each other and therefore completing the closed loop system.

For the development of our autonomous and conscient controller, they were also developed computer vision techniques with a special focus on the Optical Character Recognition (OCR).

Keywords Digital Twin, Real-Time Simulation, Rapid Control Prototyping, MATLAB, Computer Vision, Touchscreen Testing.

Resumo

Esta dissertação propõe abordar o problema de automatização de um braço robótico utilizado para testagem de ecrãs táteis, utilizando como base a metodologia de *model-based design*, para o desenvolvimento de um controlador de alto nível.

Com estudos recentes sugerindo um crescimento substancial do uso da tecnologia tátil na próxima década, isto pode colocar em risco a produção de ecrãs táteis necessários em diversos sectores. Nesta investigação, é apresentada uma nova alternativa que permitirá não só suportar este aumento na demanda de produtos, como também tendo em consideração outros fatores como por exemplo a sustentabilidade e eficiência.

O controlador aqui apresentado foi desenvolvido através de duas etapas. A primeira, também conhecida como *model-in-the-loop*, consistiu no desenvolvimento e uso de um gémeo digital, utilizado num ambiente virtual, que permitiu simular o comportamento do robô. Os resultados obtidos das simulações, permitiram observar e corrigir as falhas de programação existentes no comportamento do manipulador. Na segunda fase, foi utilizado um dispositivo especializado, fornecido pela Speedgoat, que nos permitiu verificar e analisar a qualidade do nosso sistema assim como a sua capacidade de operar num ambiente de simulação de tempo real. Esta segunda fase, denominada por *rapid control prototyping*, foi obtida através da ligação de cada um dos componentes completando assim o nosso *loop* do sistema.

Para o desenvolvimento do nosso controlador autónomo e consciente foram também desenvolvidas técnicas de visão computacional com um foco especial no *Optical Character Recognition* (OCR).

Palavras-chave: Gémeo Digital, Simulação em Tempo Real, Prototipagem Rápida, MATLAB, Visão Computacional, Testagem Ecrãs Táteis.

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF SIMBOLS AND ACRONYMS/ ABBREVIATIONS

List of Symbols

$\boldsymbol{J_A}$ – Analytic Jacobian

$\boldsymbol{J_G}$ – Geometric Jacobian

$\boldsymbol{v}$ – Linear velocity of the end-effector

$\dot{\boldsymbol{\theta}}$ – Angular velocity vector for the joint rotation

$\dot{\boldsymbol{p}}$ – Linear velocity vector for the end-effector

$\boldsymbol{w}$ – Angular velocity vector for the end-effector

$\dot{\boldsymbol{\phi}}$ – Change of orientation of the end-effector in frame Ø

Acronyms/Abbreviations

2D – Two-Dimensions

3D – Three-Dimensions

ABS – Acrylonitrile butadiene styrene

AGV – Automated Guided Vehicle

AI – Artificial Intelligence

BFGS – Broyden-Fletcher-Goldfarb-Shanno

CAD – Computer-Aided Design

CAGR – Compound Annual Growth Rate

CBDM – Cloud-Base Design and Manufacturing

DEM – *Departamento de Engenharia Mecânica*

DH – Denavit-Hartenberg

DOF – Degrees of Freedom

DPI – Dots per Inch

FCTUC – *Faculdade de Ciências e Tecnologia da Universidade de Coimbra*

FPGA – Field-Programmable Gate Array

FRF – Flange Reference Frame

GUI – Graphical User Interface

HIL – Hardware-in-Loop

HMI – Human-Machine Interaction

I/O – Input/Output

IAT – Image Acquisition Toolbox

IBVS – Image-Based Visual Servoing

IoT – Internet of Things

IPT – Image Processing Toolbox

MBD – Model-Based Design

MIL –Model-in-Loop

PBVS – Position-Based Visual Servoing

PLA – Polylactic Acid

PLC - Programmable Logic Controller

RAM – Random Access Memory

RCP – Rapid Control Prototyping

RGB – Red Green Blue

RST – Robotic System Toolbox

SDLC – Software Development Life Cycle

SIL – Software-in-Loop

SLERP – Spherical Linear Interpolation

SM – Smart Manufacturing

TCP/IP – Transmission Control Protocol/Internet Protocol

TRF – Tool Reference Frame

URDF – Unified Robotic Description Format

VS – Visual Servoing

1. INTRODUCTION

In the twenty-first century the number of automated processes has grown substantially with companies always looking for new ways to improve efficiency.

The theme of this dissertation focusses on the development of a high-level real-time supervisory robot manipulator that resorts to computer vision techniques, digital twins, and on a new ground-breaking methodology known as MBD (Model-Based Design) to achieve an efficient yet inexpensive method to test touch screens.

Model-based design (MBD) played an important role in this research since it allowed to expedite the development of our embedded software, by eliminating manual coding that is generally error-prone and by enabling test automation that in turn accelerated testing and development.

Through testing the models in the different phases of the project from Model-in-the-loop (MIL) to Rapid Control Prototyping (RCP) we were able to continuously improve the functionalities of the controller.

The advantages of this methodology became apparent when the planned robot was not available duo to unforeseen circumstances, leading to a change in the robot necessary for the practical experiments. However, with just a few modifications we were able to remain with the same Simulink model (controller) we were working on. The robots used for this research were the Meca500 (planned) and the Kassow KR810 (utilized). Their characteristics will be discussed further on Chapter 2 and Chapter 4.

This dissertation was integrated in a collaboration agreement between Speedgoat, a Swiss company that specializes in state-of-the-art systems for real-time testing, and the Industrial Robotics Laboratory of the University of Coimbra Department of Mechanical Engineering, a laboratory with a strong focus on innovative industrial projects. With this partnership Speedgoat intends to design an up-and-running industrial solution for their clients.

This chapter describes in more depth the aim of this project, the advantages of this methodology alongside the hardware provided, while introducing the Speedgoat

company. The chapter concludes by presenting the main objectives and the dissertation's structure.

1.1. Motivation

The aim of this dissertation is to design and develop a high-level robot controller to perform automated tests in touch screen devices. This controller makes use of some of the most advance techniques in software development and computer vision with the goal to innovate and increase the automation level in this industry field. A schematic of the implemented system can be seen in Figure 4.1.

The commonly used method consists of the usage of a vision guidance camera. This research intends to present a solution capable of eliminating the need for a camera, without losing any information or perception, with the aim of achieving sustainability.

More than never, we need to find new solutions that not only improve a product's quality and durability, but at the same time reduce the amount of electronic waste. The Industry 4.0 aims to solve this problem by creating a connected industry where all process is horizontally (all parts of the supply chain are connected) and vertically (connecting operations at production level with information at the enterprise level) integrated, and virtualized through Digital Twins (Shane McLaughlin, 2020). We can approach the advantages that come from having a digital twin from two different points of view. In one hand, from a technological point of view, this allows the engineers to visualize how the process will occur and help prevent damage to equipment or technical related failures. On the other hand, from an economic perspective, whereby preventing errors and failures, we can avoid unwanted costs related to wrong handling and bad maintenance.

Robots are remarkable tools that can be seen implemented in a diversified range of applications, from the medical to the manufacturing industries, mostly because they are becoming more precise, flexible, collaborative, and autonomous. They are flooding the manufacturing industries due to the fact that they can perform tasks that would be otherwise difficult or very monotonous for a human being. Nonetheless, even though they have autonomy, they still need to be programmed to work as intended (Vaidya et al., 2018) .

In the past decade, the market for robots has grown exponential. If we take a closer look at the graphics present in Figure AA.0.1 a) in appendix A and compare it to the

graphic in Figure AA.0.1 b) we can see that, the actual number of operational industrial robots for 2019 exceeded the expectations by almost 3%. This growth in the robotic field is directly related to the new advances and needs of the technology industries.

The report from the (Global Industry Analystic, 2020) shows that the global touch-screen technology is expected to have a substantial growth in the next decade. A recent report estimated a 60.3 billion dollars market for the year 2020 and projected 100.2 billion dollars by 2027, reflecting on growth at a CAGR (Compound Annual Growth Rate) of 7.5%. With the implementation of new features such as stylus pen and multiple fingers recognition, it is imperative to ensure the proper functioning before it gets to the user's hands. This increase in demand for touchscreens will create future problems in manufactures. With the purpose of addressing the forthcoming problems an alternative procedure for testing will be described and outlined in this document.

This dissertation can be applied to the following hypothetical research problem: As engineers, in charge of developing a solution capable of improving the efficiency of automated touch-screen tests, what is the best approach to follow? When dealing with a dilemma like this, there are two possibilities:

1) Since we know what we intend to do, we can order all the material (robot arm, and PLC (Programmable Logic Controller)) that we believe meet the necessary requirements, and then program everything by hand knowing that when problems surge, we will try to correct them on the spot.
2) Since we know what we are going to test but we still do not know what the robot arm requirements are (cycle time, precision, degrees of freedom, tool), and we are also not sure what communication protocol to use, since this varies from one PLC manufacturer to another and is also different from one robot manufacturer to another, we opt to start developing the system digitally and, in the end, we will buy the necessary equipment.

The first method is known as the traditional engineering approach, where risks are taken and errors are made, reflecting in a longer period to finish the task and at a higher cost. The second approach is based on the MBD methodology (Bhatt et al., 2005).

During this dissertation, we will demonstrate the benefits of the second approach. In fact, if we had taken the first approach in this research, we would likely have to spend more money and time to accommodate the change of the selected robot, caused by

the unforeseen problems when trying to obtain the robot, partially due to the pandemic. Also, the Meca500 was expected to work based on an EtherCAT protocol, implying the need of a EtherCAT capable PLC, but when we opted for the other robot, the protocol was change to TCP/IP. In the end, we would have spent money and time on a PLC that was not usable, needing to acquire new equipment.

To understand the importance of using a prototyping target versus using a PLC we need to ask ourselves the following questions: Can we program the PLC and go back and forward to desktop environment, to change design? How easily can we monitor the signals/parameters on the PLC? How much memory do we have for a low-level protocol implementation? Are we sure about the intended PLC? How much computational resources does the controller need? Will it fit in the PLC? Do all the computational methods run on PLC? Which communication protocol will be use?

The answers for these questions were practically unknown at the start but thanks to the Speedgoat target machine we just needed to worry about them after our controller was completely developed, tested and ready to be deployed into a dedicated embedded system.

1.2. Speedgoat

The Speedgoat company was incorporated in 2007 by MathWorks® employees and is currently based in Bern, Switzerland. They are specialized and providers of real-time target computers.

With their solutions they fill a gap between the software provided by MathWorks and real-time target machines. By offering the opportunity to their customers to verify and validate their designs alongside a complete MBD workflow, including requirements specification, simulation, RCP, HIL (Hardware-in-Loop) simulation, and deployment.

The range of applications and industries is quite vast. For example, in Aerospace we can use a HIL solution to simulate two interconnected engines, in automotive the Speedgoat mobile real-time target machine can be used to test computer vision technology for autonomous driving, in the energy and automation field can be used to simulate and test power systems or hydraulics that are not available or are too costly, in medical devices one of the best references is the PRECEYES surgical system used for vitreoretinal surgery , in

electronics and academia, that is going to be the case of this dissertation, is used for verification and testing of theoretical concepts or to teach students.

Speedgoat offers a wide range of products from baseline real-time target machines, same as the one used in this project, to personalized rack systems and a vast range of I/O and protocols supported by over 200 I/O modules. A list of products is show in Figure AA.0.3 in annex A.

1.3. Objectives

The main contribution of this dissertation is a new and efficient approach for touchscreen validation applications. The main purposes are:

- Rapid prototyping and testing of perception and supervisory control with I/O and real-time constraints using a high/medium-fidelity digital twin.
- Design of a wrist end-effector.
- Integration and validation of a controller prototype with a real 7 DOF (Degrees of Freedom) robot arm.
- Decision-supporting tool for planning in new testing routines.

1.4. Dissertation Outline

This dissertation was a culmination of a year of work to develop an industrial supervisory logic controller for a collaborative robotic arm with the aim to test touchscreen devices while relying on computer vision techniques. This dissertation is divided in 7 chapters and is structured as follows:

- **Chapter 1**: Motivation for this project, contextualization, and objectives.
- **Chapter 2**: The state of the art.
- **Chapter 3**: The mathematical formulation of the controller.
- **Chapter 4**: Hardware technical specifications, initial setup of the system and communication protocols.
- **Chapter 5**: Software description and project steps.
- **Chapter 6**: Controller validation during the different phases of the project.
- **Chapter 7:** Critical discussion on the presented controller and future work perspective.

2. LITERATURE REVIEW

Since the first Industrial Revolution, manufacturing has evolved from water and steam powered machines to electrical and digital automated production, as illustrated in Figure 2.1. This evolution makes manufacturing processes more complex, automatic and sustainable but with the advantage that people can operate machines more simply and efficiently (Vaidya et al., 2018).

The Third Industrial Revolution, in the second half of the twentieth century, lead the way for the rise of electronics, the appearance of computers, communications, and automation, with two of the major inventions being, PLC's and robots.

The Fourth Industrial Revolution, also known as "Industry 4.0" or "Smart Factory", is happening right now. What makes this revolution different is the fact that its appearance is a consequence and was preannounced by Siegfried Dias and Henning Kagerman (Kagermann et al., 2011). We can say that the "Industry 4.0" is the Third Industrial Revolution with the implementation of new and advance autonomous systems, like AI and digital twins. This realistic concept, "Industry 4.0", includes Internet of Things (IoT), Smart Manufacturing (SM) and Cloud-Base Design and Manufacturing (CBDM).

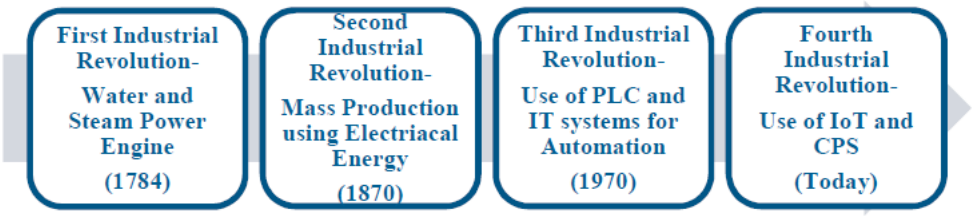


Figure 2.1 – Diagram of the industrial revolutions **Source:** (Vaidya et al., 2018)

Nowadays, automation is indispensable in many industries. The benefits are substantially more attractive to companies, since by eliminating human errors they can improve not only the quality of the products, but also the quantity, making this profitable for companies in the long term.

Repetitive and tedious works are being given to robots, improving not only the productivity efficiency but also the workers mental health. A recent study showed that 82% of people believe robots can support their mental health with 32,92% saying that automated

task would benefit them (Oracle & Workplace Intelligence LLC, 2020). This opens the doors for more automated processes where IT qualifications are required. In turn, this automation combined with Industry 4.0 will lead to a faster and stronger implementation of the digital twins.

2.1. Automated Touchscreen Testing

As previous mentioned in Chapter 1 regarding the expected growth to 2027, the need to innovate and automate this market is urgent. Assuring the supreme quality needed for this technology while producing quantities at a lightning-fast rate, is a problem that the industry is trying to deal with.

Some companies have focused on this issue and started offering gantry system robots as a solution, as illustrated in Figure 2.2. While it might seem like a good solution the cost-effective of this device becomes a problem when we start talking about dozens or even hundreds of automated touchscreen robots, since most of the equipment requires a camera to locate and interpretate what is being displayed on the screen.

Thanks to new technological advances, such as more powerful microprocessors and more memory RAM available, mobile devices like smartphones, tablets and smartwatches can easily share real-time screen information when connected to a computer via USB cable or Bluetooth (wireless). This allows for a new improvement that suits the industry needs. By exploring all the potential of these devices and by developing new computer vision techniques we can remove the cameras needed for this operation, thereby reducing the cost associated with this automation process.

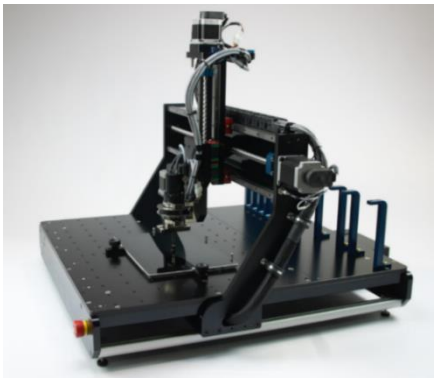


Figure 2.2 - Robotic touch tester taktouch 1000**. Source:** https://tactileautomation.com/wp-content/uploads/2019/03/TakTouch-Iso-with-SF-on-Screen-full-body.jpg

2.2. Model-based Design

Model-based design (MBD) is a mathematical and visual method of addressing problems associated with the design of complex systems.

This methodology has been growing in the last decade, and its main application is the design of embedded software. This model is based on the "V" diagram of SDLC (Software Development Life Cycle), as illustrated in Figure 2.3, providing an effective solution for an efficient and organize communication between the engineers during the design process and supporting the development cycle at the same time. MBD is an iterative process in which a test is made at every step allowing for a fast verification of the system. The process requires four basic steps in the following order:

1) **Building the plant model:** in this first step, a block diagram that implements differential-algebraic equations is built. These blocks represent the physical elements of their counterpart, and they will represent the plant model. A mathematical model will be identified from raw data acquired from the real-world system making possible the selection of the mathematical algorithm.
2) **Analysing and generating a controller for the model:** here, the dynamic characteristics will be identified by applying the previous mathematical model. The final phase is the construction of the controller based on these characteristics.
3) **Simulation of the plant and controller:** this step allows for checking all requirements, detecting and correcting errors related to the model before the next step. The type of simulations perform in here will be discuss in the next topic.
4) **Deployment of the controller:** using automatic generation code the controller previously built, will be deployed in the real world. A debugging process will be needed to guarantee that the controller will work as intended. After a few upgrades, the controller will work perfectly on the target.

MBD is quite different from previous methodologies, in the sense that, designers do not need to use complex and extensive software code. So, a model can be built just by using blocks and by making use of tools like auto-code generators to help build the code thereby preventing possible coding errors. For clarification a graphical scheme is shown in Figure AB.0.1 located in appendix B.

By using these methods, there are some crucial advantages to take into account namely: the ability to locate and correct errors early on, and therefore minimizing the time and financial impact; the model can be reused and improved in, future upgrades, etc.

However, there are still some drawbacks, for instance, the approach taken is the same for standard embedded and systems development. This often results in a considerable amount of time required to adapt processes when compared to other methods. Additionally, the ability to reuse models may not lead to good results, since sometimes the same kind of controller might not function due to hardware changes.

Although MBD can prevent hardware damage by simulating and interpreting different scenarios, when it comes to real-world production environments, it might compromise the whole process. Thus, while it is suitable for bench work, it is not a flawless method for a production system.

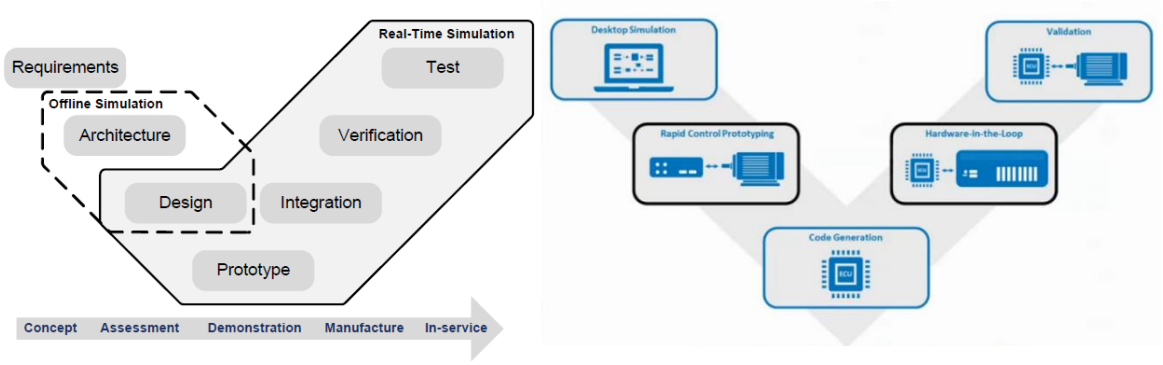


Figure 2.3 – (Left) Model-based design workflow**.** (Right) Model-based design diagram **Source:** (Bélanger et al., 2010); **Source:** (Speedgoat, 2021)

2.3. Real-time Simulation and Testing

A simulation is a representation of the operation or features of a system through the use or operation of another (American Heritage Dictionary, 2020).

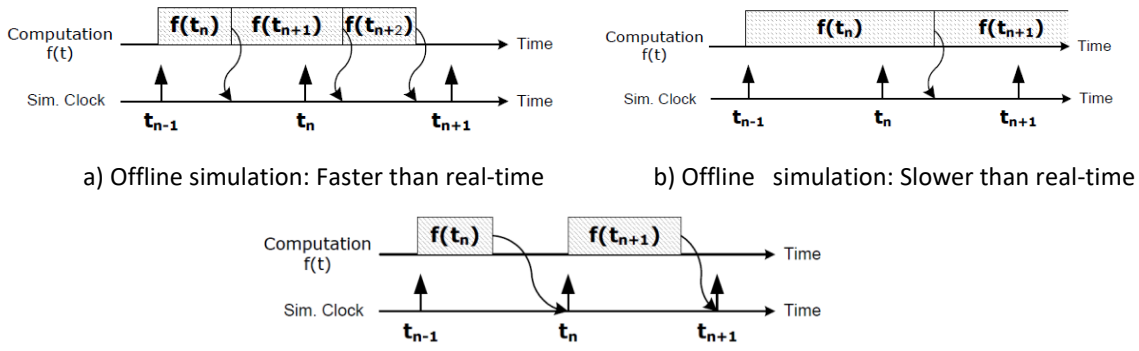
Simulation technology has evolved from analogue simulators, hybrid analogue/digital simulators to fully digital real-time simulators, as represented in Figure AB.0.2 in appendix A.

Analogue simulators were not only very expensive but required skilled personal to perform maintenance and calibration. Since the beginning, universities and research organizations have always tried to develop low-cost simulators from PC technology. It was

only when multi-core processors from *INTEL* and *AMD* became available, that this issue was addressed.

For this dissertation, we will only focus on a simulation with discrete-time and constant step duration. For a discrete-time simulation, time moves forward in steps of equal duration, commonly known as fixed time-step simulation (Dommel, 1969). However, we should point out the existence of other solving techniques that make use of variable time-steps, mainly used for solving high-frequency dynamics and non-linear systems but are not suitable for real-time simulation (Sanchez-Gasca et al., 1995).

Solving mathematical functions at a given time-step requires each variable to be solved successively as a function of variables at the end of the preceding time-step. In the case of a discrete-time simulation, the real-time needed to calculate all equations and representing a system during a giving time-step may not be the same as the duration of the simulation time-step. Figure 2.4. a) and Figure 2.4 b) represent these two possibilities.



a) Offline simulation: Faster than real-time

b) Offline simulation: Slower than real-time

c) Real-time Simulation: Synchronized

Figure 2.4. – Real-time simulation requisites and other simulation techniques. **Source:** (Bélanger et al., 2010)

These two representations are known as offline simulations, they refer to the situation when the computing time is shorter than the time-step, for instance such in case a) (also denominated accelerated simulation), or when the computing time is longer, case b).

These variances are related not only to the hardware that influences computer power but also the complexity of the mathematical model. Most of the times when executing these kinds of simulations, the goal is to obtain results as fast as possible.

In contrast, according to (Gillies, 2009) for real-time simulations, the accuracy of computations is dependent on a precise dynamic representation of the system and the length of time used to obtain results. Figure 2.4 c) demonstrates the chronological line of

real-time simulation. (Bélanger et al., 2010) also mentions that the real-time simulator must accurately produce the internal variables and outputs within the same length of time its physical counterpart would, so as to validate the model. This implies that our model simulation must correspond exactly to what is happening on the robot arm.

While offline simulation is generally used, it is also time-consuming in case no precision compromise is made on models.

State-of-the-art real-time simulators, such as the Speedgoat target-machines, can perform time-step values as low as 10 microseconds which reflects in an increase of precision. These real-time simulators are usually used in three different applications, with the first one being use in this dissertation:

RCP (Rapid Control Prototyping) – In this application, a controller is implemented through a real-time simulator and connected to a physical object. One of the advantages of using RCP over executing on an actual controller is the flexibility, fast implementation, and easy debugging.

HIL (Hardware-in-Loop) – For HIL applications, the previous controller is connected to a virtual plant that is running on another real-time simulator. HIL is a good solution for an early test of controllers when physical test benches are not available. Since virtual plants are also less expensive and constant, the results can be repeatable while allowing results from extreme testing conditions that might not be possible on real hardware.

SIL (Software-in-Loop) – This represents the third step being the combination of RCP and HIL. For this application, a powerful simulator is needed to run both controller and plant in real-time. One major advantage of SIL over the previews applications is the preservation of the signal integrity since there is no I/O. Since both models run on the same simulator, timing with the outside world is not critical anymore, this means that a simulation can be faster without compromising the results, making SIL perfect to execute a larger number of tests in a short period of time. However, if the real-time simulator is unable to reach real-time due to lack of computational power, a simulation can still run at a fraction of real-time, while being faster than on a normal computer.

Real-time simulation combined with RCP allows for fast implementation with minimal effort resulting in an accelerated integration and verification process. With real-time testing it is possible to reduce risks associated with conducting tests on physical components under normal or abnormal operating conditions, furthermore, in this type of

simulation, any model property is available during execution. Using this dissertation application as an example, the degrees of rotation and torque of every robot joint is accessible, meaning we can know the values since it is a modelled quantity. In the real robot arm, getting a precise rotation or torque value is practically impossible, Without the integration other systems (for example, sensors) and a feedback loop, making it extremely difficult to analyse and interpretate small changes.

Rapid prototyping is very important in the early stages of a product development. This technique allows for more loops in the deployment cycle, which in turn leads to a more sophisticated product.

Nowadays, real-time simulation is commonly used in areas such as power generation, automotive, aerospace, electric drive and motor development, education and research, robotics, industrial automation, etc.

2.4. Digital Twin

In the last decades, the concept and interest of digital twin has grown exponentially, mostly due to advancements in computer technology, allowing for sophisticated 3D virtual systems. These systems, as previews mention, play an important role not only in the design verification and validation but also in the product performance and characteristics (Schleich et al., 2017).

The concept of digital twin was first introduced by Dr. Michael Grieves and John Vickers in 2003, at the University of Michigan, for them, the definition would refer to a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level. At its optimum, any information that could be obtained from inspecting a physically manufactured product can be obtained from its Digital Twin (Grieves & Vickers, 2016).

Since then, many authors have tried to re-define the term. In an industrial context we can find different approaches, for instance, *Dassault Systèmes* aims to enhance their product design performance, whereas SIEMENS seek to improve their efficiency and quality in manufacturing. Another approach can be seen implemented by TESLA to develop, a digital twin for every car allowing for synchronous data exchange between the factory and the car (Coors-Blankenship, 2020).

For the purpose of this dissertation, the digital twin is seen as (Söderberg et al., 2017) describes it, a digital copy of the physical system to perform real-time optimizations while adding another definition that includes the properties, condition and behaviour of the real-life object through models and data, being developed alongside its physical twin (Haag & Anderl, 2018).

This definition represents the new era of manufacturing, consisting of smart products that are able to track and communicate their operating conditions and at the same time send data about their status, such as the environmental conditions (Schleich et al., 2017).

In sum, the digital twin describes a bi-directional relationship between a physical object and its virtual model. While the technology is not yet settling for high-fidelity models at multiple scales, due to difficulties related to the prediction of complex systems, we can already work and develop a simple high-fidelity model.

2.5. Collaborative Robots

One of the major benefits of Industry 4.0 is the use of collaborative robots. Unlike industrial robots, that are programmed to perform and function completely automatically, collaborative robots, can assist humans without putting their jobs at risk since they can co-exist, and work based on human interactions to control and adapt their process. As (Bragança et al., 2019) states humans will assume more rules of supervisory and leadership on the shop floor, making decisions that cannot be made by autonomous systems and compensating technological limitations.

With the new required laws, there are four features incorporated, such as power and force limiters, safety-rated monitored stop, hand guiding, speed and separation monitoring. With these features the robots can perform in automatic mode since they will be able to prevent injuries by avoiding or simply stopping the action if a human gets inside his workspace.

It is important to understand that collaborative robots are not adequate for all kinds of processes and applications. For example, if the cycle of the required process is faster than 10 seconds, they are not suitable, and should be replace with industrial robots that are also more expensive.

One key feature is how accurate and consistent collaborative robots can be, making them the perfect choice for tasks like tightening screws or medical surgeries. Another major advantage is how simple and intuitive it is to program these types of robots, since most of the time, builders offer their own software and workers can simply use a computer or touchpad to command it even without any programming knowledge.

2.6. Computer Vision

Vision is one of the most complicated human systems. Through our eyes, we perceive 3D objects, sometimes with irrelevant or misleading information. However, we can extrapolate and understand this chaotic information, from retinal to cognitive levels. This involves perception, recognition, and intrinsic information that comes from assumptions and knowledge.

This might seem simple given how easy it is to describe and associate the colours and the name of the object that we usually use to write with. Thus, as previous referred, these almost intuitive answers come from years of knowledge and experience, making the computer vision problem difficult to deal with.

In the early 1970s, computer vision was seen as the visual perception component that would allow the robot to mimic humans by giving them intelligent behaviour. According to some books what made this field so different from the digital image processing, already in use at the time, was the ambition to build a 3D structure of the surroundings from the images.

As (Ballard & Brown, 1982) suggests, computer vision is the automatization and integration of a wide range of processes and representations used for vision perception. It also includes techniques that are useful by themselves, from image processing (transforming, encoding, and transmitting images) to statistical pattern classification (statistical decision theory applied to general patterns). Although it also includes valuable techniques for geometric modelling and cognitive processing.

Almost four decades later we still deal with these challenges, even with all the technological advances of today. (Szeliski, 2011) introduces this topic by saying: “the dream of having a computer interpret an image at the same level as a two-year-old (…) remains elusive”. For him, this is considered as an inverse problem, in which we want to obtain a full

solution without giving sufficient information. He also believes that a marriage between the scientific/statistical approach and the engineering approach (efficient algorithms) is the only way to obtain a true solution. However, in recent years, AI technology allowed for the use of computer vision in a wide variety of applications, such as fingerprint recognition, automotive safety, medical imaging, 3D building (photogrammetry), face detection systems, etc.

In sum, we need to break the barrier and invest in more delicate and precise models to acquire the "intrinsic knowledge" so we can deploy all the potential that computer vision has to offer.

2.7. Visual Servoing

Visual servoing (VS) or vision-based robot control is a technique that uses information acquired by vision sensors (or cameras) for feedback control of the motion/pose of the robot (Corke, 1996).

(Hill & Park, 1979) first introduced this term to distinguish their approach from earlier "blocks world" experiments where the system would alternate between taking pictures and moving. In the solution presented, VS aims to interpret visual information to control the pose of the robot's end-effector in relation to an object, in our case, a touchscreen device.

(Hutchinson et al., 1996) describes visual servoing as being a fusion of results from different areas such as high-speed image processing, kinematics, dynamics, control theory and real-time computing. There are two camera configurations for this technique, as represented in Figure 2.5:

- End-effector mounted, also known as eye-in-hand, in which the camera is mounted directly on the robot's end-effector, as the name suggests.
- Fixed, often called eye-to-hand, in which the camera is mounted somewhere in the world observing the target and the motion of the robot.

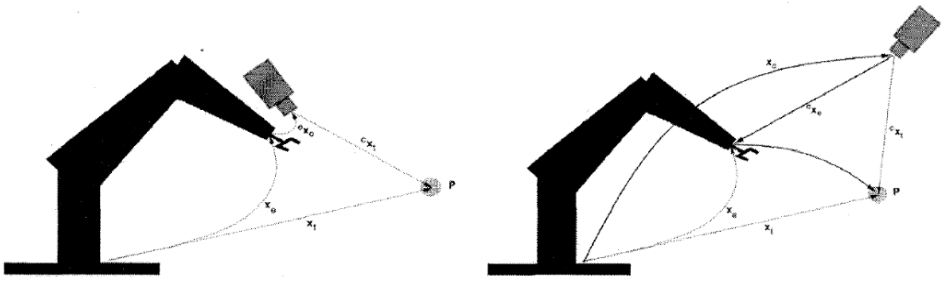


Figure 2.5 – Two possible camera mounting configurations: (Left) eye-in-hand; (Right) eye-to-hand **Source:** (Hutchinson et al., 1996)

In each situation the camera requires a calibration, prior to the tasks, in order to determine the intrinsic camera parameters: pixel pitch, focal length and principal point. However, for an eye-in-hand situation, there is still needed another calibration known as the hand/eye calibration problem with (Lenz, 1989) offering an insight to possible solutions.

According to (Sanderson & Weiss, 1983) we can categorized the visual servo systems in four major categories. The first one is referred to as a *dynamic look-and-move system*. In this case, the control architecture is hierarchical, and the vision system is used to provide set-point inputs to the joint-level controller, making use of joint feedback to internally stabilize the robot. In contrast, *direct visual servo* eliminates the robot controller entirely replacing it with a visual servo controller that directly computes joint inputs, thus using vision alone to stabilize the mechanism. A scheme of these two definitions can be found on Figure AB.0.3.

In position-based visual servoing (PBVS), a 3D reconstruction is made from information extracted from images (features) to estimate the pose of the target. A cartesian pose error (feedback) is generated allowing for the robot to achieve the desired position. A diagram representation of this process can be seen in Figure AB.0.3 a) in appendix B.

However, in image-based visual servoing (IBVS), a 3D reconstruction is not processed since the error is directly computed on the values of the images (features). Although this approach allows for a reduction of computational delay by eliminating errors from camera calibration, sensor modeling and the need for image interpretation it also presents a significantly challenge to controller since the plant is nonlinear and highly coupled (Hutchinson et al., 1996). A diagram representation of this process can be seen in Figure AB.0.3 b).

2.8. Meca500

The Meca500 is a robot developed by Mecademic. A Canadian company founded in 2013 by Jonathan Coulombe and Ilian Bonev.

With new ideas and ambition, the team developed a six-axis industrial robot named Meca500, as seen in Figure AB.0.4 in appendix B. Some key characteristics, taken from Figure AB.0.5, for this robot are: the repeatability of 0.005 mm, the payload of 500 grams, reach of 260 millimetres to the wrist centre, and an arm of 330 millimetre when fully extended, making it the current smallest industrial robot. Additionally, one of the best features is the embedded controller in the robot base resulting in a complete weight of only 4.5 kilograms. The small size and high precision make this robot the perfect choice for tight space, quality inspection, and for precision assembly in electronics, medical devices, automotive, aerospace and watchmaking industries.

Thanks to the embedded controller in the base, there is no need for external controller boxes or thick cables. After plugged-in, the robot can be programmed directly from a web-based interface, provided by the manufacturer, via TCP/IP or via EtherCAT.

The Meca500, at the time of writing this dissertation, could not be considered a complete collaborative robot by the definition of the word. However, as stated in the motors manufacturer webpage, the robot will soon become a collaborative robot, thanks to future firmware updates that will implement functions to prevent collisions (Maxon motor ag, 2021).

2.9. Kassow KR810

The Kassow KR810 is a collaborative robot developed by Kassow Robots, a company founded in Copenhagen, Denmark. This company focuses on producing seven-axes lightweight robots that offer speed and power for industrial applications. One of the many advantages of this cobot is the user-friendliness that allows small and medium-sized enterprises to accomplish sophisticated automation and programming without the necessity of robotics experts (Robots, 2020).

In Figure 2.6 is possible to observe the main characteristics with the top features being the repeatability of 0.1mm, a maximum payload of 10kg, a reach of 850mm and a maximum joint speed of 225º/sec.

This robot comes with a portable electrical cabinet and a teach pendant that contains the software application to control the robot.

With the purpose of testing our developed controller a TCP/IP connection was established so the commands could be sent between the Speedgoat real-time target machine and the Kassow KR810.

General specifications	KR 810
Reach (mm)	850
Payload (kg)	10
Weight (kg)	24
Joint Speed (deg / sec)	225
Joint Ranges	J2 and J4: -70°/+180°. J1, J3, J5, J6 and J7: ±360°
Brakes on joints	Yes
Absolute encoders on joints	Yes
Repeatability (mm)	+/- 0.1
Degrees of freedom	7
Foot print (mm)	130x130
Operating temperature (C°)	0-45
Body material	Anodized aluminum

Figure 2.6 – Mechanical characteristics of the Kassow KR810 **Source:** (Robots, 2020)

2.10. Capacitive Touchscreen

The concept of a touchscreen was first introduced in 1965 by Eric Johnson, an engineer at the Royal Radar Establishment, for air traffic control. A few years later, 1972, an engineer at CERN, Bent Stump, published the second application for a touchscreen, in which the buttons and switches of the control room would be replaced. In the early 1980s, General Motors, replaced the traditional car stereo, fan, heater and air conditioner controls for a touchscreen but unfortunately, it did not catch the public attention due to often costly technical problems. In the early 2000s, we started to see new implementations of touchscreen either in videos consoles, like the case of the Nintendo DS, 2004, or in mobile phones with LG Prada being the first mobile phone with a capacitive touchscreen, in 2006, and the iPhone being the first to offer a multi-touchscreen in 2007.

Capacitive touchscreens rely on conductors that are embedded onto a sheet of glass. Since this technology works with electrostatic charges it can only interpret the

following inputs: human fingers, since the human body naturally holds an electrical charge, and conductive pens (stylus), like the one chosen for this dissertation. Figure 2.7 represents the basic principles behind this technology (Technology, 2012).

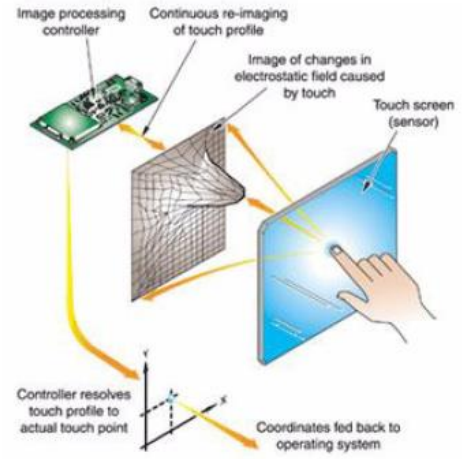


Figure 2.7 – Representation of the basic principles for a capacitive touchscreen **Source:** https://www.lorextechnology.com/self-serve/how-touch-screen-monitors-work/R-sc3100030

This technology comes in two key types: surface and projective. The first one concerns the basic of capacitance technology since there is only one side of the insulator coated with a conductive layer. In contrast, the other method consists of making use of a matrix of rows and columns of a conductive material that can have one or two layers. This grid provides an enhance on accuracy and multi-touch features. The functioning of a capacitive touchscreen is quite simple. When the display turns on, the conductors create an electrostatic field. The contact from one of the previous elements creates a distortion on the electrostatic field. From this distortion, the image processor is able to calculate where the touch occurred.

The benefits from capacitive touchscreens such as high touch sensitivity and accuracy, sharp and vibrant images, multi-touch capability, reliability (since they can still work when cracked or pierced) make it the most suitable for manufactures.

Although the market for this technology is still recent, the growth has been exponential. In 2007, 93% of touchscreens were resistive, with only 4% being capacitance. By the end of 2013, only 3% were resistive, with 90% being capacitance (Walker, 2014).

The device chosen was a Samsung $S10^{+}$ equipped with a capacitive touchscreen. Since this device does not come with a stylus pen, it was necessary to choose one that met the necessary requirements.

3. MATHEMATICAL MODELING

In simulation and control-oriented studies an essential and crucial procedure is the design of a model capturing the behaviour of the systems being studied. For instance, the manipulator controller developed in Chapter 5 was only possible due to the numerical formulations implemented on specific blocks. This preceding stage is usually referred to as mathematical modeling or system modeling and comprises the subject of this chapter. In general, matrices calculations or other mathematical calculations were not a time-consuming problem since everything was formerly set up inside the Simulink blocks. However, it is important to understand the concepts behind robotic manipulators in order to understand this dissertation. A brief description of these concepts will be presented.

3.1. Kinematic Modeling

This section is dedicated to the derivation of the kinematic model of the robot arm and analysis of its motion capabilities and singularities. The motive of this kinematic study is to establish the mathematical formulation that describe the relationship between the temporal variations of the robot end-effector pose (position and orientation) and the desired reaching point. This kinematic approach is an utterly geometrical study in which the dynamic proprieties such as mass, inertia, or friction are disregarded. Also, at this point, forces or torque inputs are not considered.

Since the invention of the first robots, it was important to describe their movement through mathematical equations, usually referred to as kinematic equations. Generally, their links are modelled as rigid bodies connected by joints, with a fixed end (base) and with a free end (end-effector). In analogy to a “human arm”, these robots normally use their first three joints to achieve the position and the last remaining to orienting the end-effector. The first joints form a structure known as arm and the structure created from the last one is denominated as wrist. A kinematic study requires the knowledge of some intrinsic characteristic of the robot, i.e., type of robot arm, type of wrist, type of joints, DOF, singularities, and workspace.

We can describe two possible wrist configurations depending on the last three joint axes, as illustrated in Figure AC.0.1 in appendix C: pitch-yaw-roll (YXZ) or roll-pitch-roll (ZYZ), also known as spherical wrist. This one is the commonly used in the industry and is also the configuration present on the Meca500.

We can also talk about joint space and task space (or Cartesian space). The first one is defined by a vector with translational and angular displacements of each joint of the robot manipulator while task space is defined by the position and orientation of the end-effector (Yu, 2018).

The position and orientation of the end-effector are normally obtained by the angular positions and forward kinematics of the robot. In fact, the forward kinematic gives the position and orientation of the end-effector from the angular positions of every joint. However, it is also possible to do the inverse. This is usually referred to as inverse kinematics, where the desired cartesian position/orientation of the end-effector are given, and the angular positions of the joints are computed (Pires, 2007). The necessary torque to produce the required motion is also computed in both cases. Although, when using this second method, it is possible to obtain multiple solutions to reach the desired position resulting in an unsolved computational solution, known as singularity.

After understanding the last two paragraphs it becomes clear the correlations between joint space/forward kinematics and cartesian space/inverse kinematics. This will be extremely important for the purpose of understanding the behaviour of the controller built during this dissertation.

To obtain either a forward or inverse kinematic model a (4x4) homogenous transformation matrix presented by Denavit and Hartenger is required. This matrix describes the spatial transformation between two consecutive links in function of the joint position that connects the two links. For a 6 DOF robot would be needed 24 parameters to describe the whole structure, with four parameters for each joint. In this dissertation we will use the modified DH (Denavit-Hartenger) parameters, as illustrated in Figure AC.0.2 in appendix C, in which the only difference is the placement of the coordinates system attachment to the links as well as the order of the transformations. For the purpose of creating the diagram for the Meca500 represented in Figure 3.1 the following rules need to be respected:

1. The links are numerated from the fixed base link, starting from 0, to *n* for the end-effector link. The enumeration for the joints starts from 1 for the joint

responsible for connecting the base link and the first movable link, and increases consecutively up to *n*.

2. The origin must be located at the point of intersection between two consecutive axis or at the common normal of intersection.
3. The z axis is aligned with the direction of the rotational joint axis.
4. The x axis is defined along the common normal between the previous axis and the joint.
5. The y axis is defined after the other two and follows the right-hand rule, which also dictates the positive direction of rotation.

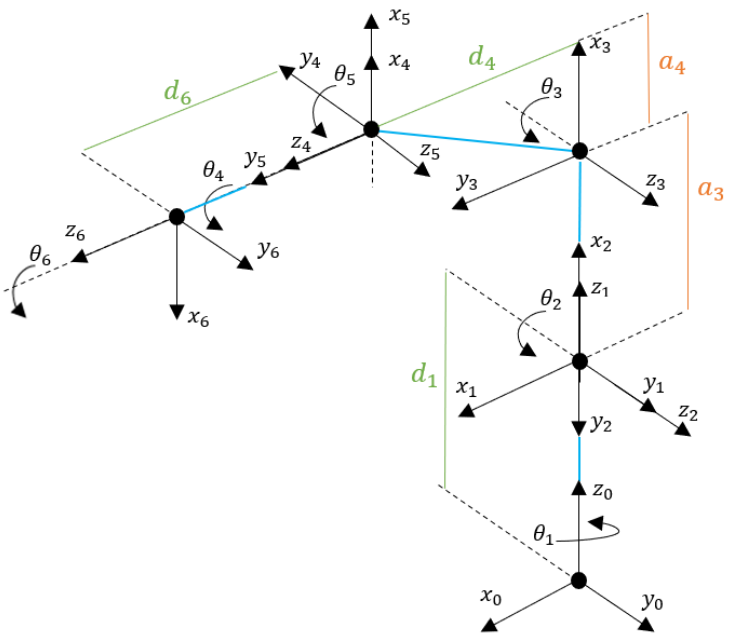


Figure 3.1 – Schematic representation of the DH parameters for the Meca500

The table for the DH parameters has 4 columns, one for each parameter ($a_i, \alpha_i, d_i, \theta_i$) and *n-1* rows. The four following parameters are used to define the geometry of link i:

- a_i = distance from z_{i-1} to z_i measured along x_{i-1} axis.
- α_i = angle from z_{i-1} to z_i measured about x_{i-1} axis.
- d_i= distance from x_{i-1} to x_i measured along z_i axis.
- θ_i = the rotation angle measured about z_i between x_{i-1} to x_i axis.

The Table 3-1 represents the DH parameters for the Meca500 and are already represented on the previous diagram.

Link	θ_i (º)	α_{i-1} (º)	a_{i-1}(mm)	d_i(mm)
1	0	0	0	135
2	-90	-90	0	0
3	0	0	135	0
4	0	-90	38	120
5	0	90	0	0
6	180	-90	0	70

Table 3-1– Table with the determined DH parameters for the Meca500

It is also important to obtain the Jacobian matrix, which correlates the cartesian velocities of the end-effector with the angular velocities of each joint. The Jacobian is a matrix with $n \times m$ elements, with *n* corresponding to the DOF and *m* to the number of joints.

$$v = \begin{bmatrix} \dot{p} \\ w \end{bmatrix} = \begin{bmatrix} J_P \\ J_w \end{bmatrix} \dot{\theta} = J_G \dot{\theta} \tag{3.1}$$

$$\dot{x} = \begin{bmatrix} \dot{p} \\ \dot{\emptyset} \end{bmatrix} = \begin{bmatrix} J_P \\ J_\emptyset \end{bmatrix} \dot{\theta} = J_A \dot{\theta} \tag{3.2}$$

$$J_A = \frac{\partial k(\theta)}{\partial \theta} \tag{3.3}$$

Where,

$\dot{\theta} = \left[\dot{\theta}_1, \dot{\theta}_2, \dots, \dot{\theta}_n\right]^T$, represents the angular velocity vector for the joint rotation.

$\dot{p} = [\dot{p}_1, \dot{p}_2, \dots, \dot{p}_n]^T$, represents the linear velocity vector of the end-effector.

$w = [w_1, w_2, \dots, w_n]^T$, represents the angular velocity vector of the end-effector.

$\dot{\emptyset} = \left[\dot{\emptyset}_1, \dot{\emptyset}_2, \dots, \dot{\emptyset}_n\right]^T$, represents the change of orientation of the end-effector frame $\emptyset$.

The Jacobian can be calculated from two different ways. The first one, known as geometric Jacobian, takes into consideration the velocity components (rotation and translation) of the end-effector, as shown in (3.1). The second one, called analytic Jacobian, is usually computed by partial differential as (3.2) using $x = k(\theta)$. In general, the geometric Jacobian and the analytic Jacobian are different.

$$J_G = \begin{bmatrix} Z_0 * (o_n - o_0) & \cdots & Z_{i-1} * (o_n - o_{n-1}) & \cdots & Z_{n-1} * (o_n - o_{n-1}) \\ Z_0 & \cdots & Z_{i-1} & \cdots & Z_{n-1} \end{bmatrix} \quad (3.4)$$

The Meca500 and the Kassow KR810 only have revolute joints and their geometric Jacobian can be constructed from the matrix (3.4). Where $\boldsymbol{Z_i}$ are the first three elements of the third column of transformation matrix $\boldsymbol{T_i^0}$, and $\boldsymbol{o_i}$ are the three elements of the fourth column of $\boldsymbol{T_i^0}$.

When performing the mapping between joint space and task space we will need to identify some singular points in the Jacobian matrix. Those points, called singularities, represent a configuration where the end-effector becomes physically blocked in some directions, due to the loses of degrees of freedom. Usually, a robotic arm can have two types of singular configurations:

- Boundary singularities: This type of singularity is related to the workspace of the robot. It occurs when the robot is outstretched to its full extent trying to reach a position outside his workspace, losing freedom to move in every direction.
- Internal singularities: This type of singularity is intertwined with the inverse kinematic approach and occurs when two or more axes are aligned. In this case, an action of one joint can be cancelled by another. Since we are working in cartesian space there are endless possibilities for the trajectory, resulting in an undetermined solution.

The boundary singularities can be detected and prevented by ensuring the position is within reach, thus preventing the robot to be fully extended. The internal singularities are the most important to deal with since they can exist anywhere inside these limits.

For the Meca500, the axis of the last three joints intersects at one point (the centre of the robot's wrist). This makes it easier to notice if the robot is close to a singularity and therefore avoiding it.

When the determinant of the Jacobian is too small, we will have very small cartesian velocities but very large joint velocities. It is possible to calculate the singular points in the matrix by inverting the Jacobian in (3.1) that results in the equation (3.5). The singularities in the robot can be found by solving $det(J) = 0$.

$$J_G^{-1} v = \dot{\theta} \tag{3.5}$$

There are three types of possible singularities: shoulder, elbow, and wrist, as illustrated in Figure AC.0.3 in appendix C. This must be taken into consideration while choosing the trajectory since not only is impossible to crosse them but there is also a huge joint velocity resulting from passing near them (Bonev, 2019). When unexpected, this situation may cause not only harm to the robot itself but also safety risks to everyone around it.

Another important parameter is the Euler angle convention. The Euler angles, denoted by α, β, γ, allow defining the different orientations of the rigid body elements in space with respect to the world reference frame, with α corresponding to a rotation along the x axis, β corresponding to a rotation along the y axes and γ corresponding to a rotation along the z axes. The tree basic rotation matrices are presented from (3.6) to (3.8).

$$\mathrm{R_x}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{3.6}$$

$$R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \tag{3.7}$$

$$R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

The rotation matrix corresponding to the Euler angles used by Mecademic is:

$$R(\alpha, \beta, \gamma) = \begin{bmatrix} \mathrm{c}(\beta)\mathrm{c}(\gamma) & -\mathrm{c}(\beta)\mathrm{s}(\gamma) & \mathrm{s}(\beta) \\ \mathrm{c}(\alpha)\,\mathrm{s}(\gamma) + \mathrm{s}(\alpha)\mathrm{s}(\beta)\mathrm{c}(\gamma) & \mathrm{c}(\alpha)\,\mathrm{c}(\gamma) - \mathrm{s}(\alpha)\mathrm{s}(\beta)\mathrm{s}(\gamma) & -\mathrm{s}(\alpha)\mathrm{c}(\beta) \\ \mathrm{s}(\alpha)\mathrm{s}(\gamma) - \mathrm{c}(\alpha)\mathrm{s}(\beta)\mathrm{c}(\gamma) & \mathrm{s}(\alpha)\,\mathrm{c}(\gamma) + \mathrm{c}(\alpha)\mathrm{s}(\beta)\mathrm{s}(\gamma) & \mathrm{c}(\alpha)\mathrm{c}(\beta) \end{bmatrix} \tag{3.9}$$

With $c = cos$ and $s = sin$.

This differs from manufacturer to manufacturer. For instance, FANUC and Kuka use the fixed XYZ Euler angle convention, ABB uses the mobile ZYX Euler angle

convention and Kawasaki uses the mobile ZYZ Euler angle convention. The robots from Mecademic, like the Meca500, use the mobile XYZ Euler angle convention, $x' \rightarrow y' \rightarrow z'$, the explanation for this can be found on the Mecademic website. The Kassow robots use the fixed XYZ Euler convention.

3.1.1. Forward Kinematics

Forward kinematics is the mapping from the joint space to the cartesian space, in this case the cartesian position of the stylus pen. In this section it will be demonstrated how to determine these calculations only for the Meca500. However, the process is the same for the Kassow KR810, for two exceptions that will be explained.

The transformation matrix that describes the system i in relation to $i-1$ is given by:

$$ {}^{i-1}_{i}T = R_x(a_{i-1}).D_x(a_{i-1}).R_z(\theta_i).D_z(d_i) \tag{3.10} $$

Applying this multiplication, we obtain the following multiplication matrix:

$$ T_i^{i-1} = \begin{bmatrix} \cos\ \theta_i & -\sin(\theta_i) & 0 & a_{i-1} \\ \sin(\theta_i).\cos\ (\alpha_{i-1}) & \cos(\theta_i).\cos\ (\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -d_i.\sin(a_{i-1}) \\ \sin(\theta_i).\sin(\alpha_{i-1}) & \cos(\theta_i).\sin\ (\alpha_{i-1}) & \cos\ (\alpha_{i-1}) & d_i.\cos\ (\alpha_{i-1}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11} $$

or in abbreviated form:

$$ T_i^{i-1} = \begin{bmatrix} c_i & -s_i & 0 & a_{i-1} \\ s_i.c\alpha_{i-1} & c_i.ca_{i-1} & -s\alpha_{i-1} & -d_i.s\alpha_{i-1} \\ s_i.s\alpha_{i-1} & c_i.s\alpha_{i-1} & c\alpha_{i-1} & d_i.c\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12} $$

To transform forward over a number of links, we just have to multiply the link matrices sequentially.

$$ T_6^0 = \prod_{i=1}^{6} T_i^{i-1}(\theta_i) \tag{3.13} $$

$$ T_1^0 T_2^1 T_3^2 \ldots T_n^{n-1} = T_n^0 \tag{3.14} $$

From (3.12) and applying the D-H values from Table 3-1 we obtain:

$$T_1^0 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \; T_2^1 = \begin{bmatrix} c_2 & s_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s_2 & -c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \; T_3^2 = \begin{bmatrix} c_3 & -s_3 & 0 & a_3 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.15)$$

$$T_4^3 = \begin{bmatrix} c_4 & -s_4 & 0 & a_4 \\ 0 & 0 & 1 & d_4 \\ -s_4 & -c_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \; T_5^4 = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \; T_6^5 = \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ -s_6 & -c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.16)$$

By decomposing this matrix into a combination of submatrices', the first 3x3 matrix would correspond to the degrees of rotation, θ, about *x, y, z* axis and the last 3x1 matrix to the position along *x, y, z* (translation) of the end-effector in relation to the base. This means that T_6^0 can be represented as in (3.17). The values of each element were obtained using MATLAB Symbolic Math Toolbox™ and optimized afterwards (firstly using MATLAB and posteriorly by hand, which allowed to obtain more compact and efficient solutions). The elements for the matrix equation (3.17) can be found in annex C.

$$T_6^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x^0 \\ r_{21} & r_{22} & r_{23} & p_y^0 \\ r_{31} & r_{32} & r_{33} & p_z^0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.17)$$

The stylus pen is rigidly connected to the robot flange and therefore this transformation matrix can be extended with a translation matrix to obtain the transformation matrix from the base frame to the position of the pen. In (3.18) it is shown how. In this research, $d_y = 0\ mm$ and $d_z = 41{,}42\ mm$.

$$T_7^0 = T_6^0 T_7^6 = T_7^0 = \begin{bmatrix} & & 0 \\ I & & d_y \\ & & d_z \\ 0 & & 1 \end{bmatrix} \quad (3.18)$$

For the Kassow, the equation (3.14) is still valid but now we need to consider $n = 7$. The transformation matrices can be obtained from applying the Kassow DH parameters listed in Table C.0-1 in annex C into equation (3.12).

This will result in a final transformation matrix T_7^0 regarding the relation between the base to the end-effector. Since, our stylus pen is rigidly connected to the TCP (Tool Centre Point) it is possible to represent the relation between the base and the tool, by multiplying with the translation matrix, as represented in equation (3.19).

$$T_8^0 = T_7^0 T_8^7 \tag{3.19}$$

3.1.2. Inverse Kinematics

Solving an inverse kinematic problem is not linear and simple as a forward kinematic one. When dealing with the mapping from a cartesian space to joint space, the joint angles are unknown and as shown in Figure AC.0.3 {a), c), d), e), g), i)} there are eight possible configurations for the same cartesian coordinates. According to (Pieper, 1968), the sufficient condition to obtain a solution in an inverse kinematic problem, describing a 6 DOF robot arm is to have three consecutive joints where the axis intersect at one point. For a robot with a spherical wrist, like the one found in Meca500, this condition is always verified. However, the same is not verified on the Kassow KR810, since a 7-DOF manipulator includes redundancies. In this case, the number of joints ($n = 7$) is greater than the dimension of the manipulation variables with three position coordinates and three orientation angles ($m = 6$). Redundancy plays an important role in the kinematic control as redundant joints allow a manipulator to avoid joint limits, singularities, or obstacles. The redundancy is also utilized to minimize joint velocities or actuator torques in case of following a desired end-effector trajectory. Due to the existence of redundancy, joint velocities cannot be obtained directly by solving the differential kinematics equation since the inverse of non-square matrix J cannot be obtained (Wang et al., 2010).

Due to the extensive use of sensory feedback processes and the implementation of real-time automated tasks, where the end-effector's pose (position and orientation) is always changing, inverse kinematics oversees most of the robot control.

There are two methods to solve this problem. The first is the closed-form solution based on the geometrical properties and can solve the inverse kinematics directly.

The second method is the inverse of the Jacobian, an iterative technique. In this dissertation we will only make reference to inversion of the Jacobian since the inverse kinematic Simulink block resorts to iterative techniques. The cost function for this optimisation algorithm is obtained from the transformation matrix T_7^0 presented in (3.18).

The iterative technique used in the Simulink block is the BFGS (Broyden-Fletcher-Goldfarb-Shanno) gradient projection. The algorithm is a quasi-Newton method that uses the gradients of the cost function from past iterations to generate approximated second-derivative information. This second-derivative is used to determine the step needed to take the current iteration. Then a gradient projection method deals with the boundary limits on the cost function created by the joints limits of the robot (Mathworks, 2021b).

If the inverse of the Jacobian is known, is possible to increment changes in the joint variables that will result in the desired incremental change in the end-effector position and orientation (Welman, 1993).

Since we are mapping from the angular velocity to the velocity of the position and orientation, the strategy is to invert the transformation matrix (3.13) and then multiply it by the position vector $\boldsymbol{p_i}$. The process is summarized in (3.21) where $\boldsymbol{p_i}$ corresponds to the position and orientation vector for each joint and x_i is the vector with the desired joint angles.

$$p_i = \begin{bmatrix} x_i \\ y_i \\ z_i \\ \alpha_i \\ \beta_i \\ \gamma_i \end{bmatrix} \tag{3.20}$$

$$\dot{\theta} = J_G^{-1}\dot{x} \tag{3.21}$$

$$\dot{x_i} = \left(T_i^{i-1}\right)^{-1}.p_i \tag{3.22}$$

This method will be the same for the Kassow KR810 controller, with the exception of the cost function for the optimization algorithm since it will be obtained from a transformation matrix T_8^0.

3.1.3. Meca500

This subsection will present an overview over the singularities considered for the development of the Meca500 Simulink model.

3.1.3.1. Singularities

In a six-axis robot arm, like the Meca500, where the axes of joints 2 and 3 are parallel and normal to the axes of joints 1 and 4, the axis of joint 5 is normal to the axes of joints 4 and 6, and these last three axes intersect at one point is usually called as an anthropomorphic robot arm (Bonev, 2019). Due to this architecture detecting possible singularity states is an easy task since it does not require much effort to describe it geometrically.

3.1.3.2. Wrist Singularity

This singularity occurs when the axes of joints 4 and 6 become coincident. This corresponds to a condition $\theta_5 = 0°$. In this situation, the robot cannot move in the direction of the axis of joint 5. For this condition, there are limitless solutions to the inverse kinematics of the robot. For example, if $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$ is a solution, then $\{\theta_1, \theta_2, \theta_3, \theta_4 - \gamma, \theta_5, \theta_6 + \gamma\}$ can also be a solution, where γ is an arbitrary angle (Bonev, 2019).

3.1.3.3. Elbow Singularity

It occurs when the wrist centre (the intersection point of the axils of joints 4, 5 and 6) reclines on the plane crossing through the axes of joints 2 and 3. This kind of singularity is defined only by the position of joint 3. This only occurs when $\theta 3 \approx -72.43°$. This singularity is the easiest to expect making it also the easiest to avoid (Bonev, 2019).

3.1.3.4. Shoulder Singularity

The third type of singularity in an anthropomorphic robot arm is the shoulder singularity. It occurs when the centre of the robot wrist intersects the plane passing through the axes of joints 1 and 2. This singularity unlike the other two is not dependent on a single joint position, making it the most complex of them all. In this case, the robot is not able to move in the direction of the axis of joint 2. Unfortunately, there is no simple mathematical

expression to denote a solution for this singularity due to the infinite solutions to the inverse kinematics of the robot.

3.2. Trajectory Planning

One of the main problems for the controller is finding a trajectory that connects an initial to a final configuration while satisfying other specified constraints, such as velocity, position, and acceleration. The trajectory planner is used to obtain the parameters in the joint space or cartesian space, as illustrated in Figure 3.2. In this dissertation, the final position is obtained in the cartesian space extracted from computer vision and image processing techniques. Then this information is passed on to the transform trajectory block, as illustrated in Figure 5.6.

This block generates a trajectory between two homogenous transformation matrices, being respectively the initial and final configuration. The positions are computed using linear interpolation and the rotations applying a spherical linear interpolation (or SLERP). The quaternion SLERP is an extension of linear interpolation along a plane to spherical interpolation in 3D. For q_1 and q_2, SLERP interpolates a new quaternion, q_0, along the great circle that connects q_1 and q_2. The interpolation coefficient, T, determines how close the output quaternion is to either q_1 or q_2 and θ is half the distance between q_1 and q_2 (Mathworks, 2021e). The SLERP can be described in terms of sinusoids:

$$\boldsymbol{q_0} = \frac{\sin\left((1-T)\theta\right)}{\sin\left(\theta\right)}\boldsymbol{q_1} + \frac{\sin\left(T\theta\right)}{\sin\left(\theta\right)}\boldsymbol{q_2} \tag{3.23}$$

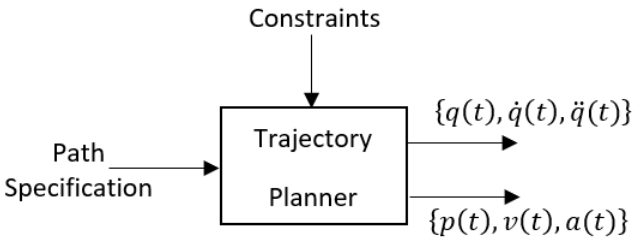


Figure 3.2 – Schematic representation of the trajectory planner

4. SYSTEM OVERVIEW

This section starts by reviewing the technical specifications of the Speedgoat real-time target-machine and explaining how to correctly establish the connection between the host computer and the referred hardware. The second part is dedicated to the selected robots for this research, with focus on their singularities and communication protocols. For each robot, the basic concepts are explained together with the respective developed Simulink model. Figure 4.1 illustrates the closed loop of the final physical system.

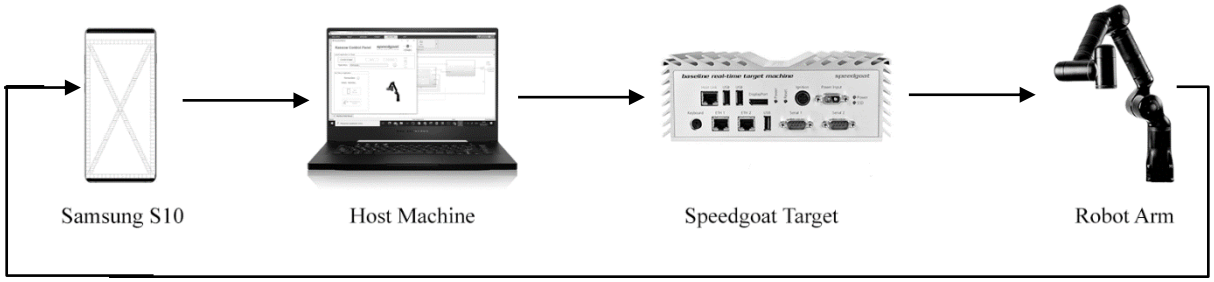


Figure 4.1 – Schematic representation of the system loop

4.1. Speedgoat Real-time Target Machine

As mentioned in Chapter 2, the variety of products offered by Speedgoat are optimized for different applications, from RCP to HIL.

For this dissertation, Speedgoat provided a baseline real-time target machine, with enough computing power to perform the needed RCP application's simulation.

Regarding the hardware and I/O specifications our baseline is equipped with an Intel Celeron 2 GHz with 4 cores, 4 GB DDR3 RAM, 1 x display port, 1 x USB 3.0, 2 x USB 3.0, 1 x host-target Gigabit Ethernet link, 2 x Gigabit Ethernet, 2 x RS232 serial ports, 4 x mPCIe slots for I/O modules and 32 GB SSD.

The TCP/IP blocks that will be described in this chapter required precise information about the Ethernet ports configuration. To find this information we used the command "getPCIInfo (tg, 'ethernet')" that displayed the information illustrated in Figure D.0.1 in Annex D. The target machine I/O ports are illustrated in Figure 4.2 and the Ethernet ports are configure as described below:

- Ethernet port “Host Link” (green), in the PCI Bus 17, and slot 0 was reserved for primary host-target communication. This port was also responsible for transmitting the waypoints captured by the OCR model, running on the host computer, to the Speedgoat target.
- Ethernet port “ETH1” (red), in the PCI Bus 14, and slot 0 was used for sending/receiving TCP/IP messages between the Speedgoat target and the Kassow robot arm.
- The Ethernet port “ETH2” (yellow), in PCI bus 15, and slot 0 could be used to feed another Speedgoat target, running the digital twin model of the robot arm in real-time, or to send/receive signals from other devices such as another robot, sensor, etc.

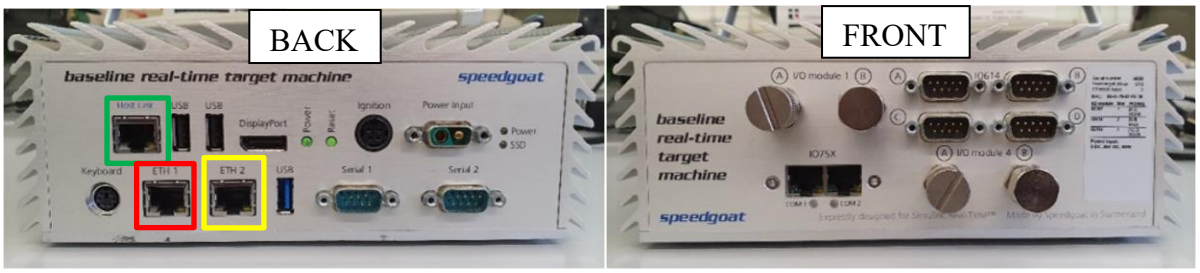


Figure 4.2 – Speedgoat real-time target machine back view (left); Speedgoat real-time target machine front view (right)

4.1.1. Target Machine Initial Set-up

Firstly, it was necessary to establish the connection between the host computer and the target machine. For this, we started by plugging-in an Ethernet cable to the Ethernet “Host Link” port present on the target computer and to the Ethernet port on the host computer. Then, we configured the TCP/IP connection between the target and the host by going to the Control Panel > Network & Internet > Network and Sharing Center > Change Adapter Settings > (Right-click on Ethernet Cable Connection Icon) > Properties > Internet Protocol Version 4 (TCP/IPv4) > Properties Button > (Changed to) Use the Following IP address > (Changed IP address to 192.168.7.2). This information can also be found in the MathWork website.

By using the “slrtexplr” instruction on the command window. This opened a new window dedicated to displaying the target’s information, such as the status of the connection, and the applications load to the target, as represented in Figure 4.3.

In this new window, on the top-left, a target icon should appear with the name “Target PC1”, however the connection should not be settled yet, and a red led indicator should be on. Afterwards, by simply pressing the previous referred icon the red indicator should turn green meaning the target is now properly connected to the host computer and ready to communicate with MATLAB. To verify if the setup was performed correctly, we used the “!ping” command on the prompt command window that displayed a confirmation message with the target network information.

With the connection properly setup, the version of Simulink Real-Time™ running on the target machine must be the same as the one on the host computer. This verification can be performed by connecting a monitor to the Speedgoat target machine via the display port, or by using the command “tg=slrt” follow by “tg.viewTargetScreen”. When using these commands, a new window with the real-time target machine information at that instant will be displayed. The target information is presented on the bottom panel of the window, as illustrated in Figure D.0.2 in annex D.

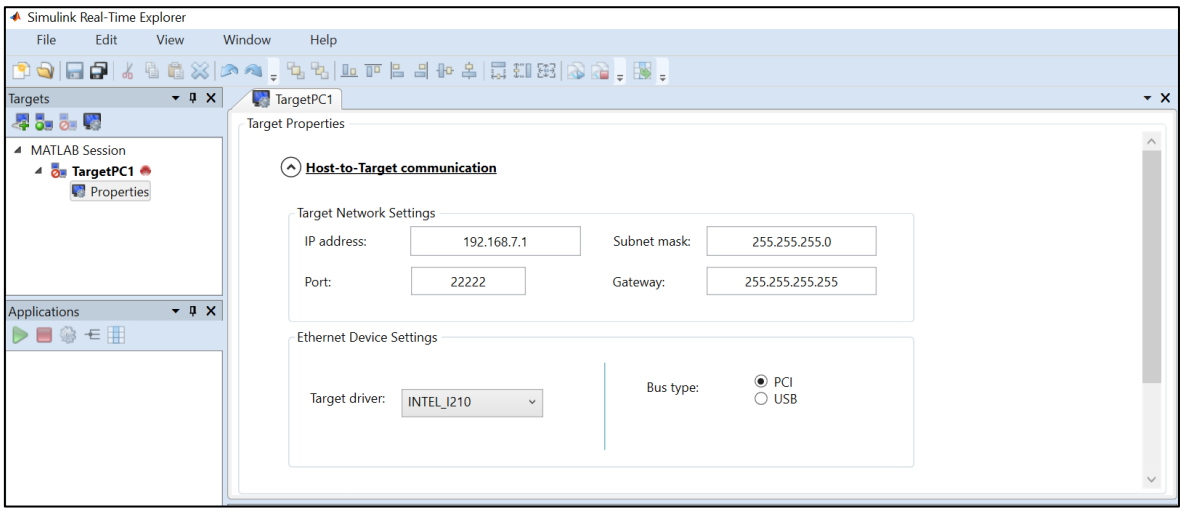


Figure 4.3 – Simulink real-time explorer window, displayed after the command “slrtexplr”.

4.2. Communication Protocol

4.2.1. Meca500

Although not utilized in the final application, the Meca500 was simulated using MATLAB and Simulink software, laying the foundation for the later developments with the Kassow robot.

This subsection will present an overview over the communication protocols considered for the development of the Meca500 Simulink model.

4.2.1.1. EtherCAT

EtherCAT or Ethernet for Control Automation Technology is a real-time industrial protocol developed by Beckhoff Automation and introduced in April 2003 (EtherCAT, 2021).

The reason for choosing this protocol in the first place was related to how much information was available on the programming manual in contrast with the TCP/IP protocol. However, it is worthy to point out advantages that came from using EtherCAT over TCP/IP, such as a faster response time (in our case 0.2 ms), suitable for deterministic operations, and also a great synchronization.

An EtherCAT Master device can be configured in Simulink using the "EtherCAT Init" driver block, which requires an ENI file. This file can be generated as follows:

- Connect the host PC to the EtherCAT network where the future Master node will be located.
- Scan the network using Beckhoff TwinCAT or EtherCAT Configurator.
- Create at least one Input and Output task.

Figure 4.4 represents the Simulink model created for sending the commands using EtherCAT protocol. This model has incorporated switches and buttons on the actual model that are linked to specific subsystems blocks, resulting in an improved testing time since it was not necessary to create a dedicated application.

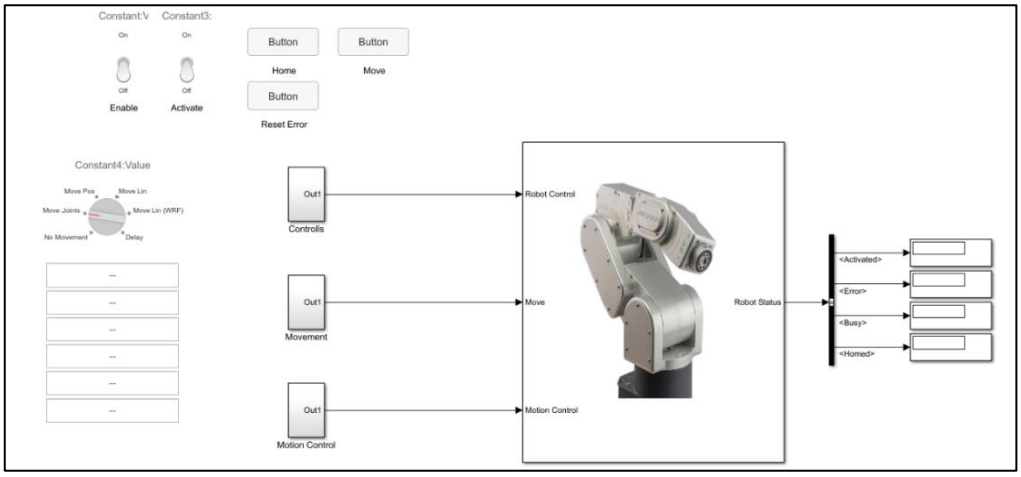


Figure 4.4 – Meca500 Simulink model developed from the commands in the user programming manual

The Meca500 EtherCAT protocol has five main objects available for interaction:

- Robot control – This controls the robot's initialization and it has four subindexes: Enable, Activated, Home, Reset error.
- Motion Control – This controls the movements of the robot. It has four subindexes: Clear move, Pause, Resume, SetPoint.
- Movement – This consists in a pair of registers and regroups the main motion commands. In total, there are seven registers with the first one being exclusive for the indication of the motion commands and the remaining six subindexes for the arguments of the motion command. The motion commands are: No movement (0), MoveJoints (1), MovePose (2), MoveLin (3), MoveLinRelTRF (4), MoveLinRelWRF (5) e Delay (6).
- Robot Status – This object gives the information of the actual robot state. It has five subindexes: Busy, Activated, Homed, SimActivated and Error.
- Motion Status – This object gives the information of the actual robot motion state. It has three subindexes: FIFO space, Actual conf, and Paused.

These objects were divided into four different categories (robot control, motion control, movement, and robot status) and can be found inside the subsystem containing the MECA500 pictures in the previous figure, as illustrated in our Simulink model present on Figure D.0.3 in Annex D.

4.2.2. Host Computer, Kassow KR810 and Speedgoat Target Machine

The Kassow was the robot used to validate the controller. The previously presented model and protocol is just valid for the Meca500 since the protocol utilized with the Kassow was the TCP/IP. Since the laptop, Speedgoat and Kassow both used the same communication protocol the configuration used in each pair (Host-Speedgoat and Speedgoat-Robot) will be explained for each pair in this section.

From this chapter onwards we will refer to the model running on the development computer as the OCR (Optical Character Recognition) model, and the one running on the target computer as the real-time model.

This subsection will present an overview over the communication protocol considered for the development of the final controller. The relation between the different devices is schematized in Figure 4.5.

Figure 4.5 – Schematic representation of the different relations in the TCP/IP protocol

4.2.2.1. TCP/IP

The communication between the different components of the system represented in Figure 4.5 was established through TCP/IP, or Transmission Control Protocol/Internet Protocol. This protocol consists in two layers: The top layer, TCP, oversees the compilation of the data into packets and sends it to a fellow TCP layer, which converts these packets back to data. The bottom layer, IP, is the internet location of the pair (Sanchez, 2015).

Speedgoat provides a library with dedicated I/O blocksets to work with every available protocol. Since the TCP/IP is a fairly simple protocol there were no significant different between using the common Simulink blocks or the Speedgoat specific blocks, so to keep it reproducible we used the ones available from the Simulink Real-Time toolbox.

As Schematize in Figure 4.5, the Speedgoat target machine will operate as a server when exchanging data with the host computer and as a client when exchanging data with the robot arm. The reason for choosing the host computer as the client is correlated to the building process of the real-time model. The TCP/IP blocks from the instrumental toolbox are only available as client blocks, and cannot be deploy into the real-time model, since during the building process a server with the specific port is not yet available leading to an error and therefore stopping the building process. As it can be seen in Figure B.0.1 from lines 263 to 278 in the real-time application code, the model on the target machine launches before the one on the host computer. This is done in order to activate the TCP/IP server necessary for the client blocks to start the execution of the OCR model.

In short, in a TCP/IP communication both client and server can send and receive messages, and the differentiation is only made based on how the connection is created. The server waits for a conversation at a specific port but does not start one, in contrast, the client initiates a conversation at the specified port. When this conversation is initialized so is the connection. This can be seen in Figure 6.6 b), in which the blue signal capture from the TCP/IP server block, running on the Speedgoat target only goes high when the OCR model, with the client TCP/IP block, starts executing.

Although the main communication between the development computer and the target is through TCP/IP and that it was already explained how to configure this in our host computer, in the first section of this chapter, it was also necessary to use and configure another set of TCP/IP blocks for the communication between the real-time model and the OCR model (Target PC).

Given that the target operates as client and server it requires two pairs of TCP/IP blocks as exhibited in Figure 4.6.

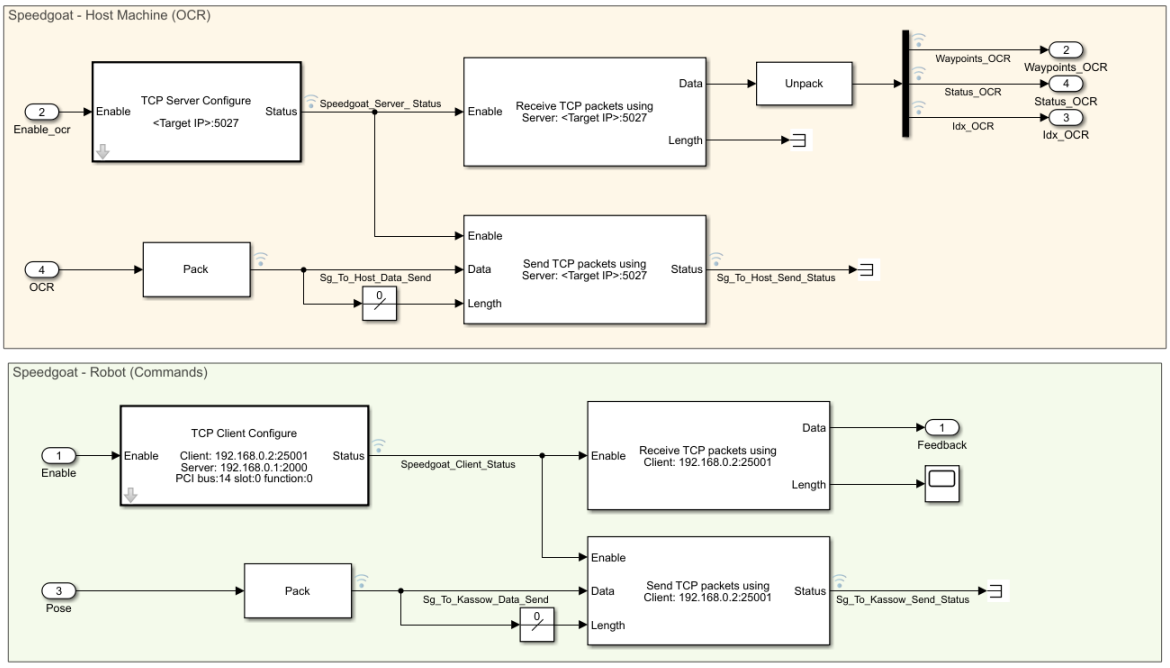


Figure 4.6 – Representation of the TCP/IP blocks used in the TCP/IP subsystem of the Simulink real-time model. Simulink blocks used to send the messages between the target and the host computer (top). Simulink blocks used to send the messages between the target and the robot arm (bottom).

The top red area is reserved for the OCR messages. In here we find the "TCP Server Configure" block. The procedure to configure this set of blocks is illustrated in Figure D.0.4 in Annex D. This server configure block had the capability to setup the intended communication just by checking the square box "Use host-target connection". After selecting this option, we just defined the server port we wanted to use. In our case we selected the 5027. It is worth mentioning that the ports 2222 and 2223 are not available since Simulink reserves these ports for its own use.

The send block was configured just by selecting the correct connection available from the drop-down menu. However, to configure the receive block, we also needed to give the proper width of the signal. Since a byte has eight bits, the dimension will be the result of the multiplication between the number of signals received and the eight bits, resulting in $5 \times 8 = 40$ width. It was also important to use and configure the "byte pack" and "byte unpack" blocks, since they allowed to organize and divide the different signals sent and received in the packages data blocks. To properly configure the "byte pack" block, we needed to know the correct data type of signal input and the desired output data type. This specific "byte pack" block receives a uint8 data type signal and needs to output a signal of

the data type of double. To properly configure the “byte unpack” block, it was required to know the desired output data type and the dimensions of the cell array. The output signal was of the data type double, and the cell array dimensions was the number of signals sent, in our case five. Note that the size of each signal was pre-define in the Simulink model explorer, otherwise the received signals might not match with the sent ones, Figure 4.7 illustrates the example for the waypoints signal.

The bottom green area is reserved for the robot arm commands. In Figure D.0.5 in annex D are presented the figures that illustrate the configuration of the blocks used. In here we find a “TCP Client Configure” block. To setup this block we filled the information about the IP client address, server IP address, server port, client local port, PCI Bus, slot, and function. The information for the first three was obtained from the robot arm controller, since usually when dealing with robots the server IP address and server port are already configured. The client local port could be a random number between 1 and 65535 apart from 22222 and 22223, in our case we selected the 25001. The information for the last three requirements, was found from the method already explained in section 4.1. The send block was once again configured just by selecting the correct connection available from the drop-down menu. To configure the receive block, the width of the signal was once requested. Although, for this the width was not obtained from the previous multiplication but is rather the maximum dimension of the ASCII signal received. The reason behind this, is the fact that the response from the robot arm controller had a variable size depending on the type of data sent (if we were sending joint angles or cartesian positions). Note that in here we do not need an unpack block since the type used in the model is the same as the one received.

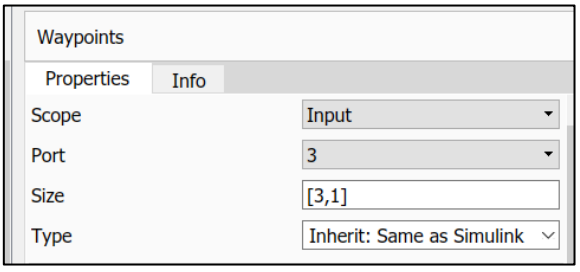


Figure 4.7 – Waypoint’s window properties

5. CONTROLLER DEVELOPMENT

In the present chapter, the steps and methodology implemented in the different software used is given. Over the past few years, hardware prototypes are being replace by hardware/software simulations due to the high cost and extensive times required to create those prototypes. To build a digital twin and to establish the communication between the robot and the user, several were the software used to guarantee the human-machine interaction.

To start, we will mention the computer-aided design (CAD) software and explain how we developed the adapter indispensable to attach the end-effector to the stylus pen and also how we created an exact virtual representation of the robot arm. Afterwards, a brief description of the Ultimaker Cura software needed to convert our CAD files into G-code will be given. Lastly, it will be explained in great detail the software used to set and run the simulation environment, develop the controller, and create the graphical user interface (GUI) along with the different toolboxes required to implement the different tasks.

5.1. SolidWorks®

This software was developed in 1995 exclusive for Windows by SolidWorks Corporation. However, it became a product of *Dassault Systèmes* in 1997 (Systèmes, 2021).

SolidWorks is a CAD software capable of performing 3D mechanical design, simulation and 2D documentation. It aids in the development of a virtual representation of a component by establishing dimensions to the component and validating it before it is manufactured.

During this dissertation, this software was extremely useful for the development of the stylus pen adapter, the assembly of Meca500 CAD files and the Kassow KR810 CAD files, and finally for the assembly between the robots and the tools. In short, it was responsible for the creation of our digital twin's geometrical model.

5.1.1. Development of the Stylus Pen and Adapter

For the intended application, it was necessary to choose the right stylus pen. In an early stage of the development, it was considered the possibility of creating our own stylus, but preliminary tests showed that the performance was not consistent, and the idea was withdrawn. However, there is a good range of options available on the market. Since we intended to present a low-cost and simplistic solution and that the tasks did not require high precision, there was no need for an active stylus pen. From this, the pen chosen for this application was a small generic stylus pen easily found in any store, as illustrated in Figure E.0.1 in annex E.

After the selection process was completed, it was necessary to create a digital representation of the pen. With the help of a pachymeter, we took precise measurements, and a 2D sketch was drawn in SolidWorks. Then, after double-checking everything, it was converted into 3D. With these measurements and the virtual model, we moved on to the next step: developing an adapter. For the intended purpose, this adapter was constructed with the following considerations:

- **Simplicity** – The adapter should respect the drilling pattern present on the robot arm and accommodate the pen but at the same time should keep the same orientation as the robot flange. In this way the TRF (Tool Reference Frame) will be the same as the FRF (Flange Reference Frame) leading to the simplification described in (3.18) – (3.19).
- **Low-cost and fast production** – It should be easy to 3D print. The material chosen was Tough PLA (polylactic acid), a material with similar strength and higher stiffness when compared to ABS (acrylonitrile butadiene styrene) making it perfect for this application.
- **Practicality** – Since Speedgoat intends to use the robot arm as a demo on fairs, the adapter should be easily fitted inside the robot arm case.

Once these requirements were fulfilled, a sketch was made considering the previous dimensions. Figure G.0.4 shows the CAD measures. However, it is extremely important to consider the printer resolution and the material proprieties due to the hole's diameters tolerance. Since it is extremely difficult to get this exactly right, we applied an

iterative printing process, in which we incremented 0.1 mm each time. The chosen stylus pen and the drawings can be found in Figure G.0.2 and Figure G.0.3.

5.1.2. Meca500 Assembly

The CAD model for the Meca500, was provided by Mecademic in a STEP file format.

After evaluating the provided assembly, some corrections were made. The modifications consisted in creating the respective centre origins, rotation axes, and coordinate systems for each joint. The result is illustrated in Figure 5.2 a).

The software responsible for the simulation can only accept XML or URDF files as inputs. The URDF file describes both the kinematic structure and the visual appearance of the robot. This means that the DH parameters presented in Chapter 3 are automatically calculated. SolidWorks does not export to these formats. However, to solve this issue there are two available options. The first one is a plug-in called SW2URDF, that allows to directly save the 3D part into a URDF file. The other option is a plug-in offered from MathWorks, called Simscape™ Multibody™ Link. For this dissertation, the first option was used.

In the newer versions of SolidWorks, the plug-in will appear in the tab “Tools” in an option named “Export to URDF”. When proceeding to the creation of the URDF file a new side window named “URDF exporter” will show up. In here, we will need to select some parameters such as the coordinate system and rotational axis for each link. It is not advisable to select the method “automatically generate” since this will lead to errors resulting from random axis generation within the model. When everything is properly defined, we can click on the button “preview and export”, where we will need once more to define some variables/parameters (i.e., weight).

Firstly, since we only know the total weight (4,5 kilograms) and not the individual mass for each link of the robot arm, a speculative division was made. Assuming that, the base is the heaviest element of the robot, it was given a weight of 1 kilogram. Then the remaining 3,5 kilograms were divided as follows: 70% is distributed along the vertical axis and 30% on the horizontal axis. With this division, we tried to follow a common approach while avoiding high shear stress forces and momentums created in the robot arm that would damage the electric motors, leading to a reduction of its useful lifetime. Figure 5.1 illustrates this partition along with the estimated mass values.

Secondly, it was critical to define the type of joints and their minimum/maximum range. From the information presented in Figure AC.0.4 in appendix C, a joint with no translation and just one rotation is known as revolute joint. The information regarding the mechanical limits of the joints is given in Figure AB.0.5. This information can also be found on page 3 of the Meca500 programming manual.

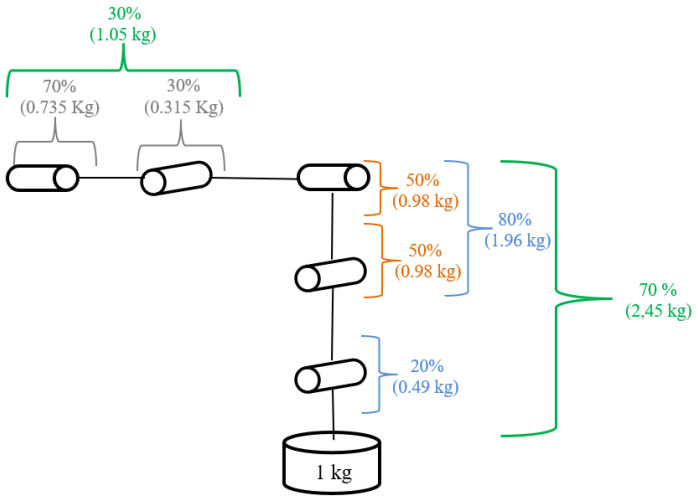


Figure 5.1 – Schematic of the mass distribution for the Meca500

Finally, we completed the process by filling the values on the locations marked in Figure E.0.2 in Annex E. Once again, this is only the representation for the first link. The process is the same for the remaining joints. Although, it is worth mentioning that the column named "Axis" tends to invert the *z* values and the user should pay attention to this situation when exporting.

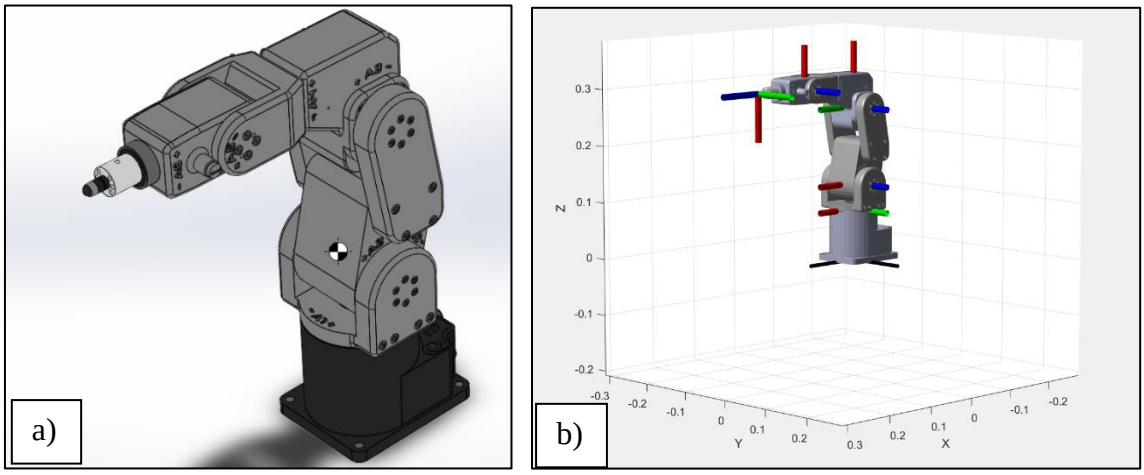


Figure 5.2 – (a) Meca500 final assembly in SolidWorks. (b) Meca500 model used in MATLAB.

5.1.3. Kassow KR810 Assembly

For the Kassow KR810 the CAD model was also provided in STEP files. By assembly each single item one to another, in SolidWorks, we were able to obtain the complete assembly that would match our robot arm.

Then, it was also necessary to build a final assembly (robot + adapter + screws + tool). Since the robot available was the Kassow, we were able to select the specific screws used to attach the adapter to the robot end-effector, and the stylus pen to the adapter as represented in Figure G.0.6. In this case, we can say that our model would be a trusty representation of his real-life counterpart.

When this final assembly was completed, the new reference frames and rotational axis were created since they would be required as input data for the plug-in. Given that the mass values for the Kassow and screws were correct, there was no need to perform the previous estimation process described for the Meca500.

Then, the SW2URDF plug-in was once used to export our 3D model to a URDF format file, so it could be imported to the Simulink environment. The steps followed the same methodology previously explained.

The final 3D model present in the simulation environment is represented in Figure 5.3.

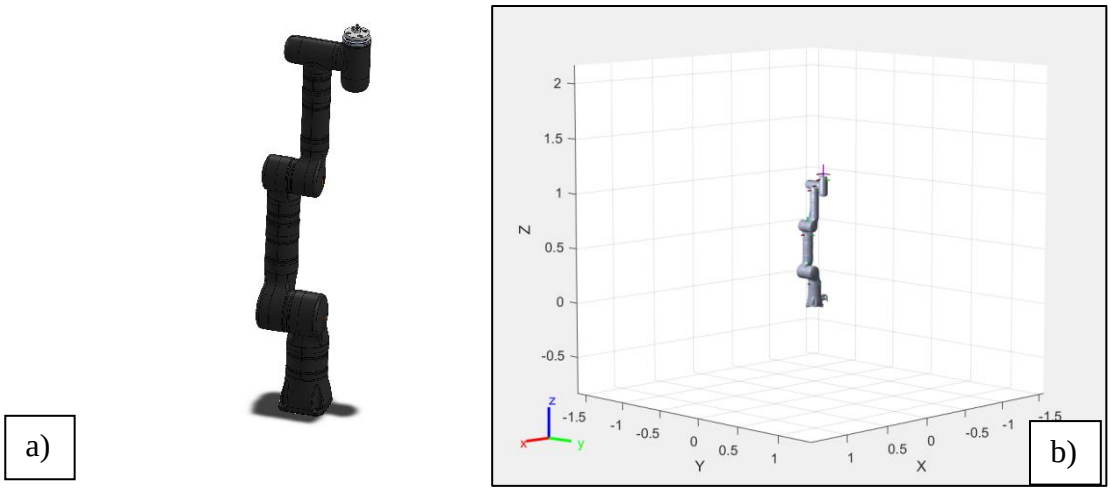


Figure 5.3 – (a) Kassow KR810 final assembly in SolidWorks. (b) Kassow KR810 model in MATLAB.

5.2. Ultimaker Cura 4.8.0

The Ultimaker Cura is an application developed by Ultimaker that allows to define and preview the 3D printing of a 3D object by reading a CAD file and turning it into a G-code file. This type of format is used in CNC machines and 3D printers since it provides essential data, including the velocity of the tool, position, and orientation.

The printer at our disposal was the Ultimaker S3. With the Ultimaker Cura, we were able to define the material, the infill density, and the infill pattern. The material was mentioned in section 5.1.1.

The infill density refers to the quantity of plastic on the inside of the item. A higher infill density leads to a reinforced object. Normally, the range of density for a small item varies between 20% to 80%.

For the Meca500 adapter it was shown that due to his size, shape, and presence of holes the best results were achievable from an octet pattern with a 70% of infill density. Figure 5.4 illustrates the software environment as well as the time and weight for our printable item.

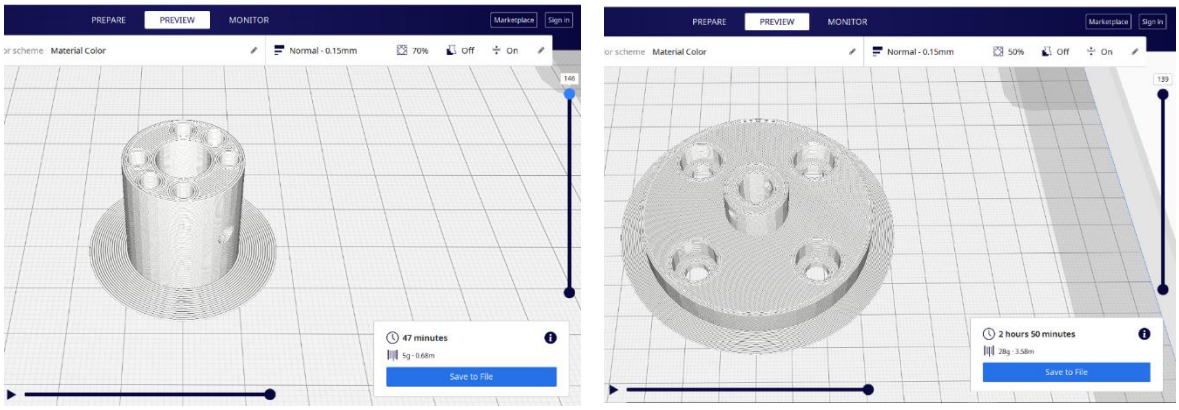


Figure 5.4 – Ultimaker Cura software with the Meca500 adapter (left) and with the Kassow KR810 adapter (right).

The adapter produced for the Kassow has a different shape and size that will result in different parameters. The pattern selected was the same as the previous one, but the infill was decreased to just 50%. This resulted in a printing time of 2 hours and 50 minutes and a weight of 28 grams. The material is the same for both items.

5.3. MATLAB® & Simulink® R2019b

MATLAB® (or matrix laboratory) is a programming software released in 1984 by MathWorks. It can be used in Windows, macOS and Linux. MATLAB is drawn upon a matrix-based language where problems and solutions are expressed in mathematical notation making it rather intuitive to learn and operate (Mathworks, 2016).

In this chapter, a description on how the capabilities of this IDE (Integrated Development Environment) were essential to develop algorithms, create models and applications.

5.3.1. Simulink®

This software is an add-on to MATLAB with a user-friendly graphical environment that allows to design, model, and simulate the dynamic behaviour of a virtual system. Simulink uses the MBD methodology, making it suitable for rapid control prototyping applications. Simulink is a very powerful and important tool in the fields of aerospace, automotive, and industrial automation. In fact, MBD with Simulink is transforming the way engineers work (Mathworks, 2020).

For modeling, the provided GUI (graphical user interface) enables to build models and block diagrams just by simply drag-and-drop the respective blocks from the library. It is worth mentioning the pre-install extensive library with the possibility of including even more diagram blocks as we install additional toolboxes suited for our needs. Simulink® Coder™ generates the C and C++ code from the model that will be deployed into the real-time target machine or embedded system. As a result, the user/system integrator can obtain an "up-and-running" model in a fraction of the time needed if he had to write each line of code from scratch with the added bonus of eliminating errors-prone from handwriting code.

For the construction of the model required for this dissertation the toolboxes installed will be described during this chapter. These toolboxes were crucial for the implementation of the several functionalities of the controller.

5.3.2. CAD Import

Simscape Multibody™ provides a multibody simulation environment for 3D mechanical systems, such as robots. These systems bodies, joints, constraints, force elements, and sensors are represented by blocks as demonstrated in Figure 5.5. With it, it is possible to import complete CAD assemblies properties, including all masses, inertias, joints, constraints, and 3D geometry (Mathworks, 2021c).

With this tool we were able to import our CAD model, from the generated URDF file, and consequently we created the digital twin of our robotic arm. The surrounding elements of the simulation such as the floor, the device holder, and the device itself were created directly in Simscape without the need to resort to our CAD software.

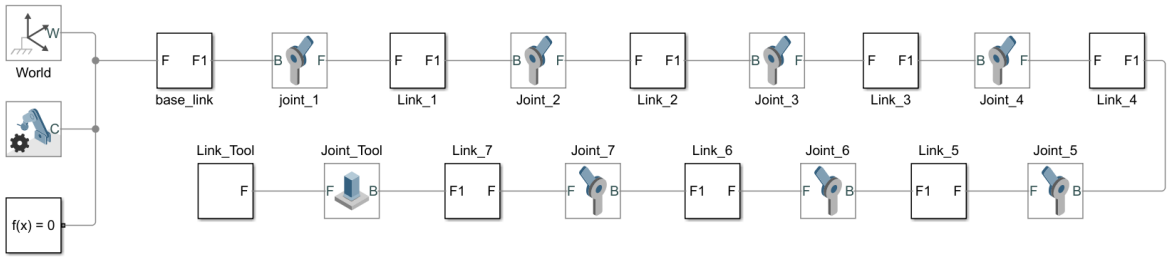


Figure 5.5 – Simscape Simulink blocks for the joints and links of the Kassow KR810

5.3.3. Kinematic Configuration

Due to the proximity between MATLAB and Simulink that allows for a combination among kinematics and dynamic models, it is possible to create and make the robot execute a trajectory in just a few steps.

The Robotics System Toolbox offers the possibility to design, simulate and test manipulators for different type of robots such as mobile robots, humanoid robots, industrial robots, and collaborative robots (MathWorks, 2020). A rigid body tree model can be created either manually by adding each individual link as a body or by importing the CAD, URDF or XML assembly file with a prompt command. This rigid body tree object is used to represent the robot configurations required in the forward and inverse kinematics calculations.

This toolbox offers several blocks that are important to perform and obtain the forward and inverse kinematics as well as run collision free operations. In this dissertation, these blocks were used to accomplish the kinematics of the robot model. The used blocks were: the inverse kinematic blocks (red), the forward kinematic block, the transform trajectory block (green) and the coordinate transformation conversion block (orange).

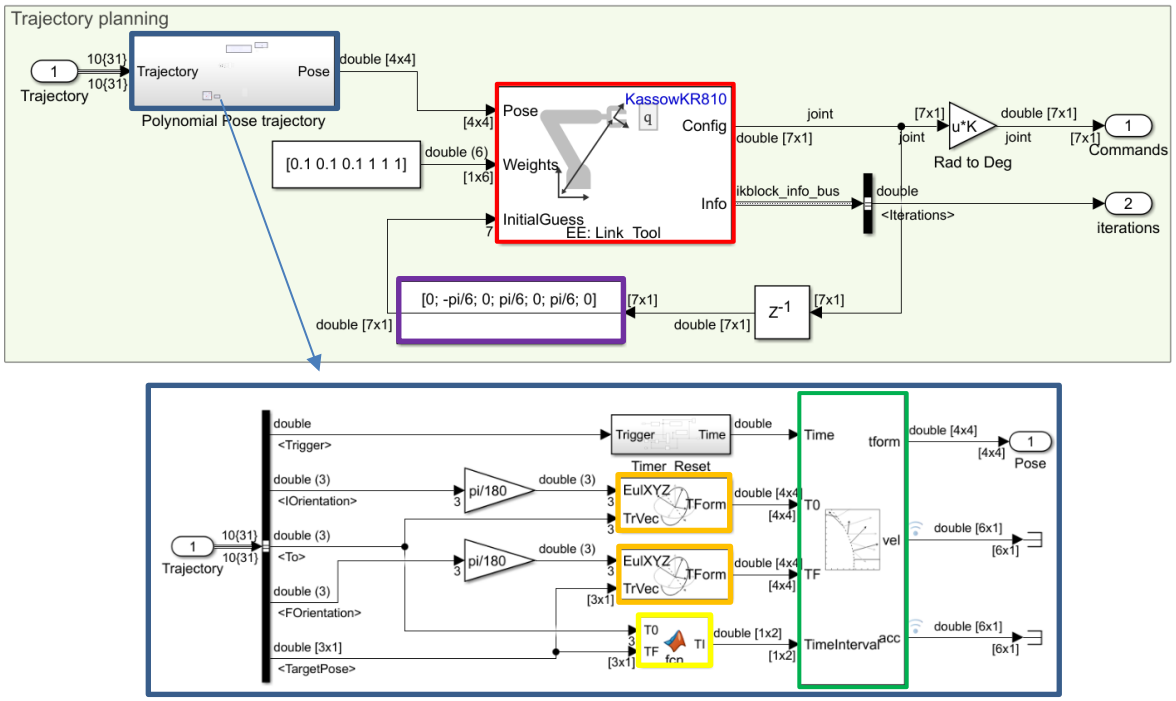


Figure 5.6 – Inverse Kinematic Subsystem used in the Digital Twin Simulink model (top); Functions and blocks used to obtain the polynomial trajectory (bottom)

By utilizing the blocks provided from this toolbox we end up saving a considerable amount of time in the design of our model, since the coding for the kinematics was already provided inside these blocks. Our principal task consisted, in providing the correct inputs that guarantee the correct output results.

The inverse kinematic was obtained from the following sequence of steps: Firstly, the stateflow chart outputs four vectors (in degrees) corresponding to a [3x1] vector with the initial pose, a [3x1] vector with the initial orientation, a [3x1] vector with the final pose, a [3x1] vector with the final orientation, and a one-dimensional signal that can be high or low (1 or 0). Secondly, the transform trajectory block only accepts a [4x4] homogenous transformation matrix, one with the initial and another with the final pose. This meant grouping the previous vectors in two pairs (initial pose + initial orientation) and (final pose + final orientation) and then converting them into the required transformation matrix. This

represented in Figure 5.6 where the signals “IOrientation” and “T0” containing the initial values serve as input for the coordinate transformation conversion block and the happens for signals “FOrientation” and “TargetPose” containing the final values.

The transform trajectory block also requires two more inputs the time and the time interval. The block performs the transformation calculations over a specific period, known as time interval, at a time instant, provided by the timer input, but it is only configured to do it once per simulation, meaning that the block would only work for one execution. To overcome this difficulty a timer reset subsystem, controlled by the one-dimensional signal, was created so whenever a new waypoint was given the time goes back to zero and the block could start the calculations all over again and keep doing it for as long as it was required. To setup the correct time interval, we created a MATLAB function where the yellow block from Figure 5.6 performs the mathematical calculations. In here, the distance in space between two given points is determined and then converted into a time interval simply by dividing the determined distance value for the chosen linear velocity.

Finally, the generated transformation matrix provided from the “Polynomial Pose Trajectory” subsystem is inputted into the inverse kinematics block (red). Since we were working with a seven-axis robot arm, there were redundancies that needed to be considered. This implied the correct configuration of the initial guess input signal provided from the initial block (purple), otherwise the behaviour of the digital twin would not match to the real-life robot arm. The configuration for the initial guess was also configured in the real robot arm.

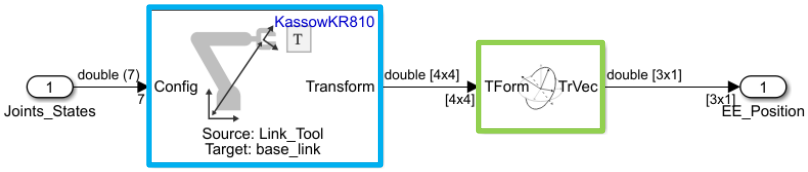


Figure 5.7 – Representation of the Simulink blocks used in the forward kinematic subsystem used in the digital twin Simulink model

The forward kinematic is a much simpler approach since we only fed a [7x1] vector with the joint positions provided from the blocks in Figure E.0.3 in annex E into the forward kinematic block (blue) and then converted the output (a [4x4] transformation matrix) into a translation vector by utilizing once again to the coordinate transformation conversion block (green), as illustrated in Figure 5.7.

5.3.4. Supervisory Control Logic

Stateflow is used to describe how MATLAB algorithms and Simulink models react to events, signals and time-based conditions (Mathworks, 2021f).

In this research Stateflow was used to design and developed our supervisory logic controller. Since Stateflow has graphical animations during execution, analysing and debugging the logic behind it was easy and intuitive.

In this dissertation we will have two different supervisory logic controllers: one for the desktop simulation and the other one for the real-time simulation. Although the logic behind them is the same, due to the inverse kinematics subsystem, represented in Figure 5.6 and present on the desktop model, the number of control variables changes. In both cases the Stateflow is divided in two main stateflows charts, as represented in Figure 5.8:

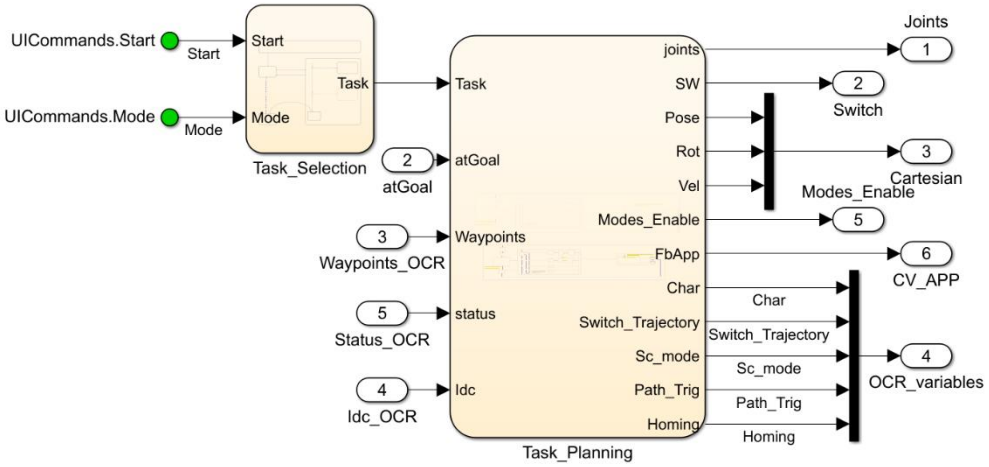


Figure 5.8 – Representation of the Stateflow charts used in the real-time Simulink model

The "Task_Selection" chart receives the input signals from the two UI blocks in inside the green area present on the Simulink model, shown in Figure E.0.4 in annex E, that allows to select between the states of "Home" and "Testing", depending on how the condition is set by the GUI, explained in the next section and represented in Figure 5.9.

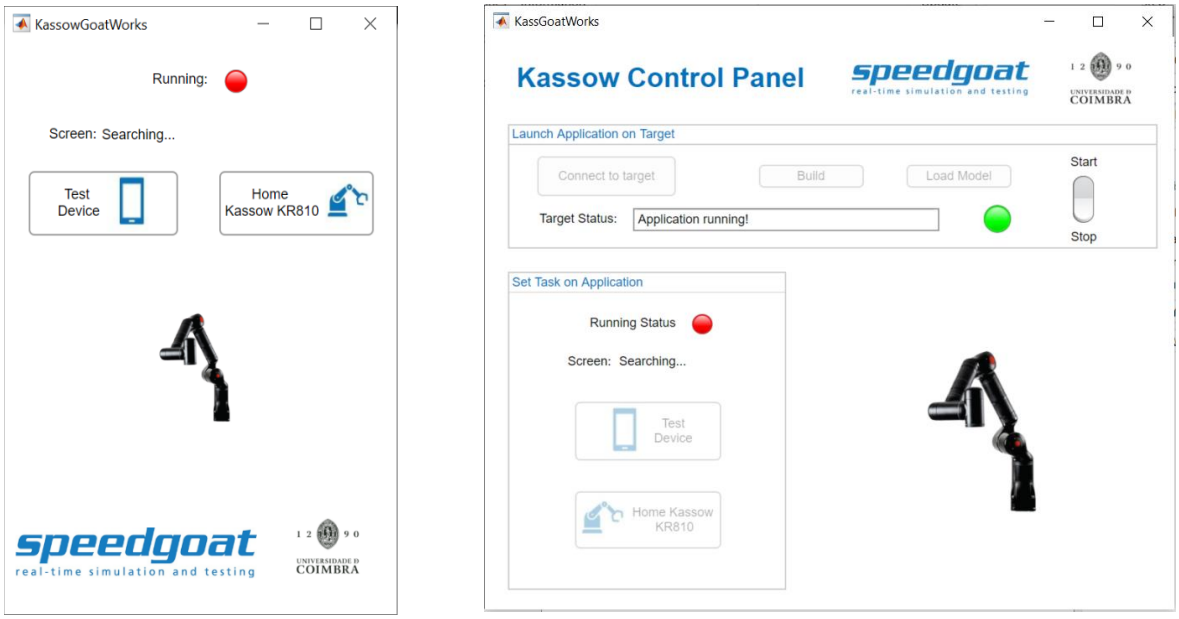
The "Task_Planning" chart was programmed with all the controller logic. This chart is responsible for interpretating the signals streaming from the other chart, from the OCR model (when running the real-time model) or the computer vision subsystem (when running the digital twin model) and from the atGoal subsystem. Both models are represented in Figure E.0.4 and Figure E.0.5 respectively.

The full logic for this Stateflow charts is explained in detail in ANNEX E.

5.3.5. User Interface

The App Designer is an interactive framework for designing and programming the behaviour of an application. With this, the user can build an "up-and-running" application in a matter of minutes given that the app designer will create the code for the graphical components and leave blank spaces for the main functions and callbacks.

The concept of designing and creating an efficient app for this project was simple. Although, it was necessary to build two applications: one to control the digital twin simulation model, named "KassowGoatWorks", and the other to control the model running on the real-time target machine as well as the OCR model named "KassGoatWorks".



a) b)

Figure 5.9 – (a) GUI application used to control the digital twin Simulink model. (b) GUI application used to control the real-time Simulink model.

The main purpose of the applications was to establish a connection with the respective Simulink model, in a way that from the press of a button, we could interact with it.

The digital twin's GUI is the simplest one since it only requires two push buttons: one for executing the task and the other to home the robot, as illustrated in Figure 5.9 a). The programming code for this application is shown in annex A.

The real-time application, represented in Figure 5.9 b) is divided in two parts. The first one concerns the establishment of the connection between the host machine and the target, the building of the model, and the download of these generated files into the Speedgoat target machine. The second one, takes care of the commands necessary to execute the routines. The buttons in the "Launch Application on Target" panel were configured to be activated by order from left to right to ensure the correct understanding and functioning of the process. The status information is provided by the appearance of a feedback message accompanied by a status LED lamp. After the model is loaded into the target, the user just needs to press the Start/Stop switch button to start the two simulations: the one running on the Speedgoat target and the one running on the host computer, as we will explain on Chapter 6. The buttons on the button-down panel in Figure 5.9 b), operate in the exact same way as the ones on the "KassowGoatWorks".

The LED running status lamp present in both GUIs has three possible colour states:

- Green – The robot is executing a task correctly. This one occurs when the robot is performing the testing.
- Yellow – The robot is in stand-by and waiting for a command. This one occurs in two instances: when it reaches the initial home position and when a test is finished.
- Red - The robot is executing a task and cannot be interrupt. This one occurs when the robot is going to the home position.

5.3.6. Computer Vision

The Computer Vision Toolbox™ played an important role throughout the development of this project since it provided functions and blocks that could perform object detection, feature detection and extraction, and it also allowed to train our custom model used for the OCR model (Mathworks, 2021a).

The Computer Vision Toolbox requires the installation of the Image Processing Toolbox™, to enable additional features such as the Optical Character Recognition (OCR) and other important functionalities that are going to be explained in this chapter.

The computer vision subsystem inside our model resorts not only to image processing techniques (for the path generation needed to execute the intended test) but also to the OCR functions for text recognition.

The OCR model used in this research was trained using the app OCR Trainer, a preview of what it looks likes is shown in Figure 5.10. With it, we were able to differentiate the zero from the "o", train specific characters such as the cardinal, the asterisk, and the back button symbol "<". While it might seem that training a lot of images will improve the overall result this is not always true.

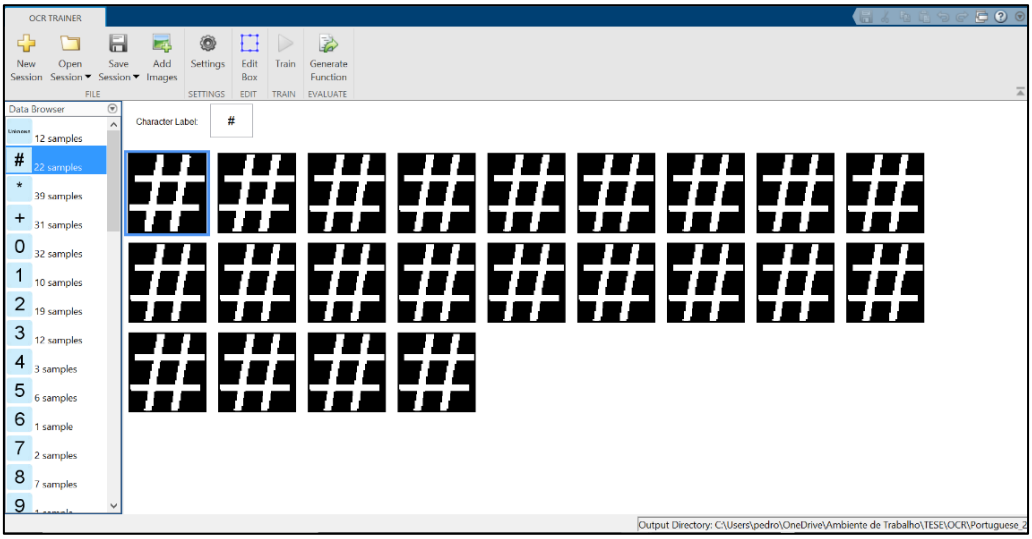


Figure 5.10 – Window of the OCR trainer used to train the OCR model

When training a model, the images should obey to the following rules: have good pixel resolution and have no distortions, since faulty images will result in a decrease in the level of confidence leading to unwanted and faulty solutions. In total, 60 images were used to train the model, leading to the values of confidence shown in Figure 5.11.

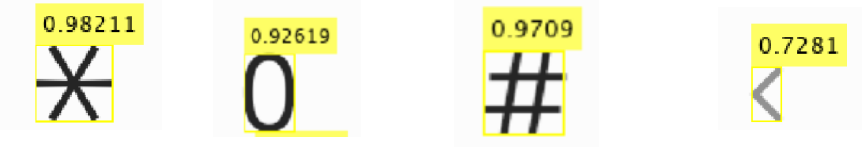


Figure 5.11 – Representation of the levels of confidence for some of the desired characters.

After our trained data reached the desired levels of confidence, two functions were built from this model. The first one “Kassow_OCR_v2” utilizes the OCR model to look for specific characters or words that are being displayed on the screen.

The last one, “Kassow_Lookforcharacter” function locates the intended character. The dimensions on an image are not in millimetres but rather in pixels. This function also performs this conversion and outputs the location in cartesian coordinates in meters [m]. The flowchart that describes the logic of this function can be found in Figure G.0.1. In short, we can describe de OCR as the eyes of the controller.

The image processing techniques were used to obtain and generate a path trajectory from the image represented in Figure 5.12 a). This image had multiple squares from different sizes and shapes. So, to successfully evaluate the image, it was required to process the image multiples times until it was clean enough to generate the correct path.

The first step consisted in converting our image from RGB (Red, Green, blue) to grey, then we filtered the areas with a dimension between 400x1700 pp (this eliminated irregular square shapes and the four white triangles in the centre of the image). The result is pictured in Figure 5.12 b). Then by using the “regionprops” function we were able to draw a circle inside each square, the indexes of these circles were saved in a variable that would later be used to sort the path, Figure 5.12 c). To be able to generate the path for the robot, we needed to make sure that the robot arm would pass through all the circles but only once. To ensure the correct sequence, the index variables were looped until the nearest neighbour was found and then were eliminated right afterwards guaranteeing that the same coordinate would not be chosen again. This loop stops when there are no more coordinates left. The generated trajectory is illustrated in Figure 5.12 d) as well as image obtained after all the refinement.

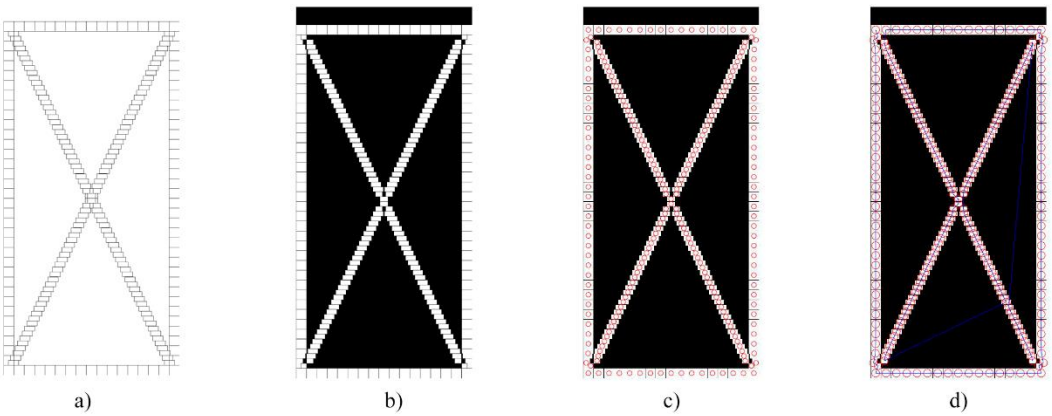


Figure 5.12 – (a) Initial imaged used for path generation. (b, c, d) Representation of the image processing techniques applied to obtain the path trajectory.

This conversion varies from device to device since it depends on the screen resolution and DPI (Dots per Inch). To find the right conversion factor we used a function that works as a spatial calibration. By selecting the image to analyse, we can draw a line between two intended points. This line needs to be measure in our device using a ruler and the distance should be place in the demo. The user needs to trace a line in both X and Y direction. Finally, the program indicates the px quantity by unit that will serve has the multiplication factor for the coordinate's conversion equation. This is an iterative process since it requires some skill to draw a line between the exact points and to read the exact distance from the ruler. After some trials this was the results that most suited our measurements, for X and Y respectively:

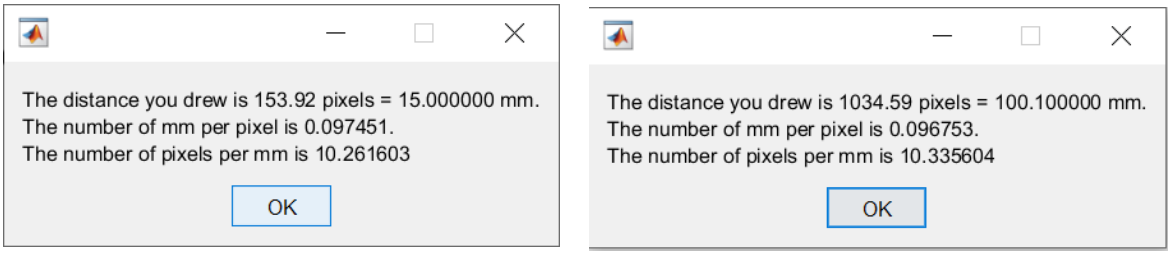


Figure 5.13 – Left: Distance of pixels measure in x direction. Right: Distance of pixels measure in y direction

The input for this subsystem came from a USB video capture device that was connect to the host machine via USB cable and to the device through a HDMI to USB-C cable, as schematize in Figure 5.14. However, it was essential to install another toolbox that provided the blocks that could pass the data from the video capture device to MATLAB and Simulink. This toolbox is called Image Acquisition Toolbox™, and from it we were able to run early tests with images that mimicked the behaviour of the device, detect and configure our hardware, and finally pass live video data from the frame grabber to the Simulink.

The images used to test and validate the model were captured directly from the device. These images were captured with a resolution of 720x1520 px and saved in a PNG format. Then, they were imported into MATLAB by typing the command "imread ('name_of_file.data_type')" in the command window and saved as a variable, for example, HomeS = imread ('Screenshot_Home.png'). Finally, the block "Image from workspace" was drag-on to the Simulink model and we were able to select the desired variable. The images used were save under the following variables name: HScreen, DDial, ST, TT and Module and they are illustrated in Figure E.0.8 annex E.

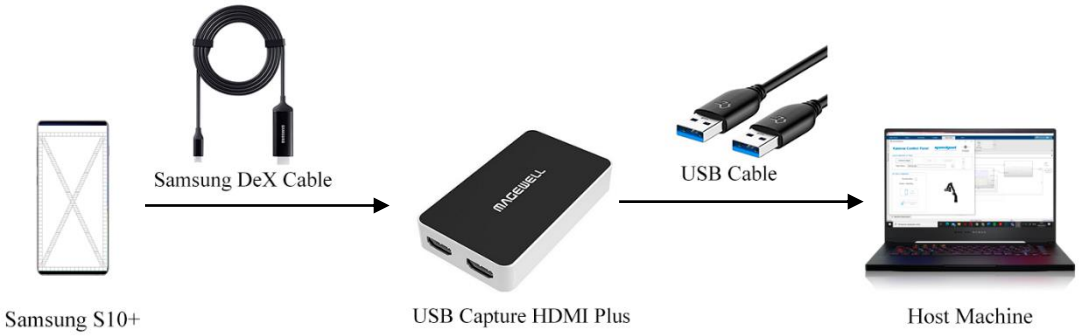


Figure 5.14 – Schematic diagram of the computer vision system

5.3.7. Converting to Simulink Real-Time

Simulink Real-Time™ Toolbox allows the creation of real-time applications from Simulink models and the option to run them on a Speedgoat target computer hardware connected to a physical system. It is designed for real-time simulations and testing tasks, including RCP, and HIL simulation (Mathworks, 2021d). The code generation for the executable real-time application requires the installation of the Embedded Coder Toolbox™.

By going into the app menu and selecting Simulink real-time, Figure 5.15, our model is automatically converted into a real-time model. This will impact the behaviour of the Simulink blocks, since some will not work in a discrete-time simulation. This toolbox also provides specific blocks that can handle real-time changes.

To observe the behaviour of the real-time system and edit the block parameters at real-time, we needed to set Simulink's external model. From this external mode, the block parameters can be edited while running the simulation. New parameter values are automatically transferred to the compiled version of the model during real-time execution (Valera et al., 2001).

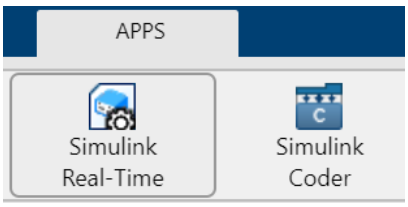


Figure 5.15 – Representation of the Simulink real-time application used to convert the model

6. CONTROLLER VALIDATION

In the present chapter, the controller development problem is brought to a higher instance, where all the subsystems are connected to each other completing the loop of the system. In this chapter, we will also be able to verify if our model is able to run on the Speedgoat target machine and finally what optimizations can be implemented. For this, we will convert the model into a real-time model, using the Simulink real-time app, illustrated in Figure 5.15.

This chapter will be divided into three validation phases. The first one concerns the validation of the digital twin model built and simulated using only offline simulations, the second is focusses on the compatibility between the developed model and the Speedgoat target machine, and the third is towards the deployment of the model to the physical machine.

6.1. Offline Simulation Testing

The first step in the model-based design methodology concerns the offline simulation. This step is performed directly within the Simulink environment, and it is essential before executing real-time tests, assuring the complete requirements-based testing. From this we were able to debug our model and make design changes that satisfy the requirements, debug the stateflow sequence and improve his efficiency, test the controller, and observe the digital twin's behaviour. These kinds of simulations are also easier to configure, modify and improve.

This offline simulation can run slower or faster than real-time depending on the development computer's hardware, but it can also be forced to run in real-time. To achieve this, the simulation pacing must be set to one and configured as represented in Figure 6.1. In this research we forced the simulation pace to real-time. However, due to the high processing power required the system could not ran in real-time at some moments and a warning was displayed during execution. From these initial tests, we were able to conclude that the host computer would not be able to execute every coded function in real-time.

Since the simulations run only on the development computer, Simulink and MATLAB, can easily exchange data in background. This facilitated the data exchange between the GUI application and the Simulink model, since the functions developed in MATLAB could be called to run directly in the Simulink model without limitations and with no data lost during execution.

By running offline simulations, we were able to capture digital signals directly from the model. These signals behaviour can be observed and interpretate inside the data inspector and we can visualize our data, as represented in Figure 6.2.

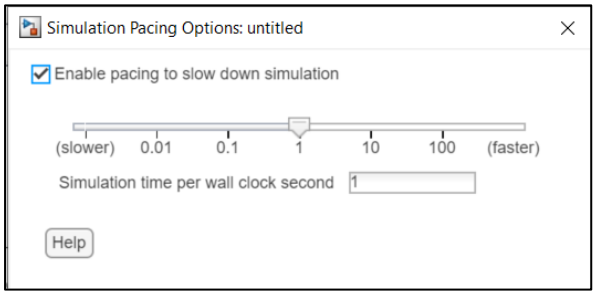


Figure 6.1 - Window for the simulation pace configuration

6.1.1. Digital Twin Results

The digital twin model was the first goal in our project timeline. With is development we wanted to demonstrate how a digital representation can improve the controllers design and efficiency while saving time.

Before deploying the developed controller to the robot arm, we were able to simulate and visualize the robot behaviour due to the powerful simulation environment in Simulink, as represented in Figure 6.3. This 3D simulation combined with the obtained graphics in Figure 6.2 helped us analyse and correct the controller performance, assuring a safe deployment in the physical system.

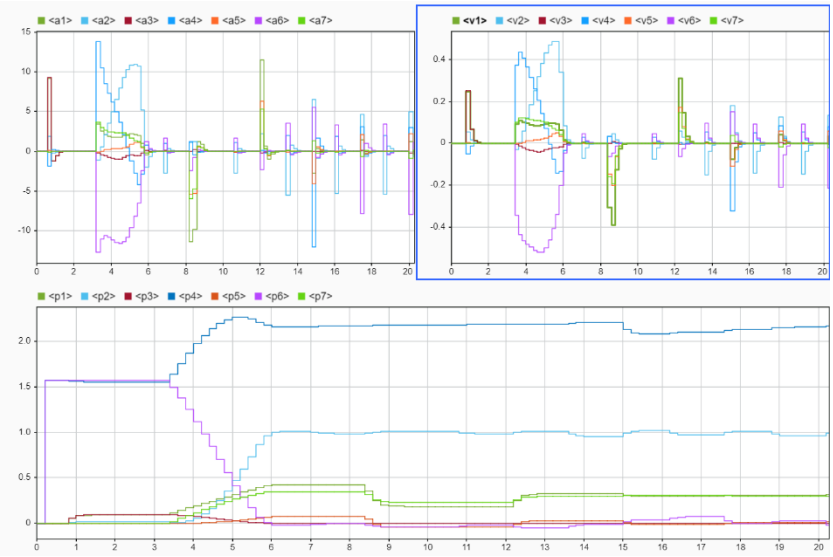


Figure 6.2 – Kassow KR810 joint properties for 20 seconds of simulation: Joints acceleration values in $[\boldsymbol{rad/s^2}]$ (top left). Joints velocity values in$[\boldsymbol{rad/s}]$ (top right). Joints position values in $[\boldsymbol{rad}]$ bottom.

Since we were working within a digital system, everything is measurable, meaning that we could obtain every signal running inside the system. In Figure 6.2 is represented the graphs with the values for the acceleration, velocity, and position for all the joints of the Kassow KR810 for a time interval of 20 seconds of simulation. By looking for the maximum attained values during a complete execution test and compared them to the ones presented in Figure 2.6 we were able to validate if they met the requirements. The GUI present on the Kassow teach pendant, in Figure 6.8, only shows the rotation of the joints or end-effector cartesian pose (position + orientation). From graphics like this one, we can tune our parameters without provoking unwanted damages to the motors or surroundings.

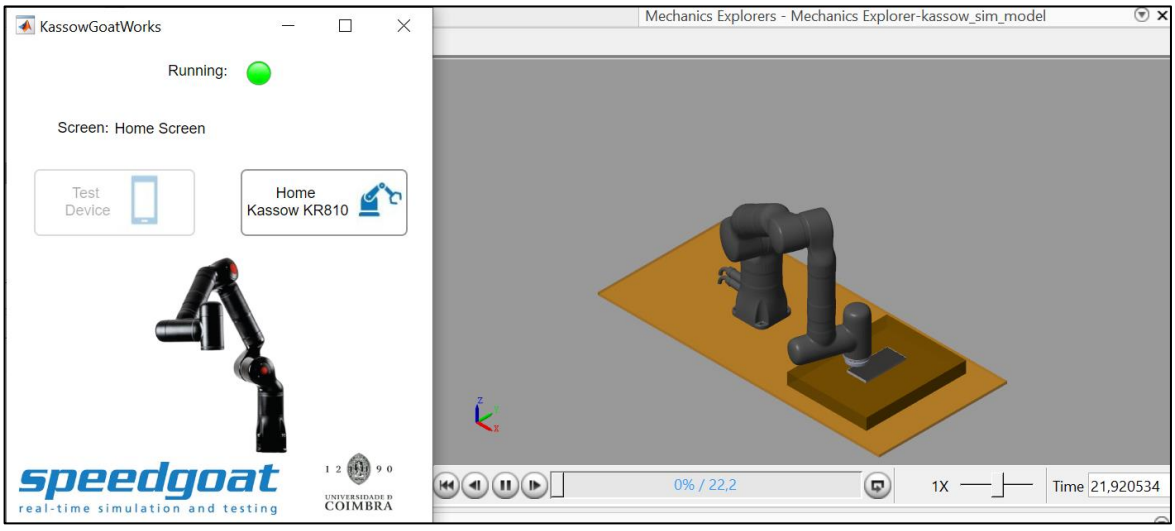


Figure 6.3 – Left: GUI for the offline simulation. Right: Graphic representation of the simulation in the Simulink environment

6.2. Real-Time Simulation Testing

Real-time testing often takes longer time than comparative model testing, especially when executing a variety of real-time tests that cover several scenarios. The real-time test can be set to run on a standalone target computer connected to the physical model, with the use of the Simulink Real-time.

One of the advantages of performing real-time simulations and RCP is the possibility of capture the signals running inside the model, since we are still working in a digital system and not in a physical one. These signals can be utilized to debug the model, improve the design, and observe changes and parameters only possible within a digital system.

6.2.1. Validation on Speedgoat Target Machine

When working with a real-time target machine it is important to understand that not all MATLAB/Simulink functions and blocks will be compatible and why is so important to test the model systems individually.

The first verification made with the provided target was exactly this. We started by creating standalone models to each individual systems. Then we converted them into real-time, using the approach described in section 5.3.7, and finally we deployed them into the target one at a time. If the routines were compatible the simulation would start. However, if they were incompatible the simulation would not start, and a warning would be displayed. From this, we found out that the only incompatible system was the computer vision, containing the OCR functions. Usually, functions that work based on neural networks, such as machine learning, deep learning, and some computer vision ones like the ORC, they required a GPU to be able to perform multiple computations simultaneously. Nonetheless, this obstacle can be overcome since MATLAB offers the possibility to run multiple models at the same time. The strategy consisted in running one simulation on the Speedgoat target and the other one on the development computer as schematize in Figure 6.4. In essence, with this approach, we could run everything apart from the computer vision blocks on the real-time target machine while keeping all the required functionalities.

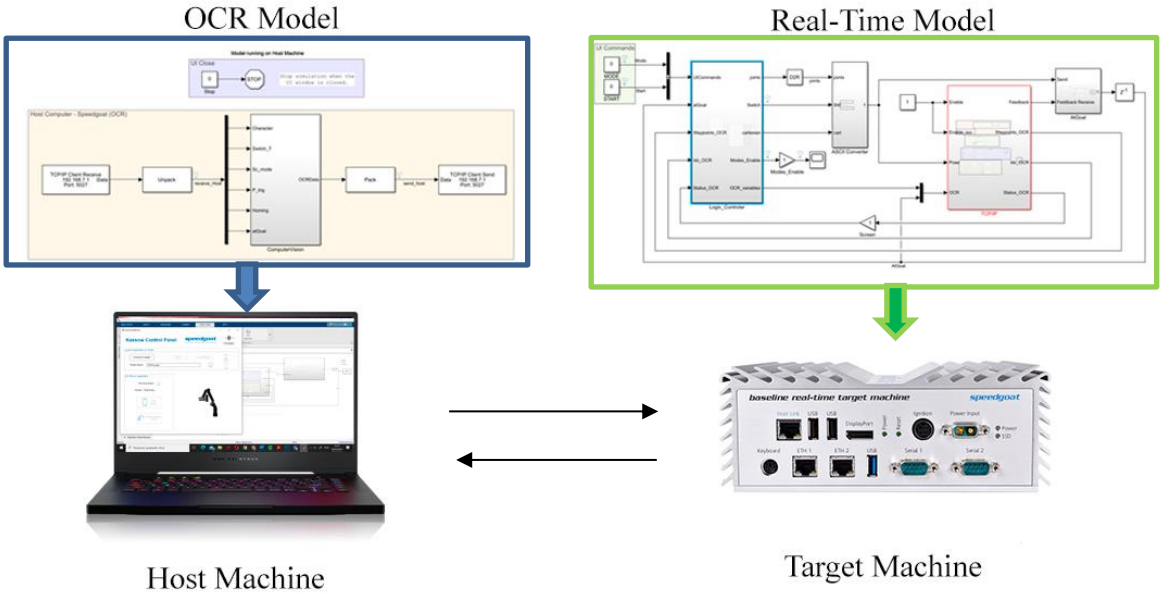


Figure 6.4 – Diagram of the deployment of the models used for the real-time simulation

Then we tested if our application, "KassowGoatWorks", created with app designer could also be used to control the real-time model. The result turned out negative. We conclude that more functions should be implemented and that the code needed to be rewritten. So, we developed and coded a new application, already described in Chapter 5, and referred to as "KassGoatWorks".

Afterwards, we tested the functionalities of the stateflow and the configuration of the TCP/IP blocks already explained in section 4.2.2.1. From this set of tests, we realised that, unlike what happened on the closed loop digital twin model, we could not send strings messages directly from the model to the robot arm because the robot controller would only process ASCII messages. Therefore, by adding two "ASCII Encode" blocks, one configured to send joint angles and the other for cartesian moves, we were capable of converting our strings before they were sent out to the robot arm. And by adding one "ASCII to String" block we manage to convert the ASCII messages from to robot arm into strings so it could be interpretated by our controller.

Then we developed the feedback subsystem incumbent with the task of ensuring that the messages were correctly delivered and read. Within this subsystem we were also capable of receiving the end-effector reached position. From this information a comparison between the received value and the sent one was made and if the values were the same, for

a maximum error of 0.01, we assumed that we had reached the intended position and the next waypoint was sent to the robot arm.

The last validation was focused on the communication protocol blocks. In this test we needed to make sure that the data was successfully exchange between the host computer, the Speedgoat target, and the Kassow KR810, as schematized in Figure 4.5.

Figure 6.5 – Final system setup

6.2.2. Speedgoat Target Machine Results

The results obtained from the Speedgoat target simulations helped us understand in more depth the performance of the controller that led to the correction of some unexpected behaviour and an improvement of the overall efficiency.

Although, even by forcing the simulation pace to real-time, as described in section 6.1 there were no guarantees that this would run at real-time for all the simulation duration. In fact, this can be observed from the signals in Figure 6.6 captured during simulation. In the graphics bellow the blue lines correspond to the signals running inside the Speedgoat hardware and the oranges to the ones on the host computer. A closer observation of the first graphic reveals a significant delay between the signals. One of the possible reasons for this event is the fact that the model running on the host computer is simulated

slower than real-time, maybe due to hardware limitations, which in turn provokes the disharmony.

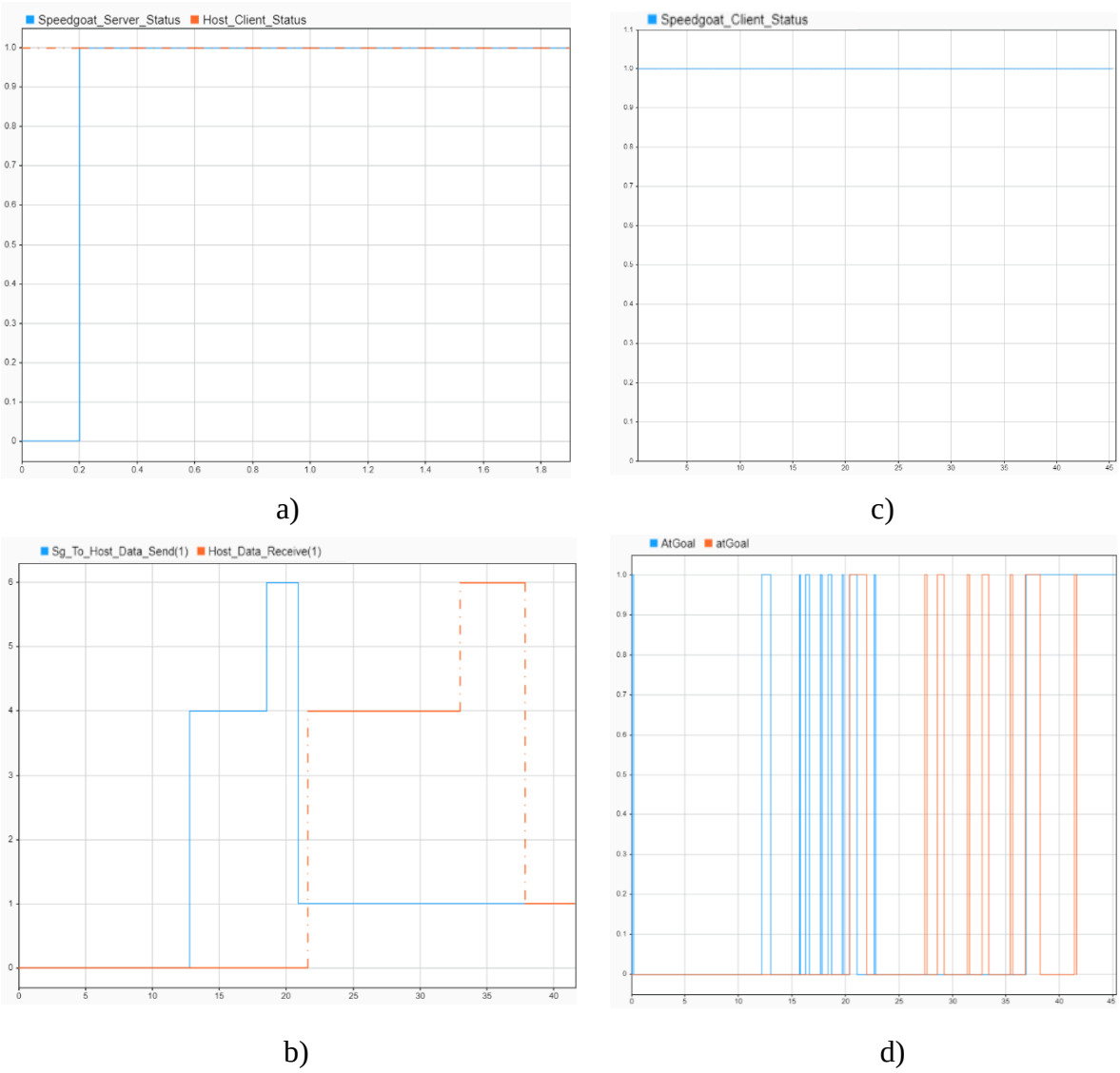


Figure 6.6 – Real-time simulation signals: Connection status between the host and target (a). Desired character signal value (b). Connection status between the target and Kassow (c). AtGoal signal status (d).

The graphic a) represents the TCP/IP signal established between the server (Speedgoat target) and the client (Host computer). As it can be observed from the graphic the blue line (server) initially starts at zero but at $t = 0.2$ seconds it goes to 1 meaning that a connection has been accomplished. Since at the start of the first simulation there is no available client, the server will not be able to find a connection, resulting in a signal with the value zero, but when the second simulation is running a client will be available and the server signal will be set to one. The client signal is always set to one because by default the

Simulink checks for a listening server before starting the execution. If we were working with both models in real-time, we would expect to observer both signals being set to one at the exact same time. This graphic also tells us the exact time that it takes to launch the OCR model.

The graph b) represents the signal related with the numerical values associated with the different characters that are intended to be located during execution. A table for the built enumeration is given in Table F.0-1 Annex F. The stateflow that runs inside the real-time model is responsible for the selection of the desired character. This means that this signal is generated inside the target, and the first signal to change should be the Speedgoat one, as it does. Then, this signal value is sent to the OCR model resulting in the change of this second signal to the value achieved by the blue line.

The graphic c) represents the TCP/IP signal established between the server (Kassow) and the client (Speedgoat). However, we could only obtain the signal for the client since the server was outside of the Simulink range and could not be measured. In the previous paragraph we stated that a client signal should also start at zero but for this case this is not true. Since the TCP/IP program developed for the Kassow was already running inside the Kassow controller box, the server was already in stand-by waiting for a client. The connection represented in c) is established when the ethernet cable from the robot control box is connected to the target port and not when the simulation starts as in the situation in b).

The graphic d) represents the atGoal signal that is generated inside the real-time model and sent back to the OCR model. This signal is responsible for the good functioning of our controller and for the correct delivering of the waypoints. Once again is expected the same behaviour for a signal generated inside the target machine as the one in b). In this case, we can observe the disharmony provoked by the hardware limitations of the host computer.

In sum, this graphics were crucial since they helped us locate signals/parameters, that were lost while sending data packages or that were being associated in the wrong channel, improve the efficiency, and also understand the overall behaviour of the controller.

6.3. Real-Time Setup

In the present subchapter we will described how we have tested the whole system developed in MATLAB and Simulink with the robot arm.

The Kassow KR810 has a dedicated portable electric cabinet with a teach pendant that contains a dedicated application to control the robot. Since the Kassow had no installed communication protocol, a TCP/IP application was built to allow the data exchange for specific messages. This can be seen in Figure 6.7.

Figure 6.7 – Kassow GUI application with the TCP/IP application

The signals and commands were set to call this specific messages. However, to prevent damages from initial tests, we set up a few predefined commands that were associated with specific positions manually selected by us, as can be seen in Figure F.0.1 in annex F. In these preliminary tests, the real-time model would only call these commands and the data was only being exchange between the Speedgoat target and the robot arm controller. After the initial validation, the setup was changed to the one represented in Figure 4.5 which was also the final one.

The Kassow dedicated application had some functionalities like the joint space and workspace values, as illustrated in Figure 6.8, that in combination with the results obtained from the digital twin simulation, helped us validate the obtained results.

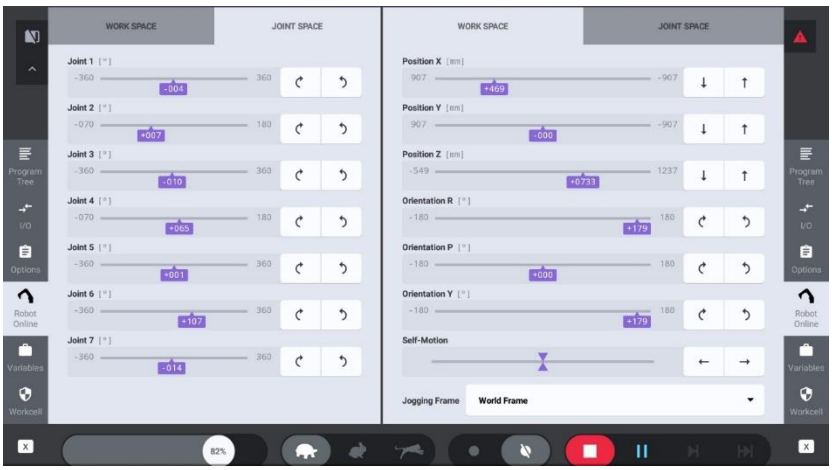


Figure 6.8 – Kassow GUI application with the joint space and workspace information

7. CONCLUSIONS AND FUTURE WORK

The dissertation addresses the automation problem stated for a robot arm in charge of performing touch-screen testing. For that purpose, a two-step strategy was followed with the objective of finding feasible and optimized functionalities suitable for an automated touchscreen testing robot arm.

First, an offline simulation approach is proposed which divides the motion planning problem into three different and successive stages: the geometric path evaluation, the path optimization and the trajectory evaluation. This step also takes focus on the image processing technique and in the supervised control logic. In the end, the real-time simulation model is the outcome of the developed functionalities with some modifications and improvements.

As mentioned, one of the advantages of MBD is the ability of testing the system at every new iteration. In fact, the controller was being tested at each step leading to a continuous verification and improvement.

In this dissertation we only covered the development process for the first and third stage. However, to cover the full development of the system there are more steps in the process some already describe in Chapter 2 and schematize in Figure 7.1.

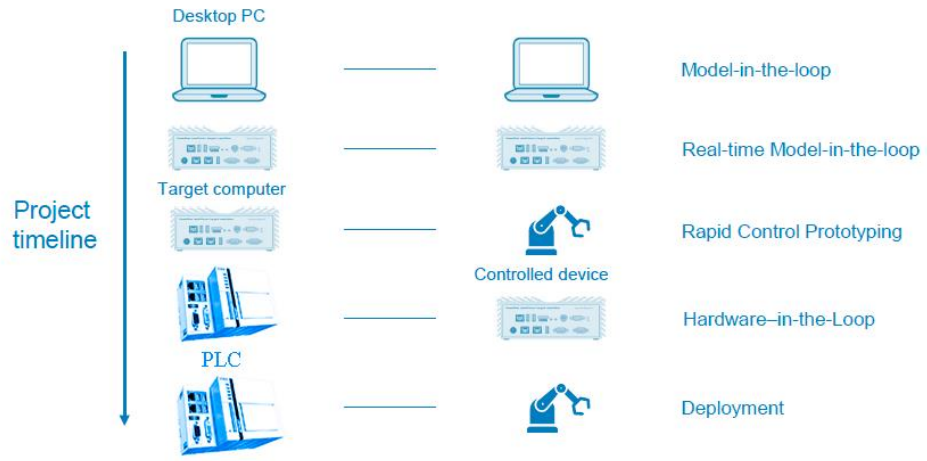


Figure 7.1 – Project timeline diagram.

The first stage concerns the first tests performed in an offline simulation environment, known as MIL. The second stage, that was not carried out in this dissertation, is usually perform between two targets in which one is running the controller under testing and the other the plant or digital twin. The main difference between this and the previous step is the fact that this one runs in real-time. This step is one of the most important ones since it allows the understanding of the controller's capabilities in a real-time environment. For example, for this dissertation, if we had performed this step, we would have encountered the problems sooner and subsequently solve them when moving on to the third stage. This stage is often used to avoid damaging the hardware.

The third stage is the one described in subchapter 6.2 where we performed our rapid control prototyping but with the robot in safe mode conditions to prevent unwanted damage from these initial tests on the physical hardware.

The fourth step in the timeline is a testing stage since the code is now generated and deployed for the first time to the PLC. Once again, the safety of the hardware is the top priority and since many things could go wrong, the PLC is only connected to the target machine that is running the digital twin model used also in step 1 and step 2.

The last step is considered to be a validation stage in which the PLC runs directly with the robot arm. The last two steps can be differentiated based on the performance gains between the simulation results and the real results, since the simulation model is never a truly high-fidelity model due to the time required to tune all the parameters. In our case, for the offline simulation, the electric motors parameters were not taken into account for the path and optimization process. This means that sometimes the obtained digital results might not be the exact same ones as the ones in the real counterpart. Finally, the real controller is now deployed in the final cycle.

The target machine revealed to be a strong investment due to the unforeseen events that happened during the realization of this dissertation research. As stated, in this dissertation, the initial plan was to use the Meca500, an EtherCAT capable robot, implying the use of an EtherCAT capable PLC like, for example, a Beckhoff. However, the desired robot was not delivered, and an alternative solution was provided. This alternative was, the Kassow KR810, a robot with no currently integrated communication protocol, since it was only a prototype version. Nonetheless, the existence of a TCP/IP module capable of exchanging messages previously built by another student for a different application helped

make the transition easier. This is where the Speedgoat real-time target machine really shines. Since this target is compatible with almost every known protocol, in this situation, we were not stuck with only one protocol. The target has also more processing power than a PLC, giving us the flexibility to test and design new ideas that otherwise would not be possible directly from the PLC usage.

For future developments, the author recommends or foresees the necessity of including safety systems to ensure the safety of the device under testing by adding a pressure sensor to the base of the device holder to guarantee a maximum value pressure on the screen.

Now considering the calibration process of the physical system, a further step involves the precision and accuracy of the accomplished results. As underlined in the dissertation, these experiments will accept or refused the behaviour of the developed controller. For instance, the calibration made for the performed tests was not very rigorous. In fact, during each execution the device was susceptible to vibrations provoked not only by the robot arm movement but also from external sources that might appeared. The robot was fixed to an AGV and the device was fitted on the top of a wooden box with some 3D printed adapters, as illustrated in Figure 6.5. However, this box was not fixed to the AGV. The centre of the box was also not perfectly aligned with the centre of the robot arm, which could lead to faulty results. So, a better testing environment with a more rigorous calibration is advisable.

Regarding the fact that the OCR model was not able to run on the target machine implies two possible solutions. The first one is related to how much time could be spent doing research and development for a new technique that is compatible with the provided target. In the second one, the final user can opt to use a more powerful embedded board capable of processing neural networks or even add a dedicated external graphic card to the selected PLC.

In the overall functionalities, it would be interesting to see implemented three new functions. The first, should be a function capable of detecting if the screen device is looked or unlocked, to prevent the robot from executing unnecessary tests. The second, should be an option for selecting the type of device under testing since the analysed image from which the end-effector pose is being determined depends on the screen device properties. The last one, could be a dynamic tunning feature capable of optimizing the

consumption of energy associated with the robot arm trajectory, that might lead to a better positioning of the robot in relation to the device under testing.

On a final remark it would be also interesting to see implemented a second connection between the Speedgoat target and an offline simulation running the digital twin. From this we would be able to achieve a synchronization between the digital system and the physical one making the most use of our digital twin technology. To obtain this it would also be necessary to change the code in the program installed in the Kassow, the one represented in Figure 6.7, since at this stage only sends a message with the final attained coordinates and not with the coordinates at every instance.

BIBLIOGRAPHY

American Heritage Dictionary - Search. (n.d.). Retrieved November 14, 2020, from https://ahdictionary.com/

Ballard, D. H., & Brown, C. M. (1982). *Computer Vision*. Prentice-Hall, Inc. http://homepages.inf.ed.ac.uk/rbf/BOOKS/BANDB/Ballard__D._and_Brown__C._M.__1982__Computer_Vision.pdf

Bélanger, J., Venne, P., & Paquin, J. (2010). *The What , Where and Why of Real-Time Simulation*. 37–49.

Bhatt, D., Hall, B., Dajani-Brown, S., Hickman, S., & Paulitsch, M. (2005). Model-Based Development and the Implications to Design Assurance and Certification. *24th Digital Avionics Systems Conference*, *2*, 10.D.3-1-10.D.3-13. https://doi.org/10.1109/DASC.2005.1563401

Bonev, I. (2019). *What are singularities in a six-axis industrial robot arm?* Mecademic. https://www.mecademic.com/en/what-are-singularities-in-a-six-axis-robot-arm

Bragança, S., Costa, E., Castellucci, I., & Arezes, P. M. (2019). A brief overview of the use of collaborative robots in industry 4.0: Human role and safety. *Studies in Systems, Decision and Control*, *202*, 641–650. https://doi.org/10.1007/978-3-030-14730-3_68

Coors-Blankenship, J. (2020). *Taking Digital Twins for a Test Drive with Tesla , Apple*. https://www.industryweek.com/technology-and-iiot/article/21130033/how-digital-twins-are-raising-the-stakes-on-product-development

Corke, P. I. (1996). Dynamic Issues in Robot Visual-Servo Systems. In *Robotics Research* (pp. 488–498). Springer London. https://doi.org/10.1007/978-1-4471-1021-7_54

De Luca, A. (n.d.). *Robotics 2 Visual servoing*. Retrieved November 3, 2020, from http://www.diag.uniroma1.it/deluca/rob2_en/17_VisualServoing.pdf

Dommel, H. W. (1969). Digital Computer Solution of Electromagnetic Transients in Single- and Multiphase Networks. *IEEE Transactions on Power Apparatus and Systems*, *PAS-88*(4), 388–399. https://doi.org/10.1109/TPAS.1969.292459

EtherCAT. (2021). *EtherCAT Technology Group*. https://www.ethercat.org/en/technology.html

Flash, T., Stm, T., & Id, D. (2012). *Programming manual*. *500*(March), 1–50.

Gillies, D. (2009). Internet FAQ Archives. *[Online]*. http://www.faqs.org/faqs/realtime-computing/faq/

Global Industry Analystic, I. (2020). *Touch Screen Displays - Global Market Trajectory & Analytics*.

https://www.researchandmarkets.com/reports/4806452/touch-screen-displays-global-market-trajectory

Grieves, M., & Vickers, J. (2016). Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches* (pp. 85–113). Springer International Publishing. https://doi.org/10.1007/978-3-319-38756-7_4

Haag, S., & Anderl, R. (2018). Digital twin – Proof of concept. *Manufacturing Letters*, *15*, 64–66. https://doi.org/10.1016/j.mfglet.2018.02.006

Hill, J., & Park, W. T. (1979). Real Time Control of a Robot With a Mobile Camera. *Water Resources and Environmental Engineering Research Report (State University of New York at Buffalo, Department of Civil Engineering), Proc. 9th*, 233–246. https://www.researchgate.net/publication/243778292_Real_time_control_of_a_robot_with_a_mobile_camera?el=1_x_8&enrichId=rgreq-b8d6e63c448d1e195ff6d8ddc3a1cd56-XXX&enrichSource=Y292ZXJQYWdlOzIyNDU2MjI0OTtBUzoyMTUyNTQ2NzI5MDgyODhAMTQyODMzMjEyMTIxNQ==

Hutchinson, S., Hager, G. D., & Corke, P. I. (1996). A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, *12*(5), 651–670. https://doi.org/10.1109/70.538972

Kagermann, H., Lukas, W.-D., & Wahlster, W. (2011). Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution. *VDI Nachrichten*, *13*, 3–4. https://www.ingenieur.de/technik/fachbereiche/produktion/industrie-40-mit-internet-dinge-weg-4-industriellen-revolution/

Lenz, R. K. (1989). A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/Eye Calibration. *IEEE Transactions on Robotics and Automation*, *5*(3), 345–358. https://doi.org/10.1109/70.34770

Mathworks. (2016). MATLAB - Mathworks - MATLAB & Simulink. In *Www.Mathworks.Com* (p. 1). https://www.mathworks.com/products/matlab.html

Mathworks. (2020). *What Is Simulink? - Video - MATLAB & Simulink*. https://www.mathworks.com/videos/simulink-overview-61216.html

Mathworks. (2021a). *Computer Vision Toolbox - MATLAB & Simulink*. https://www.mathworks.com/products/computer-vision.html#feature-detection

Mathworks. (2021b). *Inverse Kinematics Algorithms - MATLAB & Simulink*. https://www.mathworks.com/help/robotics/ug/inverse-kinematics-algorithms.html

Mathworks. (2021c). *Simscape Multibody - MATLAB & Simulink*. https://www.mathworks.com/products/simscape-multibody.html#imcad

Mathworks. (2021d). *Simulink Real-Time - MATLAB & Simulink*. https://www.mathworks.com/products/simulink-real-time.html#build

Mathworks. (2021e). *Spherical linear interpolation - MATLAB slerp*. https://www.mathworks.com/help/fusion/ref/quaternion.slerp.html

Mathworks. (2021f). *Stateflow - MATLAB & Simulink*. https://www.mathworks.com/products/stateflow.html

MathWorks. (2020). *Robotics System Toolbox - MATLAB & Simulink*. https://www.mathworks.com/products/robotics.html

Maxon motor ag. (2021). *Small and ready to conquer the world*. https://www.maxongroup.com/maxon/view/application/Micro-industrial-robot

Mecademic Inc. (2018). *User Manual Meca500 (R3). F*. https://www.mecademic.com/Documentation/Meca500-R3-User-Manual.pdf

Oracle, & Workplace Intelligence LLC. (2020). *As Uncertainty Remains, Anxiety and Stress Reach a Tipping Point at Work*. https://www.oracle.com/a/ocom/docs/oracle-hcm-ai-at-work.pdf

Paul, P. R. (1981). *Robot Manipulators: Mathematics, Programming, and Control*. The MIT Press. https://books.google.pt/books?hl=pt-PT&lr=&id=UzZ3LAYqvRkC&oi=fnd&pg=PP9&dq=Paul,+R.+P.+Robot+Manipulators:+Mathematics,+Programming,+and+Control.+The+MIT+Press,+Cambridge,+MA,+1981.&ots=zXopSCu79T&sig=V_9JFC4LGuCWJT4bAofr3KfEoEo&redir_esc=y#v=onepage&q&f

Pieper, D. L. (1968). The Kinematics of Manipulators Under Computer Control. In *Artificial Intelligence Laboratory Memo AIM-72*.

Pires, J. N. (2007). Industrial robots programming: Building applications for the factories of the future. In *Industrial Robots Programming: Building Applications for the Factories of the Future*. Springer US. https://doi.org/10.1007/b101252

Robots, K. (n.d.-a). *About Us | Kassow Robots*. Retrieved April 18, 2021, from https://www.kassowrobots.com/about-us/

Robots, K. (n.d.-b). *KR810 Cobot - Kassow Robots*. Retrieved August 13, 2021, from https://www.kassowrobots.com/products/kr810/

Sanchez-Gasca, J. J., D'Aquila, R., Price, W. W., & Paserba, J. J. (1995). Variable time step, implicit integration for extended-term power system dynamic simulation. *IEEE Power Industry Computer Applications Conference*, 183–189. https://doi.org/10.1109/pica.1995.515182

Sanchez, R. (2015, November 17). *What is TCP/IP and How Does It Make the Internet Work? - HostingAdvice.com*. 2015-11-17. https://www.hostingadvice.com/blog/tcpip-make-internet-work/

Sanderson, A. C., & Weiss, L. E. (1983). Adaptive Visual Servo Control of Robots. *Robot Vision*, 107–116. https://doi.org/10.1007/978-3-662-09771-7_7

Schleich, B., Anwer, N., Mathieu, L., & Wartzack, S. (2017). Shaping the digital twin for design and production engineering. *CIRP Annals - Manufacturing Technology*, *66*(1), 141–144. https://doi.org/10.1016/j.cirp.2017.04.040

Shane McLaughlin. (2020). *Horizontal And Vertical Integration In Industry 4.0 For Pharmaceutical And Medical Device Manufacturers*. https://slcontrols.com/horizontal-and-vertical-integration-in-industry-4-0-for-pharmaceutical-and-medical-device-manufacturers/

Söderberg, R., Wärmefjord, K., Carlson, J. S., & Lindkvist, L. (2017). Toward a Digital Twin for real-time geometry assurance in individualized production. *CIRP Annals - Manufacturing Technology*, *66*(1), 137–140. https://doi.org/10.1016/j.cirp.2017.04.038

Speedgoat. (2021). *Simulink real-time-target machines | Speedgoat*. https://www.speedgoat.com/products-services/real-time-target-machines

Systèmes, D. (2021). *History | About 3DS - Dassault Systèmes*. https://www.3ds.com/about-3ds/what-we-are/history

Szeliski, R. (2011). *Computer Vision*. Springer London. https://doi.org/10.1007/978-1-84882-935-0

Technology, L. (2012). *HOW TOUCH SCREEN MONITORS WORK*. 0–8. https://www.lorextechnology.com/self-serve/how-touch-screen-monitors-work/R-sc3100030

Touch Screen Display Market Size - Industry Growth Report 2026. (n.d.). Retrieved October 29, 2020, from https://www.gminsights.com/industry-analysis/touch-screen-display-market

Vaidya, S., Ambad, P., & Bhosle, S. (2018). Industry 4.0 - A Glimpse. *Procedia Manufacturing*, *20*, 233–238. https://doi.org/10.1016/j.promfg.2018.02.034

Valera, A., Vallés, M., & Tornero, J. (2001). Real-Time Robot Control Implementation with Matlab/Simulink. *IFAC Proceedings Volumes*, *34*(9), 527–532. https://doi.org/10.1016/s1474-6670(17)41762-9

Walker, G. (2014). *Fundamentals of Projected-Capacitive Touch Technology*. Technical Presentation. www.walkermobile.com/Touch_Technologies_Tutorial_Latest_Version.pdf

Wang, J., Li, Y., & Zhao, X. (2010). Inverse Kinematics and Control of a 7-DOF Redundant Manipulator Based on the Closed-Loop Algorithm. In *International Journal of Advanced Robotic Systems* (Vol. 7, Issue 4).

Welman, C. (1993). *Inverse Kinematics and geometric constraints for articulated figure manipulation*.

Yu, W. (2018). Preliminaries. In *PID Control with Intelligent Compensation for Exoskeleton Robots* (pp. 1–12). Elsevier. https://doi.org/10.1016/B978-0-12-813380-4.00001-3

Kagermann, H., Dieter Lukas, W., & Wahlster, W. (2011).” *Industry 4. 0: With the Internet of Things on the way to the 4th industrial revolution”*. 3–4.

(*Touch Screen Display Market Size - Industry Growth Report 2026*, n.d.)

ANNEX A

```
    properties (Access = private)
        myTimer=timer; %Timer object
    end
    methods (Access = public)
        function CheckStatusFcn(app)
            % Export real time data from Simulink
            block='Kassow_Final/df'; %'modelname/Block Name'
            rto=get_param(block, 'RuntimeObject');
            data = rto.OutputPort(1).Data;
            rto2=get_param('Kassow_Final/Gain', 'RuntimeObject');
            data2 = rto2.OutputPort(1).Data;
            rto2=get_param('Kassow_Final/LogicController/CV_App', 'RuntimeObject');
            data3 = rto2.OutputPort(1).Data;
            if data==0
                app.RunningLamp.Color='r'; %Change to red
                app.HomeKassowKR810Button.Enable='off'; %Enable Home Button
                app.TestDeviceButton.Enable='off'; %Disable Test Button
            elseif data==1
                app.RunningLamp.Color='y'; %Change to yellow
                app.HomeKassowKR810Button.Enable='off'; %Disable Home Button
                app.TestDeviceButton.Enable='on';  %Enable Test Button
            elseif data==2
                app.RunningLamp.Color='g'; %Change to green
                app.HomeKassowKR810Button.Enable='on'; %Enable Home Button
                app.TestDeviceButton.Enable='off'; %Disable Test Button
            end
            %-------------------------------------------------
            % status = 0 -Unknown | status = 1 -Home screen | status = 2 -Dial App
            % | status=3 -Dial App but not in dial mode
            % | status = 4 -Touch Mode | status = -1 -Error/Checking Screen
            %-------------------------------------------------
            if data2==0
                app.SearchingLabel.Text= 'Unknown';
            elseif data2==1
                app.SearchingLabel.Text= 'Home Screen';
            elseif data2==2
                app.SearchingLabel.Text= 'Ready/Pressing Dial';
            elseif data2==3
                app.SearchingLabel.Text= 'Going to Phone Dial';
            elseif data2==4
                app.SearchingLabel.Text= 'Test Mode';
            elseif data2==-1 && data3==1
                app.SearchingLabel.Text= 'Executing Test';
            elseif data2==-1 && data3==0
                app.SearchingLabel.Text= 'Error/Switching to HomeScreen';
            end
        end
    end
```

```
        % Code that executes after component creation
        function startupFcn(app)
            set_param('Kassow_Final/Stop', 'Value', '0'); %reset STOP value
            % Set timer object
            app.myTimer.Period = 0.1;
            app.myTimer.StartDelay = 2;
            app.myTimer.ExecutionMode = 'fixedDelay';
            %--
            app.myTimer.TimerFcn = @(~,~) CheckStatusFcn(app); %create timer object
            start(app.myTimer) %starts timer
        end

        % Close request function: UIFigure
        function UIFigureCloseRequest(app, event)
            set_param('Kassow_Final/MODE','Value','0');    % RESET HOME
            set_param('Kassow_Final/START','Value','0');     % RESET START
            set_param('Kassow_Final/Stop', 'Value', '1');    % Activate STOP
            pause(0.1); %it's crucial to send this delay so we can assure the value has been reset
            set_param('Kassow_Final/Stop', 'Value', '0');    % RESET STOP
            stop(app.myTimer);
            delete(app)
        end

        % Button pushed function: TestDeviceButton
        function TestDeviceButtonPushed2(app, event)
                    %SET/RESET PARAMETERS
            set_param('Kassow_Final/START','Value','1');
            set_param('Kassow_Final/MODE','Value','1');
        end

        % Button pushed function: HomeKassowKR810Button
        function HomeKassowKR810ButtonPushed(app, event)
                      %SET/RESET PARAMETERS
           set_param('Kassow_Final/START','Value','0');
           set_param('Kassow_Final/MODE','Value','0');
        end
```

Figure A.0.1– Representation of the programming code used in the GUI responsible for controlling the offline simulation Simulink model.

ANNEX B

```
properties (Access = private)
  tg      % Target object
  modelName = 'Kassow_RT';    %Controlador_logico_teste_v3
  %Create timer object
  mytimer                   % Timer object variable
  timerStep = 0.01;          % Timer steps in [s]
end

methods (Access = private)
    function closeApp(app)
        answer = questdlg('Are you sure you wish to exit?', 'Exit choice', 'Yes','No','No');
        switch answer
            case 'Yes'
                if app.tg.isConnected && app.tg.isRunning
                    app.tg.stop;
                end
                bdclose(app.modelName);
                app.tg.disconnect;
                delete(app);
            case 'No'
        end
    end
    function CheckStatusFcn(app,~,~)
        % Export real time data from Simulink
        data = app.tg.getsignal('Modes_Enable');
        data2 = app.tg.getsignal('Screen');
        data3 = app.tg.getsignal('CV_App');
        if data==0
            app.RunningStatusLamp.Color      = [0.94,0.94,0.94]; %Change to red
            app.HomeKassowKR810Button.Enable = 'off'; %Disable Home Button
            app.TestDeviceButton.Enable      = 'off'; %Disable Test Button
        elseif data==1
            app.RunningStatusLamp.Color      = 'r'; %Change to red
            app.HomeKassowKR810Button.Enable = 'off'; %Disable Home Button
            app.TestDeviceButton.Enable      = 'off'; %Disable Test Button
        elseif data==2
            app.RunningStatusLamp.Color      = 'y'; %Change to yellow
            app.HomeKassowKR810Button.Enable = 'on'; %Enable Home Button
            app.TestDeviceButton.Enable      = 'on';  %Enable Test Button
        elseif data==3
            app.RunningStatusLamp.Color      = 'g'; %Change to green
            app.HomeKassowKR810Button.Enable = 'on'; %Enable Home Button
            app.TestDeviceButton.Enable      = 'off'; %Disable Test Button
        end
```

```
            %-----------------------------------------------
            % status = 0 -Unknown | status = 1 -Home screen | status = 2 -Dial App |
            % status=3 -Dial App but not in keypad |
            % status = 4 -Test Modules | status = -1 -Error/Checking Screen  |
            %-----------------------------------------------
            if data2==0
                app.SearchingLabel.Text= 'Unknown';
            elseif data2==1
                app.SearchingLabel.Text= 'Home Screen';
            elseif data2==2
                app.SearchingLabel.Text= 'Ready/Pressing Dial';
            elseif data2==3
                app.SearchingLabel.Text= 'Going to Phone Dial';
            elseif data2==4
                app.SearchingLabel.Text= 'Test Mode';
            elseif data2==-1 && data3==1
                app.SearchingLabel.Text= 'Executing Test';
            elseif data2==5
                app.SearchingLabel.Text= 'PowerPoint';
            elseif data2==-1 && data3==0
                app.SearchingLabel.Text= 'Error/Switching to HomeScreen';
            end
        end
    end
```

```
        % Code that executes after component creation
        function startupFcn(app)
            app.TargetStatusEditField.Value = 'Starting app...';

            % Generate a timer object to update the UI every second
            app.mytimer = timer;                          % Create timer object
            app.mytimer.Period = app.timerStep;           % Time step
            app.mytimer.TimerFcn = @app.CheckStatusFcn;    % Periodic Function
            app.mytimer.ExecutionMode = 'FixedSpacing';
            app.mytimer.BusyMode = 'queue';

            % Load model in memory
            load_system('Kassow_RT.slx');

            % Enable/disable buttons
            app.ConnecttotargetButton.Enable    = 'on';
            app.BuildButton.Enable              = 'off';
            app.LoadModelButton.Enable          = 'off';
            app.Switch.Enable                   = 'off';
            app.TestDeviceButton.Enable         = 'off';
            app.HomeKassowKR810Button.Enable    = 'off';

        end
```

```
        % Button pushed function: ConnecttotargetButton
        function ConnecttotargetButtonPushed(app, event)
            app.TargetStatusEditField.Value='Connecting to target...';

            % Enable/disable buttons
            app.ConnecttotargetButton.Enable    = 'off';
            app.BuildButton.Enable              = 'off';
            app.LoadModelButton.Enable          = 'off';
            app.Switch.Enable                   = 'off';
            app.TestDeviceButton.Enable         = 'off';
            app.HomeKassowKR810Button.Enable    = 'off';

            app.tg = slrt;     % Initialize target object

            if strcmp(slrtpingtarget,'success')
                app.TargetStatusEditField.Value = 'Host connected to target!';
                app.LampTarget.Color='g';
```

```
                % Enable/disable buttons
                app.ConnecttotargetButton.Enable    = 'off';
                app.BuildButton.Enable              = 'on';
                app.LoadModelButton.Enable          = 'off';
                app.Switch.Enable                   = 'on';
                app.TestDeviceButton.Enable         = 'off';
                app.HomeKassowKR810Button.Enable    = 'off';

            elseif strcmp(app.tg.ping('reset'),'success')
                app.TargetStatusEditField.Value = 'Connection reset, host connected to target!';
                app.LampTarget.Color='g';

                % Enable/disable buttons
                app.ConnecttotargetButton.Enable    = 'off';
                app.BuildButton.Enable              = 'on';
                app.LoadModelButton.Enable          = 'off';
                app.Switch.Enable                   = 'off';
                app.TestDeviceButton.Enable         = 'off';
                app.HomeKassowKR810Button.Enable    = 'off';
            else
                app.TargetStatusEditField.Value = 'Error: Failed to connect to target!';
                app.LampTarget.Color='r';

              % Enable/disable buttons
              app.ConnecttotargetButton.Enable    = 'on';
              app.BuildButton.Enable              = 'off';
              app.LoadModelButton.Enable          = 'off';
              app.Switch.Enable                   = 'off';
              app.TestDeviceButton.Enable         = 'off';
              app.HomeKassowKR810Button.Enable    = 'off';
          end

      end
       % Button pushed function: BuildButton
       function BuildButtonPushed(app, event)
           app.TargetStatusEditField.Value = 'Building model...';
           app.LampTarget.Color='0.94,0.94,0.94';                    %Set colour to white
           set_param(app.modelName,'xPCisDownloadable','off');       % Temporarily set model parameters

           % Enable/disable buttons
           app.ConnecttotargetButton.Enable    = 'off';
           app.BuildButton.Enable              = 'off';
           app.LoadModelButton.Enable          = 'off';
           app.Switch.Enable                   = 'off';
           app.TestDeviceButton.Enable         = 'off';
           app.HomeKassowKR810Button.Enable    = 'off';

           % Build model
           try
               rtwbuild(app.modelName,'OpenBuildStatusAutomatically','true');
               app.TargetStatusEditField.Value = 'Real-time application built, ready to load!';
               app.LampTarget.Color='g';

               % Enable/disable buttons
               app.ConnecttotargetButton.Enable    = 'off';
               app.BuildButton.Enable              = 'off';
               app.LoadModelButton.Enable          = 'on';
               app.Switch.Enable                   = 'off';
               app.TestDeviceButton.Enable         = 'off';
               app.HomeKassowKR810Button.Enable    = 'off';
           catch
               app.TargetStatusEditField.Value = 'Error: Failed building model (see build logs)';
               app.LampTarget.Color='r';

               % Enable/disable buttons
               app.ConnecttotargetButton.Enable    = 'off';
               app.BuildButton.Enable              = 'on';
               app.LoadModelButton.Enable          = 'off';
               app.Switch.Enable                   = 'off';
               app.TestDeviceButton.Enable         = 'off';
               app.HomeKassowKR810Button.Enable    = 'off';
```

```
            end
        end
        % Button pushed function: LoadModelButton
        function LoadModelButtonPushed(app, event)
            app.TargetStatusEditField.Value = 'Loading application...';
            app.LampTarget.Color=[0.94,0.94,0.94];         %Set colour to white
            % Enable/disable buttons
            app.ConnecttotargetButton.Enable    = 'off';
            app.BuildButton.Enable              = 'off';
            app.LoadModelButton.Enable          = 'off';
            app.Switch.Enable                   = 'off';
            app.TestDeviceButton.Enable         = 'off';
            app.HomeKassowKR810Button.Enable    = 'off';
            try
                app.tg.ping('reset');
                load(app.tg, app.modelName);
                % Enable/disable buttons
            app.ConnecttotargetButton.Enable    = 'off';
            app.BuildButton.Enable              = 'off';
            app.LoadModelButton.Enable          = 'off';
            app.Switch.Enable                   = 'on';
            app.TestDeviceButton.Enable         = 'off';
            app.HomeKassowKR810Button.Enable    = 'off';
                % Update status message
                app.TargetStatusEditField.Value = 'Application loaded to target!';
                %Update Lamp
                app.LampTarget.Color='g';
            catch
                % Enable/disable buttons
            app.ConnecttotargetButton.Enable    = 'off';
            app.BuildButton.Enable              = 'off';
            app.LoadModelButton.Enable          = 'on';
            app.Switch.Enable                   = 'off';
            app.TestDeviceButton.Enable         = 'off';
            app.HomeKassowKR810Button.Enable    = 'off';
                % Update status message
                app.TargetStatusEditField.Value = 'Error: Application could not be loaded!';
                %Update Lamp
                app.LampTarget.Color='r';
            end
        end
        % Value changed function: Switch
        function SwitchValueChanged(app, event)
            value = app.Switch.Value;

            switch value
                case 'Start'
                    start(app.tg);        % Start real-time application
                    start(app.mytimer);   % Start timer
                    % Enable/disable buttons
                    app.ConnecttotargetButton.Enable    = 'off';
                    app.BuildButton.Enable              = 'off';
                    app.LoadModelButton.Enable          = 'off';
                    app.Switch.Enable                   = 'on';
                    app.TestDeviceButton.Enable         = 'on';
                    app.HomeKassowKR810Button.Enable    = 'off';
                    %Update Target Status
                    app.TargetStatusEditField.Value = 'Application running!';
                    %Set blocks to zero
                    app.tg.setparam('START','Value',0);
                    app.tg.setparam('MODE','Value', 0);
                    %Start sim model that runs in background on host machine
                    set_param('OCR_KassGoatWorks','SimulationCommand','Start');
```

```
                case 'Stop'
                    %Stop sim model on host machine
                    set_param('OCR_model','SimulationCommand','Stop');

                    app.RunningStatusLamp.Color ='0.94,0.94,0.94';  %Set lamp colour to white
                    stop(app.tg);               % Stop real-time application
                    stop(app.mytimer);          % Stop timer
                    % Enable/disable buttons
                    app.ConnecttotargetButton.Enable    = 'off';
                    app.BuildButton.Enable              = 'off';
                    app.LoadModelButton.Enable          = 'off';
                    app.Switch.Enable                   = 'on';
                    app.TestDeviceButton.Enable         = 'off';
                    app.HomeKassowKR810Button.Enable    = 'off';
                    %Update Target Status
                    app.TargetStatusEditField.Value = 'Application stopped!';
                    %Set blocks to zero
                    app.tg.setparam('START','Value',0);
                    app.tg.setparam('MODE','Value', 0);
            end
        end

        % Close request function: KassGoatWorksUIFigure
        function KassGoatWorksUIFigureCloseRequest(app, event)
            selection = uiconfirm(app.KassGoatWorksUIFigure,'Are you sure?','Please confirm exit','Options',{'Exit','Cancel'});

            switch selection
                case 'Exit'
                    progressBar         = uiprogressdlg(app.KassGoatWorksUIFigure, 'Title','Please wait','Indeterminate','on');
                    progressBar.Message = {'Application shut down in progress'};
                    try
                        stop(app.tg);               % Stop real-time application
                        stop(app.mytimer);          % Stop timer
                        close(progressBar)
                    catch
                        close(progressBar)
                        uialert(app.KassGoatWorksUIFigure, {'Could not turn off the target automatically.', ...
                            'Please stop the real-time application manually.'}, ...
                            'Error','Icon','Error','CloseFcn','uiresume(gcbf)')
                        uiwait(gcbf)
                    end
                    delete(app)
                otherwise
                    uialert(app.KassGoatWorksUIFigure, 'Shut down aborted','Info','Icon','Info','CloseFcn','uiresume(gcbf)')
                    uiwait(gcbf)
            end
        end

        % Button pushed function: TestDeviceButton
        function TestDeviceButtonPushed(app, event)
             app.tg.setparam('START','Value', 1);
             app.tg.setparam('MODE','Value', 1);
        end

        % Button pushed function: HomeKassowKR810Button
        function HomeKassowKR810ButtonPushed(app, event)
            app.tg.setparam('START','Value',0);
            app.tg.setparam('MODE','Value', 0);
        end
    end
```

Figure B.0.1 – Representation of the programming code used in the GUI responsible for controlling the real-time Simulink model**.**

ANNEX C

Link	θ_i (º)	α_{i-1} (º)	a_{i-1}(mm)	d_i(mm)
1	0	0	0	308.70
2	180	-90	50	85.70
3	180	-90	50	422.20
4	180	-90	40	84.05
5	180	-90	40	377.50
6	0	-90	0	111.35
7	0	90	0	66.35

Table C.0-1 – Table with the determined Kassow KR810 DH parameters

$$r_{11} = s6(c4s1 - s4(c1c2c3 - c1s2s3)) - c6(s5(c1c2s3 + c1c3s2) - c5(s1s4 + c4(c1c2c3 - c1s2s3)))$$

$$r_{12} = s6(s5(c1c2s3 + c1c3s2) - c5(s1s4 + c4(c1c2c3 - c1s2s3))) + c6(c4s1 - s4(c1c2c3 - c1s2s3))$$

$$r_{13} = -c5(c1c2s3 + c1c3s2) - s5(s1s4 + c4(c1c2c3 - c1s2s3))$$

$$p_x^0 = a4(c1c2c3 - c1s2s3) - d6(c5(c1c2s3 + c1c3s2) + s5(s1s4 + c4(c1c2c3 - c1s2s3))) - d4(c1c2s3 + c1c3s2) + a3c1c2$$

$$r_{21} = -c6(s5(c2s1s3 + c3s1s2) + c5(c1s4 - c4(c2c3s1 - s1s2s3))) - s6(c1c4 + s4(c2c3s1 - s1s2s3))$$

$$r_{22} = s6(s5(c2s1s3 + c3s1s2) + c5(c1s4 - c4(c2c3s1 - s1s2s3))) - c6(c1c4 + s4(c2c3s1 - s1s2s3))$$

$$r_{23} = s5(c1s4 - c4(c2c3s1 - s1s2s3)) - c5(c2s1s3 + c3s1s2)$$

$$p_y^0 = a4(c2c3s1 - s1s2s3) - d6(c5(c2s1s3 + c3s1s2) - s5(c1s4 - c4(c2c3s1 - s1s2s3))) - d4(c2s1s3 + c3s1s2) + a3c2s1$$

$$r_{31} = s4s6(cs3 + c3s2) - c6(s5(c2c3 - s2s3) + c4c5(c2s3 + c3s2))$$

$$r_{32} = s6(s5(c2c3 - s2s3) + c4c5(c2s3 + c3s2)) + c6s4(c2s3 + c3s2)$$

$$r_{33} = c4s5(c2s3 + c3s2) - c5(c2c3 - s2s3)$$

$$p_z^0 = d1 - a3s2 - a4(c2s3 + c3s2) - d4(c2c3 - s2s3) - d6(c5(c2c3 - s2s3) - c4s5(c2s3 + c3s2))$$

ANNEX D

```
>> getPCIInfo(tg,'ethernet')

List of installed PCI devices:

Intel                    I210_COPPER
     Bus 5, Slot 0, Function 0, IRQ 5
     Ethernet controller
     VendorID 0x8086, DeviceID 0x1533, SubVendorID 0x15bd, SubDeviceID 0x100a
     Released in: R2015a
     Notes: Intel Ethernet Controller I210 series

Intel                    I210_COPPER
     Bus 14, Slot 0, Function 0, IRQ 16
     Ethernet controller
     VendorID 0x8086, DeviceID 0x1533, SubVendorID 0x15bd, SubDeviceID 0x100a
     Released in: R2015a
     Notes: Intel Ethernet Controller I210 series

Intel                    I210_COPPER
     Bus 15, Slot 0, Function 0, IRQ 5
     Ethernet controller
     VendorID 0x8086, DeviceID 0x1533, SubVendorID 0x15bd, SubDeviceID 0x100a
     Released in: R2015a
     Notes: Intel Ethernet Controller I210 series
```

Figure D.0.1 – Speedgoat target machine Ethernet ports configuration

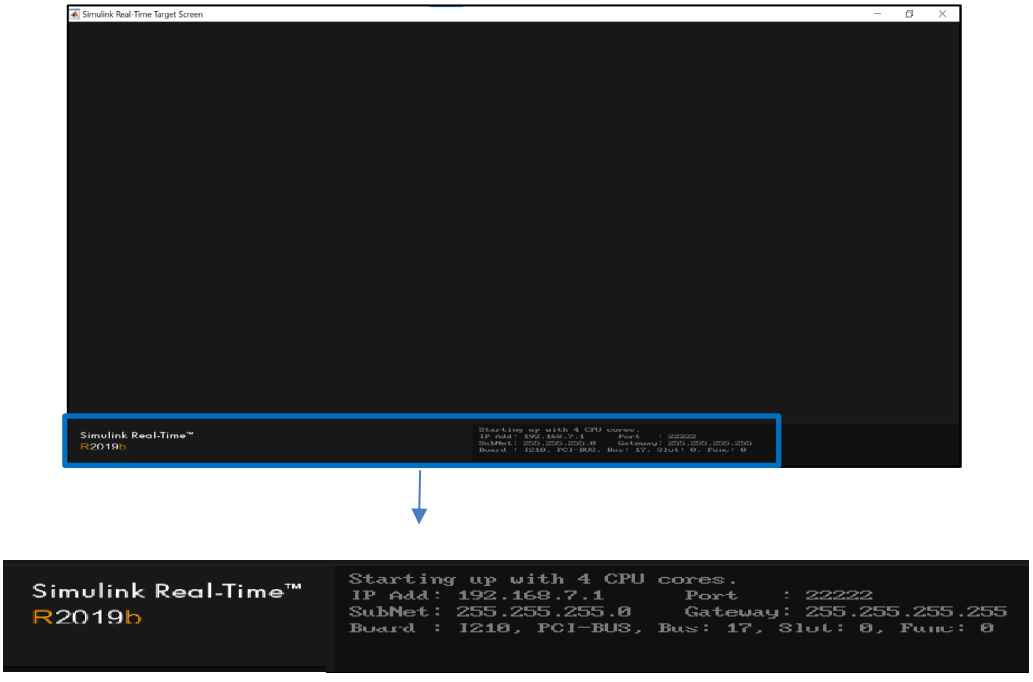


Figure D.0.2 – Speedgoat target machine screen with target information

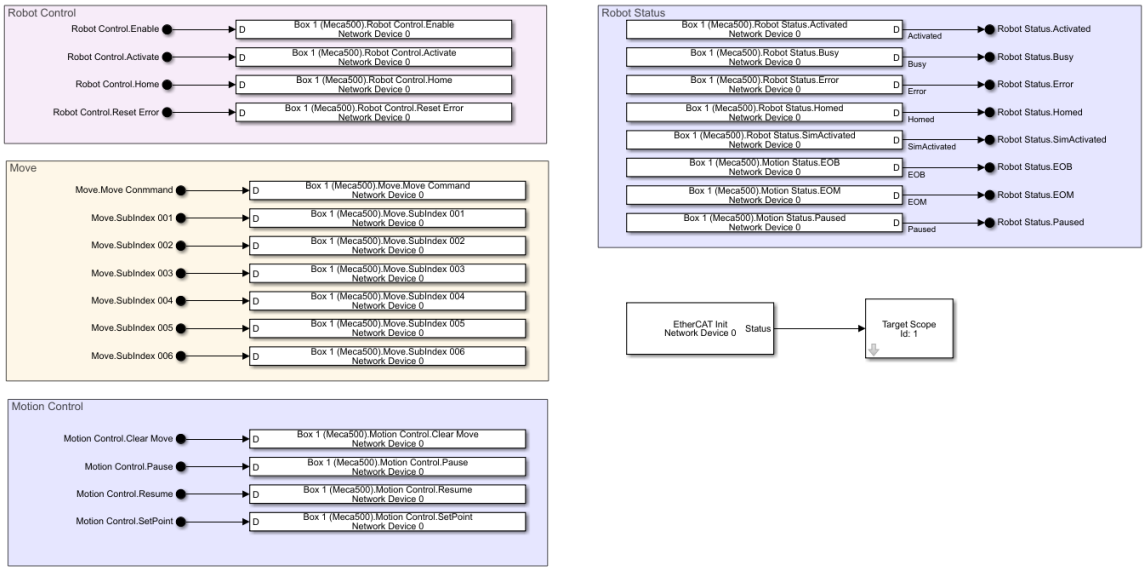


Figure D.0.3 – Meca500 Simulink model EtherCAT blocks configuration

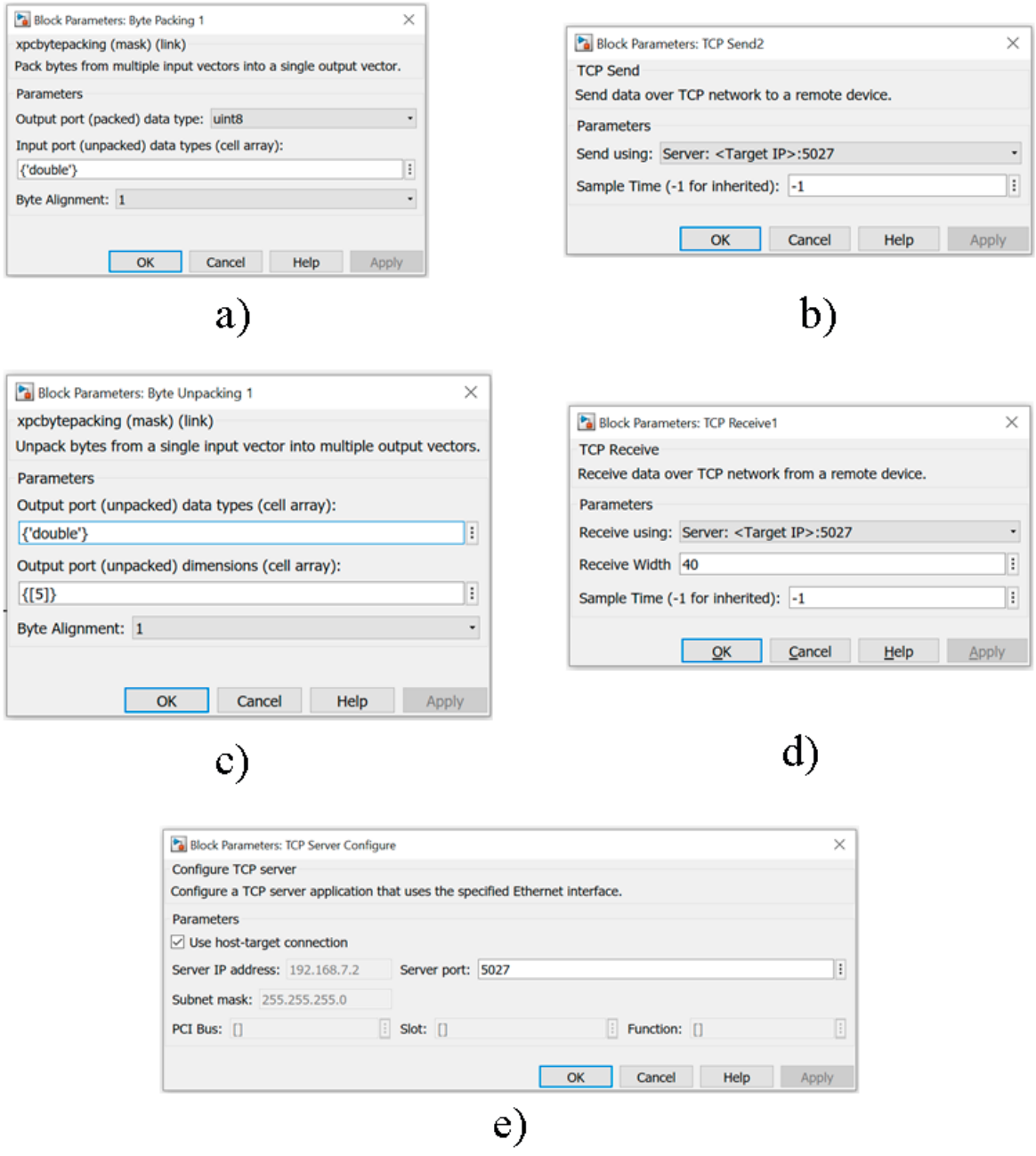


Figure D.0.4 – Simulink blocks present in the real-time model inside the Top red area in Figure 101.

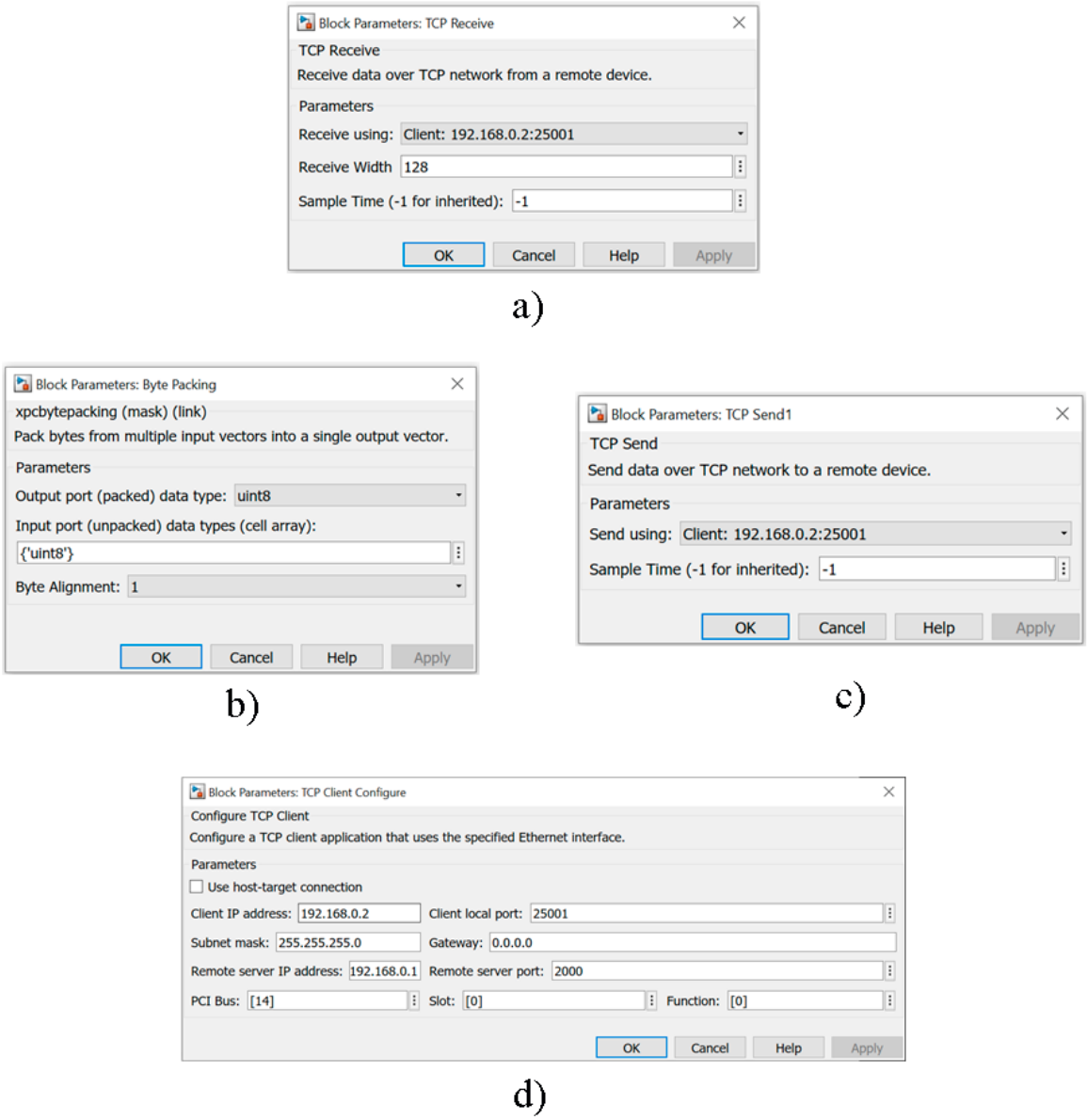


Figure D.0.5 - Simulink blocks present in the real-time model inside the bottom green area in Figure 101.

ANNEX E

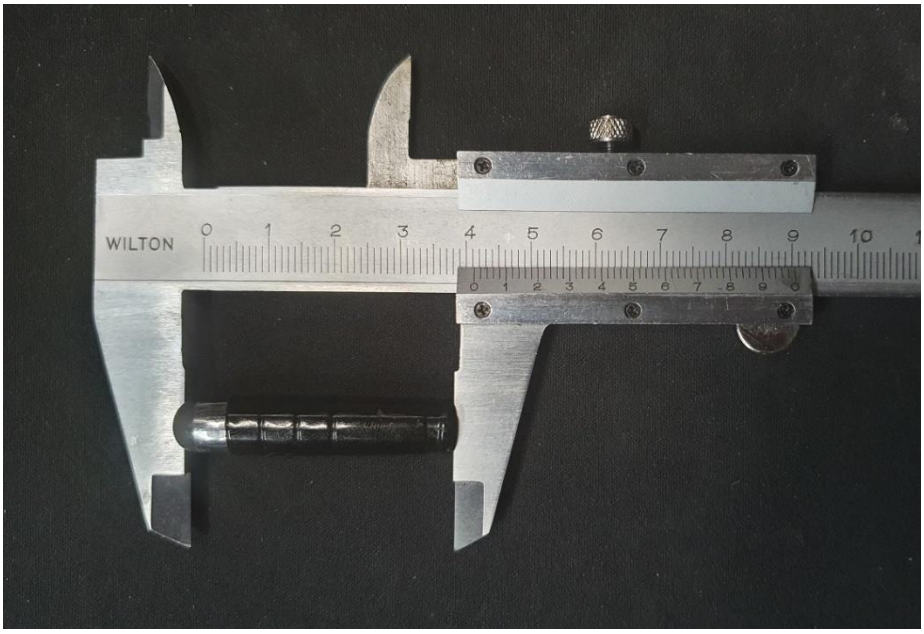


Figure E.0.1 – Measurement and representation of the selected stylus pen.

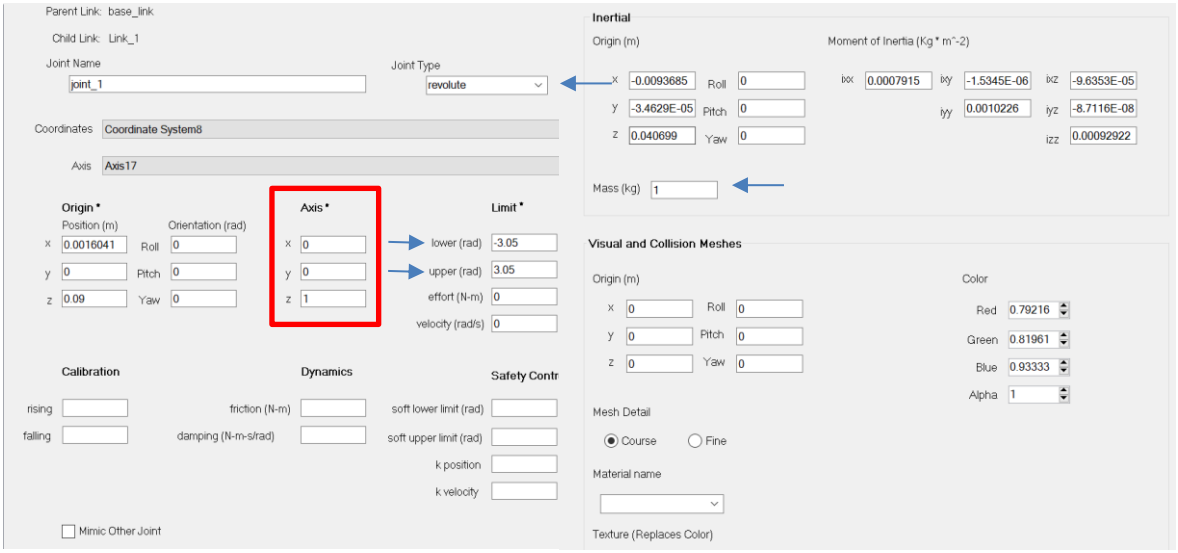


Figure E.0.2 – Example of the requered parametersfor the “joint_1” in the URDF Explorer plug-in.

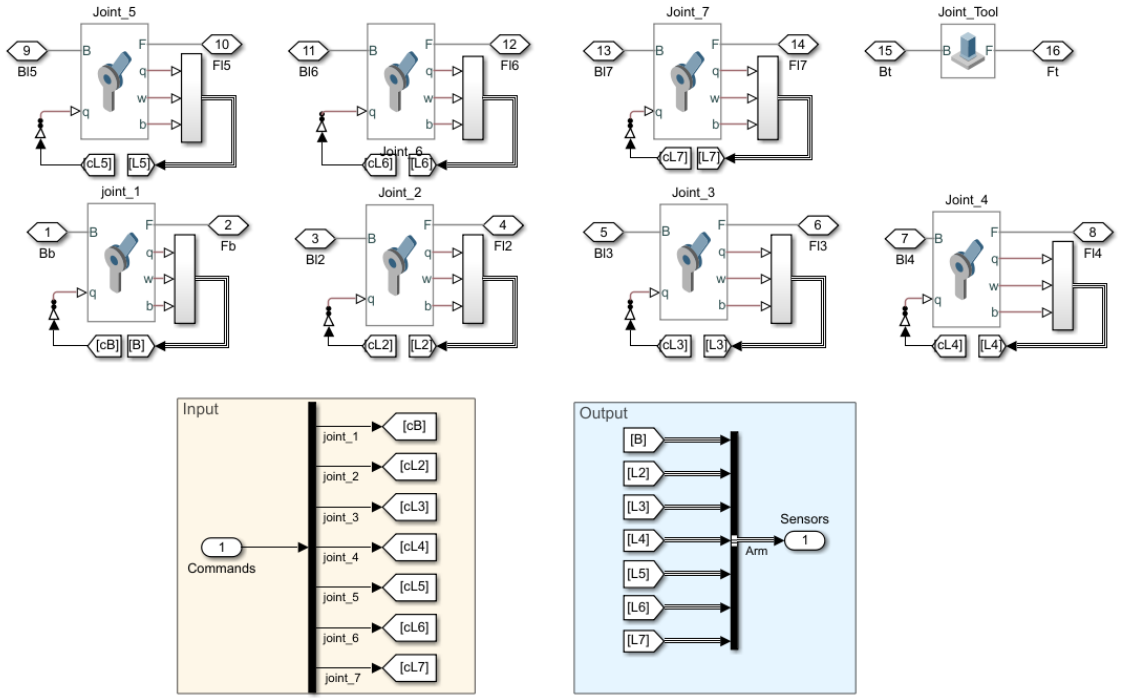


Figure E.0.3 – Schematic view of the Simscape blocks for the desktop Kassow Simulink model inside the subsystem Kassow KR810 represented in Figure E.0.7

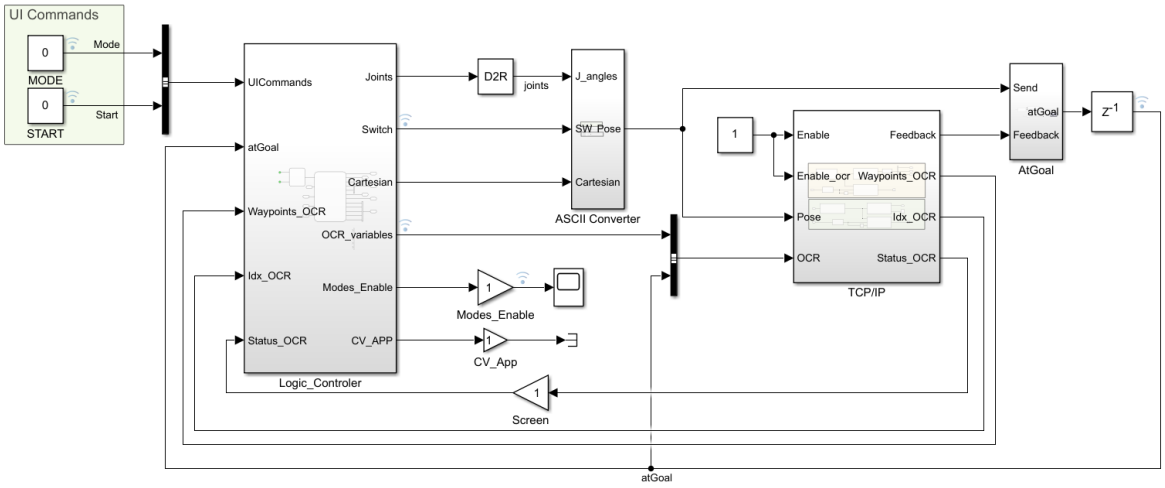


Figure E.0.4 – Representation of the real-time Simulink model running inside the Speedgoat target.

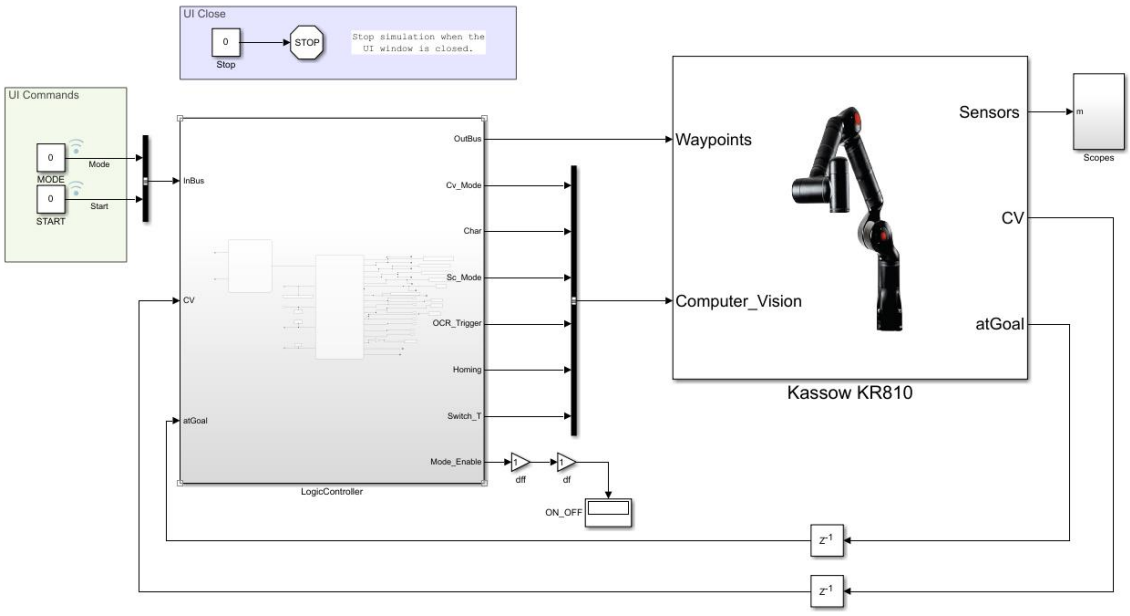


Figure E.0.5 – Representation of the desktop Kassow Simulink model.

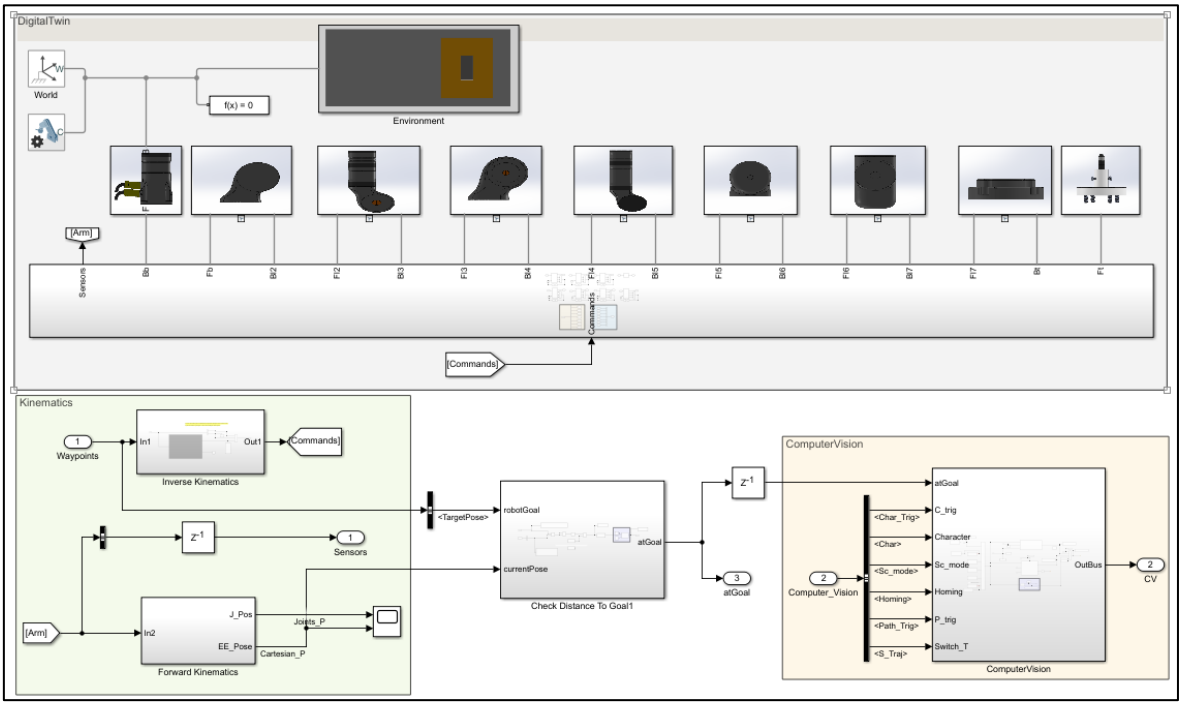


Figure E.0.6 – Schematic view of the subsystem “KassowKr810” represented in Figure E.0.5

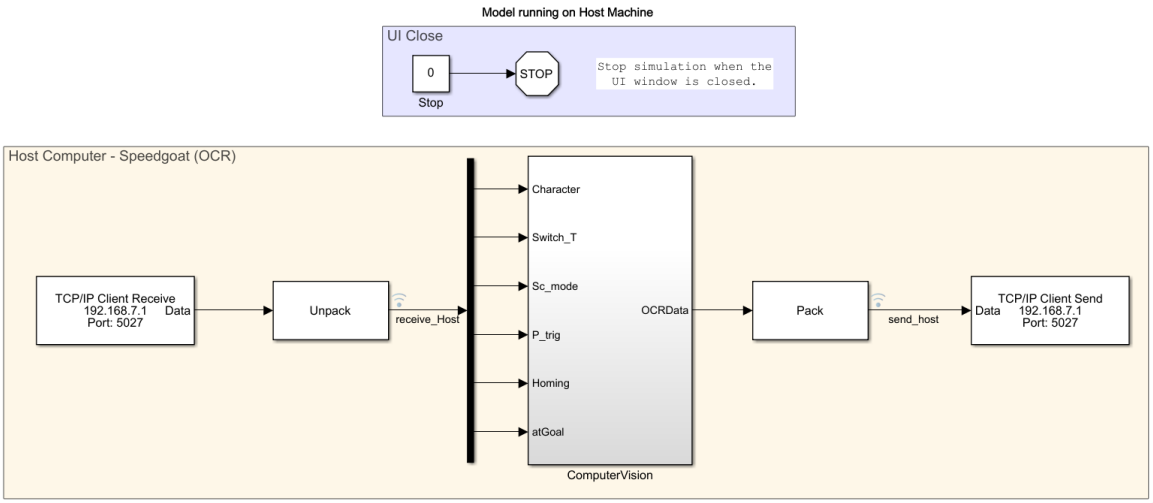


Figure E.0.7 – Representation of the OCR model deployed in the host computer

Task_Selection chart

These blocks have the following configuration: Start is 1 when the robot is homing and 0 when executing a task; Mode is 0 at home position and 1 when executing the task. However, if we intended to program more than one test, we just need to increase the value of this variable. The output parameter was created by selecting the chart in the property inspector and then checking the box “Create output for monitoring” and selecting the “Leaf state activity” from the dropdown menu. Our enumeration name was set to “TaskType”, since this will allow to set multiples tests in the future. This process allowed to define the necessary conditions inside the second stateflow chart, either for starting the required test or to stopping it. By setting up this enumeration this stateflow chart can be reutilize for more tests in future. The programmer will only need to add the new state for the test as well as the condition to activated it.

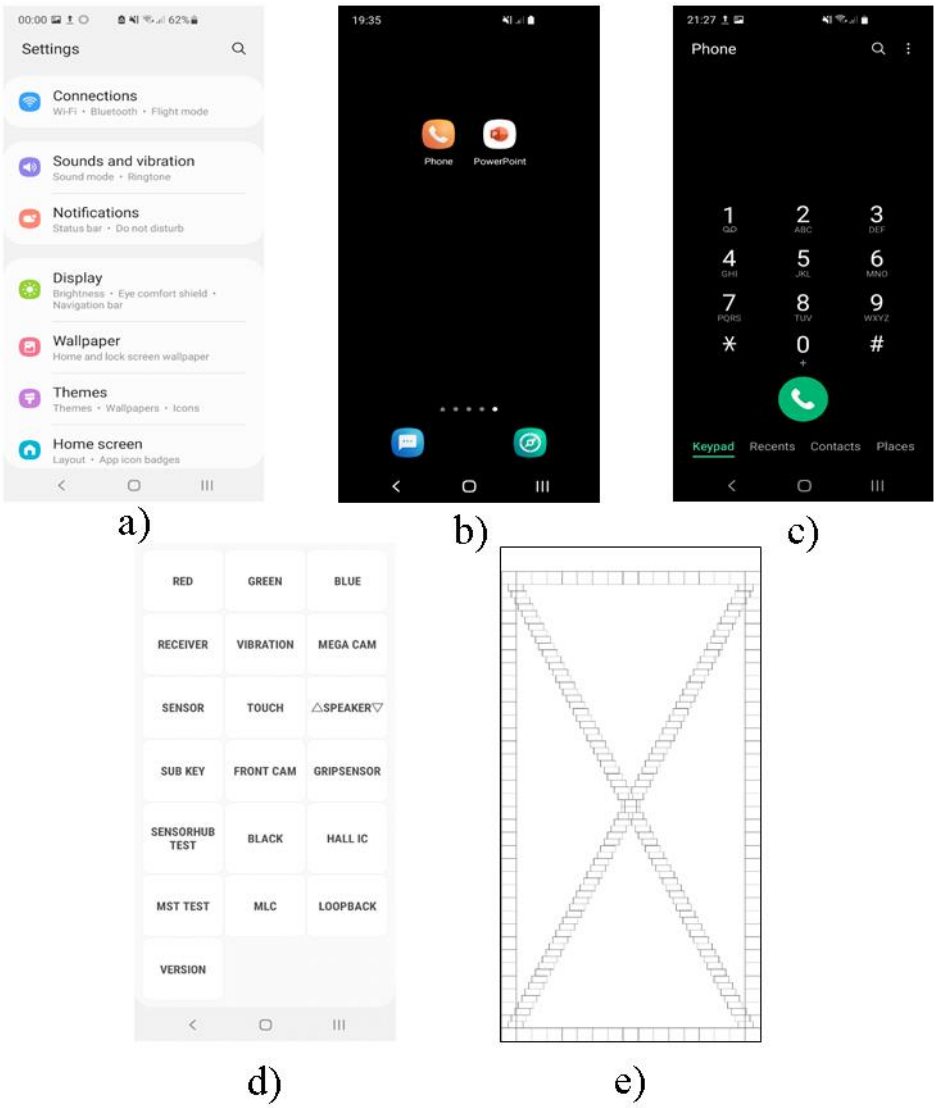


Figure E.0.8 – Example of the images used for testing: (a) Image used as error screen (ST). (b) Home screen (HScreen). (c) Dial screen (DDial). (d) Test modules screen (Module). (e) Test screen (TT)

ANNEX F

Character Enumeration	Character String
1	*
2	0
3	#
4	<
5	Touch
6	Keypad
7	PowerPoint

Table F.0-1 – Table with the correspondence between the used numbers and characters

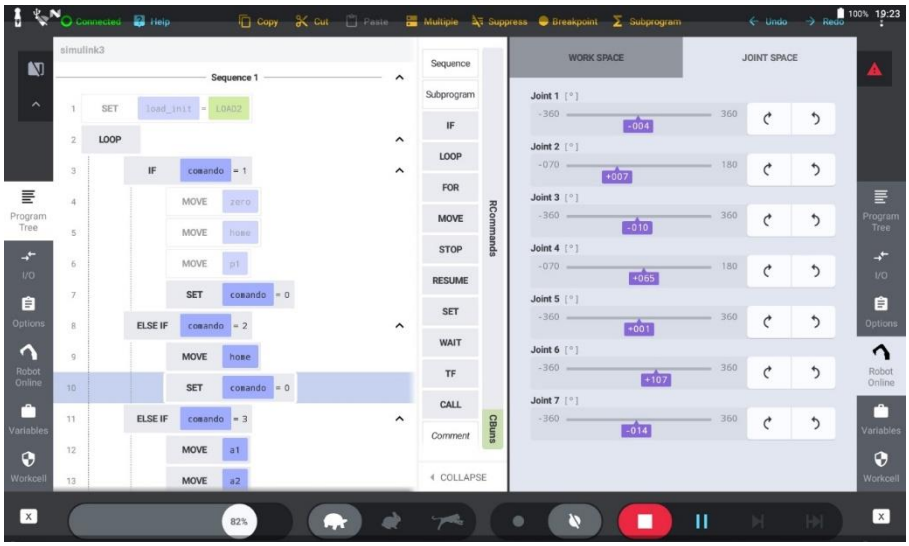


Figure F.0.1 – Kassow GUI application with the pre-programmed commands and joint space information

ANNEX G

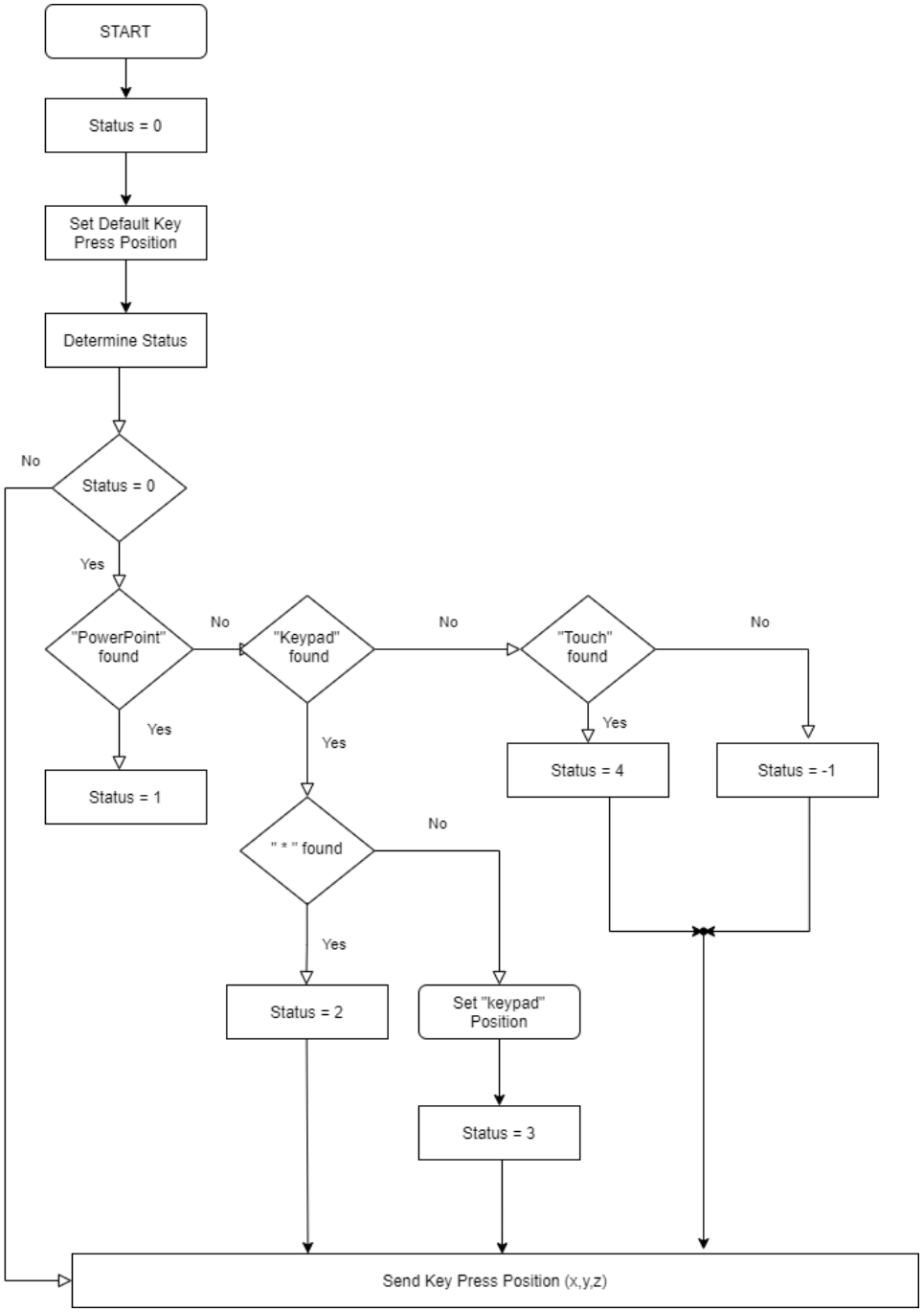


Figure G.0.1 – Schematic fluxogram of the logic for the “OCR_Status” function

Status	Screen Status	GUI Label
-1	Error/Cannot decided	-
-1 && 0	Error/Cannot decided	Error. Switching back to Home screen
-1 && 1	Touch Mode	Executing Test
0	Unknown	Unknown
1	Home Screen	Home Screen
2	Phone App	Ready/Pressing Keypad
3	Phone App but not in Keypad	Changing to Keypad
4	Touch Mode	Test Mode
5	PowerPoint	PowerPoint

Table 0-1 – Table containing the relationship between the status acquired by the "OCR_Status" function, the actual application shown in the device screen and the message displayed on the GUI

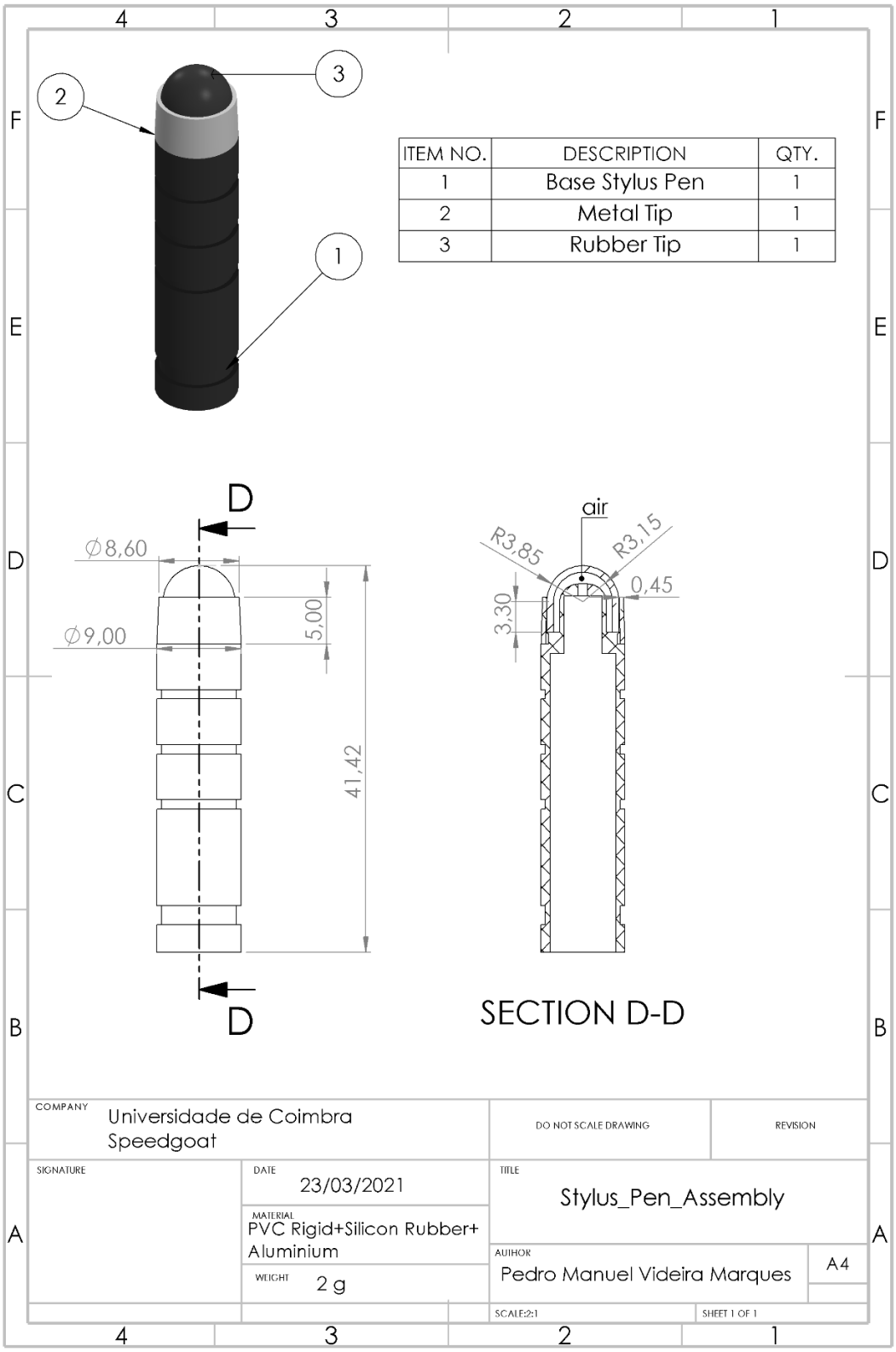


Figure G.0.2 – Representation of the CAD sheet with the “stylus_pen_assembly” information

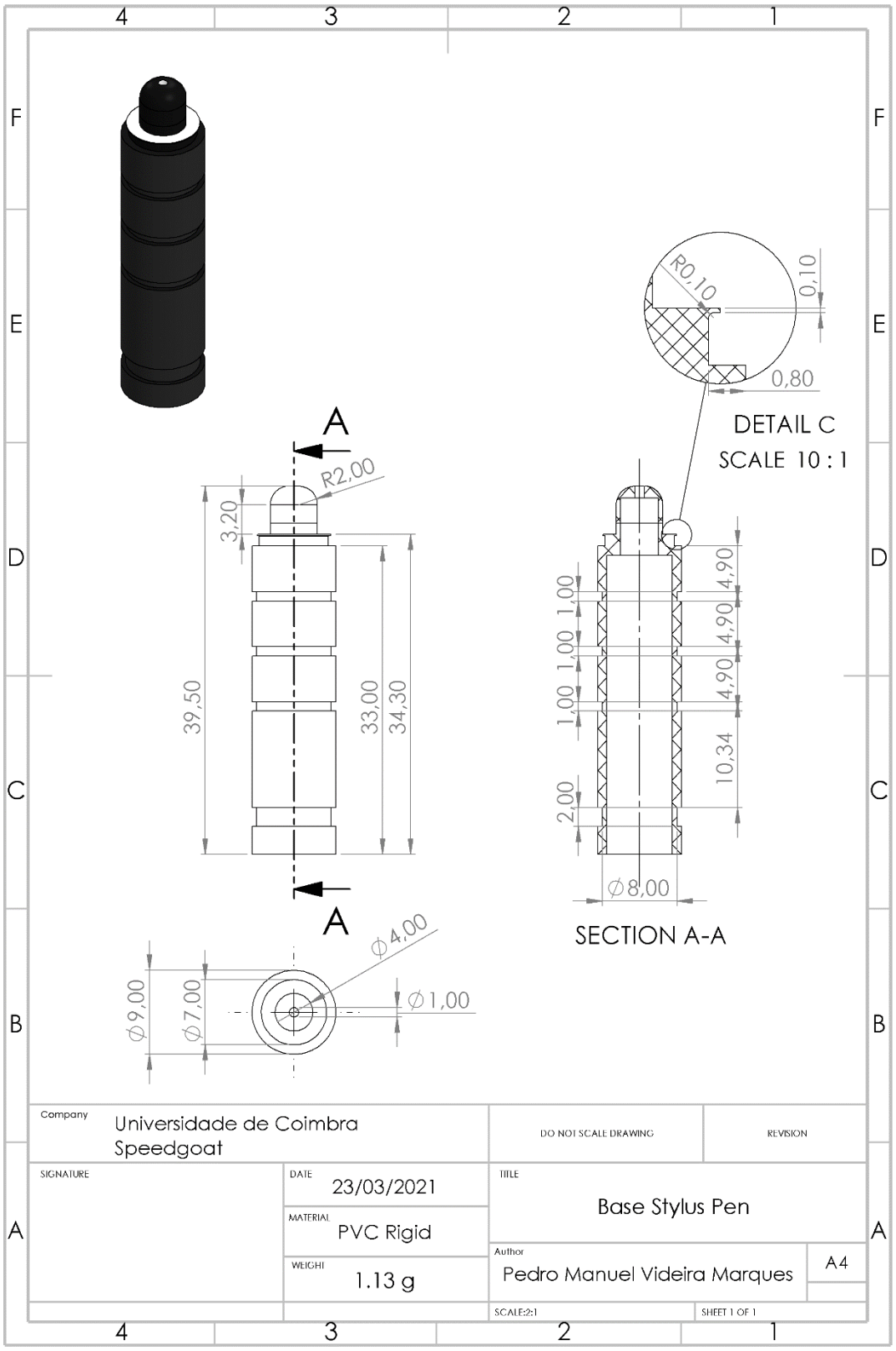


Figure G.0.3 – Representation of the CAD sheet with the *stylus* pen information.

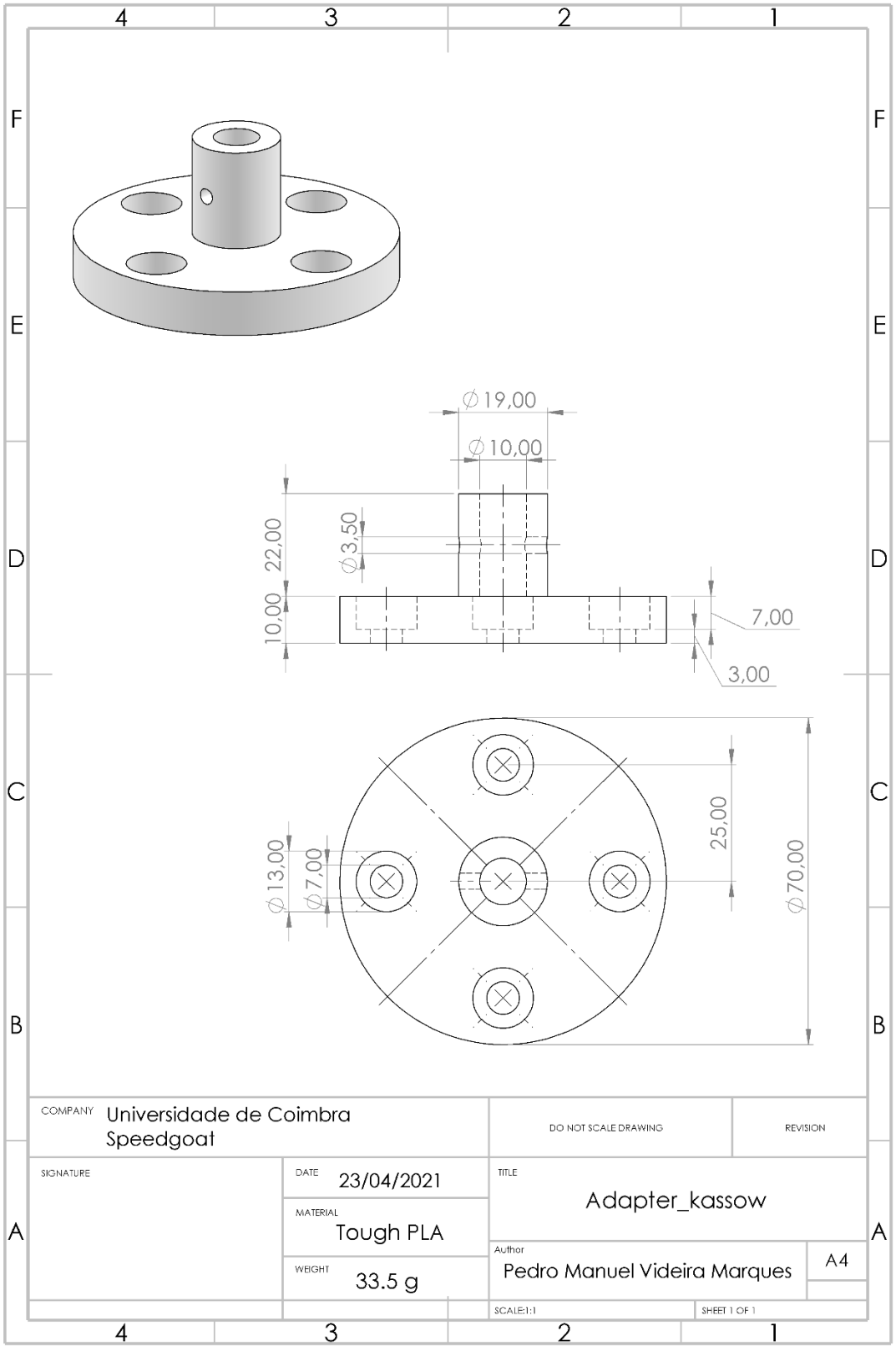


Figure G.0.4 – Representation of the CAD sheet with the measurements of the adapter developed for the Kassow KR810.

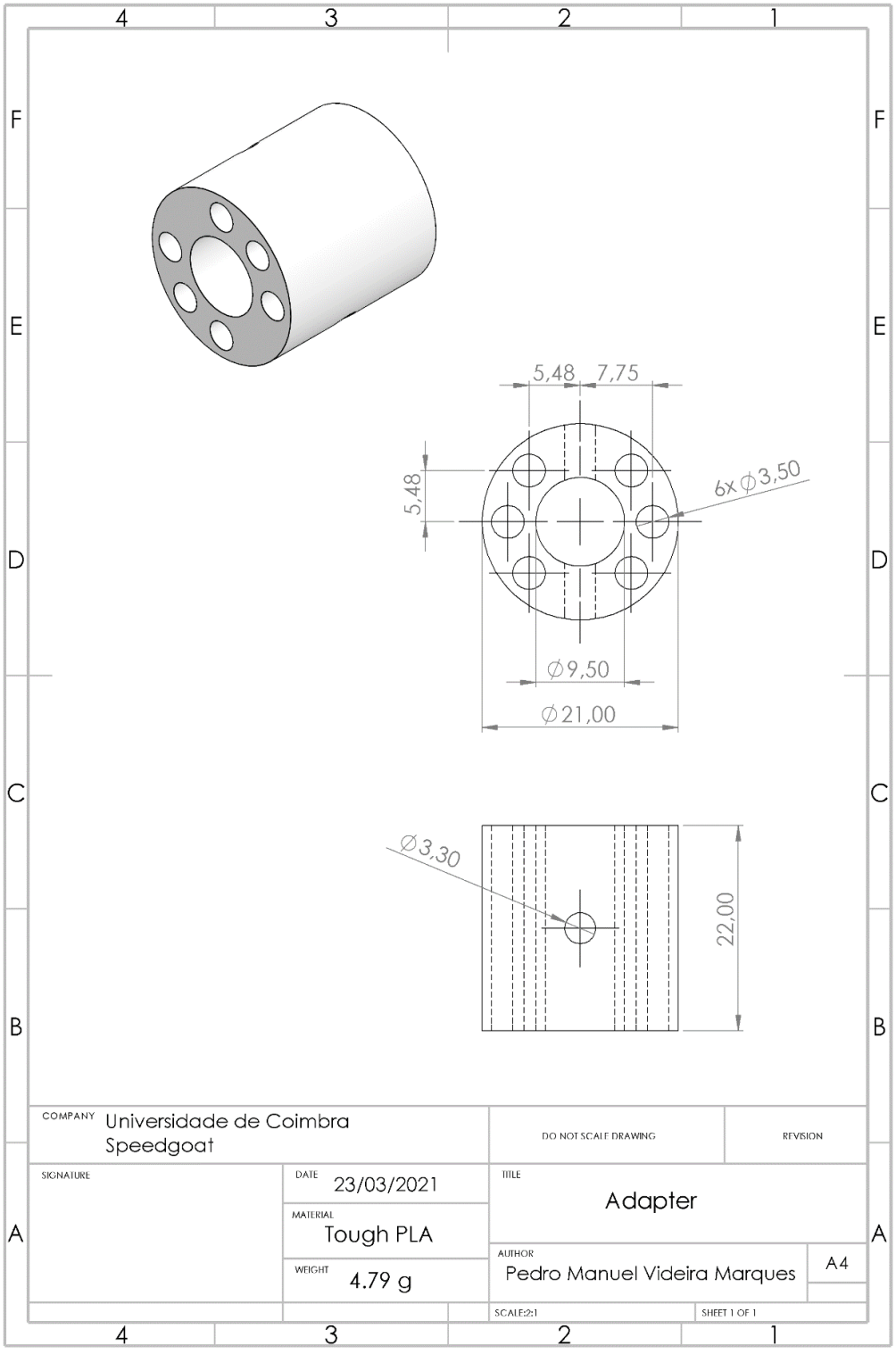
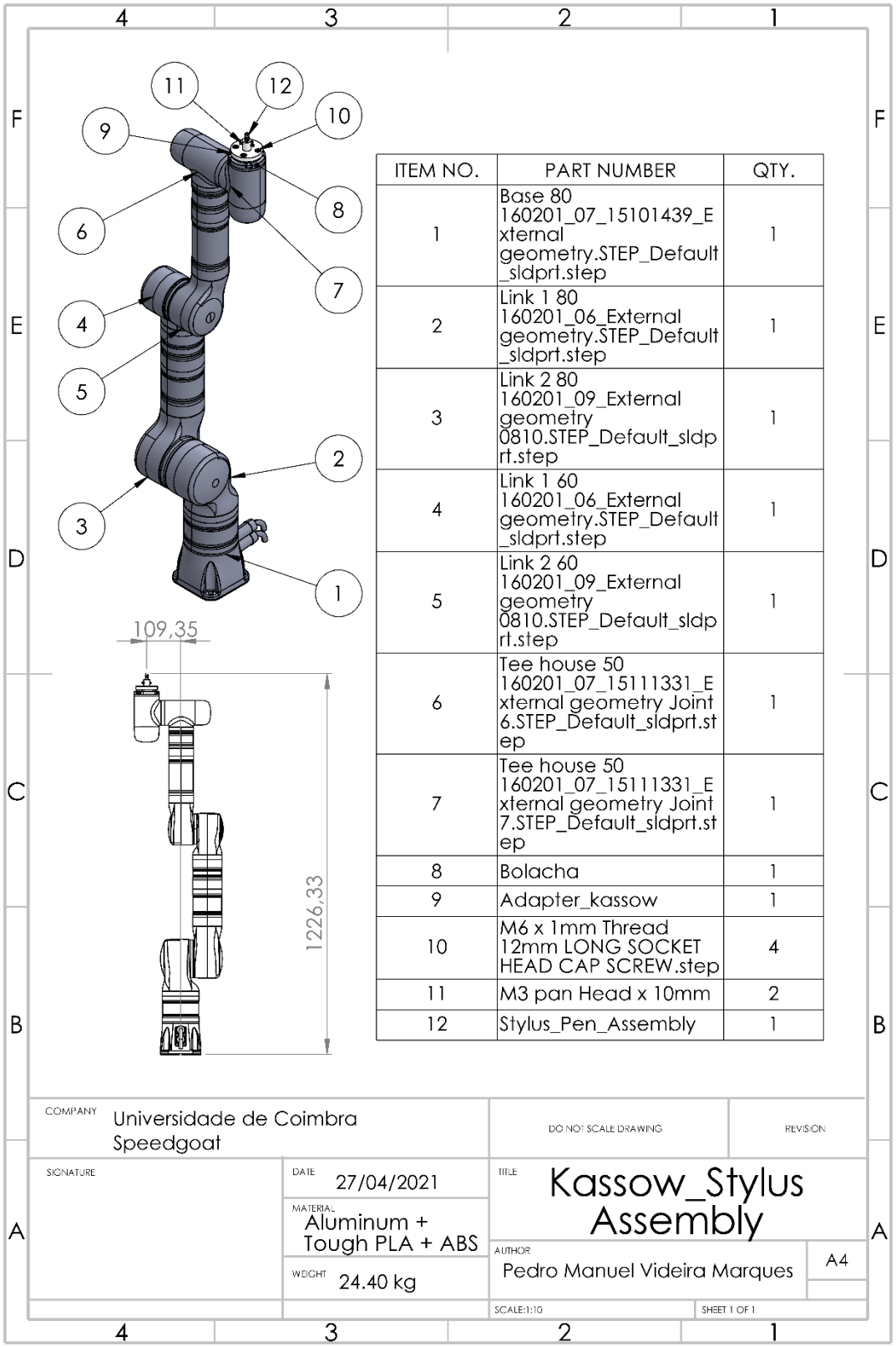


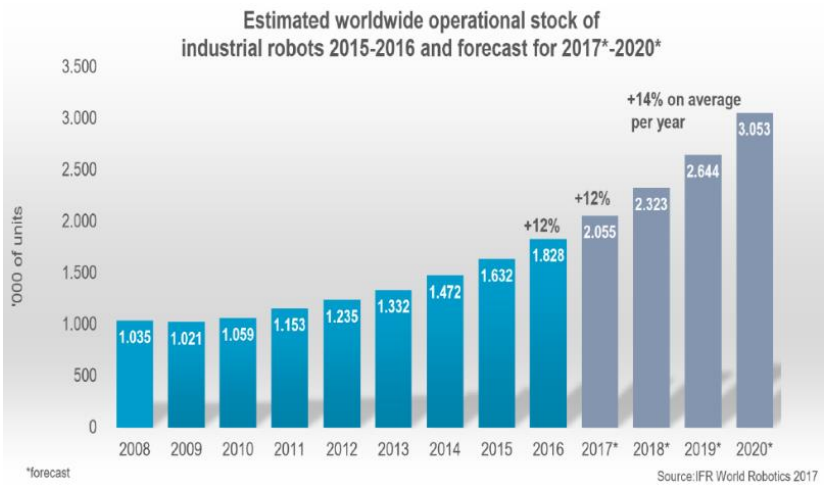
Figure G.0.5 – Representation of the CAD sheet with the measurements of the adapter developed for the Meca 500.



ITEM NO.	PART NUMBER	QTY.
1	Base 80 160201_07_15101439_External geometry.STEP_Default_sldprt.step	1
2	Link 1 80 160201_06_External geometry.STEP_Default_sldprt.step	1
3	Link 2 80 160201_09_External geometry 0810.STEP_Default_sldprt.step	1
4	Link 1 60 160201_06_External geometry.STEP_Default_sldprt.step	1
5	Link 2 60 160201_09_External geometry 0810.STEP_Default_sldprt.step	1
6	Tee house 50 160201_07_15111331_External geometry Joint 6.STEP_Default_sldprt.step	1
7	Tee house 50 160201_07_15111331_External geometry Joint 7.STEP_Default_sldprt.step	1
8	Bolacha	1
9	Adapter_kassow	1
10	M6 x 1mm Thread 12mm LONG SOCKET HEAD CAP SCREW.step	4
11	M3 pan Head x 10mm	2
12	Stylus_Pen_Assembly	1

Figure G.0.6 – Representation of the CAD sheet with the all the componenents presentes in the system (robot arm+adapter+pen+screws).

APPENDIX A



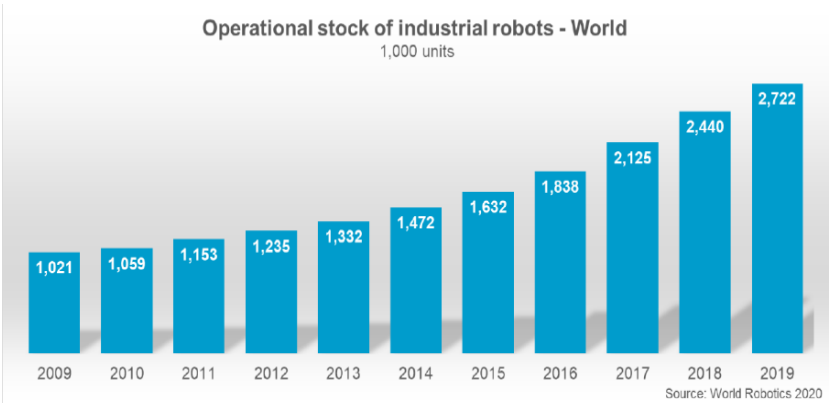


Figure AA.0.1– a): estimated operational industrial robots between 2017 and 2020, worldwide: b): Exact number of operational industrial robots between 2009 and 2019, worldwide. **Source:** IFR World Robotics 2017; IFR World Robotics 2020

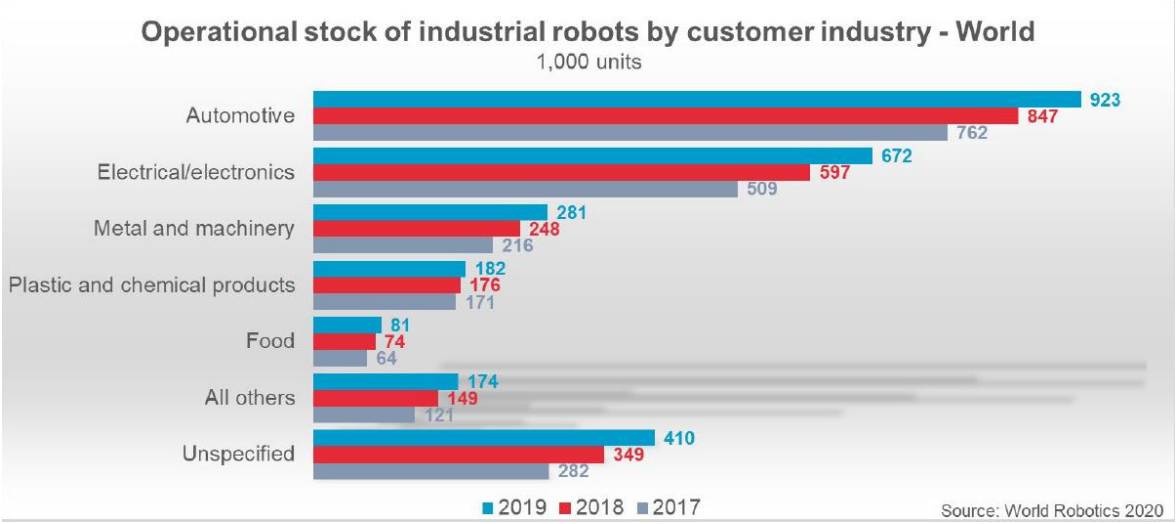


Figure AA.0.2 – Operational stock of industrial robots by customer industry**. Source:** IFR World Robotics 2020

Figure AA.0.3 – Speedgoat available products. **Source:** (Speedgoat, 2021)

APPENDIX B

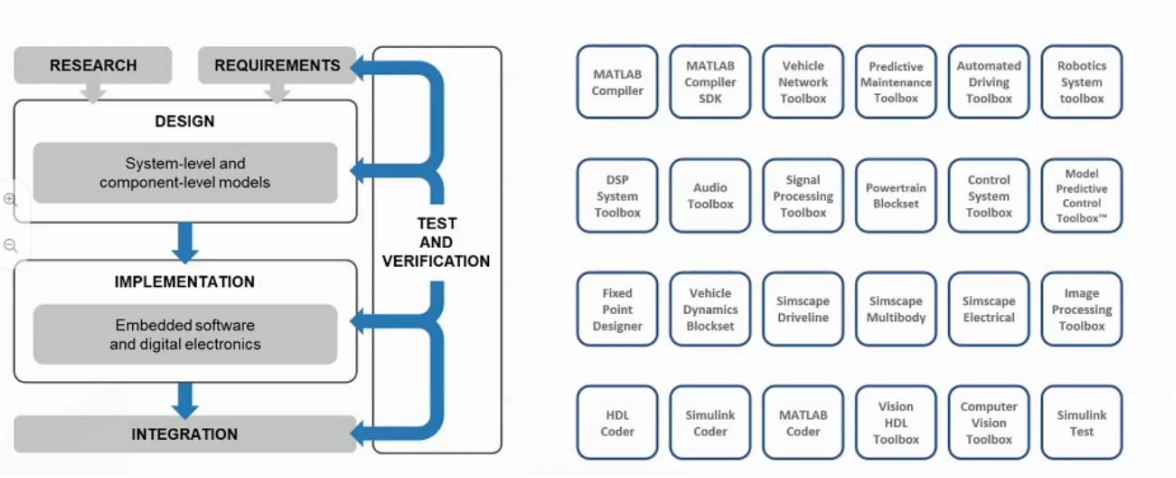


Figure AB.0.1 – MATLAB tools integration in the model-based design

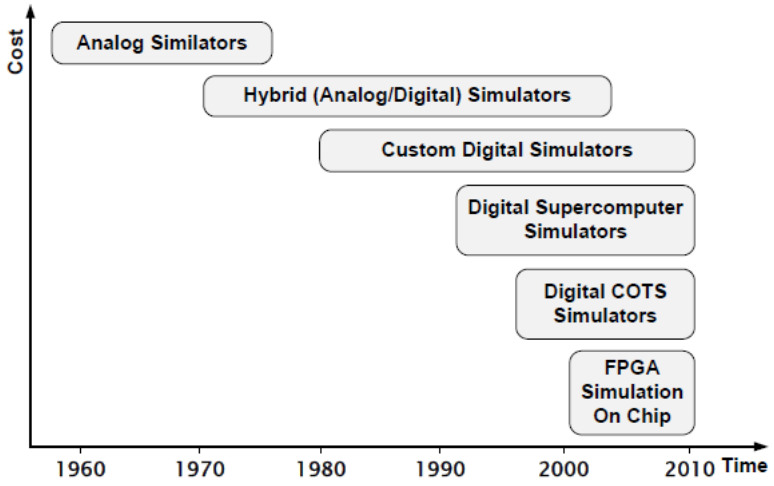
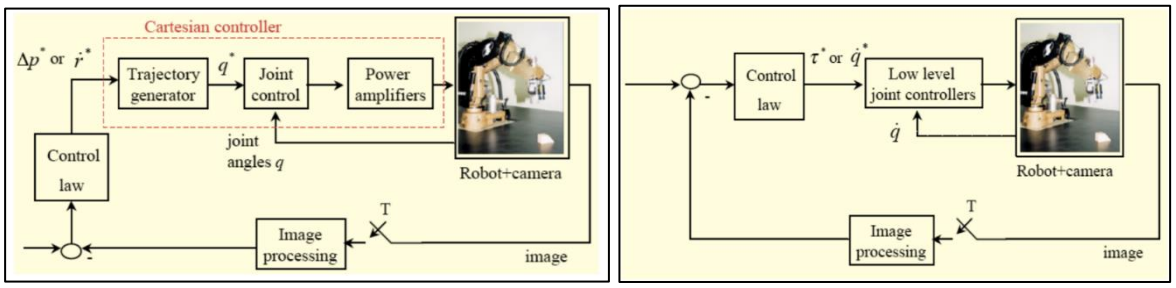


Figure AB.0.2 – Evolution of real-time simulation technologies. Source : (Bélanger et al., 2010)



a) b)

Figure AB.0.3 – (a) Indirect/external visual servoing. (b) Direct/internal visual servoing. Source: (De Luca, n.d.)

Figure AB.0.4 - Meca500 (R3) Source: https://www.mecademic.com/products/Meca500-small-robot-arm

Position repeatability	0.005 mm
Rated payload	0.5 kg
Max. payload	1.0 kg (under special conditions)
Weight of robot arm	4.5 kg
Range for joint 1	$[-175°, 175°]$
Range for joint 2	$[-70°, 90°]$
Range for joint 3	$[-135°, 70°]$
Range for joint 4	$[-170°, 170°]$
Range for joint 5	$[-115°, 115°]$
Range for joint 6	$[-36,000°, 36,000°]$
Max. speed for joint 1	150°/s
Max. speed for joint 2	150°/s
Max. speed for joint 3	180°/s
Max. speed for joint 4	300°/s
Max. speed for joint 5	300°/s
Max. speed for joint 6	500°/s
Max. TCP linear velocity in joint mode	more than 2,000 mm/s
Max. TCP linear velocity in Cartesian mode	500 mm/s

Figure AB.0.5 - Technical Specifications for the Meca500. **Source:** (Mecademic Inc., 2018)

APPENDIX C

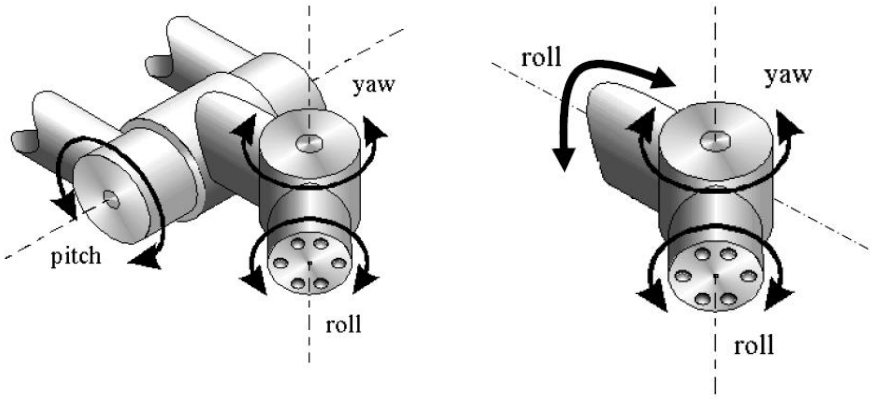


Figure AC.0.1 – Wrist configurations: Left: spherical wrist (roll-pitch-roll). Right: pitch-yaw-roll. Source: (Pires, 2007).

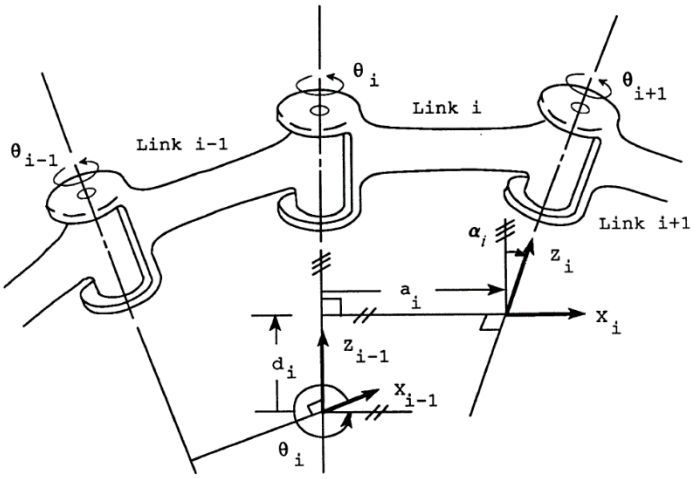


Figure AC.0.2 – This can be represented, by having the link *i* connected between the lower link *i-1* by joint *i* and the next link *i+1* by joint *i+1*. All joints have an axel, which can be rotational or translational. **Source:** (Paul, 1981)

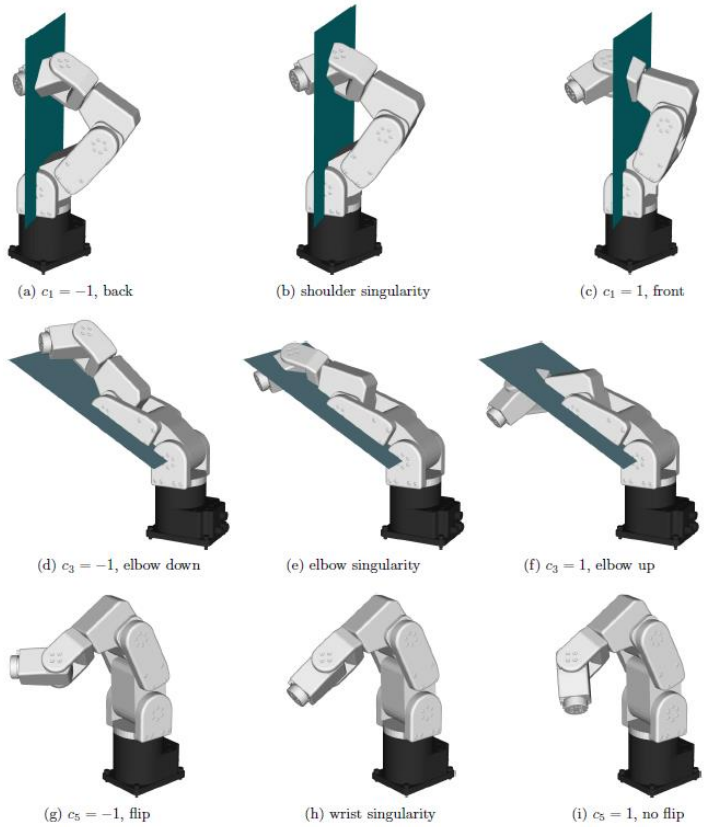


Figure AC.0.3 – Meca500 inverse kinematic configuration parameters and the three types of singularities. Source: (Flash et al., 2012)

Joint Block	Translational DoFs	Rotational DoFs	Total DoFs
6-DOF Joint	3	3	6
Bearing Joint	1	3	4
Bushing Joint	3	3	6
Cartesian Joint	3	0	3
Constant Velocity Joint	0	2	2
Cylindrical Joint	1	1	2
Gimbal Joint	0	3	3
Leadscrew Joint	1 (coupled translational-rotational)		1
Pin Slot Joint	1	1	2
Planar Joint	2	1	3
Prismatic Joint	1	0	1
Rectangular Joint	2	0	2
Revolute Joint	0	1	1
Spherical Joint	0	3	3
Telescoping Joint	1	3	4
Universal Joint	0	2	2
Weld Joint	0	0	0

Figure AC.0.4 – Example of the possible joint blocks in Simulink. **Source:** https://www.mathworks.com/help/physmod/sm/ug/joints.html?s_tid=srchtit