Tampere University

Olatz De Miguel Lázaro

IMPLEMENTATION OF AN ANTHROPOMORPHIC ROBOT CELL VIA WEB SERVICES AND FREE PATH INTERPOLATION IMPLEMENTATING DE CASTELJAU'S ALGORITHM

Engineering and Natural
Sciences
Master of Science Thesis
May 2019

ABSTRACT

Olatz De Miguel Lazaro: Implementation of an anthropomorphic robot cell via Web Services and free path interpolation implementating De Casteljau's algorithm
Master of Science Thesis
Tampere University
Automation Engineering
May 2019

Industrial automation technology evolves rapidly, and automated manufacturing systems need to find a way to improve the implementation and installation of new devices in the system. Adaptable and flexible systems reduce the time and cost of integration. One of the aspects to consider when building an adaptable and interoperable systems is the management of data flow between devices in the system. New technologies like web services have been implemented in manufacturing systems under a Service-Oriented Architecture (SOA).

Currently, robots are capable of performing simple paths composed of Point-to Point (PTP), linear and circular motions, which are adequate for most applications. However, some applications require the use of complex smooth paths to complete their operations. Robots require a simple and robust method to model complex paths within the industrial controller.

There are two main components to this thesis. One of the objectives of this thesis is to propose an approach to implement a robotic cell into a production line by using web services. This approach exposes the functionalities of the robot as services to the system, where those services can be requested via RESTful web services. The other component of the thesis presents a way for a robot to model and perform free shape paths through the evaluation of Bezier curves and the implementation of the De Casteljau algorithm into the robot controller.

The proposed approach was successfully deployed and tested on a production scenario. The testbed used was the FASTory production line, in the Factory Automation Systems and Technologies Laboratory (FAST-Lab) in Tampere University. The results of the tests show that the implementation of the approach dotes the system with flexibility and configurability and simplicity of installation.

Keywords: Anthropomorphic robot cell, system integration, Web services, Service-Oriented Architecture, free shape paths, Bezier curves, De Casteljau algorithm

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AMR	Advanced Manufacturing Research
API	Application Program Interface
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CIP	Common Industrial Protocol
CNC	Computerized Numerical Control
CRUD	Create, Read, Update, Delete
DCS	Distributed Control System
DPWS	Device Profile Web Services
ERP	Enterprise Resource Planning
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transmission Protocol
ISA	International Society of Automation
JSON	JavaScript Object Language
KRL	KUKA Robot Language
KSS	KUKA System Software
MES	Manufacturing Executing System
MESA	Manufacturing Enterprise Solutions Association
PLC	Programmable Logic Control
PTP	Point-to-Point
R&D	Research & Development
REST	Representational State Transfer
RFID	Radio Frequency Identification
RTU	Remote Terminal Unit
SCADA	Supervisory Control And Data Acquisition
SCARA	Selective Compliance Assembly Robot Arm
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
ST	Structured Text
TCP	Tool Center Point
TCP/IP	Transmission Control Protocol/Internet Protocol
WIP	Work In Progress
WSDL	Web Services Description Language
W3C	World Wide Web Consortium
XML	Extensive Markup Language

.

1. INTRODUCTION

1.1 Background

During the last few decades, there have been great advancements in industrial automation technologies. It is estimated that in 2008, there were around one million industrial robots in operation worldwide, and that number increased to over 1.8 million industrial robots in 2016 [1]. According to the International Federation of Robotics' forecast in 2017, this figure is expected to raise to 3 million in 2020. Manufacturing systems are required be highly adaptable and reconfigurable in order to accommodate rapidly to the changes in the industry. Systems need to be able to adapt to new products or services in the manufacturing company, as well as changes in the production line [2].

Adaptability is an important aspect to take into account during the integration of new systems, or when existing ones are upgraded. System integration and upgrades are accompanied by high costs, so by investing in highly adaptable and flexible systems, the costs of changing the production line in the future are reduced.

Another important aspect to consider during system implementation is interoperability. Interoperability between devices allows them to share knowledge between each other. This factor helps to achieve a better integration among components in production systems.

During the last years, Service-Oriented Architecture (SOA) has been widely implemented in order to make systems more flexible and reusable. An example of this trend is the SIRENA project [3], which describes a SOA framework for an industrial system composed of intelligent devices.

Web services are used extensively for realizing SOA systems [4]. Web services simplify interoperability between components of a system, and their validity on manufacturing systems has been proved in various test-beds, like the SOCRADES project [5].

Another factor that must be addressed during integration of an industrial robotic system is the functional requirements of the robot in regards of the application. Simple industrial applications like assembly do not normally require very complex path planning, as most of the operations that take place in the workspace are reduced to simple pick and place tasks. However, applications like welding or painting require more complex path and trajectory planning in order to ensure the correct operation of the robot and quality of the product [6]. Path planning is also important for collision avoidance purposes.

In the las few decades, some models and algorithms have been designed in order to model smooth complex paths, like Bezier curves or B-Splines.

1.2 Problem definition

Nowadays, production systems must be enhanced by doting its devices with adaptability, flexibility, reconfigurability and interoperability. When changes are made to the system, or when the system is expanded or upgraded, it must be able to adapt to the changes and make the integration as economical and fast as possible. Web services provide a tool for easy integration of the communication between devices in an industrial system.

On the other hand, some industrial applications require the modelling and calculation of complex paths. Industrial robots are capable of performing simple motions, like linear or circular movements, while more complex motions are not typically implemented within the controller. In applications where free shape paths are required, a time-effective and computationally simple algorithm is required.

This thesis focuses on describing the implementation of new robotic equipment into an existing production line. The production line is built with SOA, and the functionalities of the robot are exposed to the system as services. The integration of the new cell in the system is done via web services, using RESTful requests to invoke services from the robot. The services provided by the robot require the use of smooth complex paths, which can be modelled by using Bezier curves. Evaluation of the curves with the De Casteljau algorithm provides a simple and numerically stable program that is easily integrated in an industrial robot.

1.3 Objectives and scope

The objective of this thesis is to find an answer to a series of questions related to the implementation of a robot cell into a production line. The research work and case study presented in this thesis tries to answer the following questions:

- How can a new robot cell be integrated as a service provider into a SOA system?
- How can free shape paths be implemented into an industrial controller?

With these questions, the scope of this thesis is established. The work presented in this is related to the integration and control of a robotic arm within an industrial process. The communication between the robot cell and a higher-level orchestrator will be defined so that the robot acts as a service provider for the line, where the orchestrator requests a service and the robot executes its operations in response.

On the other hand, this work researches and implements a method that allows industrial robots to perform free shape paths. The method should minimize the complexity of the code, and it should be available to be implemented in all industrial controllers, regardless of the topology of the robot, its manufacturer or the programming language they use.

The scope of this thesis does not include all aspects of the cell implementation into an industrial production line. The safety interface, for example, is not included. Safety on the cell should be integrated to the performance of the entire production line. The safety control must be able to detect errors in one of the cells and coordinate the line in order to prevent the propagation of the error. That can be achieved with a centralised safety control, in which a safety PLC monitors all cells.

The orchestration and monitoring of the production line is not covered by this thesis either. However, the approach for the cell implementation in this document is proposed in a way that allows the cell to be integrated seamlessly with a higher-level control

1.4 Outline

The structure of this thesis is as follows; chapter 1 provides an introduction to this document and established the problem and the objectives of the thesis work. Chapter 2 provides a theoretical background and defines the concepts that will be explored later in the document. Chapter 3 presents a proposal to achieve the objectives established in chapter 1. In chapter 4, the implementation of the proposed approach is described. Chapter 5 presents the tests done during the implementation stage in order to prove the validity of the proposal, and the obtained results. Finally, chapter 6 presents the conclusions of the work and defines some future development that can be applied to expand on this thesis.

2. STATE OF THE ART

Normally the factory shop floor of a manufacturing automated system is composed of several workstations dedicated to performing some specific tasks. Nowadays, a more distributed approach is being taken when implementing manufacturing systems, and flexibility in production is a key component in the development of the operations. When implementing a new production system or updating an existing one, both the physical devices and the communication technologies must be carefully examined and selected in order to ensure flexible and efficient operations. This section examines the state of the art and current trends for some of the technologies that are related to this thesis.

This chapter is divided into six sections; the first section defines the different topologies of industrial robots, emphasizing on anthropomorphic robots. The second section describes MES systems and their place in an automated process, as well as the main functionalities that it provides to the plant's management. The third section discusses the position of web services in MES and in the factory shop floor. In the fourth section, an overview of the importance of describing free shapes paths in industrial robots is given. The fifth section expands on free shape paths by explaining how they can be described using Bezier curves and B-splines. The last section explains how Bezier curves can be implemented into industrial robots by using the De Casteljau algorithm.

2.1 Anthropomorphic robots

The automation of manufacturing systems has rapidly increased in the las few decades. One of the biggest trends in industrial automation has been the use of industrial robots along with computer-aided design (CAD) and computer-aided manufacturing (CAM) systems.

The density of robots in manufacturing industries raises evert year. Robots have increased their speed and repeatability, significantly improving the quality of the final product; meanwhile, the costs of industrial robots have decreased significantly. In 2009, industrial robots had an average selling price of 63 000 US dollars, while in 2016 that figure had decreased to 46 000 dollars [7]. Furthermore, robots provide safety to the system. Robots are tasked to perform operations that are repetitive or dangerous for human workers, thus avoiding any possible injuries or accidents that might occur. In turn, human workers are transferred to more challenging and safer task.

The leading robot manufacturing companies worldwide (as of 2017) in terms of revenue from industrial robot sales and their revenue [8] are shown **in Figure 1**:

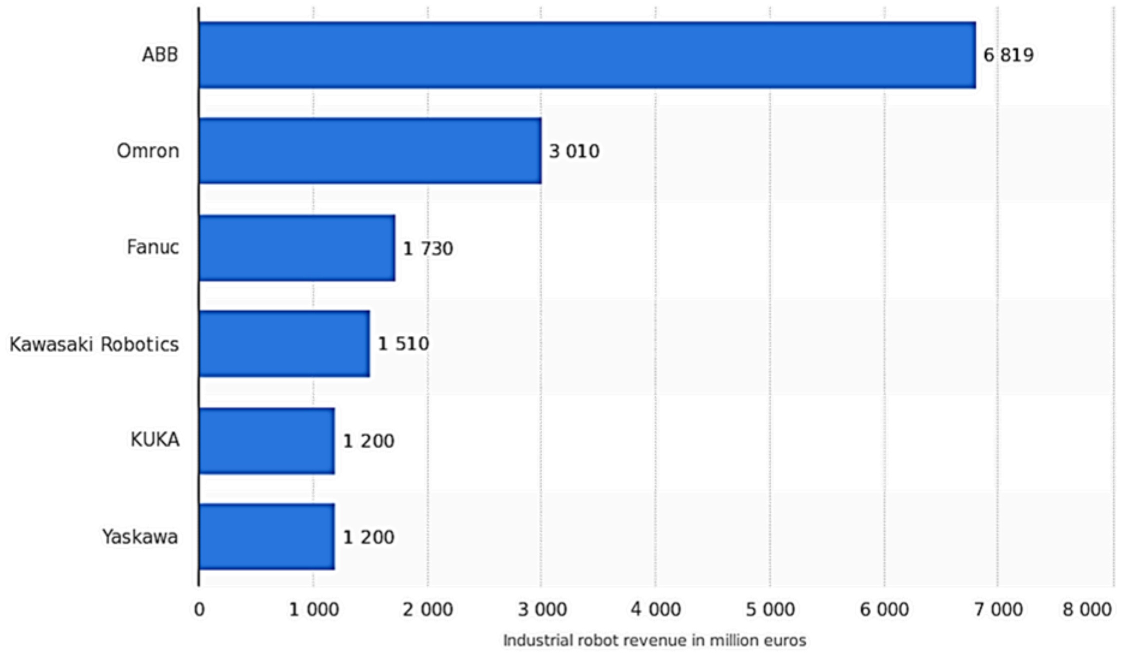


Figure 1*. Leading robot manufacturers worldwide by revenue [8]*

Based on their mechanical structure and the combination of the joints, commercially available industrial robots are classified into 6 main topologies [9]: cartesian, cylindrical, polar, parallel, SCARA and articulated. Each of these topologies offers different configurations.

- Cartesian robots: These robots have linear axes that move in X, Y and Z direction [10]. They use linear actuators to position a tool. Cartesian robots have high precision and positioning accuracy, due to their rigidity. Cartesian robots are widely used in pick and place operations and machining applications where very tight tolerances are required. The most common types of cartesian robot are Gantry robots.

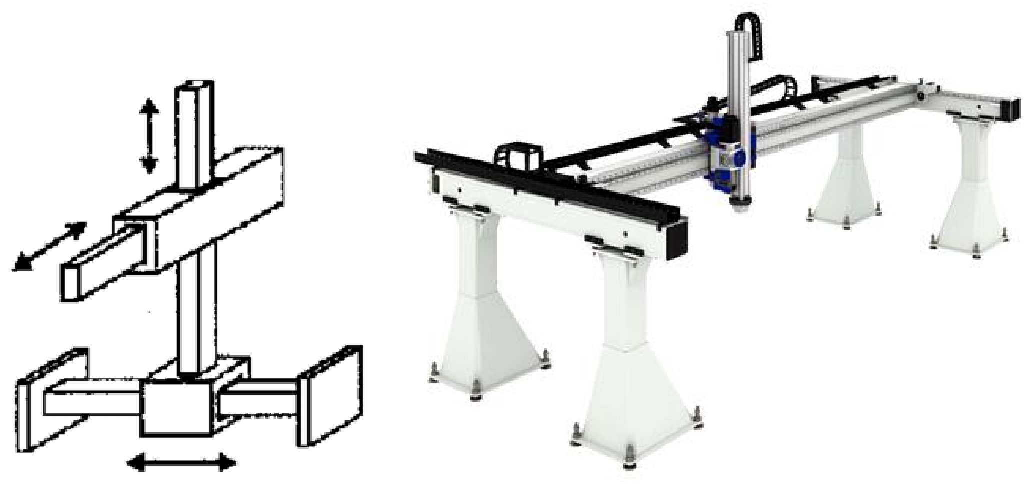


***Figure 2**. Cartesian robot axis configuration [9] and FlexMotion4 Gantry robot from Automated Motion*

- Cylindrical robots: Their first joint is revolute, while the second and third ones are prismatic. The work envelope of these types of robots has a cylindrical shape. They are used for assembly operations, machine tool handling, die casting and spot-welding applications.

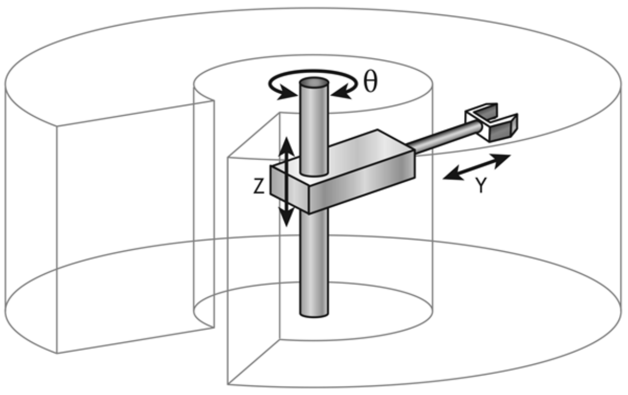


***Figure 3**. Cylindrical robot axis configuration [11]*

- Spherical or polar robots: They are composed of two revolute joints orthogonal to one another, followed by a prismatic one that provides radial extension [12]. This results in a spherical work envelope. They are commonly used for applications involving die casting, material handling and welding.

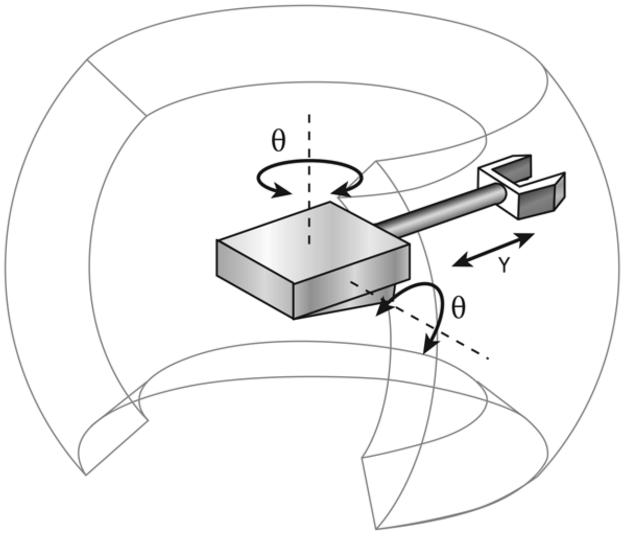


***Figure 4**. Polar robot axis configuration [11]*

- Parallel robots: More commonly known as Delta robots. They have 3 parallelograms that can rotate with respect to a fixed base [13]. The parallelograms are connected to a moving platform, in a way that keeps the platform parallel to the base at all times. They have low payload capacity and can work at high speeds with precision, which makes them valuable for pick and place applications.

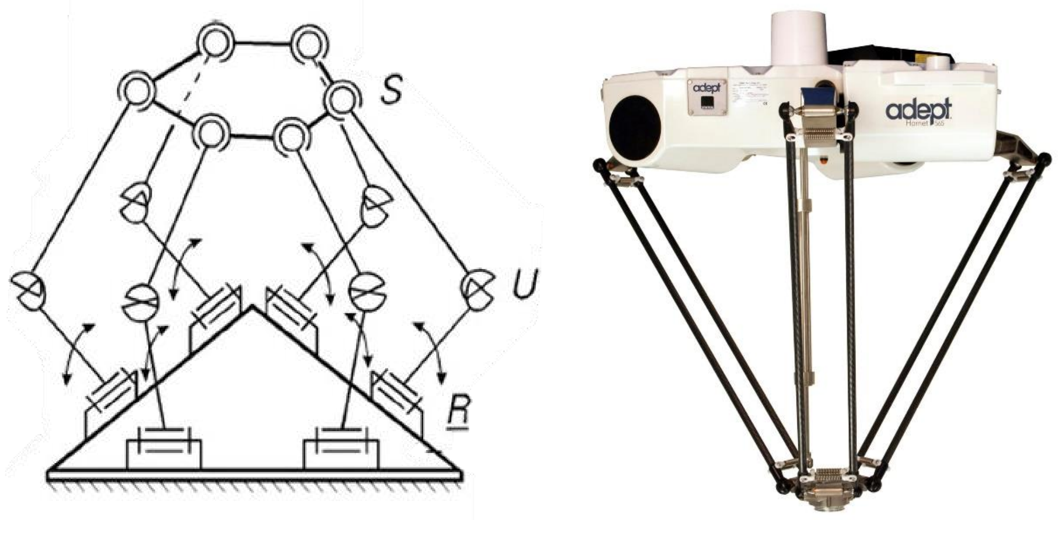


***Figure 5**. Parallel robot structure [14] and Adept parallel robot from Omron Electronics*

- SCARA robots: SCARA stands for Selective Compliance Assembly Robot Arm. A SCARA robot is composed of two parallel revolute joints and one prismatic joint. The revolute joints position the tool in the XY plane while the prismatic one moves the tool along axis Z. This makes them very useful for vertical assembly applications where the payload is small.

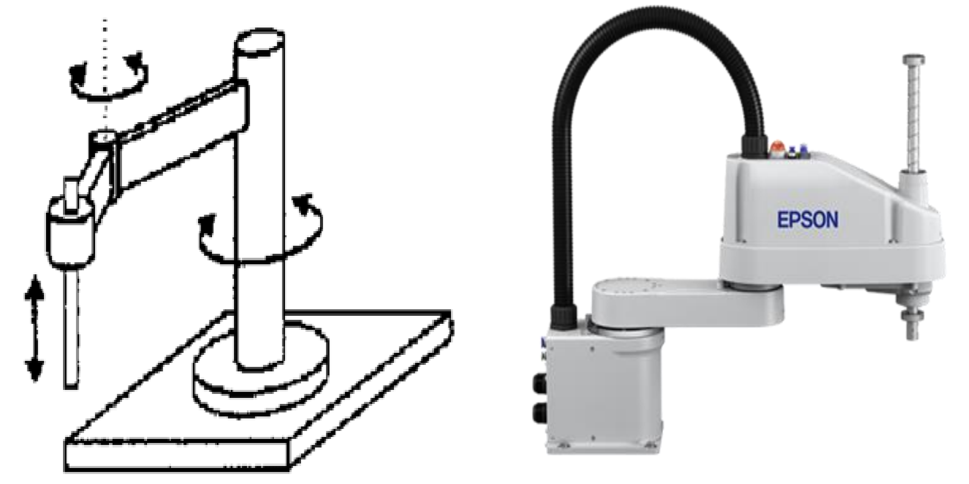


Figure 6. *SCARA robot axis configuration [9] and LS6 SCARA robot from Epson*

- Articulated or anthropomorphic robots: All the joints of these type of robots are revolute. Most articulated robots have 6 axes, which gives the manipulator 6 degrees of freedom. An anthropomorphic robot consists of two ”shoulder” joints, a ”elbow” joint and two or three ”wrist” joints.

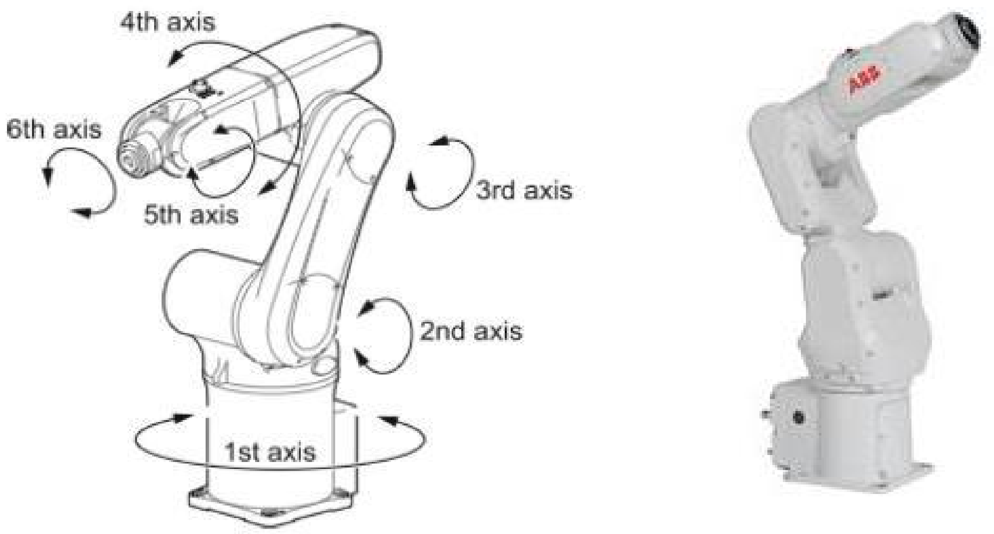


Figure 7*. 6-axis anthropomorphic robot axis configuration [15] and Anthropomorphic robot IRB 1100 from ABB*

Articulated robots are the most commonly used in manufacturing industry. With their 6 degrees of freedom, the tool can reach any position in the workspace with the desired orientation [10]. Due to the versatility of these robots, granted by the number of degrees of freedom they possess, they are used in a vast amount of applications, like welding, assembling, palletizing or painting.

Historically, they have had lower accuracy than other topologies of robots like Cartesian robots, due to errors that occur in the position of the joint angle, which accumulate through the arm [12]. However, both speed and accuracy have increased through the years. Nowadays, anthropomorphic robots are available in a wide variety of dimensions, payloads and speeds. As was mentioned earlier, their prices have dropped, making them a very affordable option.

In the 2018 report by Global Market Insights [16], it is stated that articulated robots hold the highest share of sales in the market, and the demand is forecasted to increase greatly in the following years. Figure 8 shows the market size of the different robot topologies in China during 2016 (in USD million) and the forecast for 2024.

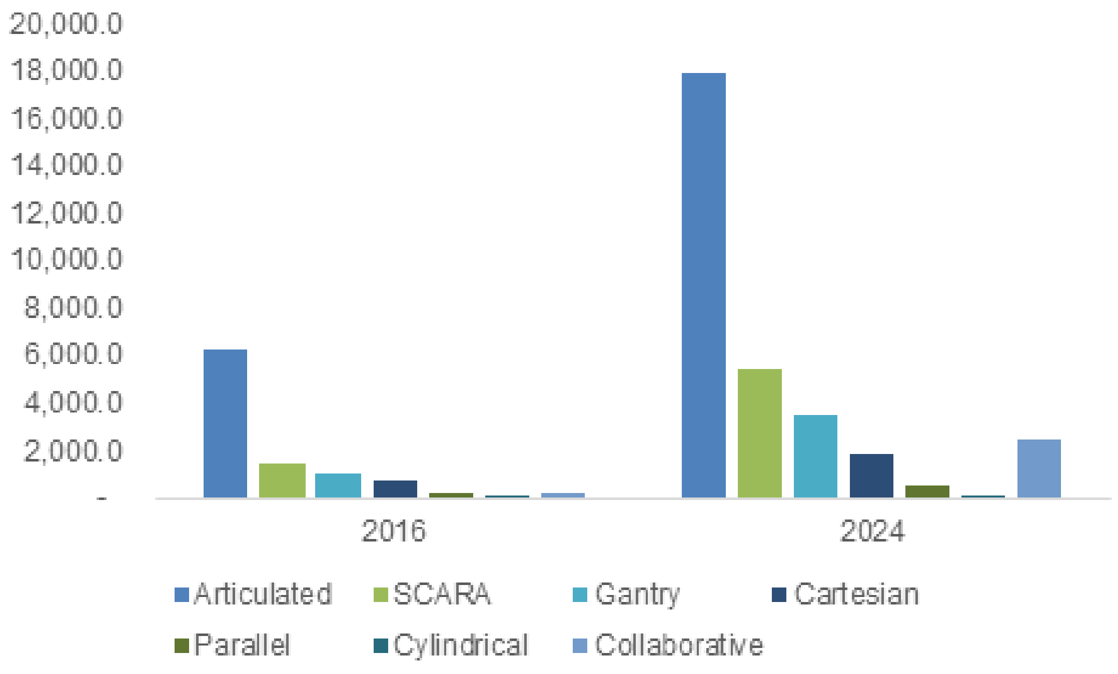


***Figure 8**. Market size of different robot topologies in China in 2016 and forecast for 2024 (in USD million) [16]*

2.2 Manufacturing Executing Systems

Manufacturing Executing Systems were first introduced in the 1970s. Before MES, the Enterprise Resource Planning Layer (ERP) and the control of the factory floor had been mostly isolated. As said in [17], MES provides an intermediate layer to integrate ERP and Distributed Control System (DCS). During the 1990s, a company called Advanced Manufacturing Research (AMR) proposed an integration model based on these three layers (ERP-MES-DCS). In this model, ERP acts as a plan management system, DCS acts as a control system and MES works as an intermediate between the two. The different layers that constitute an automated process can be seen in Figure 9**Figure 9.** Layers of the pyramid of automation [18].

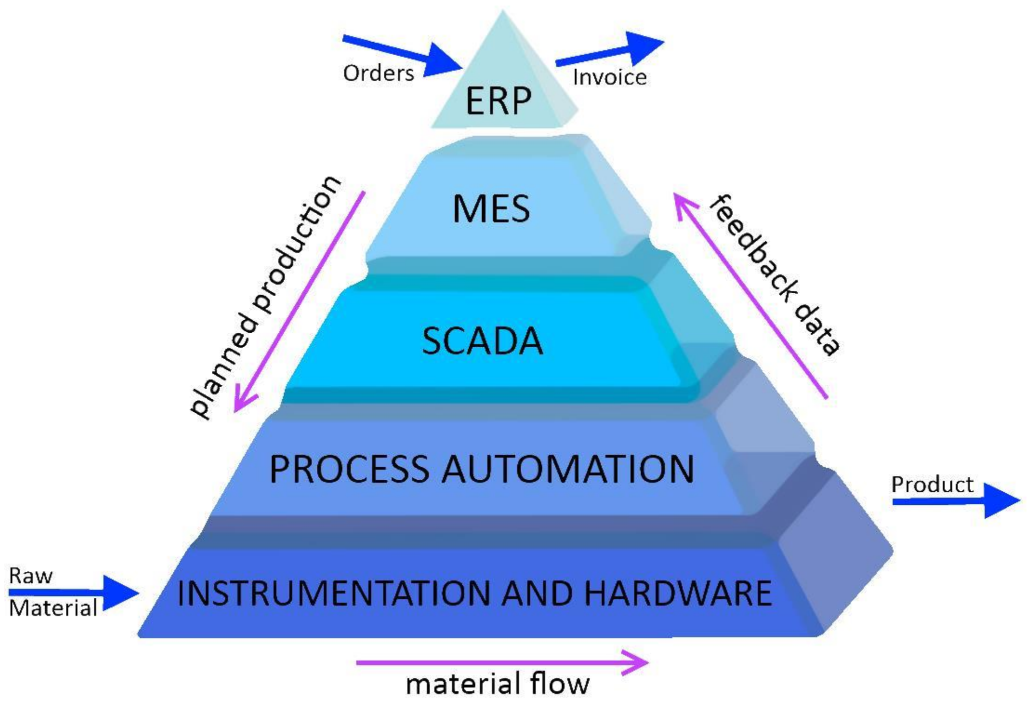


Figure 9*. Layers of the pyramid of automation [18]*

In the pyramid of automation, the physical process takes place in the lowest layers [19]. This is where instrumentation, like sensors and actuators, are found. The process automation functions are carried out by CNC (Computerized Numerical Control) or PLC (Programmable Logic Control) machines, while in the DCS layer, supervision of the process is done by a SCADA (Supervisory Control And Data Acquisition) system. The top layer is where all the planning and scheduling is implemented in an ERP system.

The objective of MES is to provide production scheduling by reporting product and material availability, schedule orders and closely monitor the shop floor activities [20]. MES ensures that the plans coming from the top layer are implemented on the shop floor. The real time data gathered from the shop floor is reported back to the ERP.

The creation of the ERP-MES-DCS model led to the standardization of the MES functionalities, which were developed by Manufacturing Enterprise Solutions Association (MESA) in the 1990s. In [21], MESA[1] analysed the performance of various manufacturers who used MES systems. Some of the benefits that MES introduced were reduced manufacturing time cycle, reduced data entry time, reduced WIP (Work In Progress) and improvement in product quality. MESA also defined 11 functions of MES which provide the core information for the management of a plant, which are described in Table 1. The

[1] http://www.mesa.org/en/index.asp

International Society of Automation (ISA[2]) used these recommendations to extend the guidelines for batch processes (S88 Standard) and for general processes (SP95) [22].

***Table 1**. Definition of MES functions by MESA*

1	**Resource Allocation and Status**	Manages production resources in the plant, like machines, materials, tools, labor skills and other entities.
2	**Operations/Details Scheduling**	Provides sequencing and timing of production operations to minimize the setup time.
3	**Dispatching Production Units**	Manages the flow of production units in form of jobs, orders, batches, lots and work orders. The information is presented in the sequence in which it needs to be done and the changes in real time as they happen in the shop floor.
4	**Document Control**	Provides the control records or forms that are necessary for a smooth production. This functionality makes information like work instructions, drawings, recipes, diagrams and charts available to operators.
5	**Data Collection/Acquisition**	Gathers and manages the data about production from the shop floor.
6	**Labor Management**	Provides information of personnel, tracks and directs the use of operators during a shift, and interacts with resource allocation to determine optimal assignments.
7	**Quality Management**	Provides measurements collected from manufacturing to ensure proper product quality control and to identify problems that require attention.
8	**Process Management**	Monitors production and either automatically corrects or provides support to operators for

[2] https://www.isa.org/

		correcting in-process activities. It directs the workflow of the planned and actual production activities.
9	**Maintenance Management**	Tracks and directs the maintenance of equipment and tools to ensure their availability for manufacturing. It is used to schedule periodic or preventive maintenance and solve immediate problems.
10	**Product Tracking and Genealogy**	Provides visibility of where work is at all times. It creates a record with the full history of the product.
11	**Performance Analysis**	Provides reports of operation results and a comparison to past history and the expected business result. Performance results include measurements like resource allocation, product unit time cycle, productivity and schedule information.

2.3 Web Services

During the last few years, there has been a trend of moving towards more intelligent and distributed systems. A large amount of costs and time in an automated system goes towards reconfiguration and re-implementation of the control when a new machine is introduced to the system [23]. Manufacturing automation systems are challenged to become more flexible and reconfigurable. One technology that can be used to solve this problem is Service-Oriented Architecture (SOA) [24].

SOA is an architectural style that is used to provide services to software applications within a network, such as the web. This allows for device interoperability in complex automation systems where changes are frequent [25].

The implementation of intelligent robotic devices in automation systems, such as robots or controllers, demands an extensive and elaborated implementation into the factory shop floor [26]. Along with programming the tasks executed by each of the devices, the communication with neighbouring devices must also be considered. As concluded in [25], SOA can be extended to low level devices in the manufacturing system by means of web services. This implementation allows the use of a uniform technology base throughout the entire automation system, making it more agile, reconfigurable and flexible.

The W3C defines web services as a software system that allows interoperable machine-to-machine communication over a network [27]. Web services provide a structure and protocol for agents to communicate by sending and receiving messages. The service is a resource that is characterised by a set of functionalities that are described by means of Web Service Description Language (WSDL).

Web services enable the creation and deployment of components into a distributed environment. It allows those components to communicate through a network in order to share functionalities or information. [28] provides an integration framework to integrate a web-service based MES system. In that paper, the authors propose the integration of the business, service and resource layers of a MES system are communicated by means of web services, as can be seen in Figure 10.

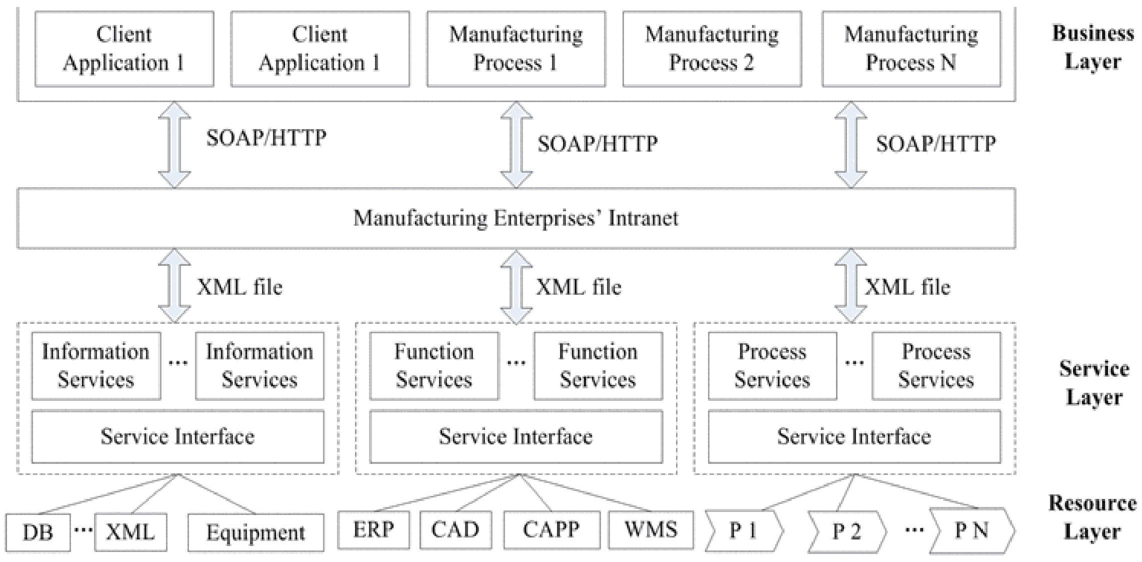


***Figure 10**. Integration framework for web services-based MES system.*

MES has a set of functionalities that are used to support and track the main production functionalities, like process management, execution management or performance analysis, among others. These functionalities can be implemented as a collection of web services in order to control the data flow in the MES system [29]. When information or a service is needed, it can be invoked by an application client through web services. At a high level, web services can be used to request data on the status of a work order before scheduling a new job. On a lower level, web services can also be included to request a specific type of service or information from the devices on the shop floor, such as the state of a machine or a sensor.

W3C also defines Representational State Transfer (REST) as an architectural style for reliable and loosely coupled web applications [30]. REST is based on HTTP methods as a means to communicate. REST is used to develop scalable, lightweight and modifiable web services, known as RESTful Web Services. It uses HTTP requests to post data (Create, Update), read data (Read), or delete data (Delete), which constitute all four CRUD operations.

Web services also use SOAP (Simple Object Access Protocol) for sending XML-based data between applications. While REST is an architectural style, SOAP is a protocol. The messages are sent as an XML document via HTTP, and the document has a specific pattern.

SOAP requires a higher bandwidth than REST, since the amount of data that needs to be transferred in each message is higher. The data must be in XML format; on the other hand, in RESTful web services support different data formats like XML, JSON, HTML or plain text, so the size of the message can be considerably lower. It provides a ”stateless” communication, which means that the state of the conversation does not affect the meaning of the message.

2.4 Free shape paths

The problem of path and trajectory planning is a very relevant part of robotic implementation. The positioning of the robot’s end effector and the accuracy of the robot’s motion is critical to ensure the quality of the manufactured product, and to avoid collisions in the workspace. In operations where there is a high interaction between humans and robots, or between two robots, it is important to ensure the safety of all the entities in the system. Moreover, the velocity of the robot must be considered. The robot needs to have continuous velocity and acceleration throughout the process, since those values cannot be changed abruptly. For this reason, the trajectory of the machines must be planned and implemented carefully.

Most industrial robots are capable of performing point-to-point, linear and circular motions, as shown in Figure 11.

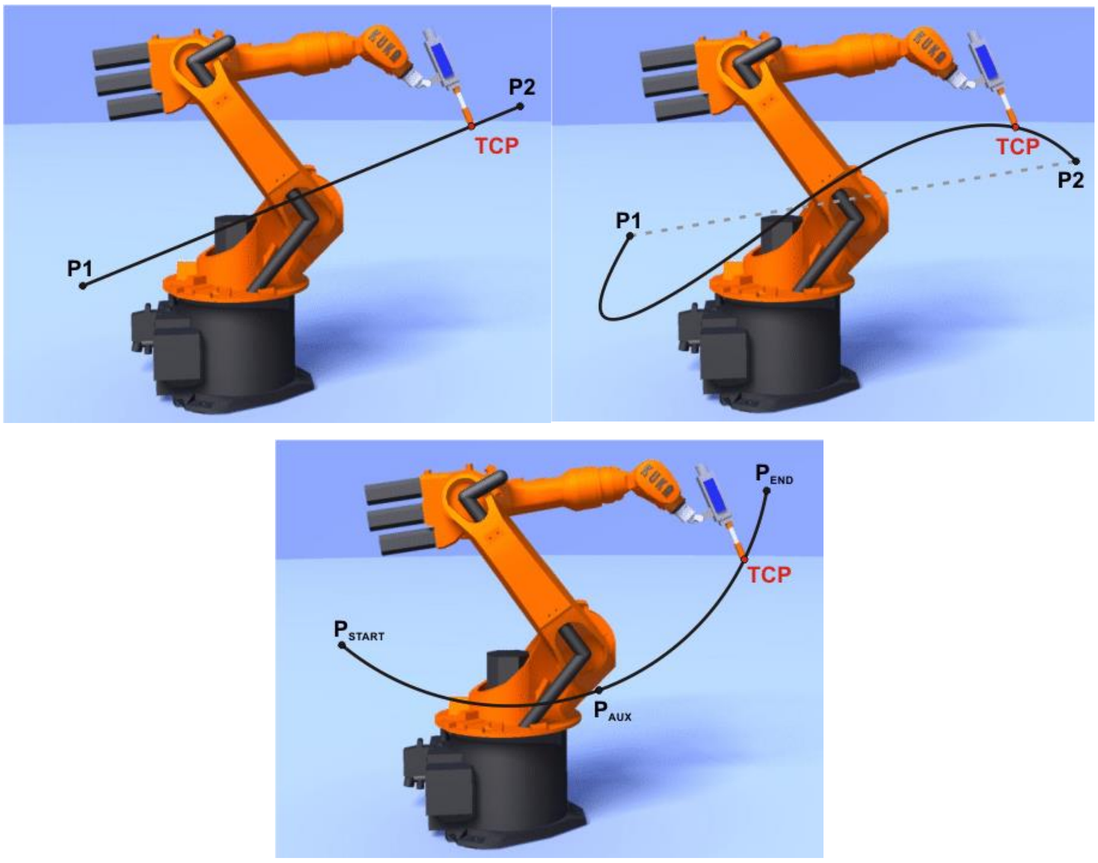


Figure 11. *Motion types in industrial anthropomorphic robots*

- Point-to-point motions: The robot moves the Tool Center Point (TCP) from one point to another in the fastest path available. This does not necessarily mean a linear path, which would be the shortest. 6-axis robots are able to perform curved paths faster than straight lines, since the movements along their axes are rotational. The trajectory of the TCP between the start and end points is not relevant.
- Linear motions: The robot guides the TCP from one point to another in a straight path.
- Circular motions: The TCP describes an arc between two points by specifying a start, auxiliary and end point.

Simple motions like the ones described above are enough to perform most tasks involved in an industrial process, such as pick and place operations in an assembly process. However, there are some applications in which the trajectory and orientation of the TCP is critical and complex. This is the case of robots performing welding, painting or fluid dispensing tasks. As a result, PTP motions are not suitable of these types of applications.

Complex trajectories can be divided into small circular and linear segments that are connected to one another. Nevertheless, when complex trajectories need to be implemented into a manipulator, concatenation of linear and circular motions does not necessarily provide a smooth path, which derives into a trajectory full of discontinuities and abrupt changes in the path's direction. Free shape paths can be implemented in a robot manipulator to smooth the end effector's trajectory.

In [31], the authors presented a method for curve reparameterization by using NURBS curves to build smooth trajectories and to reduce the jerking motions during the path. [32] presents an approach for welding complex joints, in which free shape paths are designed by robot path planning of the centroid path in a scaled Y-joint.

2.5 Bezier curves and B-splines

Bezier curves are an adequate mathematical tool to model smooth paths. They were named after French engineer and mathematician Pierre Bezier, who in the 1960s designed a method for describing curves mathematically while working for the car manufacturing company Renault [33]. Bezier curves are expressed in a polynomial form by using a series of control points and the Bernstein polynomial as a basis.

Bezier curves are widely spread in computer-aided design (CAD), computer graphics, model design and many other fields. Bezier curves have also been used in robotics to model smooth trajectories for both mobile robots and tools of fixed robots. Kolegain et al. [34] designed a methodology for off-line path programming for a friction stir welding application by approximating the welding path to a Bezier curve. Their experiments with Bezier curves proved to reduce tool deviation in comparison with paths created with other commercial softwares, as well as maintaining a constant downforce during welding. Liljebäck et al. [35] used Bezier curves to model the shape of a snake robot in order to create a control framework for the robot's locomotion.

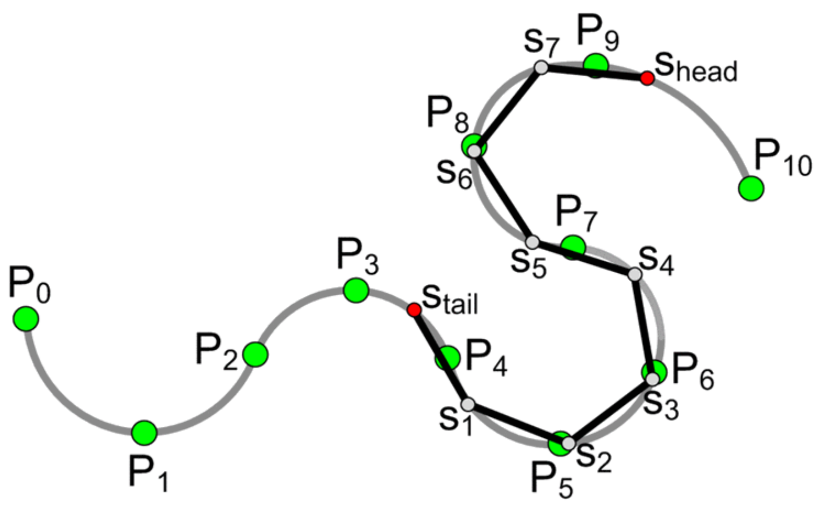


Figure 12. *Model of the motion pattern of a snake robot with Bezier curves [35]*

Bezier curves are easy to compute and evaluate, since they are expressed by a simple polynomial function. The order of the polynomial can vary, which in turn changes the number of control points or descriptors employed in the construction of the curve. By using third degree Bezier curves, or cubic Bezier curves, highly accurate and complex paths can be obtained.

A Bezier curve is defined by n+1 control points, with n being the order of the curve's polynomial function. Those points are located in a plane or space. The polynomial function that describes a third-degree Bezier curve is given as:

$$\mathrm{B}(t) = (1-t)^3\,\mathrm{P}_0 + 3t\,(1-t)^2\mathrm{P}_1 + 3t^2(1-t)\,\mathrm{P}_2 + t^3\mathrm{P}_3, \tag{1}$$

where:

P_0, P_1, P_2, P_3: control points of the Bezier curve.

t: function parameter that determines the distribution of the interpolation points.

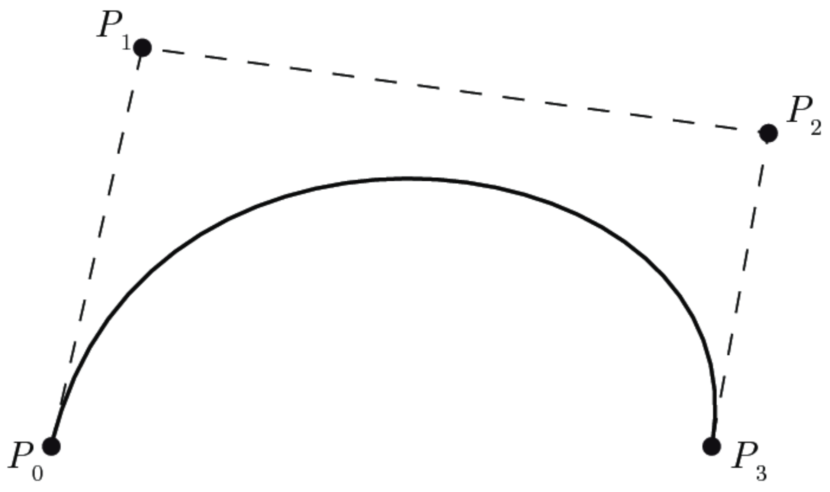


Figure 13. *Generic third order Bezier curve [36]*

The function parameter in (1) is *t*, and it comprises the values between 0 and 1. In a cubic Bezier curve there are 4 control points. The first and last control points define the beginning and the end of the curve. B(t) represents all points of the curve for the values of the parameter *t*. When parameter *t* is evaluated for different values between 0 and 1, (1) will result in a series of points that are located in the Bezier curve, as represented in Figure 14.

- When t=0, the point of the curve is the initial control point, P_0.
- When t=1, the point of the curve is the last control point, P_3.
- When $t \in (0,1)$, B is one of the points of the curve.

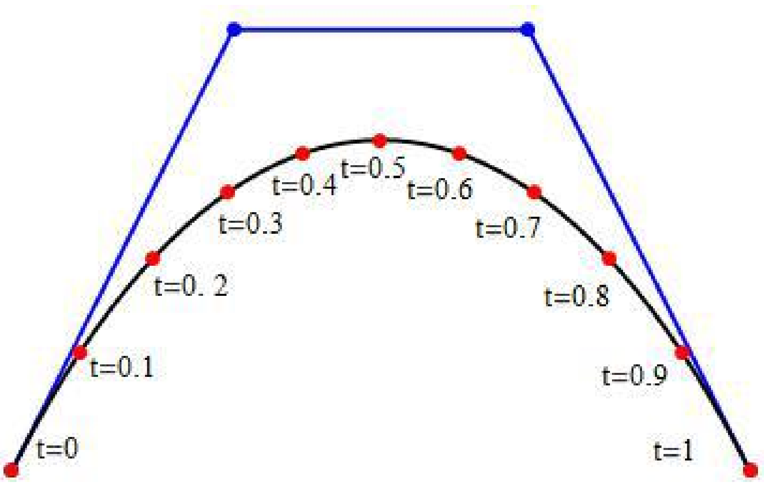


***Figure 14**. Evaluation of a Bezier curve with parameter t [37]*

A Bezier curve is contained within its control polygon, and the start and end of the curve is tangential to the start and end section of the control polygon. This property is known as the *convex hull property*.

By increasing the order of the polynomial and the number of control points, highly complex curves can be obtained. However, higher-order Bezier curves require an extensive computational cost for their evaluation and they are numerically unstable. The general n-order Bezier curve is defined by its control points P_i, where i=0,1,2,...,n [38], and it's expressed as:

$$B(t) = \sum_{i=0}^{n} b_{i,n}(t) P_i \tag{2}$$

As was previously stated, Bezier curves are based on Bernstein's polynomial [39]. $b_{i,n}(t)$ is what is known as the Bernstein polynomial, which is defined by:

$$b_{i,n}(t) = \binom{n}{i} t^i \, (1-t)^{n-i} \; , \tag{3}$$

where $\binom{n}{i}$ is the Binomial coefficient.

In a cubic Bezier curve, four Bernstein polynomials are used:

$$b_{0,3}(t) = \binom{3}{0} t^0 (1-t)^{3-0} = (1-t)^3 \tag{4}$$

$$b_{1,3}(t) = \binom{3}{0} t^1 (1-t)^{3-1} = 3t\,(1-t)^2 \tag{5}$$

$$b_{2,3}(t) = \binom{3}{0} t^2 (1-t)^{3-2} = 3t^2\,(1-t) \tag{6}$$

$$b_{3,3}(t) = \binom{3}{0} t^3 (1-t)^{3-3} = t^3 \tag{7}$$

The polynomial functions (4)-(7) are represented in Figure 15. The horizontal axis in the graph represents *t* and the vertical axis represents the values of the Bernstein polynomials; values of both axis are comprised between 0 and 1. In Bezier curves, as stated in (2), for each of the values of *t,* the Bernstein polynomials are multiplied by their respective control points.

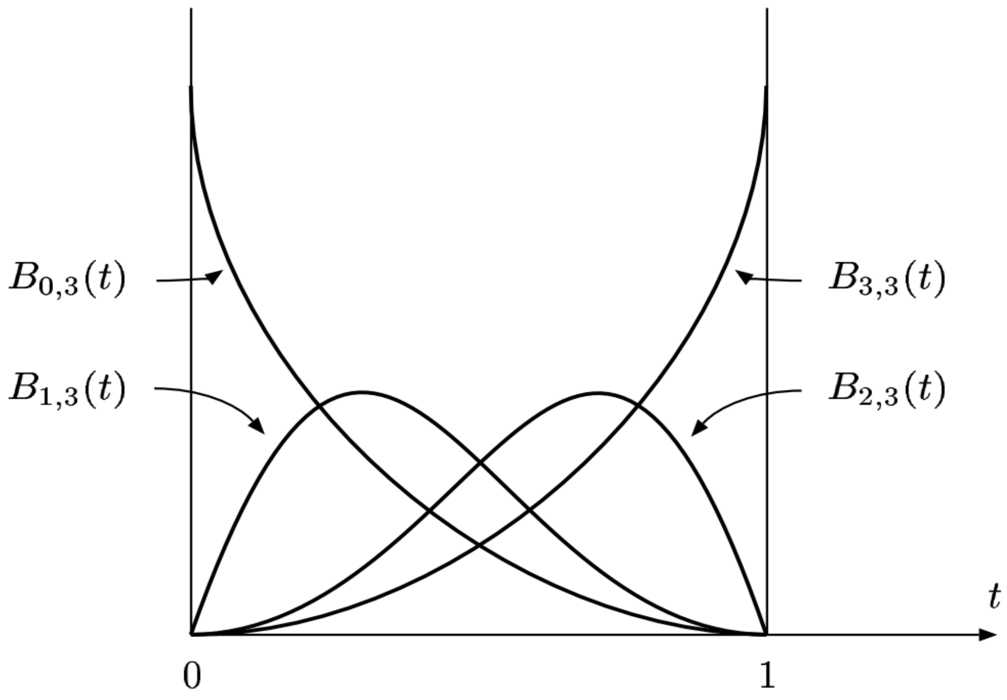


***Figure 15**. Graphical representation of Bernstein polynomials [40]*

By looking at the graph, it can be determined that as the value of *t* increases from 0, the effect of the first control point decreases, and the value of the second control point increases. When *t* approaches the value of 1, the effect of the last control point is the highest, while the effect of the previous control points is reduced.

When the desired paths are complex and the trajectory cannot be represented by a cubic Bezier curve, B-splines are employed. B-splines are generalizations of Bezier curves, where piecewise polynomials are joined together at the ends so that the last point of one curve and the starting point of the next one are the same. While Bezier curves require a high amount of control points for complex curves, thus requiring longer computation times, B-splines approximate those curves with lower degree polynomials [41].

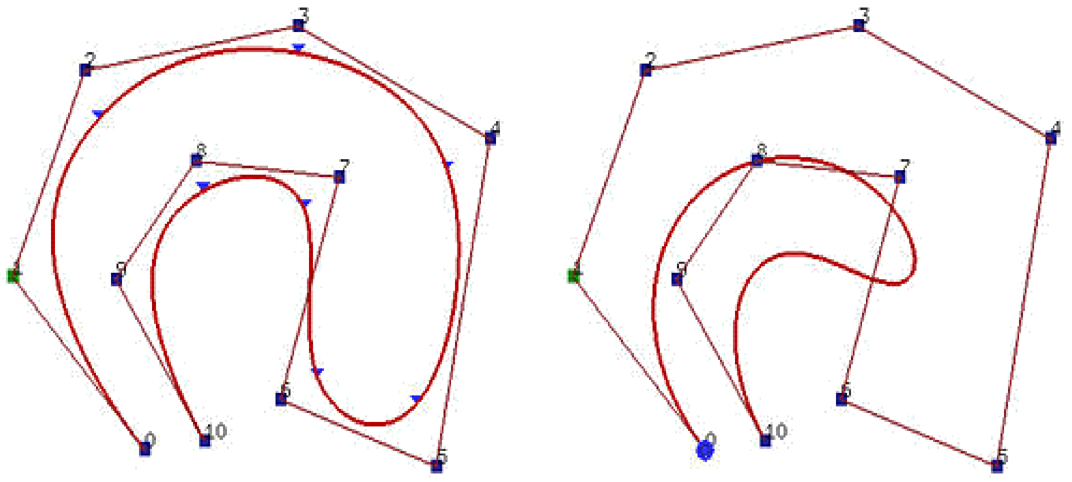


***Figure 16**. Left: Cubic B-spline curve. Right: 10-degree Bezier curve. Both have 11 control points located in the same positions [42]*

Figure 16 shows a comparison of a B-spline curve and a Bezier curve that are defined by the same set of control points. Since there are 11 control points, the degree of the Bezier curve is 10, while the spline's degree is 3. The degree of a spline is not directly correlated to the number of control points. As the degree of the spline decreases, the spline curve will follow the control points more closely. The left curve in Figure 17 has degree 7, the middle curve has degree 5 and the last curve has degree 3. As the property states, the last one, the one with the lowest order, is the one that is closest to the control polyline.

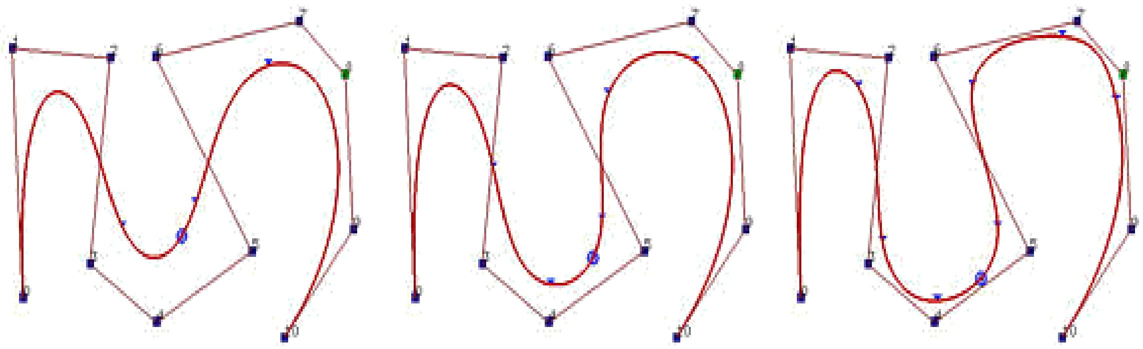


***Figure 17**. B-spline curves with varying orders and the same set of control points [42]*

B-splines maintain a continuous curvature between two consecutive Bezier curves, so the tangent of both curves in the joint point remains the same. Other interesting property of B-splines is the local control property, which is shown in Figure 18. Changing the location of one of the control points will only affect the interval of the curve in which it is located, while the shape of the rest of the curve is not changed.

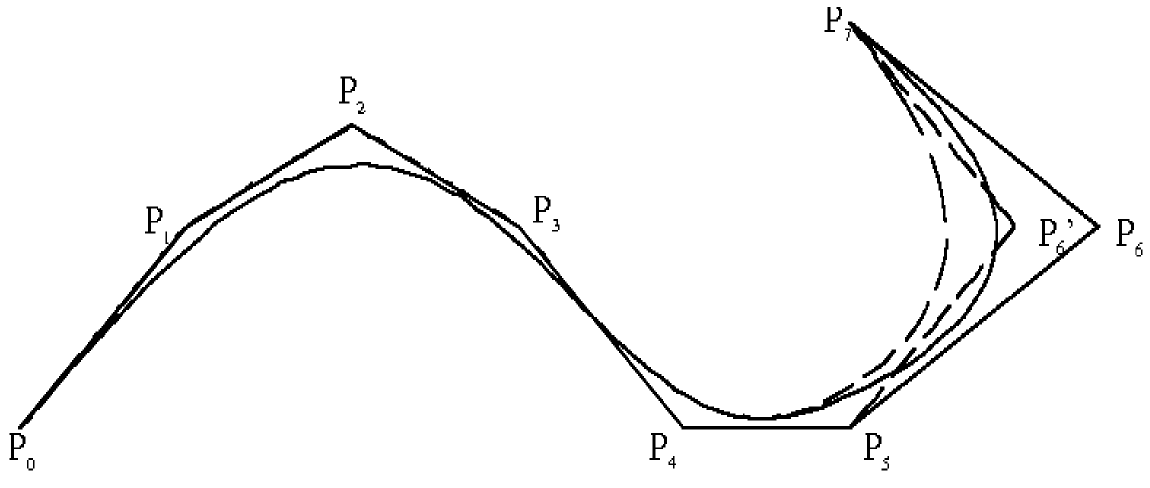


***Figure 18**. Local control property of a B-spline curve [43]*

In Figure 18, the position of P_6 is changed. This causes the shape of the curve to change locally around that point. However, the change does not propagate through the rest of the curve, so it maintains its original shape.

Pan et al. [44] employed cubic B-splines to create a path smoothing algorithm for collision avoidance. In [45], Elbanhawi et al. used B-splines to generate the path of a wheeled robot where a continuous curvature was followed.

Some arbitrary shapes do not have constant curvature along their path, thus not meeting the requirements for B-splines. In those cases, composite Bezier curves, or Bezier splines, are used, which are Bezier curves concatenated together at their ends.

As was mentioned in section 2.4, industrial manipulators support linear and circular interpolations. However, they do not support Bezier interpolations. Bezier curves can be approximated into linear and circular segments, which can in turn be implemented into an industrial robot. This way, free shape paths can be easily obtained in an industrial process. Generally, Bezier curves are evaluated by using what is called the De Casteljau algorithm, instead of computing the Bernstein polynomials [46].

2.6 Implementation of Bezier curves: De Casteljau algorithm

As mentioned in section 2.5, industrial robots are capable of performing circular and linear interpolations. Due to this limitation, in order to implement Bezier curves into a robot, they need to be approximated to one of the following:

- A composition of both circular and linear segments joined together.
- A series of linear segment.

In order to do this, there needs to be a method to approximate these segments to a Bezier curve. This method is named the De Casteljau algorithm.

The De Casteljau algorithm is a recursive method that is used to evaluate Bezier curves. This algorithm is used to find the points that comprise the Bezier curve. Through the De Casteljau algorithm, Bezier curves are easily implemented into an industrial robot.

The algorithm was named after Paul de Casteljau, a French mathematician who invented the algorithm with the use of the Bernstein polynomials in the 1960s for the French car company Citroën [47]. Both Bezier and De Casteljau developed their work at the same time while working in competing companies.

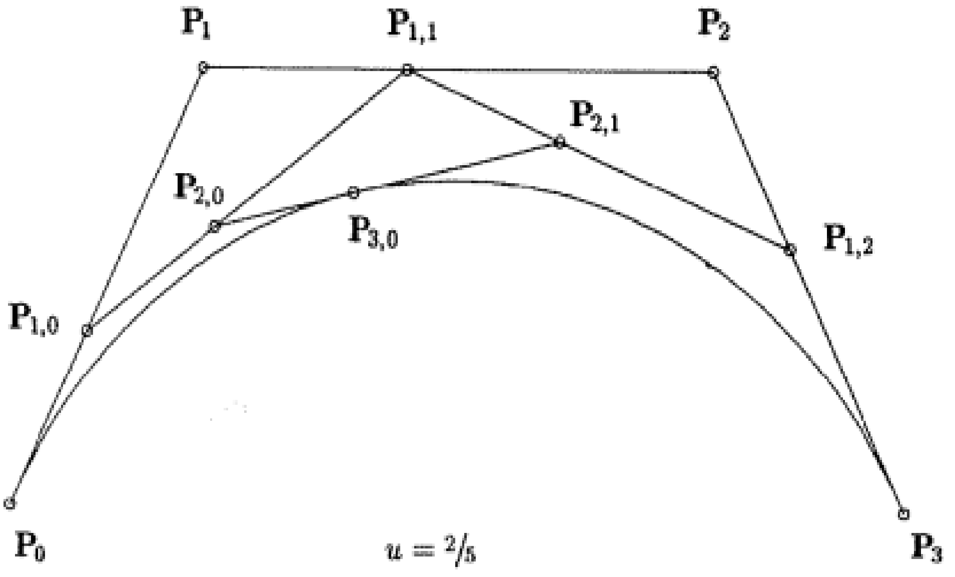


***Figure 19**. De Casteljau algorithm for a given value of parameter t.*

The De Casteljau algorithm for a cubic Bezier curve consists of computing 6 interpolations in order to obtain one of the points of the Bezier curve. Each point of the curve is related to one value of parameter *t* between 0 and 1. Firstly, each of the segments of the Bezier polygon are interpolated at a given parameter t. The resulting points, named $P_{1,0}$, $P_{1,1}$ and $P_{1,2}$ in Figure 19, are also interpolated, and points $P_{2,0}$ and $P_{2,1}$ are obtained. Lastly, those points are interpolated for the same value of *t*, which results in one single point, $P_{3,0}$. That point is the one located in the Bezier curve. The pseudocode for implementing Bezier curves by using the De Casteljau algorithm is shown in Figure 20.

```
for t=0 to I do
```

```
        P10=interPol(PO,P1,t);
        P12=interPol(P3,P2,t);
        P11=interPol(P1,P2,t);
        P20=interPol(P10,P11,t);
        P21=interPol(P11,P12,t);
        P30=interPol(P20,P21,t);
    end
```

***Figure 20**. Pseudocode of the De Casteljau algorithm for a cubic Bezier curve*

The algorithm is computed by incrementing the values of parameter *t* between 0 and 1. The size of the step between two consecutive values of *t* in which the algorithm is computed determines how precise the algorithm will be. When the step size is small, the accuracy of the resulting curve will be high, and the approximation to the original curve will be closer.

This, however, also comes with a disadvantage: as the number of points that are evaluated on the curve increases, so will the number of times the algorithm is computed, thus requiring a higher computational time and effort.

In an industrial robot, the motions of the TCP from one point of the Bezier curve to the next can be done by performing either a circular or a linear motion. A study was made in [48] that compared both arc and linear approximations, highlighting the advantages and disadvantages of each approximation.

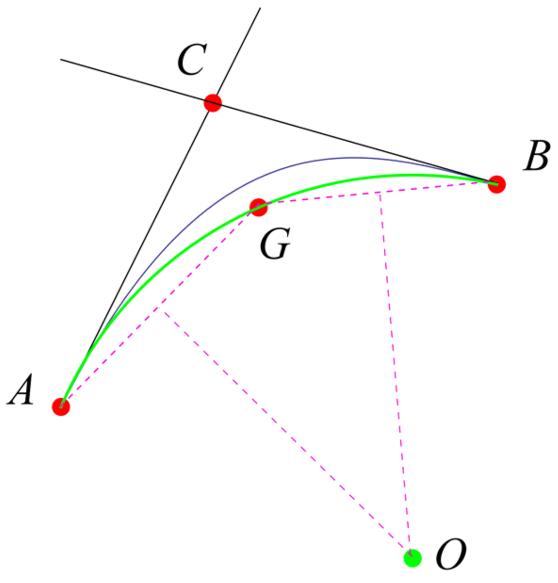


***Figure 21**. Arc approximation of a curve segment [49]*

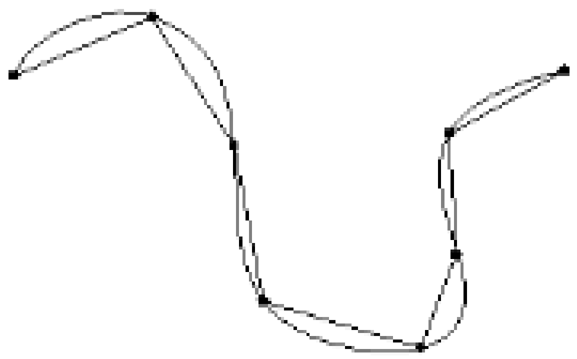


***Figure 22**. Linear approximation of a curve segment [50]*

In [49], the authors present a method to model Bezier segments by using circular arc approximations. Using arc segments to approximate a curve rather than straight lines provides the following advantages [51]:

- It creates smoother paths, since it is easier to represent the curvature of the path rather than using linear shapes. Linear movements cause sudden changes in the direction of the TCP, which can create problems if accelerations are not carefully managed.
- Arc approximations require a lower number of segments to achieve user-defined tolerance, while linear approximations need a higher number of interpolations. The amount of Bezier points that need to be transferred to the robot controller with linear approximations is higher than with arc approximations.

However, there are some major limitations for arc approximations that makes linear segments more adequate to approximate Bezier curves [48]. For starters, coding the algorithm for arc segments is more difficult, due to the higher number of parameters that are needed to implement it.

Furthermore, there are some special cases for which circular approximation leads to numerical instability. The algorithm fails when one of the following cases take place:

- The length of the Bezier curve is very small.
- The control points of Bezier curves are coincident.
- Bezier curves with high curvature require more arc segments than linear segments to achieve the same level of tolerance [51].

Table 2 shows a comparison between using linear and circular segments for approximating Bezier curves. Notice that, while arc segments provide some important advantages, its use in CNC industry is lower than that of linear approximation, due to its complexity and instability.

***Table 2**. Differences between implementing arc and linear segments for approximating Bezier curves.*

	Arc	**Linear**
Number of segments	Low	High (depending on accuracy)
Amount of data transferred	Low	High
Smoothness of the path	Good	Bad: Abrupt movements if accelerations are not constrained
Approximation algorithm complexity	High	Low
Robustness of the algorithm	Bad: numerically unstable in some cases	Good: numerically stable for all Bezier curves
Use in CNC industry	Low	High

3. PROPOSED INTEGRATION PATTERS

This chapter explains the proposal for developing a flexible robotic cell in which tasks can be assigned according to the necessities of the manufacturing line, where the service requests from the line's orchestrator will be done via web services. This proposal illustrates a general structure for the cell and explains how the industrial robot that is implemented will be able to operate regardless of its topology or manufacturer. This section also describes an approach for modelling free shape paths and implementing them on an industrial robot.

This chapter is divided into 5 sections; the first analyses the structure of the robot cell that is implemented in this thesis. The second section defines the proposal to establish a TCP/IP communication between the Remote Terminal Unit (RTU) and the robot controller. The third section explains the choice of robot topology that is installed in the cell by comparing some significant characteristics. The fourth section describes how the Bezier curves are implemented in an industrial controller, and which approximation and operation characteristics are proposed for this thesis. The fifth section presents a brief explanation on the safety configuration of the cell.

3.1 Structure of the cell

The main elements that compose the robotic cell are the robot arm, the conveyor and two RTUs. Both the robot arm and the conveyor are controlled by an RTU each. The RTU is the one that sends the orders to the robot controller and the conveyor's actuators, as well as receive data about the status of the robot from the controller and other data from the conveyor's sensors.

The RTUs on the cell are controlled by an orchestrator. The orchestrator is a higher-level control that falls on the supervisory level on the pyramid of automation. It is responsible for organizing the tasks for all the cells in the manufacturing line. The orchestrator requests services from the robot and the conveyor. Those requests are sent to the RTUs by using RESTful web services. The RTUs forward the request to the robot or conveyor. The proposed structure of the manufacturing line can be seen in Figure 23.

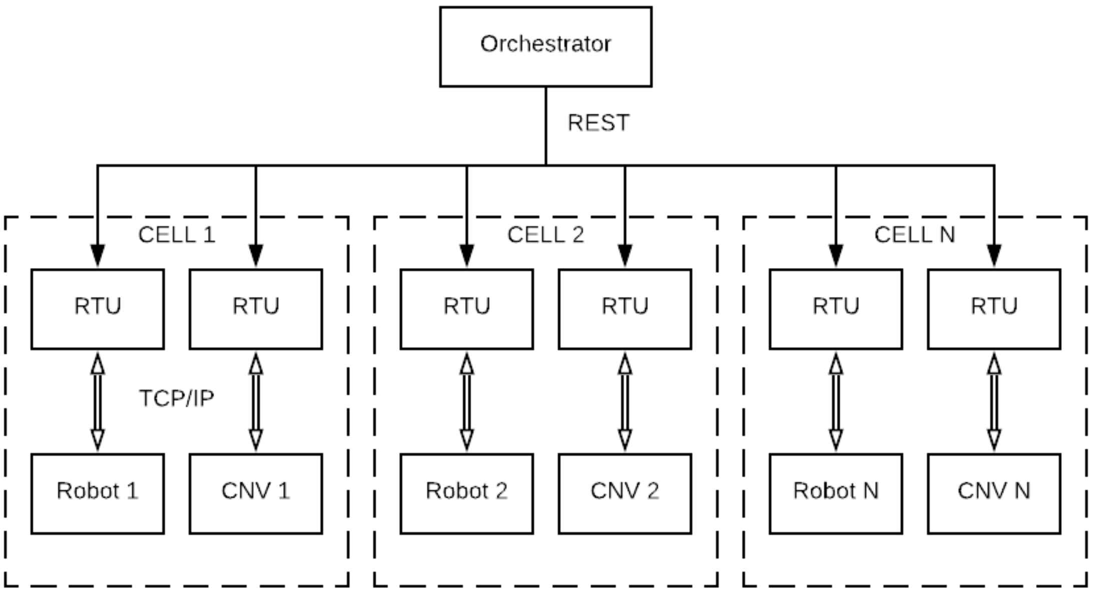


***Figure 23**. Structure of a manufacturing line with N robotic cells*

This approach proposes the use of RESTful web services over SOAP. REST requires a lower bandwidth and shorter messages for the communication, and it permits the use of more data formats than SOAP, which can only work with XML. REST is also stateless, which means that it does not maintain data from one request to another. The proposed implementation does not require information flow from one request to another, so there is no need for the stateful operations that SOAP offers. A comparison between both types of web services is depicted in Table 3.

***Table 3**. Comparison of SOAP and REST characteristics*

SOAP	REST
Protocol	Architectural style
Stateful	Stateless
Requires high bandwidth	Requires low bandwidth
Only works with XML format	Works with XML, HTML, Plain text, JSON…
Long messages	Short messages

This thesis proposes an approach to integrate the robot in the cell, as well as the RTU assigned to the robot. The approach to integrate the conveyor to the cell and to implement an orchestrator that unifies and organizes the operation of the entire assembly line have been previously developed.

Within the cell, the conveyor transports pallets in and out of the cell. The RTU controls the motor of the conveyor, as well as gathering data and activating a series of sensors and actuators located on the conveyor across the cell. The orchestrator requests to the RTU to transfer the pallet from one zone of the conveyor to the other, or to transfer it out of the cell. The RTU can sense the presence of the pallets on each zone of the conveyor by using the presence sensors located in each zone and stop them at their intended destination zone by activating the stopper actuators. By using both the sensors and actuators the RTU is able to fulfil the requests of the orchestrator.

Once there is a pallet waiting in position on the conveyor, the higher-level orchestrator sends a message to the robot’s RTU with an assigned task for the robot to perform. The RTU receives the order and forwards it to the robot controller. The communication between the RTU and the robot is done on top of TCP/IP. Messages are sent as raw binary strings through an Ethernet connection. The robot controller reads the order and proceeds with the task that was requested.

This approach for the integration of the cell and the interactions between the different objects of the system provides flexibility to the process. The orchestrator is responsible for the supervision of the line and the requests to each cell, and a supervisory level composed of the RTUs, control each cell individually. The operations of the robots are flexible, since they answer to specific service requests instead of performing the same operations at all times.

3.2 TCP/IP Communication between RTU and Robot

The previous section mentions that the communication between the Remote Terminal Unit and the robot is done on top of TCP/IP (Transmission Control Protocol/Internet Protocol). This section will expand on that by explaining the details behind the connection and the process flow during the communication.

There needs to be an open connection between the robot and the RTU in order to ensure that the robot receives the request to perform its operations. The robot acts as a service provider while the RTU is the service requester. The communication between the two is done on top of TCP/IP. Connection between the devices occurs when both devices are in the same Ethernet network.

TCP/IP uses a client/server model of communication. The robot controller acts as a server and the RTU as a client. TCP/IP is characterised by a low data overhead, and the communication is bidirectional, simple and fast. To open a connection, the client needs to know the server's IP address, as well as a port number that defines a unique communication channel between the two devices. Once the connection is open, it will remain open until either the server or the client closes it.

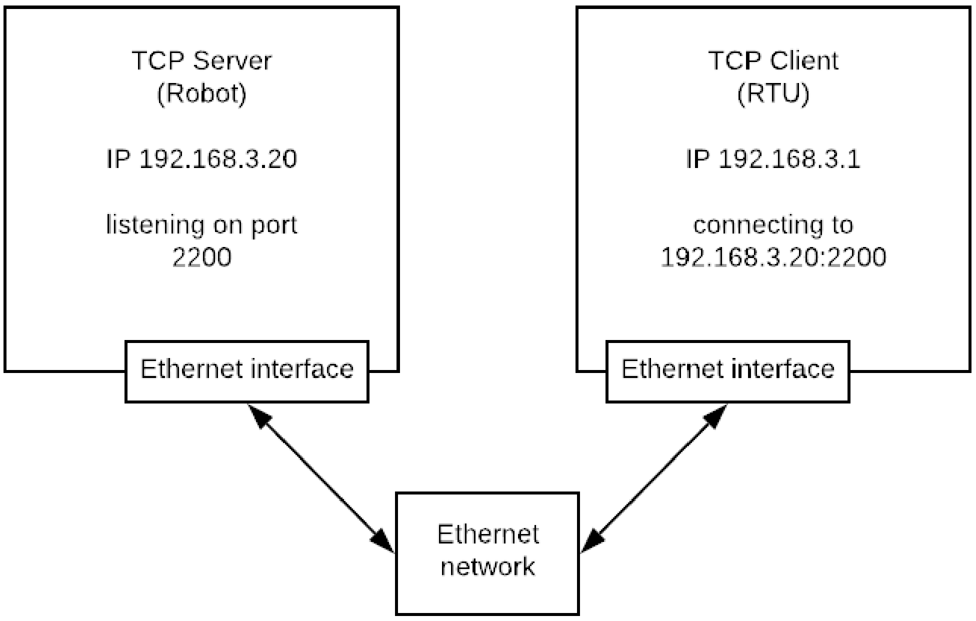


***Figure 24**. Connection example for TCP/IP client/server communication.*

For the implementation on the robotic cell, the proposed process to establish a connection and send messages is as follows: the robot, as the TCP/IP server, is listening for a connection with a client. When the orchestrator makes a request to the RTU, this last one opens a connection as a client. Once the server accepts the connection, the exchange of messages can begin. As TCP/IP is a bidirectional communication model, messages can be sent from both the server and the client.

First, the client sends the service request to the server. The server processes the request and either accepts or denies it. If the service request is accepted, the robot calls the necessary subprograms to perform the operation. Once the operation is finished, the robot notifies the RTU, and the RTU forwards the message to the orchestrator. This process is represented in a sequence diagram in Figure 25.

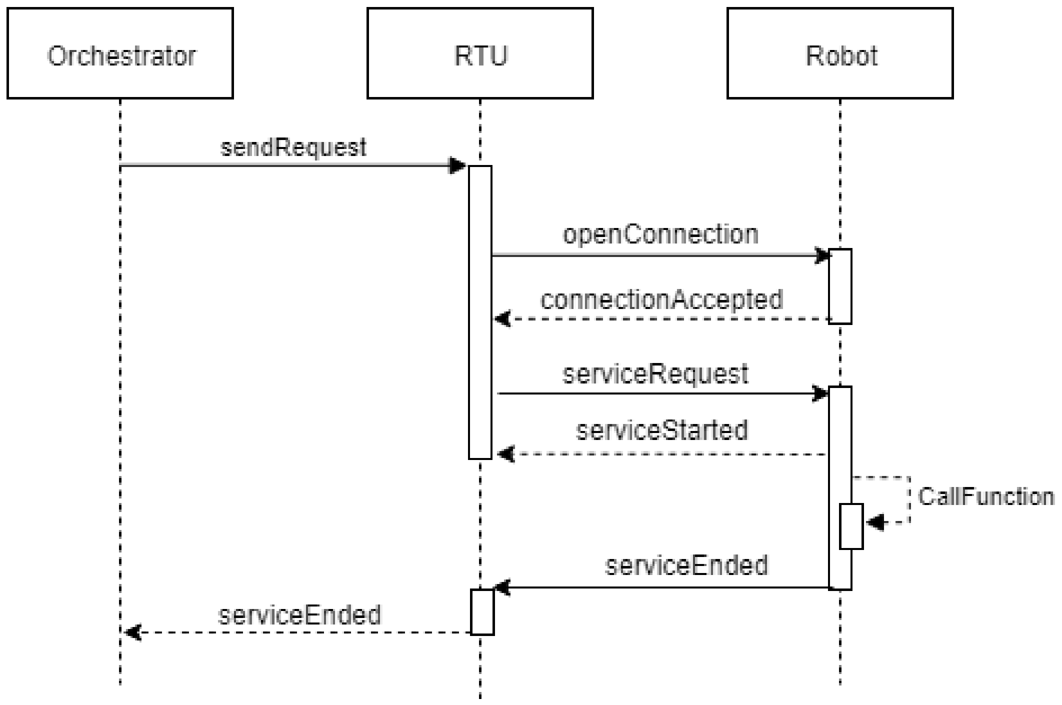


***Figure 25**. Sequence diagram for communication between entities on the cell.*

The data is sent as plain text through the TCP/IP connection. The communication between the RTU and the controller uses low level Application Program Interfaces (APIs), as the client only sends messages with a single word that describes the exact operation that is requested. The APIs that are used in the implementation of this thesis are explained in chapter 5.

3.3 Robot topology selection

In section 2.1, 6 main robot topologies were presented. In order to choose the suitable topology for the cell, the characteristics and benefits of each type must be pondered. Some of the main factors that are considered when deciding on a robot topology are the application type, the payload to be handled, the workspace restrictions and maximum reach, the precision and the speed. These factors are explained below, as well as their effect in the current proposal.

- Application type.

  The task that is going to be performed with the robot is the main factor in choosing the right industrial robot for the process. The application that is examined in this thesis consists of some simple manipulations in which small objects are picked and placed. Furthermore, the TCP performs free shape path over a horizontal plane located on top of a pallet. Most topologies can be used for performing pick and place operations; however, free shape paths are not easy to perform for all of them.

- Payload.

  The load capacity of the robot must be higher than the total weight of the payload. This includes both the piece that is picked and the tool attached to the end of the robot. Normally, anthropomorphic and SCARA robots don't have very high payloads, while other topologies like Cartesian robots can pick and place payloads in the range of 100 kg easily. In this implementation, both the pieces that will be picked by the robot and the gripper used to pick them are light, so the load capacity of the robot is not a defining factor.

- Number of axes.

  The number of axes of a robot is directly related to its degrees of freedom. SCARA robots have four axes, which allows them to move around the XY plane and along the Z axis, as well as rotate around the Z axis. They can't, however, rotate around the X and Y axes. 6-axis anthropomorphic robots, on the other hand, can rotate around those axes. For applications in which the robot needs to work in a small workspace and twist around to reach the targets, 6-axis robots are recommended.

- Workspace and reach.

  Another factor in deciding which industrial robot to use is the volume of the workspace and the maximum reach of the robot. The maximum horizontal reach of the robot is measured from the centre of the robot to the furthest point that the TCP can reach. The vertical reach of the robot is calculated from the lowest point the robot can reach to the highest.

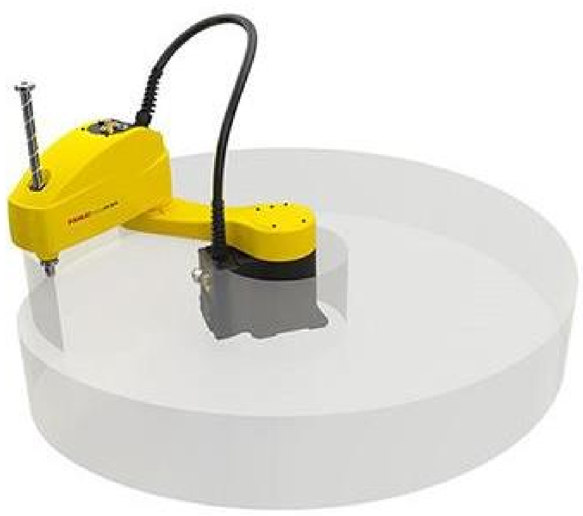


***Figure 26**. Work envelope of a SCARA robot (from Fanuc).*

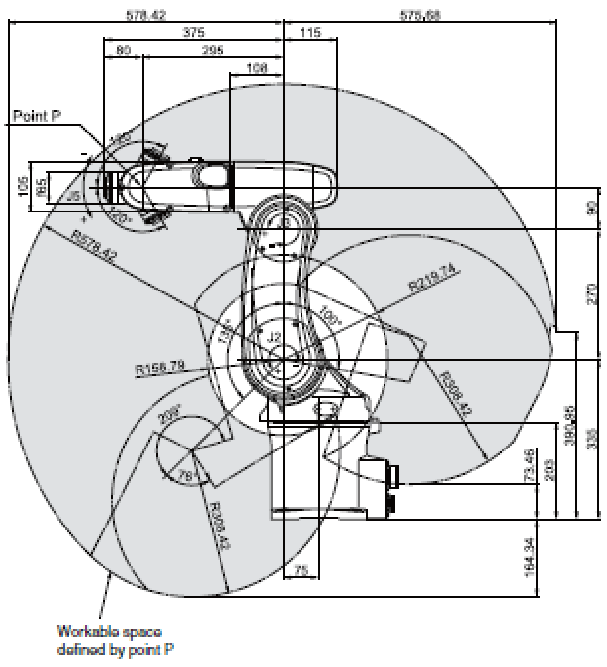


***Figure 27**. Work envelope of an articulated 6-axis robot (Viper 650 form Omron)*

- Positioning accuracy.

  There are some applications that require very precise positioning and repeatability during the process. For high precision applications, Cartesian robots are widely used. Articulated and SCARA robots don't offer precisions as high as Cartesian ones, due to arm deflection.

- Speed.

  Speed and acceleration are also taken into consideration at the time of choosing an industrial robot. The speed requirements depend on the time cycle that is needed for the operation. The speed is not a deciding factor for the implementation of this thesis.

In result of these specifications, the best fits for the application in question are either SCARA robots or anthropomorphic 6-axis robots. Both topologies are able to fulfil the necessary tasks with sufficient accuracy. However, the benefit of anthropomorphic robots over SCARA robots is the amount of degrees of freedom they possess, which means that they are able to reach the target points with the TCP in varying orientations. This extends the number of applications for which they can be implemented. Although the current application of the robot can be done with just 4 degrees of freedom, having extra degrees provides the robotic cell with more flexibility in case the tasks required change in the future.

For this reason, the implementation of the robotic cell will be done with an anthropomorphic robot. As mentioned previously, anthropomorphic robots are used for several different applications. Their work envelope is large, and the disposition if their axes allows the tool to reach a given target with multiple orientations due to their degrees of freedom. Although not as fast and accurate as other robot topologies, the speed and precision of articulated robots have increased, and their price has decreased, making them more affordable. Since the payload and size required are not big, a small 6-axis robot suffices. The specific robot arm and controller that was chosen for the implementation will be described in Chapter 4.

3.4 Bezier curves in industrial robots

In chapter 2 Bezier curves were introduced as a way to create smooth path models. Smooth path creation is important in robotic applications where the path followed by the TCP is critical. As industrial robots are not able to perform Bezier interpolations, a different method needs to be applied to implement these curves in the robot. The De Casteljau algorithm is a tool that allows the implementation of Bezier curves into an industrial controller.

This thesis proposes the use of the De Casteljau algorithm in order to perform free shape paths. The algorithm is implemented within the robot controller. The code for implementing this algorithm is simple and consist on performing a series of repetitive interpolations. In the case of cubic Bezier curves, the number of interpolations needed for one point in the curve is 6. Third order Bezier are proposed for this implementation, since they can perform complex curves without requiring extensive computational cost.

If the paths to be modelled are too complex to evaluate using cubic Bezier curves, several Bezier segments should be concatenated instead.

The approximation to Bezier curves with De Casteljau can be made by performing either linear or arc segments. This refers to the motions performed in between two consecutive points of the curve calculated by the De Casteljau algorithm.

Section 2.6 presented a comparison between linear and arc approximations. Due to its simplicity and its flexibility to be applied to any Bezier curve, linear approximations are the chosen alternative for this implementation. Approximation accuracy can be improved by decreasing the step size of parameter *t* in the De Casteljau algorithm. The step size must be decided for each specific application, depending on the tolerance requirements of the operation and the time requirements of production as well as the length and complexity of the Bezier curve.

The benefit of the De Casteljau algorithm's simplicity lays in the fact that it can be applied to any robot controller that is able to perform linear interpolations. The algorithm for implementing Bezier curves in an industrial robot controller is represented in Figure 28. The robot begins the curve by approximating the path's plane perpendicularly with a linear motion. The first position of the TCP on the path's plane matches the first Bezier control point. The interpolation begins with t=0. Once the robot has performed the interpolations and obtained the next point where the TCP must move, a linear motion is executed towards that position.

If parameter *t* is not 1, meaning that the curve has not reached the last control point yet, the parameter is incremented and the interpolations are performed again for the new value of *t*. Once the last control point has been reached, the robot checks if there's another Bezier segment to evaluate. If there is, the evaluation is repeated with the new Bezier points. To ensure continuity in the path, the last point of the previous curve must be the first point of the new one. When all segments are finished, the TCP moves away from the path's plane.

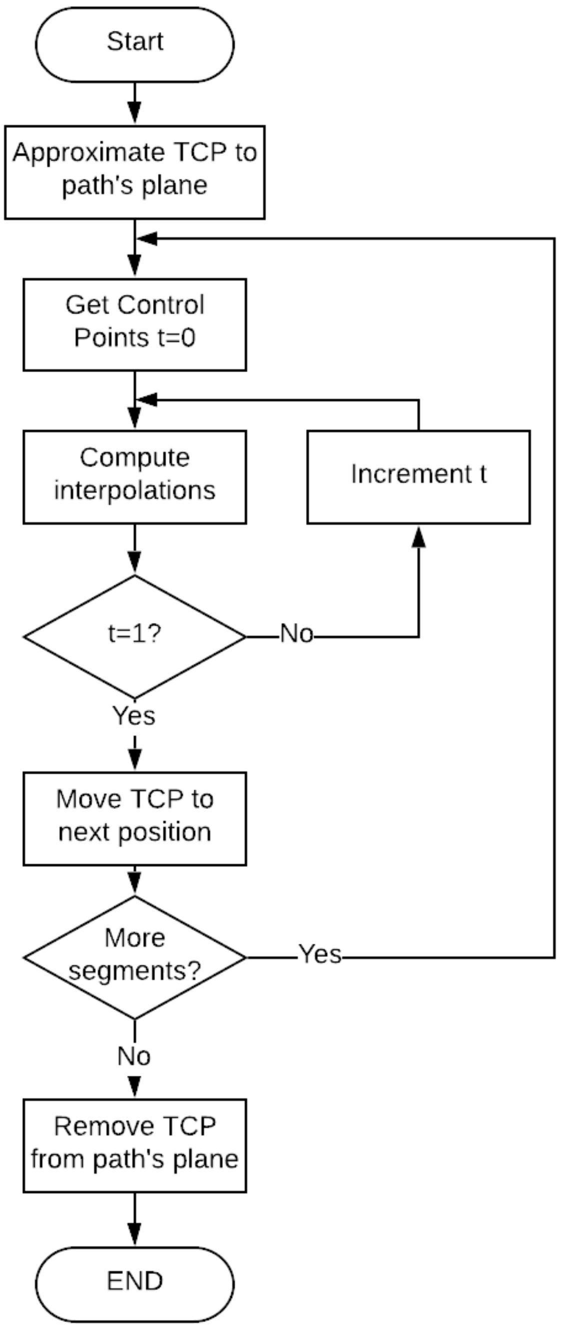


Figure 28. *Proposed flowchart for the implementation of the De Casteljau algorithm*

Earlier in this section it was mentioned that the implementation of the De Casteljau algorithm is made within the industrial robot's controller. One of the alternatives to this implementation would be to run the algorithm in an external program and introduce the evaluated points directly on the controller.

If the algorithm is computed in an external program, the values that are transferred to the controller will be the points evaluated over the Bezier curves. Instead of computing the interpolations, the robot will only perform linear motions from one point to the other until the curve is finished. This approach reduces the computational time and cost in the controller.

However, the memory requirements are significantly higher. For example, when evaluating five concatenated Bezier curves, for which the step size has been set to 0.1, the number of Bezier points needed is 16. If the computations are performed on an external program, the number of positions that will need to be transferred and stored on the controller would be 51. By implementing the algorithm within the industrial robot controller, the memory requirements are significantly reduced.

In addition, if the accuracy requirements change when an external program is used, the number of positions transferred to the controller will also be modified. When the interpolation is done inside the controller, the number of control points does not change. Only the increment of t would have to be changed from the code, since this approach is decoupled from the required approximation accuracy.

In conclusion, the proposed approach for the implementation of free shape paths in industrial robot is to use the De Casteljau algorithm directly implemented within the controller. This decision was made due to the drawbacks that the external problem presents, and the flexibility and simplicity of the algorithm when it's implemented in the controller.

4. IMPLEMENTATION

This chapter presents an implementation of the approach presented in chapter 3. This chapter describes the hardware and software used during the implementation. The implementation consists on the integration of a robot cell into an existing production line. This chapter is divided into 6 sections; the first section describes the implementation scenario, which is the FASTory production line. The second section describes the hardware and software components employed during the implementation of this case study. The third section illustrates the process taking place in the workstation and the services provided by the robot. The fourth section explains the implementation of the end effector on the robot manipulator and describes the process of workstation verification before the installation. The fifth section explains the full functionalities of the robot and the implementation of the functions that grant those functionalities. Lastly, the sixth section elaborates on the implementation of Web Services and the TCP/IP client into the RTU.

4.1 Implementation in FASTory line

The FASTory process line is used to demonstrate the assembly of mobile phones by drawing the main parts of a phone (frame, screen and keyboard) in three different colours (blue, red and green). The drawings are done on a paper located in a pallet that moves around the cell.

There are 12 workstations on the line in a loop topology: 10 identical cells containing a robot, in which the drawings are done; one workstation used to load raw materials (blank paper) and unload finished products (papers with phone drawing) from the pallets; and one to load and unload the pallets to the line. This thesis presents the implementation of one of the robotic cells, specifically workstation 3.

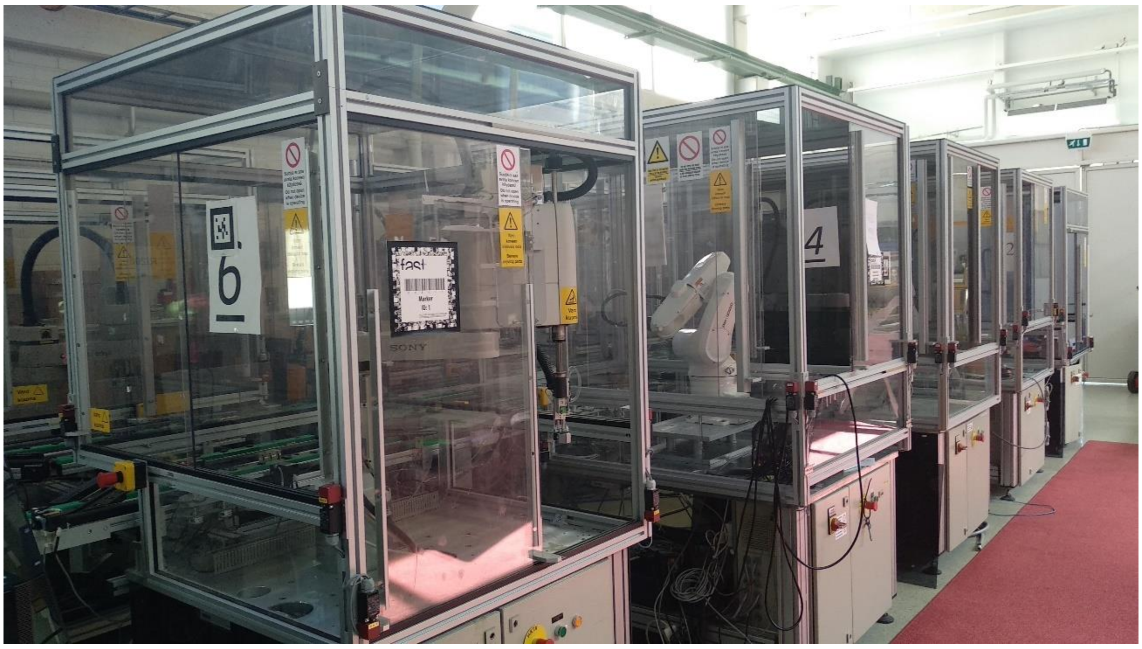


***Figure 29**. FASTory production line*

As shown in Figure 30, the workstation is composed of one robot arm and two conveyors, a main one and a bypass one. The main conveyor is used when the cell is required to perform an operation on the pallet; the bypass one is used to usher the pallet directly to the next cell. This configuration ensures that the line will not be held up during production.

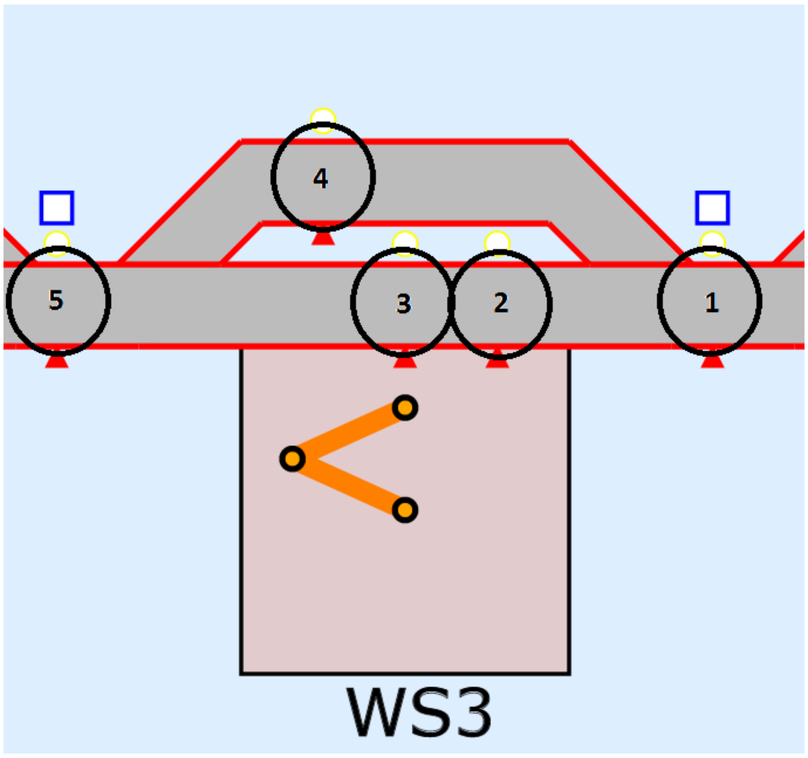


***Figure 30**. Simulation of workstation 3 made with FASTory Simulator [52]*

The workstation is divided into five zones. The functionalities of each zone are described in Table 4. Functionalities of the zones in the workstation. Each of the zones have a presence sensor and a stopper actuator, both of which are connected to the conveyor's RTU. Zone 1 also has an RFID (Radio Frequency Identification) reader used to identify the pallet that is entering the workstation.

Table 4. *Functionalities of the zones in the workstation*

Zone ID	**Functionality**
Z1	Entrance to the workstation. The RFID reader reads the ID of the pallet and the direction of the pallet is decided: through the main conveyor or the bypass conveyor.
Z2	Internal buffer. This is a waiting zone for a pallet in case Z3 is occupied.
Z3	Production zone. This is the position in which the service is provided to the pallet. The drawings are done when the pallet is located in this zone.
Z4	Bypass zone. It holds the bypassed pallet when Z5 is occupied, or when the pallet in Z3 is going to be transferred to Z5 first.
Z5	Exit zone. The pallet waits here to exit the workstation.

Coloured LED lights are installed on top of each cell in the line. These lights describe the state of both the conveyor and the robot at all times. The lights are controlled by an Arduino that is connected wirelessly to a router that is also connected to FASTory's Ethernet network. The robot can have four states: working (green), idle (blue), error (red) or calibration (yellow).

The FASTory line is located at FAST-Lab in University of Tampere, Finland. This line has been used in R&D projects like eSonia [53] and eScop [54] projects. The conveyors and control of the line have already been implemented by other parties. The FASTory line used to have SONY SRX-611 SCARA robots for production. These robots have 4 degrees of freedom and were used to perform the drawing operations on the pallets. Aside from those operations, the robots were also able to pick and place pens from a pen holder in order to change the colour of the drawing. The robots were equipped with custom-made end-effectors that were able to grab the pens.

However, recently new robots have been purchased to replace the ones in the workstations. For this thesis, one of the new robots is implemented and fully integrated into workstation 3. The approach for the integration ensures that the cell maintains its functionality, regardless of the robot's topology or the manufacturer.

4.2 Description of the components on the cell

This section describes the main hardware and software components that were used for the implementation. There are three main hardware components in the cell: the robot arm, the robot controller and the RTU. On the other hand, the software programs employed during the implementation are Workvisual 5.0 and KUKA SimPro 3.0.

The new robot on the cell is a KR 3 R540 [52] from KUKA Robotics[3]. This robot is a 6-axis anthropomorphic small robot specially designed to work in small workspaces. The maximum payload of this robot is 3 kg and the maximum reach is of 541 mm.

The robot's controller, the KRC4 Compact controller, is also from KUKA Robotics. This controller also comes with a KUKA smartPAD, which is an intuitive teach pendant used to control the robot. It can also be used for inline programming of simple tasks. The controller runs on the KUKA System Software (KSS) v8.5. The programming language of the controller is called KUKA Robot Language (KRL).

The KRC4's system architecture integrates four controllers in one: Motion Control, Robot Control, PLC Control and Safety Control. This controller allows the use of an external safety controller, for which the communication and safety functions are implemented with Ethernet-based protocols, like CIP Safety with Ethernet/IP.

[3] https://www.kuka.com/

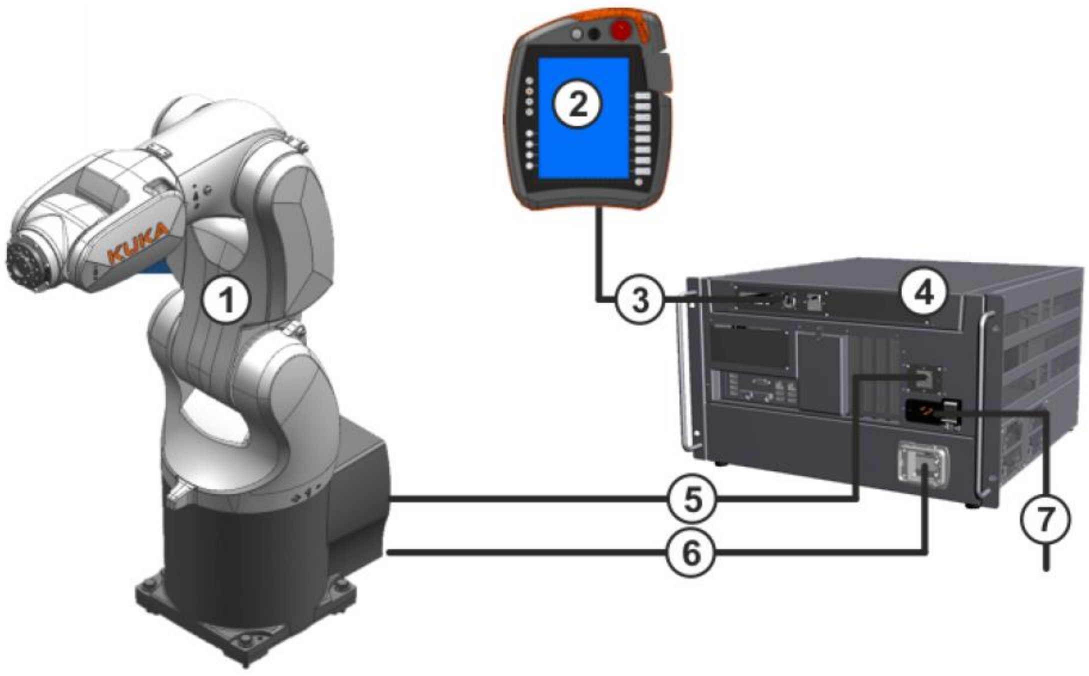


***Figure 31**. Overview of the robot system in the cell*

Figure 31 shows the set up for the robot manipulator and controller. The controller (4) and the robot arm (1) are connected by means of two cables: a data cable (5) and a motor cable (6). The power supply arrives to the controller through the device connection cable (7). The smartPAD (2) is also plugged to the controller through a cable (3).

Although the controller does not allow TCP/IP communication by itself, there is an add-on technology package that can be installed on the controller that makes it possible. This package, called EthernetKRL, was installed on the robot during the implementation in order to communicate with the RTU.

Workvisual 5.0 is KUKA's software interface in which configuration, diagnosis and programming can be made, among other things. It is a useful tool to perform offline configuration and development of the controller, as well as online diagnostics and maintenance. With Workvisual it is possible to perform I/O mapping and offline programming on a robot project. That project can be later deployed to the controller through the software.

KUKA SimPro 3., on the other hand, is a simulation software also issued by KUKA. This software is used to simulate the layout of the workstation and for offline programming. It is an efficient tool to perform workspace verification and optimization, by testing the layout of the cell and the reachability of the robot at the early stages of the project, before the physical installation is done.

The RTUs in question are INICO S1000 [56]. This smart RTU is web service based and allows the integration into a Service-Oriented Architecture. It supports RESTful web services and DPWS (Device Profile Web Services) for discovery and manipulating web services. The S1000 can also maintain a communication via TCP/IP, which makes it possible to integrate it with the abovementioned robot controller.

***Figure 32**. INICO S1000 RTU*

4.3 Deployment of the robot

As explained in section 3.2, the devices in the cell are connected to the same FASTory network through an Ethernet switch. The IP addresses assigned for devices in workstation 3 are 192.168.3.X. The X is different for each device in the workstation.

The robot is configured as a service provider. The service is requested by a higher-level orchestrator through the S1000 RTU. When a service is requested, the S1000 forwards the request to the robot controller through a TCP/IP connection. The robot receives the request and executes one of its internal functions to provide the service. When the operation is finished, it informs the S1000, who responds to the orchestrator to notify that the service has been provided. The robot's RTU also sends a REST request to the controller of the lights that show the state of the robot in order to change its colour every time that state changes.

The functionalities of the robot and the S1000 are shown in the use case diagram in Figure 33. The robot opens the connection as a TCP/IP server and, during its idle time, waits for a message from the RTU. There are three service types that the RTU is able to request from the robot:

- Select pen: A pen colour is selected. If there is already a pen in the gripper, the robot places it in its pen holder and picks the requested pen.

- Get new pen: This service is used to discard an empty pen in the pen holder and replace it with a new one from the pen feeder. A pen must already be selected in order to perform this service.
- Select drawing: One of 9 drawings are selected. This operation is only executed if a pen has already been selected.

In order to perform these services, the robot has been programmed with the following functionalities: pick pen, place pen, pick new pen, discard empty pen and draw. The functions that are used to implement these functionalities are described in section 4.5.

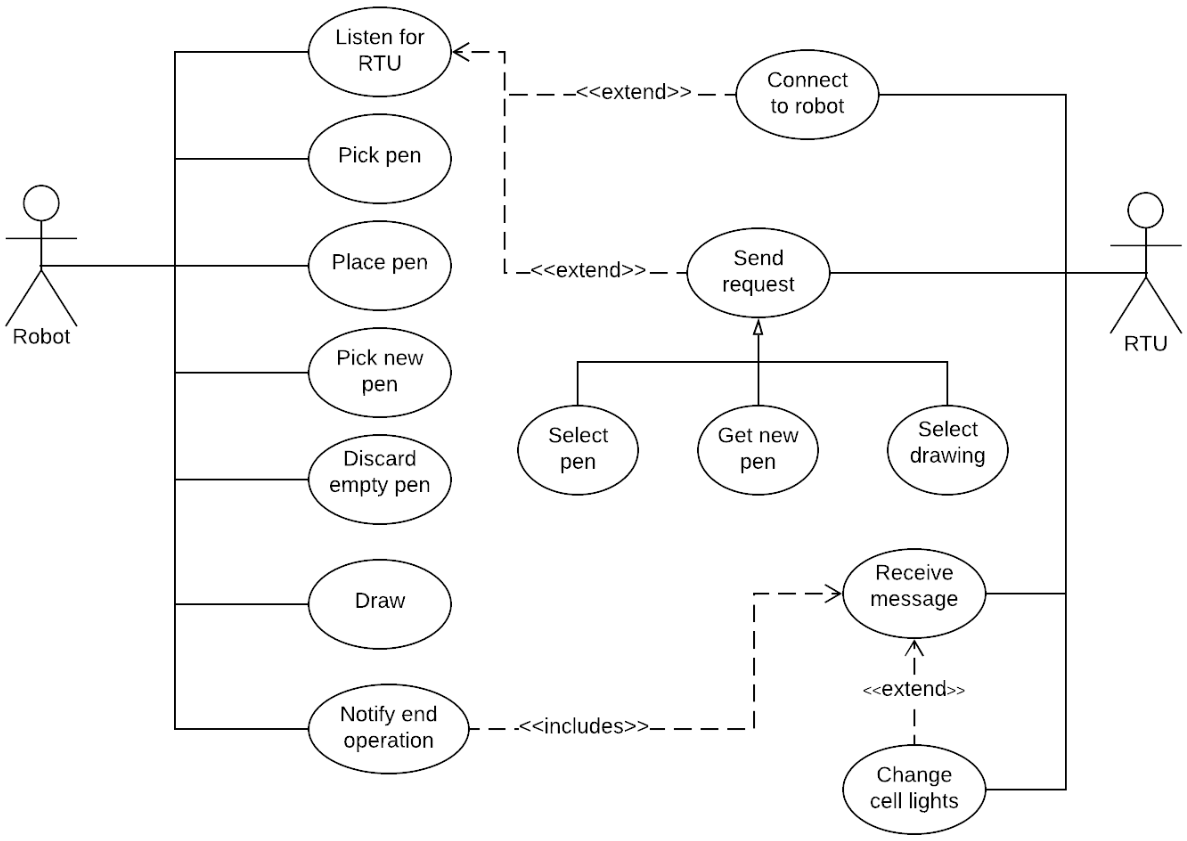


***Figure 33**. Use Case Diagram of the implemented case study*

When the service is accepted by the robot, the RTU changes the lights of the cell to green to signify that the robot is working. Once the operation is finished and the service has been provided, the robot notifies it to the RTU, who receives the message and changes the light of the cell to blue, to show that the robot is idle and awaiting new requests. The lights are changed by sending a REST POST request to the Arduino that controls the lights. The URL list and payloads for these requests can be found in the Appendix.

4.4 Installation of the end effector

During the installation of the robot in the cell, the configuration of the end effector was decided. The end effector used for the KUKA robot is the same one that was previously installed on the SCARA robots, however, an attachment plate had to be designed to be designed in order to attach it to the robot.

Before designing the plate, the orientation of the end effector was considered. In the SCARA robots, the end effector was located so that the last axis of the robot and the picked pen were in parallel. This orientation is adequate for robots with SCARA topology, since that way the drawings can be done in the XY plane by using the robot's revolute axes. In an articulated robot, there are two possible orientations for the end effector: one for grasping the pen in parallel to the robot's last joint axis, and one for grasping it in perpendicular.

In order to decide on a configuration, first it's important to know if the robot is able to reach all points in the workstation in the desired orientation. The reach of the end effector was measured for each orientation by using KUKA's simulation software, SimPro. Performing the reachability tests with an offline simulation ensures the viability of the workstation and the robot configuration before the physical installation is done.

For the tests, the layout of the cell was simulated in the program. Then, two tool frames were configured: one for the parallel orientation and one for the perpendicular. The simulated robot was then jogged around the workstation in order to prove that the end effector could reach all relevant targets, as well as the approximation targets. The following figures shows the configuration of the robot when picking the green pen from the pen holder. On Figure 34, the end effector's orientation allows the robot to pick the pen in perpendicular to the last axis; on Figure 35, the pen is picked in parallel.

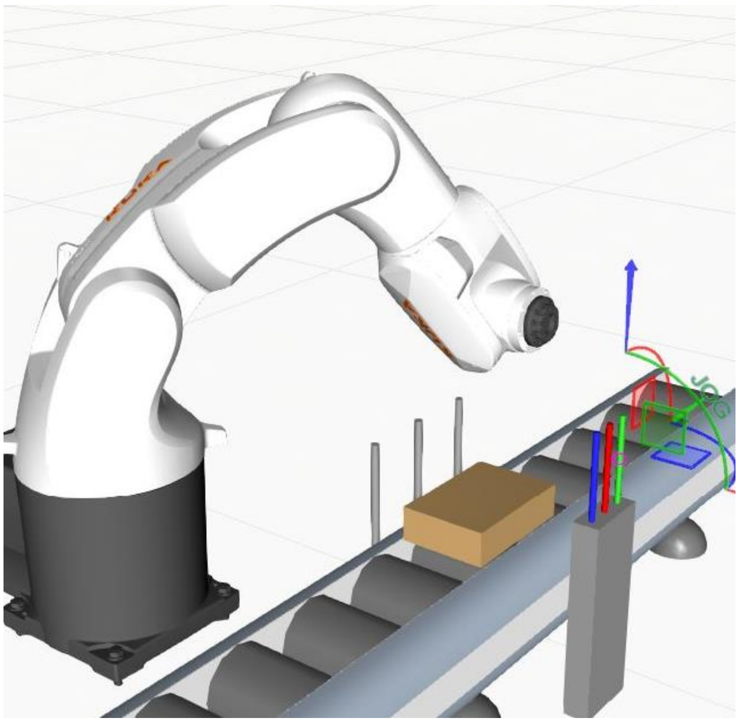


***Figure 34**. Robot grasping green pen in perpendicular to the last axis*

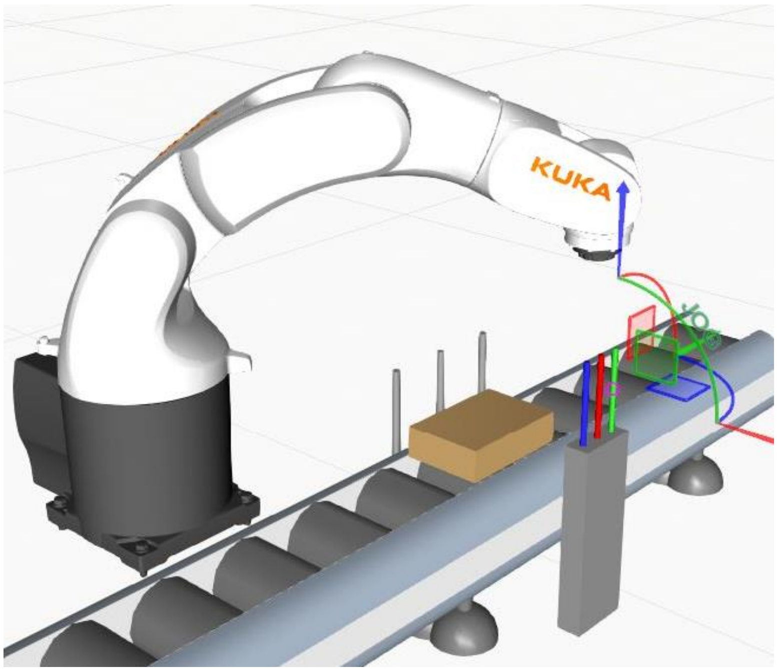


***Figure 35**. Robot grasping green pen in parallel to the last axis*

The tests demonstrated that both orientations allowed the robot to reach the intended targets; however, in the parallel configuration the robot approximated some singularity points during the removal of the pens from the pen holder. This occurs because the end effector must stay vertical during the process, so joints 2 and 3 get close to lining up, and the robot is unable to move the end effector further upwards.

For this reason, the perpendicular orientation was selected. The final installation of the end effector on the physical robot is shown in **¡Error! No se encuentra el origen de la referencia.**.

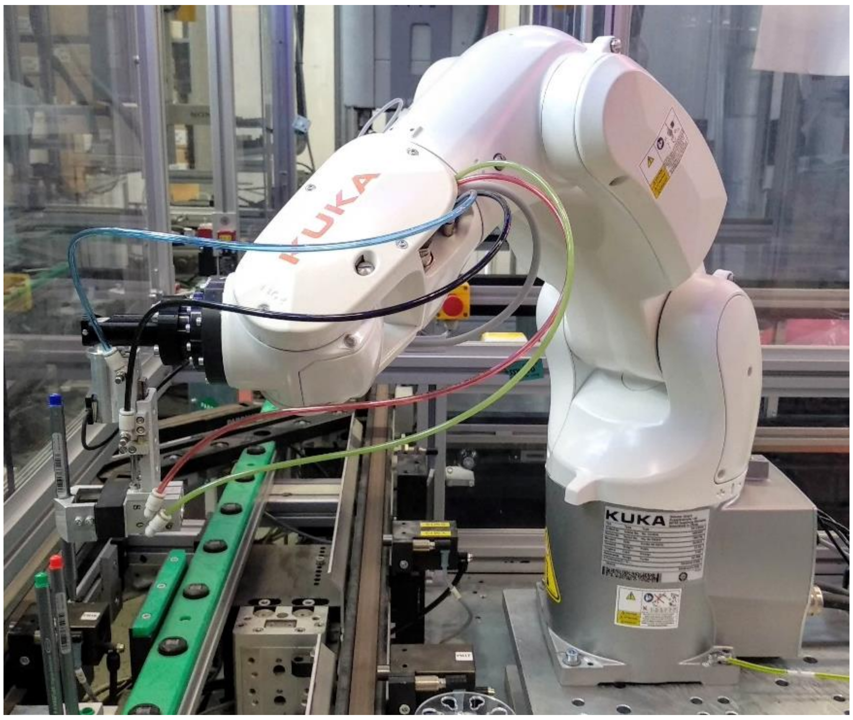


Figure 36*. Final configuration of the end effector and the attachment plate in the robot*

There are some advantages to doing the reachability checks offline rather than in the real robot workspace. For starters, performing offline tests is a quick and efficient way to perform the workstation verification and make sure that the layout of the cell is adequate. It ensures that the robot is able to reach everything inside the workstation before installing and wiring the real robot. The simulation also helps find the constraints in the workstation and the robot configurations that cause singularities, so that they can be avoided during programming.

In this implementation the simulation software was used to calculate the reachability and find constraints of the workstation before the end effector was installed. This way, the time that would have been employed for designing and machining attachment plates for both configurations was reduced, as well as the time necessary to test both in the physical workstation.

There are four pneumatic valves on the end effector: two to open and close the gripper and the other two to move the gripper up and down. The end effector also has two sensors installed. The D-A93 is a position sensor from SMC[4] intended for pneumatic

[4] https://www.smc.eu

actuators. The sensor activates when some resistance is felt in the gripper, sending an input signal to the controller; it is used to sense when the tip of the pen has touched the paper in the pallet at the moment of calibrating the base frame for the drawings. The UM-R5TVP sensor from Takex[5], on the other hand, is a photoelectric sensor used to detect the presence of objects in front of the end effector by using a LED light source.

4.5 Functionalities of the robot

The services that the robot provides are briefly explained in section 4.3. This section describes the functionalities programmed in the robot that make it possible to provide those services.

The programming language used by KUKA controllers is called the KUKA Robot Language (KRL). KRL programs consist of two files: a DAT file (.dat) or data list, where all constants and variables for the program are declared; and an SRC file (.src), where the program code is written. An SRC file and its associated DAT file are called a module. The codes of the SRC files described in this section can be found in the Appendix.

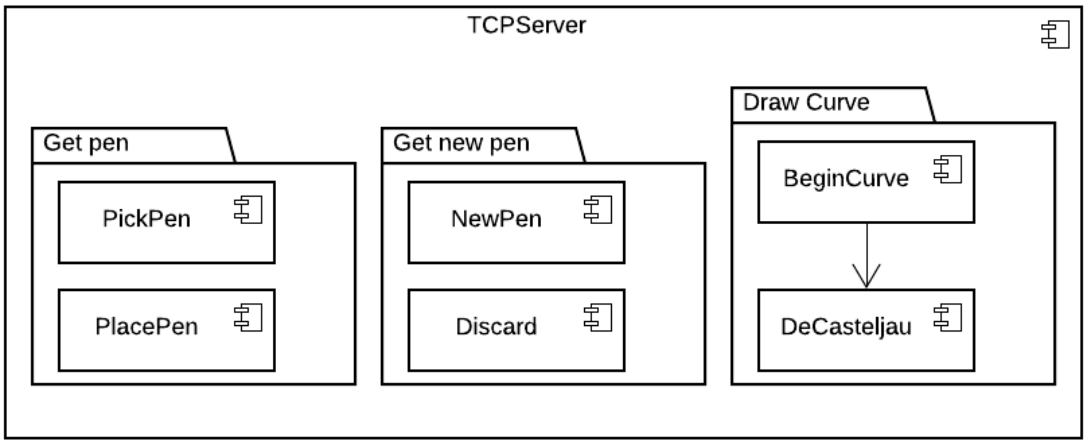


***Figure 37**. Modules and functionalities of the robot.*

Seven modules have been created for this implementation. The main module is called *TCPServer*, and it is the module that opens the TCP/IP communication and processes the received messages. The modules called from *TCPServer* are *PickPen*, *PlacePen*, *NewPen, Discard, BeginCurve* and *DeCasteljau*. Figure 37 shows the modules in the robot, divided into groups that showcase the robot's functionalities. These operations are exposed through the TCP/IP connection between the RTU and the TCPServer module,

[5] https://takex.com/

and later as RESTful web services. The exposure of the functionalities of the robot though APIs is further explained in section 4.6.

The *PickPen* function is used to pick one of the pens in the pen holder. This function requires an input variable that specifies the colour of the pen that will be picked. When this function is invoked, the end effector describes a PTP motion to the approximation position of the selected pen. Then the gripper opens and the robot moves horizontally to approximate the pen, where the gripper closes. Lastly, the robot performs an upwards linear motion vertically to remove the pen from the holder.

The *PlacePen* function performs a similar operation as *PickPen*, but in reverse. The linear approximation is done vertically from a position over the pen holder. Once the pen is lowered to its place, the gripper is opened and the robot retreats by describing a horizontal linear motion. *NewPen* is employed to pick a new pen from the pen feeder. This function also needs an input variable that specifies the pen colour. The movements described by the robot are the same, the only change is the position of the pen that is picked.

All three of these modules have the position of the three coloured pens saved in the data list. Before beginning the movements, the function first checks which of the colours is the one selected in order to know to which position to go.

The *Discard* module deposits the pen on the gripper into a box in the workstation, where the empty pens are placed. Only the position of the box is saved in the data list of this module.

The four modules above constitute the pen manipulation operations for the controller. In order to keep track of the pen that has been selected, a global variable is used. This variable is called currPen and it is an integer type. The numerical values of the variable and their meanings are indicated in Table 5. At the end of each function execution, the value of currPen is changed so that it indicates the colour of the pen that is in the gripper at the moment.

***Table 5**. Values of currPen variable for the colours of the pen.*

Value	**Meaning**
0	Gripper is empty; no pen selected.
1	Current pen is BLUE.
2	Current pen is RED.
3	Current pen is GREEN.

The other two modules produce the drawings on the pallet. The *DeCasteljau* module receives four control points as inputs. Those points are interpolated using the De Casteljau algorithm. When the set of interpolations for one value of *t* is computed, the robot performs a linear motion to the next position of the curve. The interpolations and the motion command are encapsulated inside a counting loop that increases the value of *t* for each repetition of the loop.

Finally, the *BeginCurve* module is used to start the curve and call the *DeCasteljau* module for each of the Bezier curve segments in the drawing. The execution of this function is the one proposed in section 3.4, and it's represented in Figure 28. Proposed flowchart for the implementation of the De Casteljau algorithm. The robot approximates the pen to the pallet until the tip of the pen touches the paper in the position of the first Bezier curve. Then it calls the *DeCasteljau* module with the first set of control points as inputs. The module is called as many times as curves are in the drawing. When the drawing is finished the pen is lifted from the paper and the execution of the module is finished.

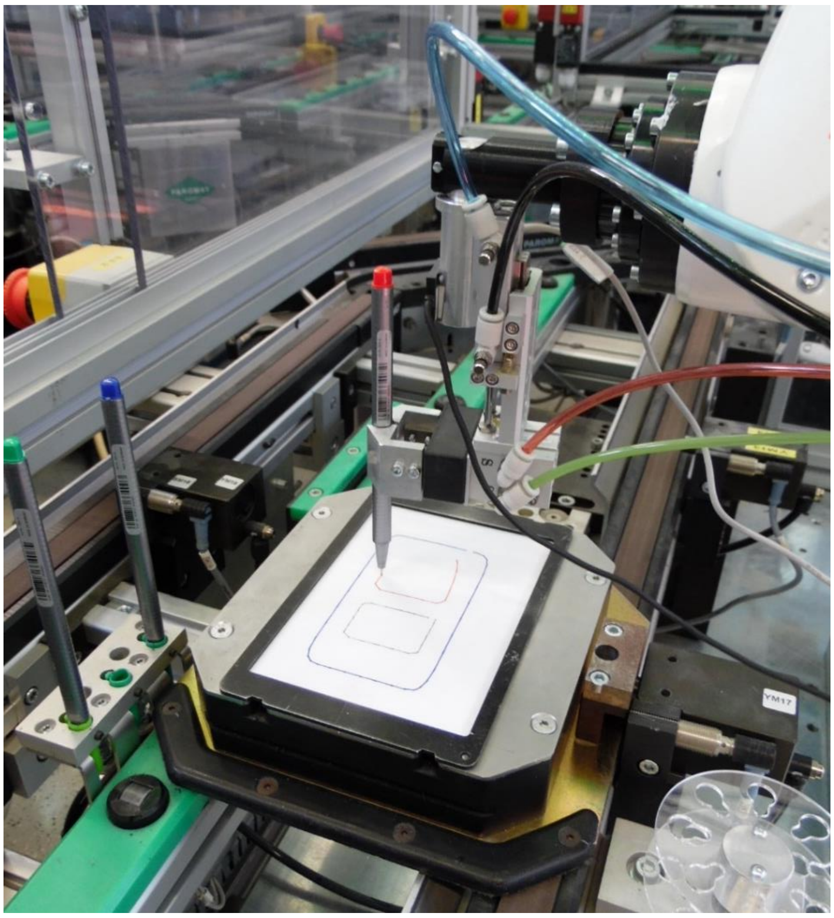


***Figure 38**. Robot performing a drawing operation on a pallet*

There are 9 possible drawings that the robot is able to draw: 3 for the mobile phone's frame, 3 for the keyboard and 3 for the screen. Each drawing is divided into a set of cubic Bezier segments described by four Bezier control points. The number of segments for each drawing varies depending on the complexity of the curve.

BeginCurve requires an input integer variable that specifies the number of the drawing to be executed. The number of segments and the list of Bezier points are stored in the data file of the *BeginCurve* module. The *DeCasteljau* module is called repetitively for all Bezier segments of the drawing.

The execution of the modules described in the previous section are orchestrated by the *TCPServer* module. Figure 39 shows a flowchart that explains the sequence of operations that occur during the execution of this module.

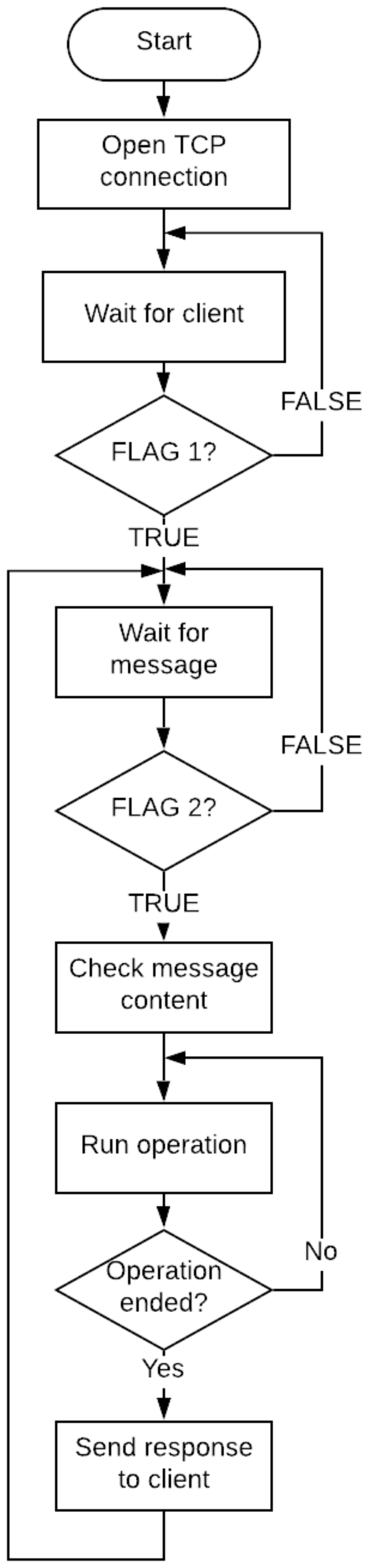


Figure 39*. Flowchart of the process in TCPServer.*

The operation begins by opening a TCP/IP communication channel. The configuration of the connection is done via an XML file that is saved in the robot controller. This configuration file, shown in Annex A, indicates the connection parameters between the external system and the robot controller as well as the reception and transmission structure of the messages.

When the robot controller acts as a server, the IP address of the controller and a port must be indicated. For the KRC4 Compact controller, the ports available for TCP/IP communication are ports 54600-54615.

The reception of data is configured to be a string of raw data with variable length. The data must contain an End Of Stream (EOS) element, which is an end string that lets the server know that the transmission of data is over. In the XML file, the EOS is set as ”69,78,68”. This is the ASCII code for ”END”. The messages sent from the S1000 need to end in this string of characters for the message to be understood by the server, otherwise FLAG 2 is not set to TRUE, and the controller is kept waiting for the end of the transmission.

Once the client tries to open the connection, FLAG 1 is set to TRUE. The flag will remain TRUE until either the server or the client closes the connection. The server listens to the connection for a message from the client. When a full message arrives, FLAG 2 is set to TRUE. The content of the message is a raw string that functions as an API. The server processes and interprets the message and, depending on the content, runs the necessary functions to complete request.

When the operations are finished, FLAG 2 is reset and the server sends a response to the client to inform that the service has been completed. Provided that the client has not been disconnected, the server goes back to waiting for a new message.

When the requested service is the selection of a pen, the message received from the client is ”BLUE”, ”RED” or ”GREEN”. In this case, the pseudocode that runs in the *TCPServer* program to perform the requested task is written in Figure 40.

```
IF currPen<>0 THEN
    PlacePen();
ENDIF;
PickPen(selectedPen);
currPen=selectedPen;
```

***Figure 40**. Pseudocode for the service request of pen selection*

If the content of the message from the RTU is ”NEWPEN”, it means that the pen that is in the gripper is empty and thus, it needs to be disposed and a new one needs to be picked. Figure 41 shows the pseudocode for this task.

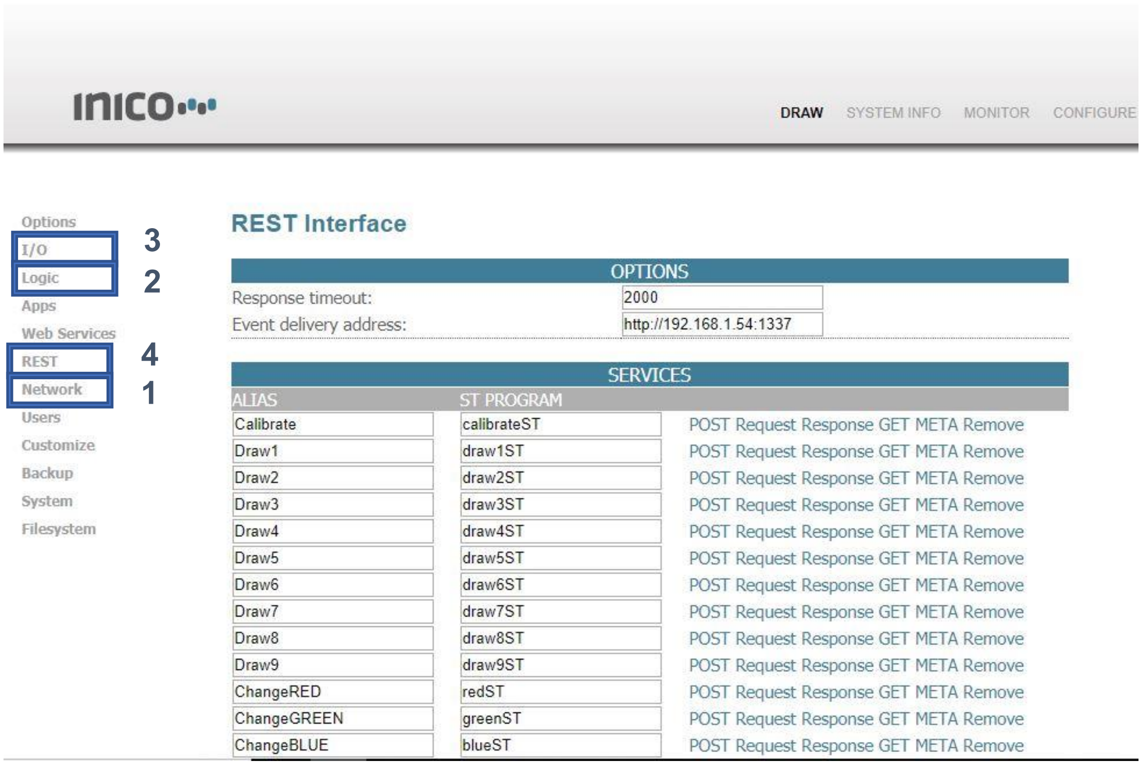
```
IF currPen<>0 THEN
    Discard;
    NewPen(currPen);
ENDIF;
```

***Figure 41**. Pseudocode for the service request of the renewal of a pen*

Lastly, when the content of the message is ”DRAWX”, with X being a number from 1 to 9, it means that the required service is one of the 9 drawings for which the Bezier points are stored in the robot controller. In this case, the program calls the *BeginCurve* module with the number of the drawing as an input.

4.6 Communication between S1000 and the robot controller

Request messages for the robot controller arrive from the INICO S1000 RTU. The S1000 can be fully configured through the web browser. Both the programming of the logic and the configuration of the I/O modules and the web services can be done in web browsers like Chrome or Mozilla Firefox.



REST Interface

OPTIONS

Response timeout:	2000
Event delivery address:	http://192.168.1.54:1337

SERVICES

ALIAS	ST PROGRAM	
Calibrate	calibrateST	POST Request Response GET META Remove
Draw1	draw1ST	POST Request Response GET META Remove
Draw2	draw2ST	POST Request Response GET META Remove
Draw3	draw3ST	POST Request Response GET META Remove
Draw4	draw4ST	POST Request Response GET META Remove
Draw5	draw5ST	POST Request Response GET META Remove
Draw6	draw6ST	POST Request Response GET META Remove
Draw7	draw7ST	POST Request Response GET META Remove
Draw8	draw8ST	POST Request Response GET META Remove
Draw9	draw9ST	POST Request Response GET META Remove
ChangeRED	redST	POST Request Response GET META Remove
ChangeGREEN	greenST	POST Request Response GET META Remove
ChangeBLUE	blueST	POST Request Response GET META Remove

***Figure 42**. Web interface of the INICO S1000 RTU*

Figure 42 shows an overview of the S1000 interface used for configuration. The steps followed to configure and program the device are numbered in the figure.

***Table 6**. Overview of the tabs in the INICO S1000 interface.*

Item	Tab	Description
1	Network	This tab is used to configure the name of the device and the IP address on the Ethernet network.
2	I/O Module	Here the I/O systems is configured. New I/O modules can be added as well as configuring existing ones.
3	Logic	Logic control is programmed here. This tab is used to declare constants and global variables and to write and debug the programs. The language used for programming is Structured Text (ST).
4	REST	Here the REST services are configured. Each REST service is linked to a Logic program, so when a REST request arrives, the Logic program is invoked.

To configure the TCP/IP connection, one of the S1000's I/O modules is used. The module is called Net Connection, and it can be used to create a TCP Client or Server on the S1000 to send or receive generic data in form of strings. The client module is configured by assigning an alias and indicating the IP address and listening port of the server.

After the connection module is created, the logic is programmed. The Logic tab allows the creation of several programs written in ST. For this implementation one function has been created per request type. In total, there are 13 functions in the controller: three for the selection of the pens (one for each colour), one for discarding the selected pen and getting a new one, and 9 for each of the drawings.

Each of these functions send a unique string that is received by the server and used to indicate which service must be provided. These words are the APIs of the connection between the RTU and the robot. The API pattern created for the application and the operations it executes are shown in Figure 43.

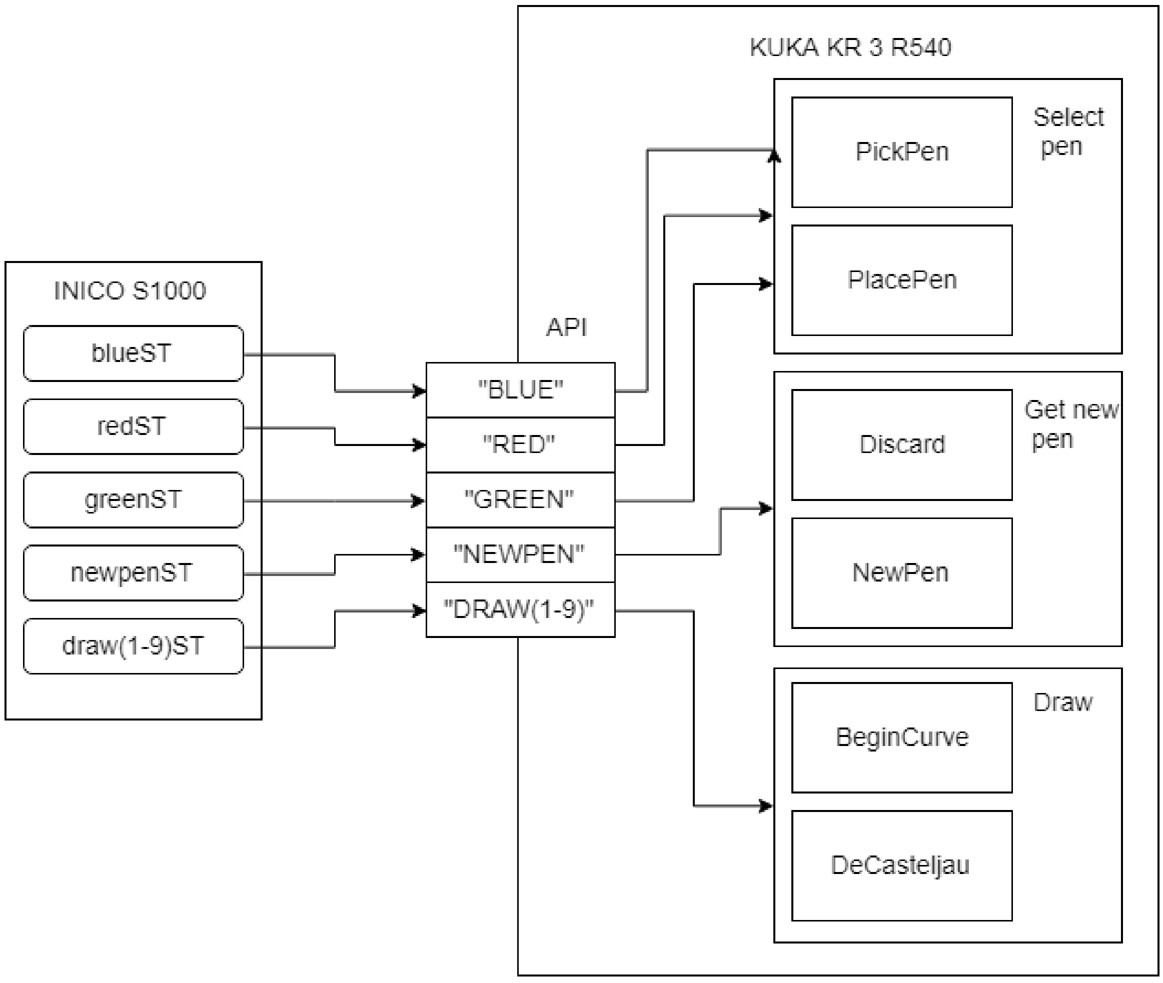


Figure 43*. API pattern for the communication between RTU and robot*

Aside from the ST programs, the logic also includes the system variables that are used globally during the execution of the logic. There are two variables that are declared globally: currPen, which is a integer variable that indicates the colour of the selected pen, and robState, which is a string that indicates the state of the robot as ”working”, ”idle”, ”error” and ”calibration”.

After the programs are created the REST services must be configured. In the REST tab, services are created with a name and linking them to one of the ST programs. All RESTful APIs services and their descriptions are collected in the Appendix. The RESTful APIs used to invoke an operation are sent with the following structure:

```
POST: http://192.168.3.1/rest/services/{operation}

Body: {"destURL":"" }
```

Figure 44*. Generic RESTful API for service request*

When a REST request is receives, the ST program linked to it is executed. The program sends a message through the Net Connection to the robot controller, which processes the message and executes the requested operations. Aside from receiving REST requests, the RTU also sends REST POST requests to the control of the cell lights in order to change them according to the robot’s state. The RESTful APIs for the control of the lights is listed in the Appendix as well.

5. TESTS AND RESULTS

This chapter describes the test that were done in order to prove the validity of the proposal presented in chapter 3, as well as the results of the tests. The chapter is divided into two parts; the first describes the tests and result obtained in order to check the implementation of the De Casteljau algorithm and decide the optimal parameter values for the application. The second part describes the test of the implementation scenario as a whole. It is important to mention that while the first tests are quantitative, the testing of the scenario shows qualitative results, since the tests measure whether the system is properly implemented.

5.1 De Casteljau algorithm testing

During the process of implementation, some tests were done in order to prove the validity of the algorithm. The objectives of the tests were the verification of approach to implement free shape paths in industrial robots. Tests are also made to analyse the effect that the number of interpolations computed for each curve segment have on the quality of product and the process. For that, the accuracy of the drawing has been tested, as well as the time it takes to perform the drawings.

Initial testing is performed by introducing the number and colour of a drawing via the robot's smartPAD. A pallet is manually introduced in Zone 3 of the workstation. When the gripper is empty, the *PickPen* module is executed, followed by the *BeginCurve* module. For the purpose of testing, the drawings performed by the robot are shown in Figure 45.

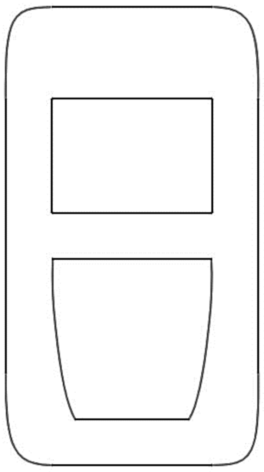


***Figure 45**. Model of the mobile phone drawing*

During the first execution of the testing, it was demonstrated that the surface of the pallet is not entirely aligned with the XY plane of the robot, and thus, the tip of the pen is not in contact with the paper at all times during the drawing process. To correct the error a calibration of the base frame was executed.

Calibration of the base frame consist of assigning a Cartesian coordinate system to the pallet so that the surface of the paper coincides with the XY plane of the system. Calibration is performed through a 3-point method. The robot is jogged to three points in the work surface: the origin of the coordinate system, a point along axis X and a point in the XY plane.

In order to perform the calibration, a pen must be selected first. The pen is lowered to the pallet until the position sensor is activated. The pen must be slightly raised in order to avoid damaging the paper on the pallet, but still keeping it in contact with the paper. These steps are followed to define all three points. The base calibration is performed on the teach pendant.

Once the work surface is properly configured, testing can be performed. The test cases that are considered are modifications of the step size of parameter t, Δt, during the interpolation process. Section 3.4 explains the effect of Δt in various factors of the curve and the process in itself. The results that are analysed in this chapter are the accuracy of the drawings and the execution time of the operation. Five values of Δt have been compared:

***Table 7**. Step sizes for the test cases*

Test Case	Step size	Number of interpolations
1	$\Delta t = 0.5$	3
2	$\Delta t = 0.25$	5
3	$\Delta t = 0.1$	11
4	$\Delta t = 0.05$	21
5	$\Delta t = 0.02$	51

The number of interpolations that are processed in the algorithm increases as the value of the step size decreases. Higher number of interpolations require a higher computational time for the robot controller to execute the algorithm. At the same time, the linear segments drawn by the robot arm after each computation are shorter, which results in a closer approximation to the Bezier curve. Figure 46 shows the outcome of the operation for each of the test cases.

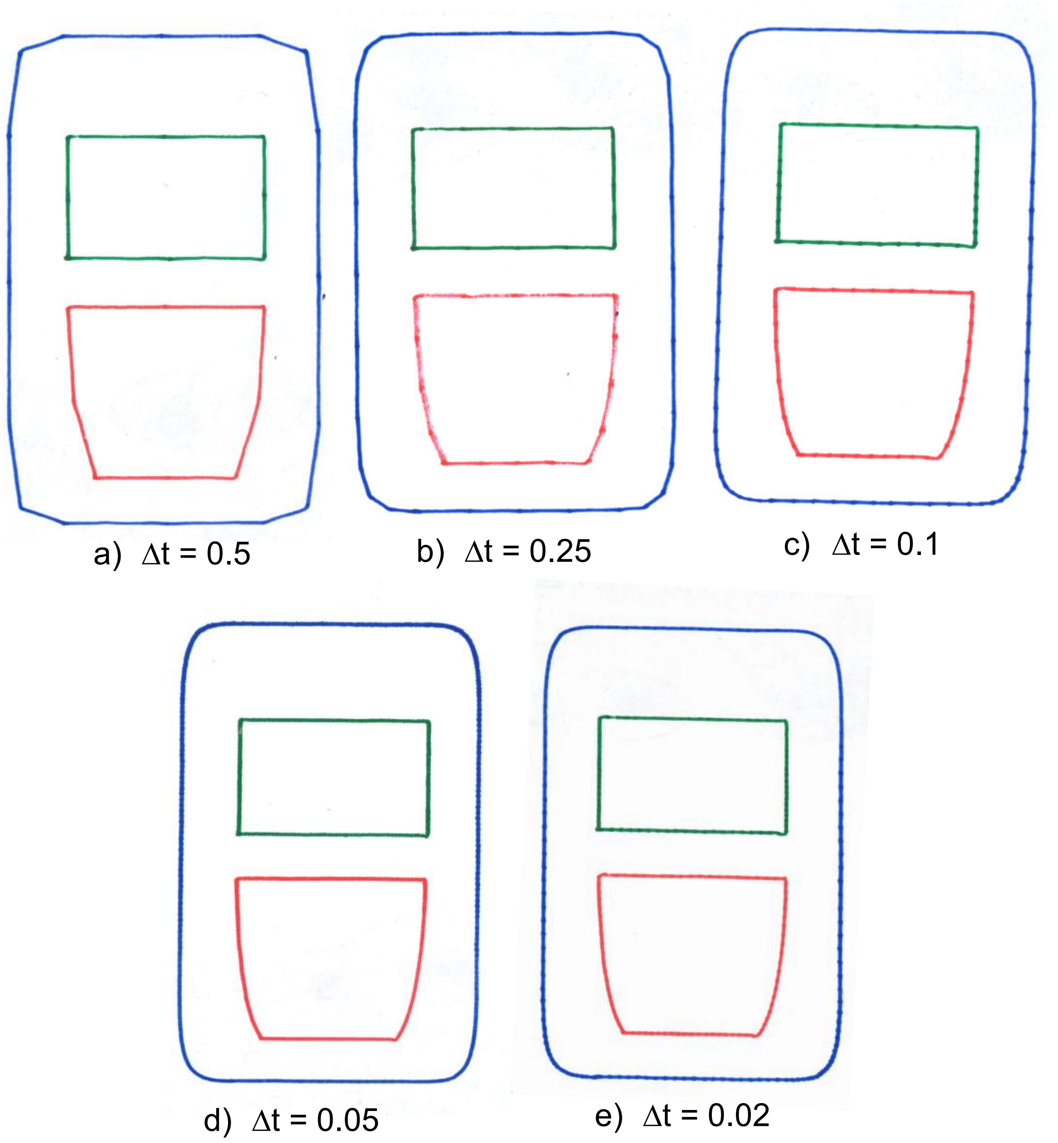


***Figure 46**. Result drawings of the test cases*

For the higher step sizes, the linear segments are visually noticeable, especially around the rounded edges of the frame. For shorter step sizes the approximation is closer and the linear segments are less noticeable, however the points between the segments can still be appreciated for $\Delta t = 0.1$.

In order to measure the computation time of the algorithm, the duration of one drawing operation is timed. The maximum velocity of the robot during linear motions is of 2 m/s, which is set to 30% during testing. The drawing chosen for the test cases is the frame of the mobile phone, drawn in colour blue in Figure 46. The time was measured from the moment the pen touches the paper to the moment the drawing is finished, removing the pallet approximation and separation times. The resulting operation times (in seconds) for each test case are shown in Table 8. Figure 47 shows that the operation time to complete one drawing and the step size configured in the algorithm are related by a potential function.

***Table 8**. Operation times for the use cases*

Test Case	Step size	Operation time (s)
1	∆t = 0.5	11
2	∆t = 0.25	16
3	∆t = 0.1	28
4	∆t = 0.05	44
5	∆t = 0.02	80

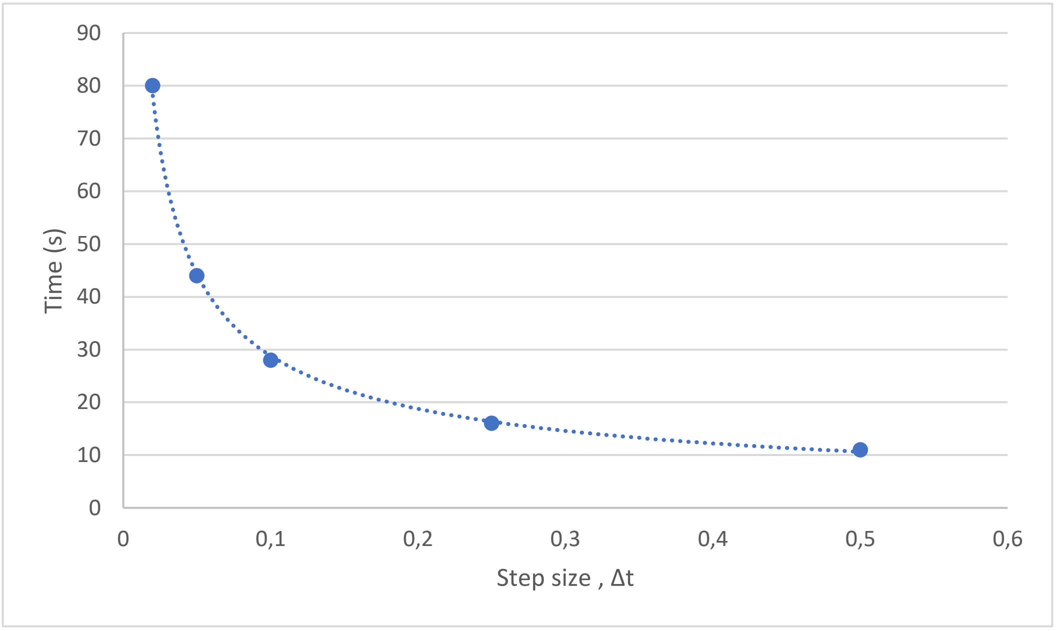


***Figure 47**. Operation times vs step size*

Not all applications have the same requirements and restrictions in terms of cycle time and accuracy. In order to make the choice of the step size value for parameter *t*, those requirements must be pondered, so that the step size is set to maximize the productivity of the line and the quality of the product. During this implementation, the chosen value is ∆t = 0.05. This value provides an adequate approximation to the designed drawing, while ensuring that the cycle time does not create bottlenecks in the system.

These tests also demonstrate the advantages of implementing the De Casteljau algorithm within the robot controller, as opposed to evaluating them in an external software. The changes done during the testing process were reduced to a simple modification of the step value within the function. By performing the computation externally, the points on the curve need to be transferred to the controller with every test case, so the data transfer time increases.

5.2 Integration of the robot in the cell

The process of testing the scenario is done in order to ensure the proper integration of the robot in the cell, and to validate the communication between the orchestrator and the RTU and between the RTU and the robot.

Communication between the orchestrator and the RTU is done via RESTful web services. Before deploying the system into the production line, testing was done separately, without the orchestrator. Services were invoked by using a REST API testing platform. The platform was used to send REST requests to the robot’s RTU, in the same way the orchestrator would to the RTUs in the production line. The RESTful APIs used to request services are listed in the Appendix.

The scenario begins by executing the *TCPServer* program from the robot’s smartPAD. The server opens the TCP connection and waits for a client to connect to the port. Then, a REST request is sent to the RTU. During the first request, the RTU connects to the server and waits for 3 seconds, in which the status of the connection is checked. When the connection is properly established, a message is shared between the RTU and the robot, and the robot performs whichever operation is requested from it.

The interactions between the objects in the implementation are shown in Figure 48. This sequence diagram exemplifies one of the scenarios performed in order to test the overall performance of the system, in which the user requests that the robot picks one of the pens.

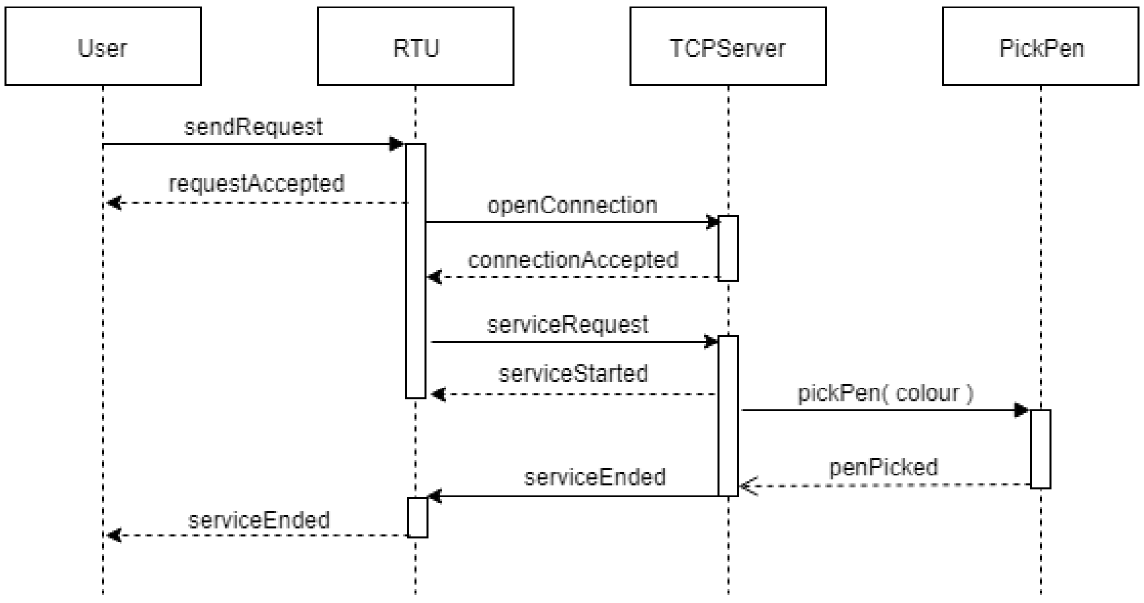


Figure 48. *Sequence diagram for one of the test scenarios.*

These tests proved to be effective, as the robot performed the tasks that were requested with REST. Once the operation is finished, the robot returns to the *TCPServer* program and, as the connection is not closed, it waits for a new message from the RTU.

Another thing that was tested is the queueing system of the TCP/IP connection. This connection is asynchronous, since the messages sent from the RTU to the robot do not need an immediate answer from the robot. The messages are deposited into a queue and will be read by the TCP server when it is unoccupied.

The queueing system was tested by sending a request while the robot was busy performing another task. During the execution, the robot received the message, but continued with the current operation. Only when the task finished and the robot went back to idle, the next message was read and the request executed.

6. CONCLUSIONS

This chapter provides a conclusion to the research work done in this thesis. The conclusions summarize the work exhibited in this document and discusses the results found in the previous chapter. The research questions asked in chapter 1 are answered and the objectives set are matched to the outcome of the implementation.

6.1 Conclusions

During the implementation and testing of the scenario, it was proven that web services can be used for integrating a robotic cell into a production line. Web services were integrated into a SOA system to expose the robot's functionalities as services within the system. The interaction and communication of the robot cell with the rest of the system establishes the robot as a service provider within the production line.

Communication between the robot controller and a higher-level orchestrator is done through a Remote Terminal Unit. Since industrial robots cannot normally communicate via web services, the RTU acts as a supervisory control interface that receives the requests from the orchestrator, sent via RESTful web services, processes them and forwards them to the robot controller via a TCP/IP connection.

Another main conclusion extracted from the implementation of the proposal made in this research work is that it is possible to implement an efficient and computationally simple algorithm in an industrial robot that allows the robot to move describing free shape paths. These paths, useful in welding or painting applications, as well as in collision avoidance operations, can be modelled using Bezier curves.

The implementation shows that the De Casteljau algorithm permits the implementation of said curves into an industrial controller with positive results. Parameter *t* in the algorithm is a factor that affects both the quality of the product and the duration of the operation, affected by the number of interpolations that are computed. As such, the parameter must be chosen in order to optimize the operation of the system and the accuracy and time requirements of individual cases.

6.2 Further work

In the future, the work done in this thesis can be extended, and the features and functionalities of the FASTory line can be improved. Here are some implementations that can further improve the performance of the production line:

1. Safety implementation

   As explained in chapter 1, safety implementations regarding the production line have not been a concern of this thesis. However, some safety protocols were explored before removing them from the work’s scope. The implementation of the robot cell can be further extended by adding safety measures to the cell, and by configuring a safety PLC for the line. The robot controller used during the implementation allows the use of CIP Safety as an Ethernet/IP-based safety interface that connects to a higher-level safety controller.

2. Another improvement that can be implemented is to add a way to transfer the Bezier points to the controller externally, while the robot is in operation. Instead of storing the points in the controller by means of configuration of the code, which requires the robot to be in set-up mode, the Bezier control points should be sent from an external program by using the same TCP/IP connection as the RTU uses.

REFERENCES

[1] IFR, ‘Robots double worldwide by 2020’, *IFR International Federation of Robotics*. [Online]. Available: https://ifr.org/ifr-press-releases/news/robots-double-worldwide-by-2020. [Accessed: 26-May-2019].

[2] B. Böhm *et al.*, ‘Challenges in the engineering of adaptable and flexible industrial factories’, p. 10.

[3] F. Jammes and H. Smit, ‘Service-oriented paradigms in industrial automation’, *IEEE Trans. Ind. Inform.*, vol. 1, no. 1, pp. 62–70, Feb. 2005.

[4] H. R. M. Nezhad, B. Benatallah, F. Casati, and F. Toumani, ‘Web services interoperability specifications’, *Computer*, vol. 39, no. 5, pp. 24–32, May 2006.

[5] ‘Socrades Project 2006-2009’. [Online]. Available: http://www.socrades.net/. [Accessed: 19-May-2019].

[6] X. Wang, L. Xue, Y. Yan, and X. Gu, ‘Welding Robot Collision-Free Path Optimization’, *Appl. Sci.*, vol. 7, p. 89, Feb. 2017.

[7] ‘Industrial robot average selling price 2018 | Statistic’, *Statista*. [Online]. Available: https://www.statista.com/statistics/830578/average-selling-price-of-industrial-robots/. [Accessed: 19-May-2019].

[8] ‘Global industrial robots - leading companies by revenue 2017 | Statistic’, *Statista*. [Online]. Available: https://www.statista.com/statistics/257177/global-industrial-robot-market-share-by-company/. [Accessed: 19-May-2019].

[9] E. M. Rosales, Q. Gan, and J. Gan, ‘Forward and Inverse Kinematics Models for a 5-dof Pioneer 2 Robot Arm’, Jan. 2002.

[10] L. Tirloni, I. Fassi, and G. Legnani, ‘Robot in industrial applications: State of the art and current trends’, 2013, pp. 1–28.

[11] ‘What’s the Difference Between Industrial Robots? | Machine Design’. [Online]. Available: https://www.machinedesign.com/robotics/what-s-difference-between-industrial-robots. [Accessed: 19-May-2019].

[12] F. L. Lewis, C. T. Abdallah, D. M. Dawson, and F. L. Lewis, *Robot manipulator control: theory and practice*, 2nd ed., rev. And expanded. New York: Marcel Dekker, 2004.

[13] W. Khalil and E. Dombre, ‘Chapter 8 - Introduction to geometric and kinematic modeling of parallel robots’, in *Modeling, Identification and Control of Robots*, W. Khalil and E. Dombre, Eds. Oxford: Butterworth-Heinemann, 2002, pp. 171–190.

[14] M. Dehghani, M. Eghtesad, A. A. Safavi, A. Khayatian, and M. Ahmadi, ‘Neural Network Solutions for Forward Kinematics Problem of HEXA Parallel Robot’, *Parallel Manip. New Dev.*, Apr. 2008.

[15] G. Renganathan, ‘Design and Control of 3-DOF Articulated Robotic Arm using LabVIEW and NI-myRIO’, *Int. J. Innov. Res. Electr. Electron. Instrum. CONTROL Eng.*, vol. 3, p. 5, Mar. 2015.

[16] ‘Industrial Robotics Market Forecast - Industry Size, Share Report 2024’. [Online]. Available: https://www.gminsights.com/industry-analysis/industrial-robotics-market. [Accessed: 19-May-2019].

[17] Muhammad Younus, Cong Peiyong, Lu Hu, and Fan Yuqing, ‘MES development and significant applications in manufacturing -A review’, in *2010 2nd International Conference on Education Technology and Computer*, 2010, vol. 5, pp. V5-97-V5-101.

[18] ‘Database Applications - Asitek Oy’. [Online]. Available: http://www.asitek.fi/en/www/1258_database-applications. [Accessed: 19-May-2019].

[19] D. Diep, P. Massotte, and A. Meimouni, ‘A distributed Manufacturing Execution System implemented with agents: the PABADIS model’, 2003, pp. 301–306.

[20] P. M P A Blanco, M. Poli, and M. Pereira-Barretto, ‘Distributed Object Technologies in Manufacturing Execution Systems’, *Prof Mello Moraes*, pp. 2231–5508, May 2019.

[21] MESA International, ‘White Paper 6’. .

[22] H. Meyer, F. Fuchs, and K. Thiel, ‘Manufacturing Execution Systems (MES): Optimal Design, Planning, and Deployment’.

[23] M. H. Ong, R. P. Monfared, S. M. Lee, A. A. West, and R. Harrison, ‘Evaluating the Implementation of the Component-Based System in the Automotive Sector’, pp. 279–287, Jan. 2004.

[24] ‘SOA in Manufacturing Guidebook’. [Online]. Available: https://services.mesa.org/ResourceLibrary/ShowResource/c604a2e1-f3b6-4411-a8f8-dbff278d2b16. [Accessed: 19-May-2019].

[25] A. W. Colombo, F. Jammes, H. Smit, R. Harrison, J. L. M. Lastra, and I. M. Delamer, ‘Service-oriented architectures for collaborative automation’, in *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005.*, 2005, pp. 6 pp.-.

[26] F. Jammes, H. Smit, J. L. M. Lastra, and I. M. Delamer, ‘Orchestration of service-oriented manufacturing processes’, in *2005 IEEE Conference on Emerging Technologies and Factory Automation*, 2005, vol. 1, pp. 8 pp. – 624.

[27] ‘Web Services Glossary’. [Online]. Available: https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice. [Accessed: 19-May-2019].

[28] W. Qifeng and W. Zhangjian, ‘Web Services-based System Integration Approach for Manufacturing Execution System’, in *2011 International Conference on Internet Computing and Information Services*, 2011, pp. 469–472.

[29] L. Canché, M. de J. Ramírez, G. Jiménez, and A. Molina, ‘Manufacturing Execution Systems (MES) Based on Web Services Technology’, *IFAC Proc. Vol.*, vol. 37, no. 5, pp. 135–140, Jun. 2004.

[30] P. Giessler, M. Gebhart, D. Sarancin, R. Steinegger, and S. Abeck, ‘Best Practices for the Design of RESTful Web Services’, *Proc. - Int. Conf. Softw. Eng.*, vol. 10, p. 392, Nov. 2015.

[31] A. Hashemian, S. Hosseini, and s. N. Nabavi, ‘Kinematically smoothing trajectories by NURBS reparameterization – an innovative approach’, *Adv. Robot.*, vol. 31, pp. 1296–1312, Nov. 2017.

[32] H. Fang, S. Ong, and A. Nee, ‘Robot path planning optimization for welding complex joints’, *Int. J. Adv. Manuf. Technol.*, vol. 90, no. 9, pp. 3829–3839, Jun. 2017.

[33] H. N. Fitter, A. B. Pandey, D. D. Patel, and J. M. Mistry, ‘A Review on Approaches for Handling Bezier Curves in CAD for Manufacturing’, *Procedia Eng.*, vol. 97, pp. 1155–1166, Jan. 2014.

[34] K. Kolegain, F. Leonard, S. Chevret, A. Ben Attar, and G. Abba, ‘Off-line path programming for three-dimensional robotic friction stir welding based on Bézier curves’, *Ind. Robot Int. J. Robot. Res. Appl.*, vol. 45, no. 5, pp. 669–678, Aug. 2018.

[35] P. Liljebäck, K. Y. Pettersen, Ø. Stavdahl, and J. T. Gravdahl, ‘A control framework for snake robot locomotion based on shape control points interconnected by Bézier curves’, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 3111–3118.

[36] B. Solemslie, ‘Experimental methods and design of a Pelton bucket’, 2016.

[37] T. Nuntawisuttiwong and N. Dejdumrong, ‘An Approach to Bézier Curve Approximation by Circular Arcs’, in *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2018, pp. 1–6.

[38] G. Xiao and X. Xu, ‘Study on Bezier Curve Variable Step-length Algorithm’, *Phys. Procedia*, vol. 25, pp. 1781–1786, Jan. 2012.

[39] R. Winkel, ‘Generalized Bernstein Polynomials and Bézier Curves: An Application of Umbral Calculus to Computer Aided Geometric Design’, *Adv. Appl. Math.*, vol. 27, no. 1, pp. 51–81, Jul. 2001.

[40] G. Timmerman, ‘Approximating Continuous Functions and Curves using Bernstein Polynomials’, p. 15.
[41] R. Goldman, ‘CHAPTER 7 - B-Spline Approximation and the de Boor Algorithm’, in *Pyramid Algorithms*, R. Goldman, Ed. San Francisco: Morgan Kaufmann, 2003, pp. 347–443.
[42] ‘B-spline Curves: Important Properties’. [Online]. Available: https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/B-spline/bspline-curve-prop.html. [Accessed: 05-May-2019].
[43] M. Pourazady and X. Xu, ‘Direct manipulations of B-spline and NURBS curves’, *Adv. Eng. Softw.*, vol. 31, no. 2, pp. 107–118, Feb. 2000.
[44] J. Pan, L. Zhang, and D. Manocha, ‘Collision-free and smooth trajectory computation in cluttered environments’, *Int. J. Robot. Res.*, vol. 31, no. 10, pp. 1155–1175, Sep. 2012.
[45] M. Elbanhawi, M. Simic, and R. N. Jazar, ‘Continuous-Curvature Bounded Trajectory Planning Using Parametric Splines’, in *IDT/IIMSS/STET*, 2014.
[46] D. Hansford, ‘Chapter 4 - Bézier Techniques’, in *Handbook of Computer Aided Geometric Design*, G. Farin, J. Hoschek, and M.-S. Kim, Eds. Amsterdam: North-Holland, 2002, pp. 75–109.
[47] *Handbook of Computer Aided Geometric Design*. Elsevier, 2002.
[48] L. E. G. Moctezuma, A. Lobov, and J. L. M. Lastra, ‘Free shape paths in industrial robots’, in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 3739–3743.
[49] P. Kaewsaiha and N. Dejdumrong, ‘Modeling of Bézier Curves Using a Combination of Linear and Circular Arc Approximations’, in *Imaging and Visualization 2012 Ninth International Conference on Computer Graphics*, 2012, pp. 27–30.
[50] ‘A Methodology for Piecewise Linear Approximation of Surfaces’, *J. Braz. Comput. Soc.*, vol. 3, no. 3, Apr. 1997.
[51] D. J. Walton and D. S. Meek, ‘Approximation of quadratic Bézier curves by arc splines’, *J. Comput. Appl. Math.*, vol. 54, no. 1, pp. 107–120, Sep. 1994.
[52] ‘FASTory Simulator’. [Online]. Available: http://escop.rd.tut.fi:3000/. [Accessed: 19-May-2019].
[53] A. H. ARTEMIS-IA <andre.hebben@artemis-ia.eu>, ‘Artemis-IA’, *artemis-ia-eu*. [Online]. Available: https://artemis-ia.eu/project/18-esonia.html. [Accessed: 19-May-2019].
[54] A. H. ARTEMIS-IA <andre.hebben@artemis-ia.eu>, ‘Artemis-IA’, *artemis-ia-eu*. [Online]. Available: https://artemis-ia.eu/project/45-escop.html. [Accessed: 19-May-2019].
[55] ‘KR 3 AGILUS’, *KUKA AG*. [Online]. Available: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/kr-3-agilus. [Accessed: 19-May-2019].
[56] ‘Inico Technologies’. [Online]. Available: http://www.inicotech.com/s1000_overview.html. [Accessed: 19-May-2019].

APPENDIX A: ROBOT CONTROLLER CODE

This chapter presents the XML file for the configuration of the TCP server/client connection and the code for the SRC files deployed in the robot controller.

TCPServer.xml

```
<ETHERNETKRL>
        <CONFIGURATION>
                <EXTERNAL>
                        <TYPE>Client</TYPE>
                </EXTERNAL>
                <INTERNAL>
                        <ENVIRONMENT>Program</ENVIRONMENT>
                        <IP>192.168.3.20</IP>
                        <PORT>54601</PORT>
                        <ALIVE Set_Flag="1"/>
                        <PROTOCOL>TCP</PROTOCOL>
                        <MESSAGES Logging="warning" Display="error" />
                </INTERNAL>
        </CONFIGURATION>
        <RECEIVE>
                <RAW>
                        <ELEMENT Tag="Buffer" Type="STREAM" Set_Flag="2"
Size="16" EOS="69,78,68"/>
                </RAW>
        </RECEIVE>
        <SEND>
                <RAW>
                        <ELEMENT Tag="Buffer" Type="STREAM" Size="16"
EOS="69,78,68"/>
                </RAW>
        </SEND>
```

TCPServer.src

```
DEF TCPServer( )
;FOLD Declaration
  INT i
  DECL EKI_STATUS RET
  CHAR Bytes[64]
  BOOL BLUE
  BOOL RED
  BOOL GREEN
  BOOL DRAW1,DRAW2,DRAW3,DRAW4,DRAW5,DRAW6,DRAW7,DRAW8,DRAW9
  REAL valueReal
  BOOL b
  ;EXT beginCurve()
  ;EXT decasteljau()
;ENDFOLD (Declaration)

;FOLD Initialize sample data
 FOR i=(1) TO (64)
  Bytes[i]=0
 ENDFOR
 valueReal=0
;ENDFOLD (Initialize sample data)

; Open communication
RET=EKI_Init("TCPServer")
RET=EKI_Open("TCPServer")

WAIT FOR $FLAG[1]

WHILE $FLAG[1]

;Bytes[]="Stream ends with:"

;RET = EKI_Send("TCPServer",Bytes[])

; Receive data in bytes
WAIT FOR $FLAG[2]
RET=EKI_GetString("TCPServer","Buffer",Bytes[])
$FLAG[2]=FALSE
RET=EKI_ClearBuffer("TCPServer","Buffer")

BLUE=StrComp(Bytes[],"BLUE",#NOT_CASE_SENS)
RED=StrComp(Bytes[],"RED",#NOT_CASE_SENS)
GREEN=StrComp(Bytes[],"GREEN",#NOT_CASE_SENS)
NEWPEN=StrComp(Bytes[],"NEWPEN",#NOT_CASE_SENS)
DRAW1=StrComp(Bytes[],"DRAW1",#NOT_CASE_SENS)
DRAW2=StrComp(Bytes[],"DRAW2",#NOT_CASE_SENS)
DRAW3=StrComp(Bytes[],"DRAW3",#NOT_CASE_SENS)
DRAW4=StrComp(Bytes[],"DRAW4",#NOT_CASE_SENS)
DRAW5=StrComp(Bytes[],"DRAW5",#NOT_CASE_SENS)
DRAW6=StrComp(Bytes[],"DRAW6",#NOT_CASE_SENS)
DRAW7=StrComp(Bytes[],"DRAW7",#NOT_CASE_SENS)
```

```
DRAW8=StrComp(Bytes[],"DRAW8",#NOT_CASE_SENS)
DRAW9=StrComp(Bytes[],"DRAW9",#NOT_CASE_SENS)

; Change pen to blue
IF (BLUE) AND (currPen<>1) THEN
   nextPen=1
   IF (currPen<>0) THEN
      placepen()
   ENDIF
   pickpen()
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF

; Change pen to red
IF (RED) AND (currPen<>2) THEN
   nextPen=2
   IF (currPen<>0) THEN
      placepen()
   ENDIF
   pickpen()
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF

; Change pen to green
IF (GREEN) AND (currPen<>3) THEN
   nextPen=3
   IF (currPen<>0) THEN
      placepen()
   ENDIF
   pickpen()
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF

; Get new pen
IF (NEWPEN) AND (currPen<>0) THEN
   discard()
   newpen()
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF

; Draw curve
IF (DRAW1) AND (currPen<>0) THEN
   Bytes[]="WORKING"
   RET = EKI_Send("TCPServer",Bytes[])
   beginCurve(1)
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF

IF (DRAW2) AND (currPen<>0) THEN
```

```
   Bytes[]="WORKING"
   RET = EKI_Send("TCPServer",Bytes[])
   beginCurve(2)
   Bytes[]="IDLE"
   RET = EKI_Send("TCPerver",Bytes[])
ENDIF

IF (DRAW3) AND (currPen<>0) THEN
   Bytes[]="WORKING"
   RET = EKI_Send("TCPServer",Bytes[])
   beginCurve(3)
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF

IF (DRAW4) AND (currPen<>0) THEN
   Bytes[]="WORKING"
   RET = EKI_Send("TCPServer",Bytes[])
   beginCurve(4)
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF

IF (DRAW5) AND (currPen<>0) THEN
   Bytes[]="WORKING"
   RET = EKI_Send("TCPServer",Bytes[])
   beginCurve(5)
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF

IF (DRAW6) AND (currPen<>0) THEN
   Bytes[]="WORKING"
   RET = EKI_Send("TCPServer",Bytes[])
   beginCurve(6)
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF

IF (DRAW7) AND (currPen<>0) THEN
   Bytes[]="WORKING"
   RET = EKI_Send("TCPServer",Bytes[])
   beginCurve(7)
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF

IF (DRAW8) AND (currPen<>0) THEN
   Bytes[]="WORKING"
   RET = EKI_Send("TCPServer",Bytes[])
   beginCurve(8)
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF
```

```
IF (DRAW9) AND (currPen<>0) THEN
   Bytes[]="WORKING"
   RET = EKI_Send("TCPServer",Bytes[])
   beginCurve(9)
   Bytes[]="IDLE"
   RET = EKI_Send("TCPServer",Bytes[])
ENDIF

ENDWHILE

RET=EKI_Close("TCPServer")
RET=EKI_Clear("TCPServer")
END
```

PickPen

```
DEF pickpen( )

DECL E6POS XPick

IF nextPen==1 THEN
   XPick=Xblueapr
ENDIF

IF nextPen==2 THEN
   XPick=Xredapr
ENDIF

IF nextPen==3 THEN
   XPick=Xgreenapr
ENDIF

WAIT SEC 0.5
$OUT[2]=TRUE

$BASE=$NULLFRAME
PTP XPick

WAIT SEC 0.5
$OUT[1]=TRUE

LIN_REL {X 85}

WAIT SEC 0.5
$OUT[1]=FALSE

LIN_REL {Z 100}

currPen=nextPen

WAIT SEC 0.5
$OUT[2]=FALSE

END
```

PlacePen

```
DEF placepen()

DECL E6POS XPlace

IF currPen==1 THEN
   XPlace=Xplaceblueapr
ENDIF

IF currPen==2 THEN
   XPlace=Xplaceredapr
ENDIF

IF currPen==3 THEN
   XPlace=Xplacegreenapr
ENDIF

WAIT SEC 0.5
$OUT[2]=TRUE

$BASE=$NULLFRAME
PTP XPlace

LIN_REL {Z -100}

WAIT SEC 0.5
$OUT[1]=TRUE

LIN_REL {X -100}

WAIT SEC 0.5
$OUT[1]=FALSE

currPen=0

WAIT SEC 0.5
$OUT[2]=FALSE

END
```

Discard

```
DEF  discard ( )


$BASE=$NULLFRAME
PTP discPos

LIN_REL {Z -300}

WAIT SEC 0.5
$OUT[1]=FALSE

END
```

NewPen

```
DEF newpen()

DECL E6POS XNew

IF currPen==1 THEN
   XNew=Xnewblue
ENDIF

IF currPen==2 THEN
   XNew=Xnewred
ENDIF

IF currPen==3 THEN
   XNew=Xnewgreen
ENDIF

WAIT SEC 0.5
$OUT[2]=TRUE

discard()

$BASE=$NULLFRAME
PTP Xmidpoint

$BASE=$NULLFRAME
PTP XNew

WAIT SEC 0.5
$OUT[1]=TRUE

LIN_REL {X 50}

WAIT SEC 0.5
$OUT[1]=FALSE

LIN_REL {Z 100}

WAIT SEC 0.5
$OUT[2]=FALSE

END
```

BeginCurve

```
DEF beginCurve(drawNum:IN)

E6POS startdraw
DECL INT drawNum
DECL INT curveNum
DECL INT I,J
DECL BOOL b
DECL REAL bPoints[50]
;EXT decasteljau()

curveNum=0

SWITCH drawNum
   CASE 1
      ; Get the Bezier points for drawing 1 and number of curves
      FOR J=1 TO 50
         bPoints[J]=BEZ1[J]
      ENDFOR
      curveNum=BEZ1Num

   CASE 2
      ; Get the Bezier points for drawing 2 and number of curves
      FOR J=1 TO 26
         bPoints[J]=BEZ2[J]
      ENDFOR
      curveNum=BEZ2Num

   CASE 3
      ; Get the Bezier points for drawing 3 and number of curves
      FOR J=1 TO 26
         bPoints[J]=BEZ3[J]
      ENDFOR
      curveNum=BEZ3Num

   CASE 4
      ; Get the Bezier points for drawing 4 and number of curves
      FOR J=1 TO 8
         bPoints[J]=BEZ4[J]
      ENDFOR
      curveNum=BEZ4Num

   CASE 5
      ; Get the Bezier points for drawing 5 and number of curves
      FOR J=1 TO 8
         bPoints[J]=BEZ5[J]
      ENDFOR
      curveNum=BEZ5Num

   CASE 6
      ; Get the Bezier points for drawing 6 and number of curves
      FOR J=1 TO 8
```

```
         bPoints[J]=BEZ6[J]
      ENDFOR
      curveNum=BEZ6Num

   CASE 7
      ; Get the Bezier points for drawing 7 and number of curves
      FOR J=1 TO 8
         bPoints[J]=BEZ7[J]
      ENDFOR
      curveNum=BEZ7Num

   CASE 8
      ; Get the Bezier points for drawing 8 and number of curves
      FOR J=1 TO 8
         bPoints[J]=BEZ8[J]
      ENDFOR
      curveNum=BEZ8Num

   CASE 9
      ; Get the Bezier points for drawing 9 and number of curves
      FOR J=1 TO 8
         bPoints[J]=BEZ9[J]
      ENDFOR
      curveNum=BEZ9Num

ENDSWITCH

$BASE=BASE_DATA[1]
PTP XP1

startdraw={X 0, Y 0, Z 0}
startdraw.X=bPoints[1]
startdraw.Y=bPoints[2]

PTP_REL startdraw

$ORI_TYPE=#CONSTANT

LIN_REL {Z -100}

FOR I=0 TO (curveNum-1)

decasteljau(bPoints[6*I+1],bPoints[6*I+2],bPoints[6*I+3],bPoints[6*I+4
],bPoints[6*I+5],bPoints[6*I+6],bPoints[6*I+7],bPoints[6*I+8])
ENDFOR

LIN_REL {Z 100}

END
```

DeCasteljau

```
  DEF                                                         decasteljau
(BEZ1_X:IN,BEZ1_Y:IN,BEZ2_X:IN,BEZ2_Y:IN,BEZ3_X:IN,BEZ3_Y:IN,BEZ4_X:IN
,BEZ4_Y:IN)

  FOR t=50 to 100 STEP 50
     PastPos=$POS_ACT
     ;Find next position in x
     INTER_A_X=(100-t)/100.0*BEZ1_X+t/100.0*BEZ2_X
     INTER_B_X=(100-t)/100.0*BEZ2_X+t/100.0*BEZ3_X
     INTER_C_X=(100-t)/100.0*BEZ3_X+t/100.0*BEZ4_X
     INTER_AB_X=(100-t)/100.0*INTER_A_X+t/100.0*INTER_B_X
     INTER_BC_X=(100-t)/100.0*INTER_B_X+t/100.0*INTER_C_X
     POS_X=(100-t)/100.0*INTER_AB_X+t/100.0*INTER_BC_X

     POS_X=POS_X-PastPos.X

     ;Find next position in y
     INTER_A_Y=(100-t)/100.0*BEZ1_Y+t/100.0*BEZ2_Y
     INTER_B_Y=(100-t)/100.0*BEZ2_Y+t/100.0*BEZ3_Y
     INTER_C_Y=(100-t)/100.0*BEZ3_Y+t/100.0*BEZ4_Y
     INTER_AB_Y=(100-t)/100.0*INTER_A_Y+t/100.0*INTER_B_Y
     INTER_BC_Y=(100-t)/100.0*INTER_B_Y+t/100.0*INTER_C_Y
     POS_Y=(100-t)/100.0*INTER_AB_Y+t/100.0*INTER_BC_Y

     POS_Y=POS_Y-PastPos.Y

     MyPos={X 0, Y 0}

     ;Change values of position in x and y
     MyPos.X=POS_X
     MyPos.Y=POS_Y

     ;Move to next position
     $ORI_TYPE=#CONSTANT
     LIN_REL MyPos
  ENDFOR

  END
```

APPENDIX B: ST PROGRAMS ON S1000

This chapter shows an example of the program that creates the connection between the S1000 and the robot. There is one ST program for each of the service request types, with the message content being the only difference between them. The following is the code for requesting a change of pen to blue:

```
PROGRAM blueST
VAR
    connStatus:int;
     index:int;
     response:int;

END_VAR

    connStatus:=netconn_status(netconn);
    if connStatus<>0 then
          netconn_open(netconn);
          wait(3000);
     end_if;

     (* send message to the robot *)

     connStatus:=netconn_status(netconn);
     if connStatus=0 then
          netconn_write( netconn, 'BLUEEND' );
         penColor:=1;
     end_if;

     (* receive message from the robot *)

   index := 0;
   response:=0;
   WHILE (index < 30 AND response = 0 ) DO

      IF netconn_avail( netconn ) > 0 THEN
          netconn_read( netconn , rob_status , 10 );
            response := 1;
      END_IF;

       wait(1000);
       index := index + 1;
   END_WHILE;

rest_respond(ChangeBLUE);

END_PROGRAM
```

APPENDIX C: RESTFUL APIS

The following table contains the RESTful APIs of the robot RTU in workstation 3 of the FASTory line.

Service ID	Method	URL	Body
ChangeBLUE	POST	http://192.168.3.1/rest/services/ChangeBLUE	{"destUrl":""}
ChangeRED	POST	http://192.168.3.1/rest/services/ChangeRED	{"destUrl":""}
ChangeGREEN	POST	http://192.168.3.1/rest/services/ChangeGREEN	{"destUrl":""}
NewPen	POST	http://192.168.3.1/rest/services/NewPen	{"destUrl":""}
Draw1	POST	http://192.168.3.1/rest/services/Draw1	{"destUrl":""}
Draw2	POST	http://192.168.3.1/rest/services/Draw2	{"destUrl":""}
Draw3	POST	http://192.168.3.1/rest/services/Draw3	{"destUrl":""}
Draw4	POST	http://192.168.3.1/rest/services/Draw4	{"destUrl":""}
Draw5	POST	http://192.168.3.1/rest/services/Draw5	{"destUrl":""}
Draw6	POST	http://192.168.3.1/rest/services/Draw6	{"destUrl":""}
Draw7	POST	http://192.168.3.1/rest/services/Draw7	{"destUrl":""}
Draw8	POST	http://192.168.3.1/rest/services/Draw8	{"destUrl":""}
Draw9	POST	http://192.168.3.1/rest/services/Draw9	{"destUrl":""}

Service ID	Method	URL	Body
RobBusy	POST	http://192.168.3.6/ROB/busy	{"destUrl":""}
RobIdle	POST	http://192.168.3.6/ROB/idle	{"destUrl":""}
RobError	POST	http://192.168.3.6/ROB/error	{"destUrl":""}
RobCalibration	POST	http://192.168.3.6/ROB/calibration	{"destUrl":""}