UNIVERSITÀ
DI TRENTO

Department of Information Engineering and Computer Science

Bachelor's Degree in
Information, Communication and Electronic Engineering

Final Dissertation

Remote controller for kuka robot

Supervisor

Luigi Palopoli

Student

Federico Adami

Academic year 2022/2023

Acknowledgements

I would like to thank my University supervisor, Prof. Luigi Palopoli for guiding me during the writing of this thesis. I would also like to extend my deepest appreciation to my company tutors Matteo Morelli, Paolo Bevilacqua, Fabiano Zenatti and to all my colleagues at Polytec Intralogistics for their support, advices and friendly working environment, that really motivated me. I am deeply grateful to my family, for their love and encouragement during this years. To my girlfriend, Sara, I want to express my profound gratitude to have found a such strong and loving person that has always supported me and given insightful advices. To my colleagues and friends, Eddie Veronese and Francesco Olivieri, the best team I could ever ask for, with whom I accomplished lots of successes. Lastly I want to express my sincere thanks to my friends for the laughter and lightheartedness, much needed in life.

Contents

Abstract

This bachelor degree thesis focuses on the development and integration of a system to remotely operate an anthropomorphic robot inside a cell for medicine sorting. This project has been developed during the curricular internship at the robotics company Polytec Intralogistics, founded by three researchers of the university of Trento. This project aimed at developing a system that would enable technicians to directly operate the robot inside the cell for testing purposes and to solve remotely situations that otherwise would take longer needing their presence on site. The explanation of the development process will cover all the three software layers of the architecture. Starting from the one nearest to the robotic arm, developed in a proprietary language called Kuka Robot Language, then will be explained the challenges and solutions implemented in the development of the middle layer called the "supervisor" that controls the correct movement and orientation of the robot's end effector inside predetermined areas. Lastly will be explained the development of the UI that would then be provided to the technicians to operate and monitor the robot status and movements.

1 Introduction

1.1 Background

The development of the following project took place during a curricular internship at "Polytec Intralogistics", previously known as "Dolomiti Robotics". Born as a startup of three researchers of the University of Trento, this company works in the robotics field and offers a wide variety of products for industrial automation and industry 4.0. Their product portfolio ranges from a smart cleaner for drugstore shelves, a robotic cell sorting medicines, to mobile robots for the transport and management of material in industrial warehouses. Furthermore they offer the integration and customization of their automatic systems based on their clients specific needs. These products are oriented towards companies that have to handle big warehouses and drugstores that need to keep their inventory organized.

1.2 The robotic cell

In my internship I worked on the robotic cell that would sort medicines. This machine is part of a bigger product that not only has the purpose to sort medicines but also to organize them into shelves by kind and by expiry date. This shelf is synched with a database to keep track of the inventory. So the whole machine is composed of two parts. One with a robotic arm (KUKA KR4 R600) that thanks to a series of belts, rotative boards and cameras identifies objects and expiration dates. Then there's a second part, a cartesian robot, that based on the information gathered by the robotic cell grabs the boxes and places them in an ordered manner on the shelves. Polytec Intralogistics is not in charge of the development of the cartesian robot but it is in charge of the development of the robotic cell.

The physical layout of the cell is divided in different areas or bases as shown in the picture below. Moreover each base has a reference frame assigned that determines the coordinates relative to the base. By following the path that a medicine package follows from the moment it enters the cell to the moment it is grabbed by the cartesian robot, we can analyze the various stages of the process.

Firstly all the various boxes are feeded into a hopper that loads them onto a conveyor or moving belt. These products then reach an area called "BELT" in figure 1.1 where a first set of cameras detects the boxes size, and pose. All this information is then used by the kuka robot to pick each single box and move it to the next base, the "ROTATIVE BOARD". Here as the name suggests each medicine is rotated by a rotating plate and all of its faces get scanned by a camera to classify the medicine and read the expiration date. Once a box has been correctly scanned it is picked up by the cartesian robot and placed on the shelf. In case something goes wrong during this entire sequence, the robot can pick the box and place it again on the

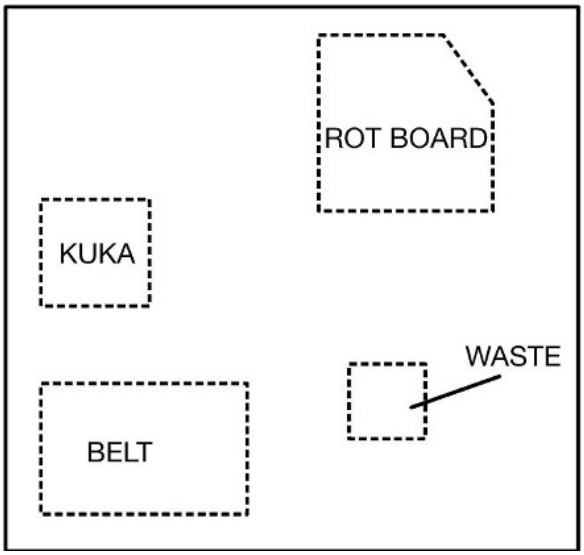

Figure 1.1: layout scheme of the robotic cell, the "KUKA" label represents the position of the base of the Kuka robot.

BELT base to restart the process or in extreme cases it can discard the box by putting it in the WASTE area.

The various operations and functions of the cell are performed by programs running on different computers and boards. Each board executes a program that corresponds to one of the layers of the software architecture that will be shown in the next chapter. Starting from the layer which is nearest to hardware we find the "dolomiti board" that controls lights, camera streams and motor actuators of the belts and rotative board. At the same level we find the KUKA controller KR C5 which is the computer which controls the movements of the Kuka robot. Going one level up we find the supervisor board which coordinates the operations of the cell and then one level more we have a windows computer that runs a user interface. All these boards are connected to each other via ethernet cable and communicate via TCP/IP protocol.

1.3 Problem statement

During the standard operating of the robotic cell, some situations may require physical intervention of an operator. For example, some boxes could get overlapped in the moving belt and even the automatic shaking of the belt couldn't spread them enough to allow camera detection; moreover after the first installation of the cell it might be necessary to test all the movements of the robot to ensure that all has been installed properly. All of these situations may lead to a halt of the machine, waiting for the errors to be dispatched and because these issues can only be solved by an operator being on site, the robotic cell may not work for hours causing a reduction in productivity. The objective of my work during the internship was the development of a software that would enable the remote control of the Kuka robot in the cell by providing a joystick-like interface to operate the arm. This way many of the scenarios that would require a human intervention could be solved remotely by an operator in a shorter period.

The development of the software had various requirements to be satisfied. The first one was that it needed to allow the safe manual movement of the robot, this would include the definition of areas in which the robot end effector could or couldn't move in order to avoid cell obstacles and arm singularities. Secondly it was necessary to build the software in a manner that it would be easily integrable and would not interfere with the already existing program that controls the standard operations of the cell. Moreover it was necessary to implement some basic functionalities to manipulate objects, in particular commands for picking and releasing objects.

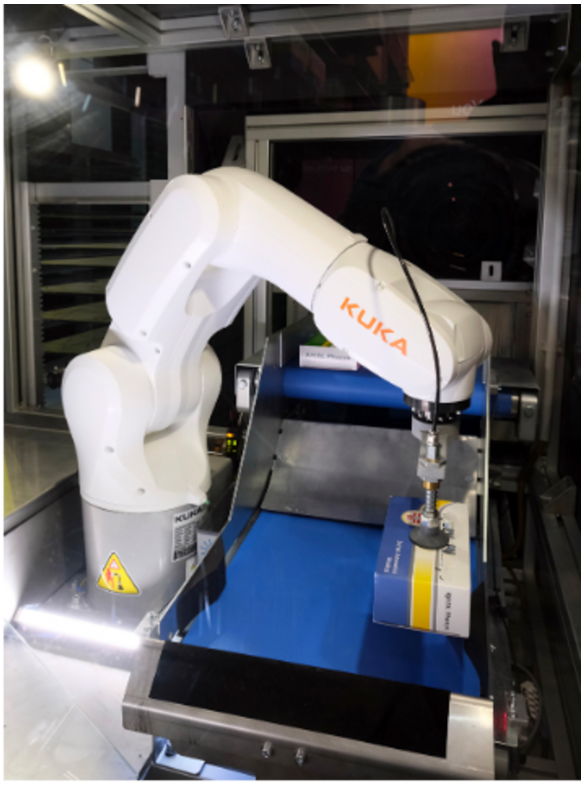

Figure 1.2: The Kuka robot operating in the cell

1.4 Software architecture

In the following chapters I will discuss in detail my implementation while right now we will focus on the structure of the already existing software that runs the cell and we will understand the functions of each part. It is important to keep in mind that my software added functionalities to the programs that control the cell and it was conceived to be completely integrable with the already existing softwares, so the architecture remains mostly the same.

1.4.1 KUKA movimentation software

This is the lowest layer of software. It runs on the Kuka KR C5 computer and it is written in Kuka Robot Language. It is in charge of two main tasks, receiving instructions from the supervisor program and then sending the actions to execute to the actual anthropomorphic arm. All the communications happen through TCP/IP protocol.

1.4.2 Supervisor software

This is the program that as the name suggests controls all the main operations in the cell, it was developed using behavior trees due to the high complexity of the operations it has to coordinate. Based on the inputs received and the behavior tree status, it sends instructions to the various peripherals. It receives data about detected boxes and maintains a database which associates each box to its pose and dimensions. Then sends instructions on how the KUKA robot needs to move, can control the rotative board and the conveyor belt and can handle unexpected events based on the feedback it receives.

1.4.3 User interface

This one last software layer runs on a windows computer. It displays an interface with many tabs each that provides some kind of information. Its main purpose is to make debugging easier and show developers and technicians useful data on the status of the operations and on the errors that arise. In addition it allows connection to the video stream coming from the webcams placed around the cell to have live feedback on what is happening. This UI is not only passive but also allows small interactions with the cell by turning on the lights or controlling the belt conveyor through some buttons.

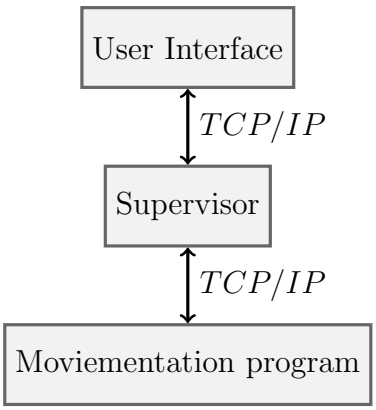

Figure 1.3: Software layers

2 The robotic system

2.1 Rotation matrices and euler angles representation

A manipulator can be schematically represented from a mechanical viewpoint as a kinematic chain of rigid bodies (links) connected by means of revolute or prismatic joints, one end of the chain is constrained to a base, while an end-effector (EF) is mounted to the other end [6]. Therefore, in order to manipulate an object in space, it is necessary to define the direct and inverse kinematic equations which describe the end-effector position and orientation as a function of the joint variables and vice versa. This allows the end-effector position and orientation (pose) to be expressed as a function of the joint variables of the mechanical structure with respect to a reference frame and with the inverse kinematics the joint variables can be described as a function of the EF pose. The frames associated of each link are expressed by roto translational matrices, in our case we won't need to define the direct and inverse kinematics because this is already embedded in the Kuka robot controller KR C5 and it is not accessible, but still it will be necessary to understand the notion of rotation matrices and euler angles representation to compute and communicate the desired pose for the EF. A rigid body is completely described in space by its pose defined by position and orientation to a reference frame. Given a position in space O' it can be written as a 3x1 vector o'

$$o' = \begin{bmatrix} o'_x \\ o'_y \\ o'_z \end{bmatrix}$$

where its components are the coordinates of the point along the reference frame axis. To describe the orientation it is convenient to consider an orthonormal frame attached to the body and express its unit vectors with respect to the reference frame. Let then O'-x'y'z' be such a frame with origin in O' and x', y', z' be the unit vectors of the frame axes. These vectors are expressed with respect to the reference frame O-xyz by the equations:

$$x' = x'_x x + x'_y y + x'_z z$$

$$y' = y'_x x + y'_y y + y'_z z$$

$$z' = z'_x x + z'_y y + z'_z z$$

The components of each unit vector are the direction cosines of the axes of frame O'-x'y'z' with respect to the reference frame O-xyz. The three unit vector describing the body orientation with respect to the reference frame can be combined into a 3x3 matrix which is termed rotation matrix [6].

$$R = \begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x'_x & y'_y & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix}$$

Rotation matrices have a redundant description of frame orientation; in fact they are characterized by nine elements which are not independent but related by six constraints due to the orthogonality conditions. Therefore three parameters are sufficient to describe orientation of rigid body in space, this is called minimal representation of the orientation and can be obtained by using a set of three angles $E = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$. Consider the rotation matrix expressing the elementary rotation about one of the coordinate axes as a function of a single angle. Then a generic rotation matrix can be obtained by composing a suitable sequence of three elementary rotations while guaranteeing that two successive rotations are not made about parallel axes. In the project was considered the set or Euler Angles ZYX (Roll-Pitch-yaw) which indicates that the rotations take place firstly about Z-axis, then about the newly obtained Y-axis and lastly about the X-axis altered by the previous rotations [6].

2.2 KUKA robot description

The robot system KR 4 R600 comprises different components:

- the manipulator (1)
- smartPAD-2 (2)
- Connecting cable, smartPAD-2 (3)
- Robot controller (4)
- Connecting cable, data cable (5)
- Connecting cable, data cable (6)

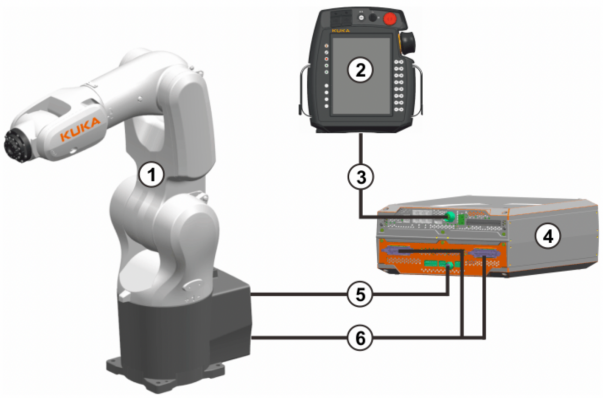

Figure 2.1: the KR 4 R600 system

The manipulator Figure 2.2 is a 6 degrees of freedom anthropomorphic arm with an in-line wrist, made up of cast light alloy, with each axis fitted with a brake. All motor components and connecting cables are protected against dirt and moisture under the cover plates. The robot movements are controlled by the Robot movimentation KR C5 PC. To this component are both connected the manipulator, the smartPAD and an external computer.

The smartPAD KUKA Figure 2.3 is used to control and command the robot. It has different features, the most relevant ones are the red emergency button (3) which actuates all the brakes of all the joints and can be used to stop the robot in any situation. The 6D mouse (4) is a tool used to operate the robot manually. Then we have a black switch (2) which allows the selection of the operating mode. There are two main modes which were the ones used during the testing of the software.

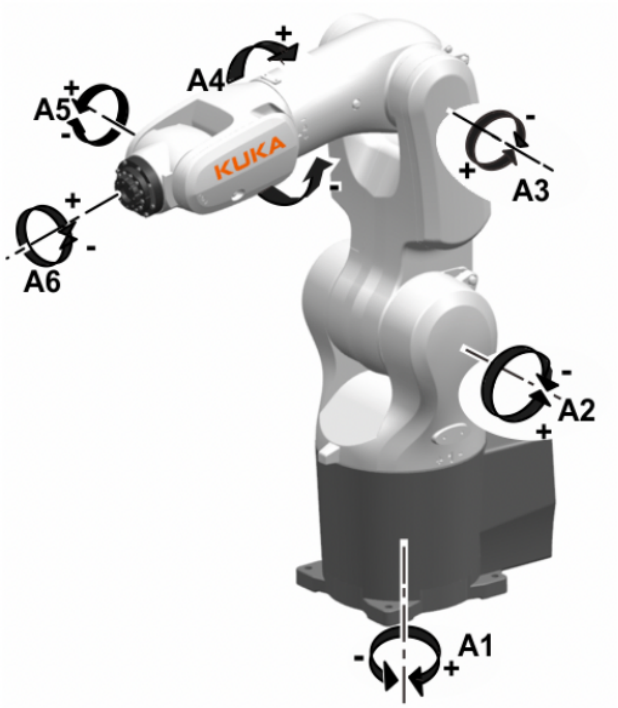

Figure 2.2: view of Kuka robot axis

Figure 2.3: smartPAD

Figure 2.4: end effector tool attached to the wrist joint

The T2 mode is used for testing new software and for the manual movement of the robot with the 6D mouse. In this case the maximum speed that the robot can move at is lower than the speed during normal operations. The second mode is the AUT mode which operates the robot at programmed velocity without any constraints, while the manual movement with the 6D mouse is not allowed.

The last component of the robot that was not part of the robot system KR 4 R600 is the end effector (EF) tool Figure 2.4 attached to the wrist of the robot. It is an L shaped tool that at the tip has a suction cup connected to a small pipe attached to a pump that would suck air to create vacuum and pick an object. The object could then be released by letting air in by opening a valve connected to the suction cup.

2.3 Movement instructions

Different kinds of movement are available depending on the task the robot needs to accomplish. There are two main kinds of instruction movements, referred to the axis (SPTP) and based on trajectory (SLIN, SCIRC). The movements based on trajectory are subject to singular positions. Singular positions are characterized by the impossibility of converting the cartesian coordinates (working space coordinates) into axis values. Singularities may even occur when small changes

in the working space produce excessive variations of the axis values [7].

2.3.1 Spline Point to Point - SPTP

This movement is referred to the robot axis and the Kuka moves the Tool Center Point (TCP) along the fastest movement to the destination point. Usually being this the fastest trajectory, it doesn't correspond to a straight line. This is explained by the fact that the axes of the robot rotate and curvilinear trajectories can be followed much faster than straight ones. One downside of this approach is that the exact trajectory followed by the TCP is unpredictable, it can be made more predictable by placing some known intermediate points. This movement finds usage in cases where the trajectory is not relevant and it must happen fast like transport of material or soldering of points [7].

2.3.2 Linear movement - SLIN

In this case the TCP follows a straight line and it is guided along the trajectory with constant velocity and with the defined orientation. The applications for this kind of movement are continuous soldering, laser cutting and gluing [7].

2.3.3 Circular movement - SCIRC

In this case the circular movement is defined from the starting point, an intermediate point and the destination point. The TCP is still moved with constant velocity and defined orientation. This movement is suitable for soldering precise curvatures [7].

3 Implementation of movimentation software

3.1 Kuka Robot Language

The Kuka Robot Language (KRL) is a proprietary programming language similar to Pascal and used to control KUKA robots. Any KRL code consists of two different files with the same name: a permanent data file, with the extension .dat, and a movement command file, with the extension .src. This language provides different motion commands to execute the different movements described in the previous chapter. These commands include SPTP, SLIN, SCIRC. The motion commands support different kinds of data structures that specify the pose of the destination point [5]:

- FRAME {X 10, Y 0, Z 500, A 0, B 0, C 0}
- POS {X 10, Y 0, Z 500, A 0, B 0, C 0, S 6, T 21}
- AXIS {A1 0, A2 -90, A3 90, A4 0, A5 0, A6 0}

The last one specifies the angle that each joint needs to reach.

3.2 Workvisual suite

WorkVisual is a PC software, running on Windows, used to configure and program KUKA KR robots. Although the writing of robotic programs can be still done on the smartPad, this software allows better tweaking of global variables and has better debug features that simplify the development phase.

3.3 Implementation

At this level there are two main softwares that are running. One is a subroutine in charge of the communications with the supervisor software (CommSUB) and then there's the actual program (Main) that sends the specific instructions based on the data received by the other subroutine. In this case the communications subroutine was already implemented so I won't describe it in depth but my main objective was to add functionalities to the movimentation program that sends instructions to the robot to implement my remote control system. Once a message arrives the communications subroutine parses it and assigns the respective values to the corresponding variables. The movimentation program consists in an infinite while loop that waits for a signal from the communication subroutine that a message has been received.

Then the variable sPrimitive[] which contains the string of the type of action to be executed is checked and based on its value the corresponding function is called. We have four main functions:

- **Hang**
  GLOBAL DEFECT INT Hang (iToolNo: IN, iBaseNo: IN, rApprIn[]:OUT, rApprOut[]: OUT, rSpeed: IN, fObjFrame: IN)

  This function indicates the robot to go to the base specified by the message received (iBaseNo). The movement to go from a base to another is made with a Spline Point To Point (SPTP) instruction, to ensure the fastest movement to the end position:

  SPTP pPoint WITH $VEL AXIS[1] = SVEL JOINT (rSpeed), $TOOL = TOOL DAT[iToolNo], $BASE = BASE DATA[iBaseNo]

- **MoveStep**
  GLOBAL DEFFCT INT MoveStep(iToolNo: IN, iBaseNo: IN, rSpeed: IN, fStepFrame: IN)

  This function is used to allow the movement of the robot inside a base. With the variable iBaseNo we are specifying in which base we are currently on and with fStepFrame we specify the 3d step vector that indicates the step to take with respect to the origin frame of the base. The movement that the EF follows is a straight line to the desired position (fStepFrame)and we use the Spline Linear instruction:

  SLIN fStepFrame WITH $TOOL = TOOL_DATA[iToolNo], $BASE = BASE_DATA[iBaseNo]

- **Release**
  GLOBAL DEFFCT INT release ()

  With this function we don't have input variables. This function opens the valve that is attached to the suction cup placed on the tip of the EF. When a box is grabbed from the robot vacuum is created by a pump connected to the suction cup, if we let air in we don't have vacuum anymore and the box is released. Therefore by opening the valve connected to the suction cup we release the object. To open the vacuum valve we need to set OUT[1] variables follows:

  $OUT[1]= FALSE

- **PickStep**

  GLOBAL DEFFCT IN PickStep(iToolNo:IN, iBaseNo:IN, rSpeed: IN, fStepFrame: IN)

  This function is used when we want to grab a box. We assume that the EF has been positioned above the desired box. Then we call the function ReachToPickZ that makes

the EF go down towards the plane of the base, the pump is activated and once vacuum is detected an interrupt is sent to signal that an object has been grabbed.

```
GLOBAL DEF ReachToPickZ(pPoint_:IN, rSpeed:IN)
   DECL FRAME pPoint_
   DECL REAL rSpeed
   pPoint_.Z = MaxR(0.75*pPoint_.Z,5)
   INTERRUPT ON 10 ;ARM ON the interrupt

   CONTINUE ; In alternativa si puo utilizzare TRIGGER (vedi sotto)
   $OUT[1] = TRUE   ; activate vacuum
   ;TRIGGER WHEN DISTANCE = 0 DELAY = 0 DO $OUT[1] = TRUE

   ; this is the movement that is expected to be interrupted!
   SLIN pPoint_ WITH $VEL = SVEL_CP(2.0*0.01*rSpeed, , LCPDAT1)

   INTERRUPT OFF 10 ;ARM OFF interrupt

   ;MsgNotify("Reached end of movement without triggering interrupt")

   WAIT SEC 0.100 ; this is compulsory: break Advance Run Pointer (ARP).
   ; This is needed to ensure the ARP does not
   ;get back to the top level program before the interrupt fires

END ;ReachItem
```

Then the robot returns to the position it had before picking the object. The movement of the EF is linear and it is performed with the SLIN instruction. The descent velocity is controlled by setting the VEL variable at 1% of the maximum speed of the robot.
$VEL = SVEL_CP(2.0*0.01*rSpeed, , LCPDAT1)

3.4 Testing

This layer was't tested as soon as it was implemented but all the debugging and thest of the functionalities was performed once its communication was established with the supervisor layer. This is because the supervisor had a terminal interface to test instructions and so the testing resulted easier to be performed simultaneously on the supervisor and on the movimentation program.

4 Implementation of supervisor software

4.1 Behaviour trees

In this software layer, the function that would then allow to remotely control the robot will become part of a node of the behavior tree of the supervisor software that controls the cell. "A Behavior Tree (BT) is a way to structure the switching between different tasks in an autonomous agent, such as a robot or a virtual entity in a computer game." Their concept is similar to Finite State machines but BTs are a more efficient way of creating complex systems that are both modular and reactive. Behavior trees have many advantages such as intrinsic hierarchy, in fact complex behaviors can be composed by making entire trees sub branches of a bigger tree. Moreover their graphical representation has a semantic meaning, so having all the logic of a system concentrated in one point allows developers to have a better understanding of the system and facilitate the debugging of errors. Last main advantage is the modularity, composability and good separation of concerns [3].

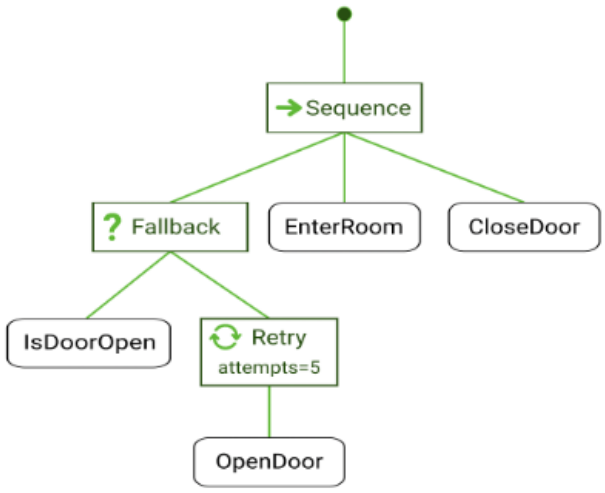

Figure 4.1: behaviour tree

4.1.1 Basic functioning

All starts when a signal called "tick" is sent to the root of the tree and propagates through the tree until it reaches a leaf node. Any treeNode that receives a tick executes its callback. This callback must return either success, failure and running. Running means the action needs more time to return a valid result. if a node has one or more children, it has the responsibility to propagate the tick and each node might have different rules on how to propagate it. The leaf nodes are the actual commands by which the BT interacts with the system [3].

4.1.2 Types of nodes

There are different types of nodes. We have the control node that usually ticks a child based on the result of its sibling and/or its own state. The decorator node must have just one child and it may alter the result of the child or tick it multiple times. Then we have the leaf nodes, the condition node used to verify a condition and neither should alter the system nor return running status. The other leaf node is the ActionNode which is the node that performs the

action [3].

4.2 Implementation

To allow a better understanding of the functioning and implementation of this layer the explanation will follow the process that goes from the arrival of an instruction message from the UI, to the sending of the message to the KUKA movimentation program. The communication between UI and this layer will be covered in the UI implementation chapter. Everything starts from a message received from the UI, which is parsed, the parameters are assigned to the corresponding variables. The obtained parameters are passed to a function of the Logic class **bool Logic::execMsgInstruction(Action_t action, Base base, Vec3 pos)**. Important to note that the Action_t action variable can assume four types of values, MOVE, PICK, RELEASE, CHANGEBASE. Each one of these actions has a direct correspondance to the four actions that were implemented in the KUKA movimentation program, respectively MoveStep, PickStep, Release, Hang. In the execMsgInstruction function we have the composition of a variable called nextMovement of the class movement which contains all the parameters to define a movement, the main ones are:

- **type** of action (Action_t action)
- **base** in which the movement will take place (Base base)
- **position** the robot will reach at the end of the movement (Vec3 pos)
- **orientation** of the end effector.

The first three values were given by the received message but the orientation of the end effector needs to be carefully calculated.

4.2.1 Computation of EF orientation

As we have seen from the structure overview of the robotic system we observed that the tip of the EF (the suction cup) is not in line with the wrist axis of the robot. Because of this it may happen that by placing the EF near an obstacle the robot wrist would collide with it. Moreover if the EF is not properly positioned it may happen that the robot collides on itself or reaches a singularity before than desired. Therefore depending on the base requirements we need to define a way to compute the EF orientation. The EF will be always kept in vertical position (suction cup face parallel to the ground) and so it will only be necessary to compute the angle about the Z axis.

Computation in belt base

From the image of the belt base Figure 4.2 we can observe have two important parts. The first one, the upper side of the image, is the slide which has steel side walls and then there's the flat part, lower part of the image, which is the conveyor belt. In computing the EF orientation we have a few requirements to satisfy. First one is to keep the wrist center away from steel

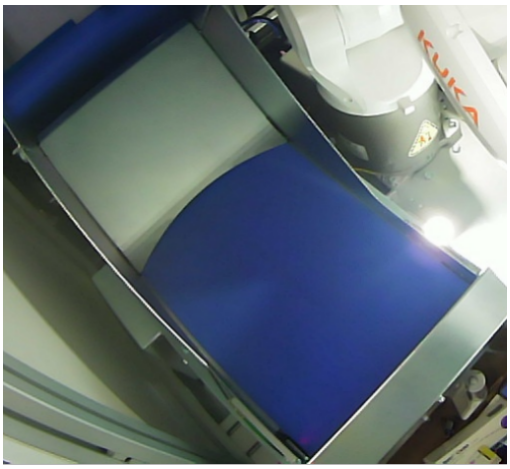

Figure 4.2: top view of the belt base

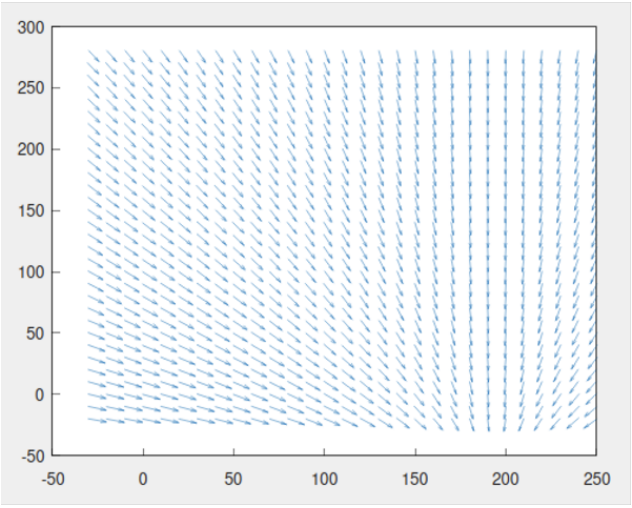

Figure 4.3: simulation on how the EF should be oriented in the belt base (the top part of the graph corresponds to the upper left section of Figure 4.2) where the slide is placed

walls (bulkheads), then we need to keep the wrist as far as possible from the base frame of the robot (in the up-right corner of Figure 4.2), otherwise it might collide with itself and so limiting the movements. Moreover if we need to reach the bottom corner of the base, the Kuka robots almost fully extends and gets very near to a singularity. Lastly it was necessary to keep the maximum delta of the orientation angle below 180° otherwise in a sequence of steps we would reach the wrist joint limit. To avoid these issues the following function was created:

$$\phi = atan(\frac{p_y - c_y}{p_x - c_x})$$

Where ϕ is the desired value to rotate the EF, p_x and p_y are the coordinates of the desired position and c_x and c_y are the coordinates of the center considered to calculate the orientation. Before being implemented was tested using the library Matplot++ [1] allowing to test the function before actually implementing it on the robot.

In Figure 4.3the picture of the function obtained by Matplot++ which allows us to visualize and simulate its behavior. The tip of the arrow is the position of the wrist while the tail of the arrow represents the EF position. As we can see the requirements specified are satisfied. The bottom left corner of Figure 4.3 we have the EF almost fully extended to reduce the possibility of reaching a singularity, in the upper side we have the arrows almost vertical to keep the wrist distant from slide and its walls and on the lower-right side we have the arrow almost horizontal to keep the wrist distant from the robot base frame. In addition we can see that the arrow is never flat in one direction or the other so the orientation delta never exceeds 180°.

Computation in rotative board base

The development of the orientation in this base followed a process similar to the one described previously. In this case we had a base that was particularly distant from the base frame of

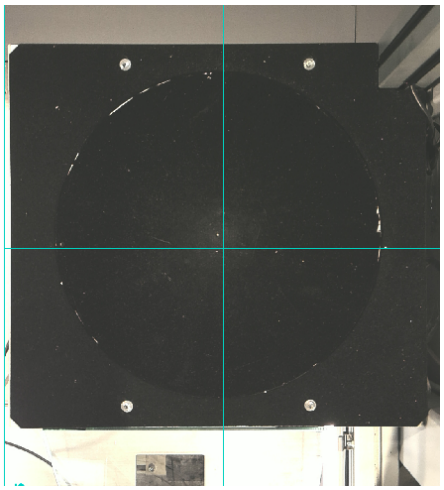

Figure 4.4: top view of the rotative board base

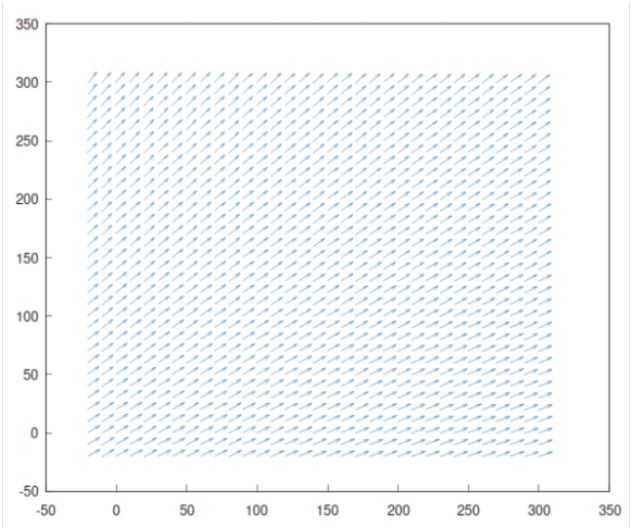

Figure 4.5: simulation on how the EF should be oriented in the rotative board base, this graph is orietned the same way as Fig. 4.4

the robot therefore not all the sections of the base were reachable and with the arm fully extended,was quite common to encounter a singularity. To ensure better reachability of the base a function similar to the one of belt base was developed, it was still

$$\phi = atan(\frac{p_y - c_y}{p_x - c_x})$$

but the center c_x, c_y was set to always have the EF extended outwards and aligned radially with the base frame of the robotic arm. This way we ensure to reach points as far as possible.

4.2.2 Bounding box check and message composition

Once all the values of the movement variable (nextMovement) have been calculated, before composing the XML message to send to the robot controller we need to check that the next position the robot is going to reach is accessible. In order to avoid collision with obstacles and the EF going outside the working base, a structure has been defined called Bounding Box (BB). This structure contains the maximum and minimum value that could be reached along each axis (xMin, xMax, yMin, yMax, zMin, zMax) of the base frame we'are currently working on. If the boundaries are exceeded the movement is not allowed. The implementation and dimensions of the BB depends on the base we're working on.

Belt base BB

Due to the regular shape of the conveyor belt base the BB was easy to define as a parallelepiped. The reference frame is set in the bottom corner of the Figure 4.2. The upper limit of the box is the slide (grey area in the Fig. 4.2) that can't be accessed by the robot tip, because it stops just before.

Rotative board BB

The definition of the BB for the rotative board was much more troublesome. The maximum height was constant and defined at 13cm above the base plane, while the tougher part was the definition of the polygon that sits on the rotative board plane.

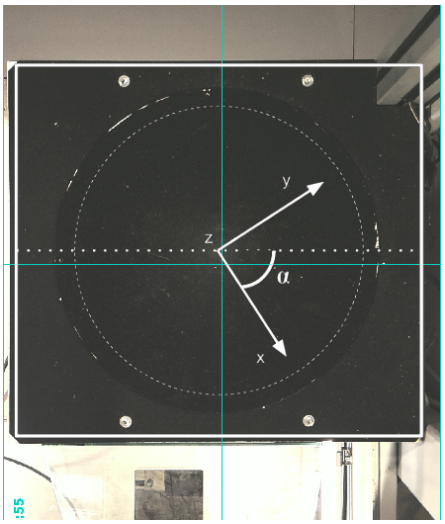

Figure 4.6: rotative base and reference frame

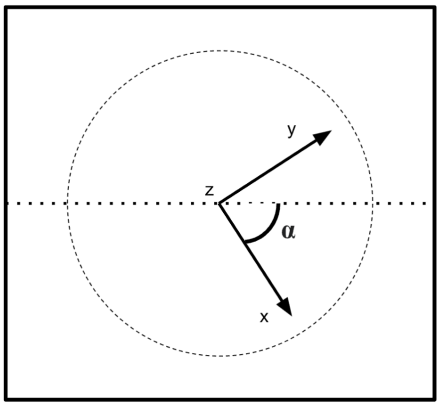

Figure 4.7: scheme of the rotative board reference frame

Figure 4.8: the images are representative of the situation but do not respect proportions

In the Fig. 4.7 which represents the rotative base we have the axis x and y correspond to the reference frame of the base and the circle represents the board that rotates. As we can observe, we see that the reference frame axes of the base aren't aligned with the borders of the base and so the definition of a bounding polygon that uses all the possible space is not as easy as before.

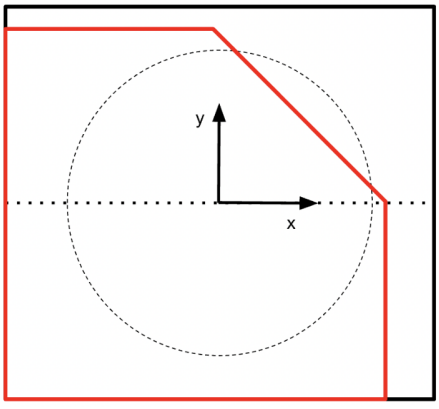

Figure 4.9: rotative board and bounding polygon

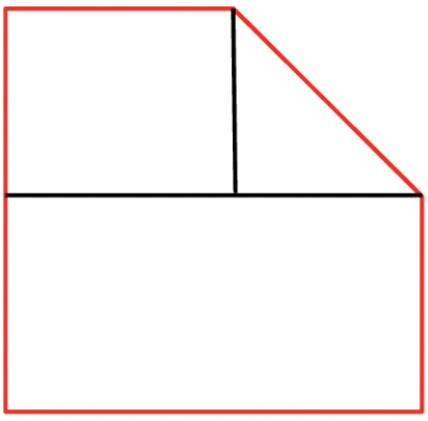

Figure 4.10

The first thing necessary to do was to define a rotation matrix that would rotate the reference frame round the z axis (coming out the plane) of an alpha amount to align x with the dotted line. Then I proceeded to define the bounding polygon and even here there were some challenges. Firstly this base is not all reachable by the robot in all points so the first thing was to find the

limits of the robot and this was done by moving the EF along the axis (x,y) at the maximum height until the robot reached a singularity. The obtained shape is the one in red in the Figure 4.9.

Due to the complexity of the resulting shape it had to be divided into three polygons each allowing the EF on the inside. The resulting polygons are two rectangles and one triangle Fig. 4.10. The function that checks if a point is inside a rectangle checks that the point coordinates x and y are inside the maximum and minimum boundaries $(x_{min}, x_{max}, y_{min}, y_{max})$ of the rectangle. For the triangle a different function is defined:

```
bool isWithinTriangle(float x, float y, float z){
    if( x>=xMin && x<xMax && y>yMin && y<yMax && z>=zMin && z<zMax ){
        float m = yMax/xMax;
        float yLine = -m*x + yMax;
        return (y <= yLine);
    }
        return false;
}
```

The function "isWithinTriangle", firstly checks that the point is inside the max and min values of x and y as in the rectangles, but then it checks if the point is under the diagonal line $y <= -m * x + y_{max}$.

Once all the movement parameters are checked the movement is transformed into an XML file and sent to the corresponding port and ip address of the Kuka controller computer. This part was't implemented by me but i used the already existing functions that were developed by my company tutor.

4.2.3 testing

For the testing of this program was developed a quick input interface from the terminal in order to compose the instructions to send the robot. In the terminal input interface could be inserted the coordinates to go to, this function was used to ensure that the robot could reach all the corners of the bounding box of the base and didn't collide with any obstacle, in addition various coordinates outside the BB where sent, to test that the position check worked properly and that the robot did not execute instructions that would send it outside the boundaries.

5 Implementation of User Interface

This interface allows the user to instruct movements to the robot. It provides a live stream from a webcam inside the robotic cell.

5.1 QT creator IDE

Qt creator is an integrated development environment for developers to create applications for multiple platforms like desktop or mobile. In our case it has been used to develop the desktop user interface, programmed in c++.

5.2 Interaction

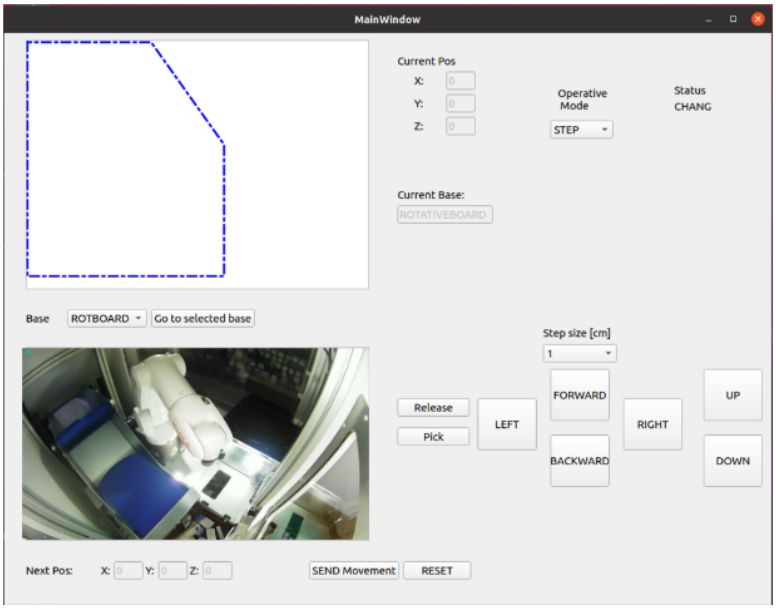

Figure 5.1: user interface

There are four main actions that the user can send to the robot. The first one is "change base" in Fig. 5.1 by selecting the desired base in the drop down menu on the center left. By clicking "go to selected base" the instruction is sent. When a base is changed also the polygon above representing the area in which the robot can move changes accordingly. Then we have the instruction to move within a base and can be performed in two ways.

The movement can be composed using the joystick on the right Fig 5.1 where you can select the size of the step with the dropdown menu with title "step size" , and then can be selected the direction of the movement with the four buttons FORWARD-BACKWARD LEFT-RIGHT UP-DOWN. This kind of interaction could be precise but not very intuitive so the second way to make the robot move within a base is to select the operative mode COORD from the drop

down menu “operative mode” on the upper side and then by clicking the image of the base on the upper left the corresponding coordinates of that position are generated. The instructions can then be sent to the robot with the “send movement” button on the center bottom. Instead the instruction can be resetted with the RESET button. To grab a box or release one the pick or release button can be clicked respectively and by clicking “SEND movement” the action is performed. It is necessary to specify that the UI to control the robot runs on a windows PC in the local network and in order to access it remotely a technician connects to the windows PC through the TeamViewer app which is an application that allows remote access to a computer.

5.3 Software implementation

All the UI software architecture is built around the movement class which contains the base, coordinates values and kind of action to be performed. The information contained in the movement class is then used to compose the message to send to the supervisor program. By clicking the buttons on the screen tha parameters are inserted into the corresponding variable of the class by the functions like **void Movement::setBase(Base_t base_)** to set the base, **void Movement::setNextPos(float x, float y, float z)** to set the next position and **void Movement::setAction(Action_t action_)** to set the actions (change base, pick, release, movestep).

The generation of the coordinates by clicking on the image of the base was done in three steps. Firstly there's a function that is called when the interrupt of a click on the image scene occurs. This function provides coordinates with respect to the upper left corner of the rectangle scene that contains the base polygon. Then there's a function provided by Qt bool containsPoint(const QPointF &pt, Qt::FillRule fillRule) that allows to check if a point is inside a polygon. If the previous check is true then the coordinates are converted to the corresponding point on the base with respect to the relative base frame, otherwise nothing happens and the coordinates are not generated.

The communication with the supervisor was implemented quite easily using the json nlohmann library [4] that provides an interface to read and write json files. The created file is then sent to the supervisor's specific port and Ip address through TCP/IP communication protocol, all handled by the ZMQ library [2].

5.4 Testing

The testing of the functioning of the ui was performed in three steps. Firstly the all the buttons interactions and the creation of the movement message were tested by just printing in the terminal the message content once the “SEND movement” button was clicked. Then the UI was connected to the supervisor program and in order to have full access on the debugging interface both softwares were run on the same computer and was simulated a TCP/IP connection internally to the computer. Once all the communications were established all the programs stack was tested by sending command to the robot and checking the correspondence between what was sent and what was executed. Sometimes it was necessary to test just the

interaction between UI and supervisor. So it was implemented in the supervisor the macro value "DEBUG", that when was set to 0 it just simulated the connection to the Kuka robot without actually sending instructions to it.

6 Conclusions

The end of this project resulted in the successful creation of a remote control system for a kuka antropomorphic arm, providing the technicians a user interface that allows control and monitoring of the robot. The system as explained is structured into three distinct layers, each playing a crucial role in the overall functionality. The lowest layer is responsible for the movimentation of the robot, ensuring accurate execution of commands coming from the UI. The middle layer "supervisor" is in charge of receiving information from the UI and sends the movement information to the movimentation program preventing the robot from exiting the operative areas or hitting into cell obstacles. The highest layer, the one in contact with the user, functions as a bridge between the technician and the robot, allowing a simplified interaction through a more intuitive interface. This is also achieved by providing live video streams and graphical representations of the robot environment. Even if the initial goal was successfully achieved there's still room for improvement. Starting from the UI, could be studied an interface even more intuitive that simplifies more the interaction with the system or could be added the support for a physical joystick for better control. When it comes to the movement of the robot, at the moment it is not able to pick boxes that aren't laying flat, so another great improvement would be adding the possibility to tilt the end effector along the x and y axis to better handle this scenario.

Bibliography

[1] c++ library for data visualization. https://alandefreitas.github.io/matplotplusplus/.

[2] communication lib. https://zeromq.org/.

[3] Introduction to behaviour trees. https://www.behaviortree.dev/docs/Intro.

[4] json files interface. jshttps://github.com/nlohmann/json.

[5] Kuka robot language. https://en.wikipedia.org/wiki/KUKA$_{R}obot_{L}anguageCITEREFBraumannE$
$Cokcan2011$.

[6] Luigi Villani Giuseppe Oriolo Bruno Siciliano, Lorenzo Sciavicco. *Robotics Modelling, Planning and Control*. Springer, 2009.

[7] KUKA company. *KUKA robot manual*. KUKA, 2018.