Development of a MATLAB Remote Control Package for Kuka Robots

By:

Almutazbellah Mohammed Asad Mesmar

Supervisor:

Dr. Mohammed Bani Younis

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Mechatronics Engineering

Deanship of Academic Research and Graduate Studies

Philadelphia University

January, 2019

تعهد بالالتزام بالأنظمة والتعليمات الخاصة

أتعهد بالألتزام بالأنظمة والتعليمات النافذة في جامعة فيلادلفيا، كما أتعهد بأن أكتب رسالتي ملتزماً بأسس الأمانة العلمية في كتابة الرسالة و الأبحاث، وأن لاتكون رسالتي منقولة عن أي رسائل أو كتب سابقة. وأتحمل كامل المسؤولية اذا ثبت عدم تقيدي بأسس الأمانة العلمية في كتابة الرسالة بما في ذلك حق جامعة فيلالدلفيا الغاء قرار منحي الدرجة العلمية التي حصلت عليها.

Amman, January 2019 _____

نموذج تفويض

أنا المعتزبالله محمد أسد مسمار، أفوض جامعة فيلادلفيا بتزويد نسخ من أطروحتي للمكتبات أو المؤسسات أو الهيئات أو االشخاص عند طلبهم حسب التعليمات النافذة في الجامعة.

التوقيع:

التاريخ:

Authorization Form

I, Almutazbellah Mohammed Asad Mesmar, authorize Philadelphia University to supply copies of my thesis to libraries or establishments or individuals on request, according to Philadelphia University regulations.

Signature:

Date:

Committee Decision

This dissertation (Development of a MATLAB Remote Control Package for Kuka Robots) was successfully defended and approved on January, 2019.

Examination committee	Signature
Dr. Mohammed Bani Younis (Supervisor) Academic Rank: Associate Professor	……
Dr. Tarek A. Tutunji Academic Rank: Associate Professor	……
Dr. Nathir A. Rawashdeh Academic Rank: Associate Professor	……
Dr. Ibrahim Izzidin Al-Naimi Academic Rank: Assistant Professor	…….

Acknowledgment

I would like to thank all who helped me in this thesis. At first, I would like to thank the supervisors of this thesis, Prof. Rolf Biesenbach, and Dr. Mohammed Bani Younis, who helped me during my work on this thesis.

I want to thank Mr. Tim Wrütz who was patient with me and was there when I was having technical problems and I want to thank him for the instructions and pieces of advice he gave me during my stay in Bochum university of applied science.

I would also like to thank Prof. Kasim Al-Aubidy and Dr. Tarek A.Tutunji for their support and contribution in enriching my knowledge and abilities.

I want also to take the chance and thank Mrs. Katharina Töws for solving my organizational issues and for her helping me during my stay in Bochum.
I would also like to thank my colleagues and my friends, Eng. Hudefah Al-Kashashneh and Eng. Mohamed Jarnaz, as they always help me during my master studies in Philadelphia university in Jordan.

Also, I want to thank Eng. Ahmad Al-Ajlouny, Iyad Al-Sattari, Mr.Raed Alobeid, Naser Musallam, Abdallah Jaber, Mohammed Warasneh, and Dana Hadian. As my stay in Bochum would not be completed without them.

Finally, this work was not to be done without the financial support from the European Tempus-JIM2l project and Bochum University of applied science represented in the department of Electrical Engineering.

DEDICATION

In dedication to my beloved family; my beloved father, for earning an honest living for us and encouraging me to believe in myself. My caring mother, who always supports me, and my dear brother, for believing in me all the time. For making me who I am and for being beside me all the time.

In dedication to my uncles, aunts; for being my guardians during my educational career. and my cousins, for supporting me all the way.

ABSTRACT

Robots have become an essential element in industry and have drew the attention of researchers and manufacturers. Many companies are competing in the robotics manufacturing field. Kuka is one of the leading innovative companies in the manufacture of industrial robots. However, Kuka robots do not have a direct communication with PC to send and receive data. To overcome this problem Kuka introduces an additional communication package to establish a communication between the robot and PC. However, the package requires the user to calculate the forward and inverse kinematics and send the data as an XML file to the Kuka controller. This Thesis presents a new method to program Kuka robots from a remote PC without the need of an additional software or the robot kinematics. The method is to use MATLAB functions to program the robot. KUKAVARPROXY was used to write and read variables in the robot. MATLAB will send data to the KUKAVARPROXY while a KRL program is running, Then the robot will move according to the changes in the data. Two case studies are used to test the package: the first is to check the motions separately while the second is to ensure that the package can be used for robot programming. The results show that the developed package is successful in transmitting and receiving data between the PC (MATLAB) and Kuka robot. Furthermore, control of point-to-point and linear motions using the developed package are tested and validated.

List of used abbreviations

RSI	Robot Sensor Interface
Ethernet	Ethernet is a networking technology used in local area networks (LAN) for data exchange between the connected devices.
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
KCT	Kuka control toolbox
KVP	KUKAVARPROXY
KRL	Kuka robot language
PTP	Point-to-point motion
LIN	Linear motion
CRC	Circular motion
XML	Extensible Markup Language
KRC	Kuka robot controller

Table of contents

TABLE OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Robot arms or manipulators are widely used devices especially in industries; these robots are deployed in heavy industries that require high precision. Robots do tasks that are difficult for a person to do, whether these actions are dangerous or boring tasks. In addition, manipulators help to get rid of human errors, which can be dangerous and sometimes costly.

Robots can be seen in areas and tasks that have the three Ds: Dirty, Dull and Dangerous. For example, environments that contain high temperatures and heavy machinery that can threat labors life, mine exploration or dirty jobs as sewer reconnaissance [1]. Therefore, it is better to have a robot working in these environments instead of a human. Moreover, with the development of sensors and interaction between robots and humans, it is possible to use manipulators in high precision surgical procedures [2]. For example, many companies produce industrial robots such as Kuka, which are widely used in industry and Medrobotics, which produces medical robots.

Robot manufacturers often focus on aiming their robots to the industrial market and neglect the research field, so it has been an important task to develop a solution for this problem. Many researchers have tried to create and develop a different type of communication platforms with industrial robots.

Kuka is one of the biggest companies in the robot's industry, their robots are widely used in industry. Kuka is known for introducing the first industrial robot with six electromechanically driven axes. Kuka robots can be seen in many industrial fields such as the automotive industry, energy, and metal industry. Figure 1-1 shows the KR 1000 titan, which was the world's largest and strongest 6-axis industrial robot in 2007 [3].

Figure 1-1: Kuka KR 1000 titan

1.1 LITERATURE REVIEW

Kuka robots are programmed using KRL, which is a proprietary programming language similar to Pascal. It has a simple interface with industrial robots, it uses a fixed and controller specific set of instruction, which makes it limited to research purposes. In addition, Kuka robots can be programmed by teaching the robot the point and position, which the robot will follow during its operation, and the user can clarify the movement type the robot will use [4, 5]. However, this technique may be tedious and may take a long time especially for long programs that contain many movements. Also, it is not suited for real-time remote applications. It does not support graphical interfaces and advanced mathematical operations such as: Matrix operations and optimization [6].

To overcome these problems researchers tried to build MATLAB abstraction layer upon KRL. One of the earlier approaches is the KUKA-KRL-Toolbox which is an extension that allows connecting the controller with an interpreter program in KUKA Robot Language (KRL) of the KUKA controller and the remote PC which includes MATLAB. The toolbox uses serial interface to connect the robot controller with the remote PC. The KRL interpreter on the robot controller will establish a bi-directional communication between the robot and the remote PC. The interpreter is also responsible for the execution of the instruction transmitted to the robot via the serial interface. However, the serial interface can limit the real-time control applications, and new controller such as KRC4 does not support serial interface [7].

Newer version of the Kuka controller does not support serial interface, so Kuka offers auxiliary software package, which enable the researchers to connect with the robot. Robot Sensor Interface (RSI) is a package provided by Kuka, it allows communication and data exchanging between the robot controller and an external computer, the communication can be made via Ethernet or via I/O system of the robot. However, data transfer must be fully consistent with the XML format. The problem with RSI is that the communication with the robot must be done at the robot cycle time. Which means that the communication will be parallel with the robot program execution, so if the robot fails to receive data within the cycle time it could cause errors and the communication will be interrupted [8,9].

RSI had been used in many approaches to establish a communication between Kuka robots and MATLAB. The communication between MATLAB and Kuka robots has been concerned by many researchers due to the computational abilities of MATLAB. Researchers had been trying to build MATLAB toolboxes to control the motion of the robot.

Kuka Control Toolbox (KCT) is a similar approach to the KUKA-KRL-Toolbox where a MATLAB toolbox is developed to control a Kuka robot from a remote PC. The toolbox was compatible with all 6-DOF (Degree of Freedom) small Kuka robots that use RSI. The toolbox had a set of functions divided into 6 categories, spanning operations such as forward and inverse kinematics computation, point-to-point joint and Cartesian control, trajectory generation, graphical display, 3-D animation, and diagnostics [6]. The difference between the KCT and the KUKA-KRL-Toolbox is that KCT is not complex as KUKA-KRL-Toolbox and it supports TCP/IP protocol. However, the toolbox was designed for the old generation controller, and do not support KRC4 [6].

A MATLAB toolbox had been made to control the motion of the Kuka KR6-R900-SIXX [8], the toolbox uses a C# code to transfer the position for the robot and the data were stored as an XML file. The toolbox performance was tested by using an Xbox controller. However, this approach requires the user to be familiar with several programming languages and require the forward and inverse kinematics of the robot so it is not effective for all models.

RoBo2L is a cross-platform mode-based that allows offline programming of a Kuka robot using MATLAB. RoBo2L uses RSI for the communication between the PC and the robot, in this case the sensor was replaced and emulated by MATLAB. The RoBo2L allows the user to move the robot to a specific point with a specific speed. Also, it allows the user to open and close the gripper and the user can read the tool current position and the gripper state. However, the RoBo2L uses RSI so it requires the robot kinematics, beside the RSI only support UDP/IP connection, which does not send a feedback in case of a transmission error, and if there was an error during data transfer in worst case a wrong packet can lead to wrong movement and incorrect and uncontrolled movement harm surroundings or destroy the manipulator [10].

Due to these problems in RSI researchers tried to find another communication platform that will be easier to use and does not require knowledge in different programming languages, KUKAVARPROXY (KVP) is a multi-client server that serves up to 10 clients simultaneously. KVP interface with the Kuka CrossComm class, this interfacing makes it possible to read and write variable along with different tasks. The variable that needs to be accessed must be declared in the predefined data file $CONFIG.DAT. JOpenShowVar is a java open source cross-platform, it is a Java library for accessing KVP, and has implemented classes for all KRL variable types [11].

KVP can be considered as a successful approach to communicate with Kuka robots because it eliminates the need for the robot kinematics and does not require so much knowledge in many programming languages. It has been used in many research projects as well as commercial software like roboDK [12].

RoboDK is a program for simulation and offline programming for industrial robots. The program has a user interface and the programs can be written in python or can be created graphically also it can generate robot specific languages like KRL. The program supports all Kuka controllers since KRC2. In addition, the program uses KUKAVARPROXY to communicate with Kuka robots by introducing a source code written in KRL and some global variables that the user should write in the configuration file in order to move and control the robot [13].

1.2 AIM

The aim of the thesis is to control Kuka robot from a remote PC without the need of an additional software provided by Kuka, in order to eliminate the need of the forward and inverse Kinematics.

1.3 OBJECTIVE

This Thesis is dedicated to use KVP to establish connection between a Kuka robot and MATLAB so that users can program the robot through MATLAB without using any additional softwares provided by Kuka (RSI). The Thesis will focus on developing a MATLAB package, which the user can use to program the robot. The package is easy to use because it does not require the user to use KRL. Also, the functions will not use mathematical data so the user will be able to program the robot without the need of any mathematical information about the robot.

1.4 STRUCTURE OF THE THESIS

The thesis is divided into five main chapters, chapter 2 presents the robotic system that had been used and the forward and inverse kinematics and the singularity for Kuka robots. Beside it discusses the robot controller and the methods of programming the robot. Chapter 3 discusses the package design and how to establish the connection between the PC and the robot. Also, it presents how to use the package and the functions syntax for the functions with the method of configuring the package. Chapter 4 presents the experiment along with the results of the experiment. Chapter 5 presents a brief conclusion for the thesis with some proposed future works to improve the package.

CHAPTER 2: ROBOTIC SYSTEM

This chapter present some basics in robotics like the robot kinematics and the Kuka robots singularity along with an overview of the Kuka robot controller and the type of data that the controller accepts and the chapter also present the type of motions in Kuka robots.

2.1 BACKGROUND

Due to the scientific development, the topic of robots has increased to be an important topic for researchers and developers. The Oxford dictionary describes the robot as "*a machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer*" [14].

Robots are classified according to the environment in which they operate, but generally, robots can be classified into mobile and fixed robots. Most fixed robots are called manipulators and they are industrial robots that are used in a robot-oriented environment and do certain and repetitive tasks such as assembling parts or in some applications in the automotive industry [2].

Mobile robots operate in a wider and unspecified environment and deal with unknown circumstances. These environments may contain unknown entities such as animals, humans, or non-living objects. An example of moving robots is self-driving cars [2]. Mobile robots have also been used in exploration such as NASA's Mars Exploration Rovers: Spirit and Opportunity, which have been used to explore the planet mars [15].

2.2 ROBOT KINEMATICS

Kinematics describes the movement of objects and points regardless the mass or force that caused the movement. Robot kinematics describes the relation between the joint movement and the resulting movement of the robot. There are two kinematics for manipulators, forward and inverse kinematics. Forward kinematics is used to determine the position and orientation of the end-effector given the value of the joint variable. On the other hand, inverse kinematics is concerned in determining the joint variable value given a desired position and orientation of the end-effector. The forward and inverse

kinematics of the Kuka KR-R900-Sixx can be seen in Appendix A. The Figure 2-1 represents the forward and inverse kinematics [16].

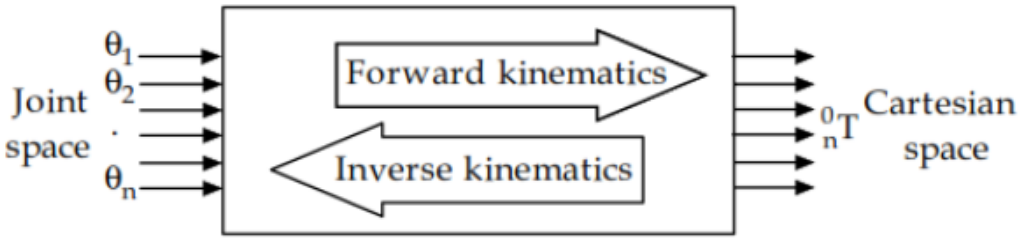

Figure 2-1 : Forward and inverse kinematics representation

2.2.1 Homogeneous transformation

The transformation of frames is a fundamental concept in the modelling and programming of a robot. This section presents a notation that describes the relationship between different frames and objects of a robotic cell [17].

Transformation matrix is used to display the coordinates of a point in a different frame from the original point's frame. Robot frames can be classified into three frames: reference frame, joints frame and tool frame. Reference frame is normally attached to the robot base, joints frame is attached to each joint of the robot and the tool frame is attached to the end-effector. Frames are used to present points; points can be presented by their x, y and z coordinate according to a specific frame. It could be represented as a vector:

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{P_x} \\ \boldsymbol{P_y} \\ \boldsymbol{P_z} \end{bmatrix} \quad (1)$$

The transformation matrix is a 4x4 matrix, which is represented as:

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{P} \\ \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \quad (2)$$

Where R is the rotational matrix and P is the position vector.

To find the transformation matrix through multiple frames, the transformation matrix for each frame with respect to its previous frame must be multiplied. For example, the transformation matrix for frame 3 with respect to frame 0 will be:

$$T_3^0 = T_1^0 T_2^1 T_3^2 \tag{3}$$

Rotational matrix is a matrix that display the rotation of a frame with respect to another frame, the rotation matrix can change according to the axis of rotation.

- If the rotation is around the x-axis the rotation matrix is:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{4}$$

- If the rotation is around the y-axis the rotation matrix is:

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{5}$$

- If the rotation is around the z-axis the rotation matrix is:

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Figure 2-2 shows a frame 1 rotating around the z-axis the rotation of frame 1 with respect to the previous frame 0, this rotation can be presented using matrix (6).

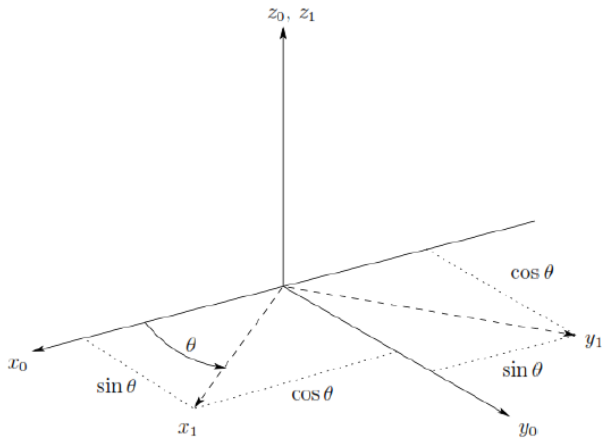

Figure 2-2 : Frame rotation

2.2.2 Euler Angles

In robotic systems, the robot orientation is a tricky thing because unlike the translation movement orientation could be described in different forms and these forms can refer to the same orientation. The most common way to describe robot orientation is the Euler

angles. Euler angles are introduced by Leonhard Euler they describe the orientation of a frame with respect to a fixed frame. Euler angles consist of three angles (α, β, γ) and have many representations depends on the order of the rotation. For example, the representation Z-Y-X means that the tool will rotate around the z-axis then will have a rotation around the current y-axis then will rotate around the current x-axis again. This corresponds with the final rotation matrix:

$$R(\alpha, \beta, \gamma) = \mathrm{Rot}(z, \alpha)\ \mathrm{Rot}(y, \beta)\ \mathrm{Rot}(x, \gamma) \quad (7)$$

$\mathrm{Rot}(z, \alpha), \mathrm{Rot}(y, \beta)$ and $\mathrm{Rot}(x, \gamma)$ are the rotation matrices (6), (5) and (4) respectively.

2.2.3 D-H Parameters

The Denavit-Hartenberg Convention is a systematic general method to define the relative position of two consecutive links. The method can be applied by determine two frames attached to two links and compute the coordinate transformations between them [8]. Figure 2-3 [18] shows the D-H parameters. The D-H parameters are a, d, α, and Θ. To calculate these parameters, the frame of each link must be defined first, to define the link frame the D-H representation rules must be applied, the rules according to [18] are:

- Choose axis z_i along the axis of joint i+1.
- Locate the origin O_i at the intersection of axis z_i with the common normal to axis z_{i-1} and z_i. Also, locate O_i at the intersection of the common normal with the axis z_{i-1}.
- Choose axis x_i along the common normal to axes z_{i-1} and z_i with the direction from joint I to joint i+1.
- Choose axis y_i so as to complete a right-hand frame.

When each frame is defined the pose (position and orientation) of each frame with respect to the previous frame can be specified by the D-H parameters:

- a_i is the distance from z_{i-1} to z_i measured along x_i. (a_i is always positive)
- d_i is the distance from x_{i-1} to x_i measured along z_{i-1}. (d_i can be negative)
- α_i is the skew angle from z_{i-1} to z_i measured about x_i.
- Θ_i is the angle from x_{i-1} to x_i measured about z_{i-1}.

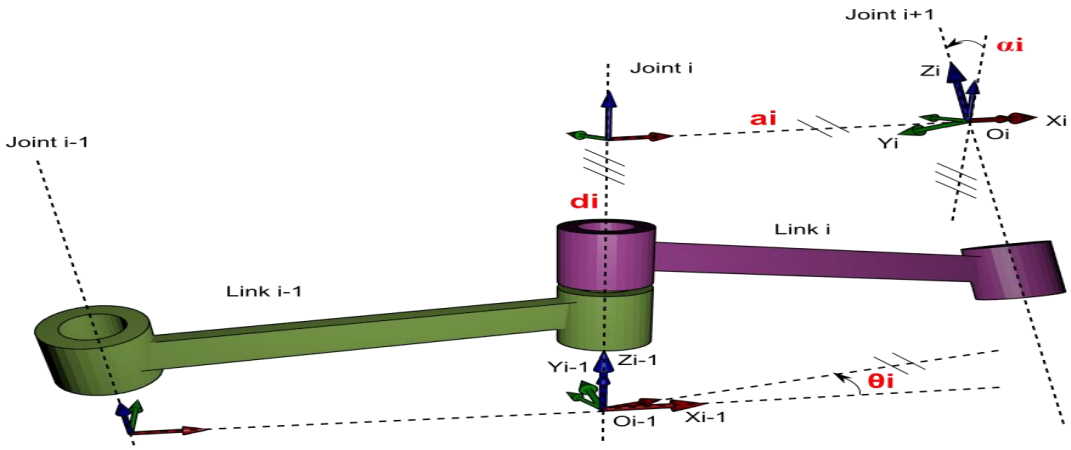

Figure 2-3 : D-H parameters

2.3 SINGULARITY IN KUKA ROBOTS

Singularity is the configuration where the robot loses a degree of freedom. Inverse kinematics can have many solutions, so the robot can have many ways to reach the desired position and orientation. If the optimal solution is not chosen, the robot can stop working because some of these ways can be impossible for the robot to take; this problem is called the kinematic singularity [19].

Singularity also occur when two or more joints are aligned, which means that two or more joints are no longer independently control the position and orientation of the tool. In standard Kuka robot's kinematics, Kuka robots have three different singularity positions, the overhead singularity, the extended position, and the wrist axis singularity.

2.3.1 Overhead singularity

This type of singularity occurs when the intersection of axis A4, A5, and A6. Is positioned on axis 1. Figure 2-4 shows the overhead singularity [20].

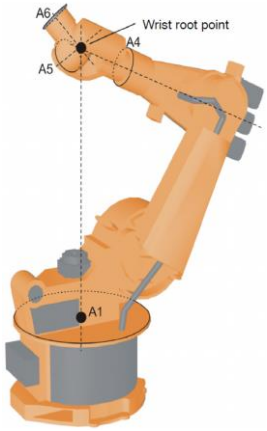

Figure 2-4 : Overhead singularity

During the overhead singularity, the position of Axis 1 cannot be determined by means of the inverse kinematics. In this situation, the controller offers two solutions: axis 1 is moved to the default position (0- degree) or axis 1 angle remains the same for the start and end points [20].

2.3.2 Extended position singularity

While the arm is fully extended, the robot will be on the limit of its work. Even though inverse kinematics can provide a solution, this position is considered a singularity position because in order to maintain a low Cartesian velocity the joint velocity have to be high which. Figure 2-5 shows the extended position singularity for a Kuka robot [20].

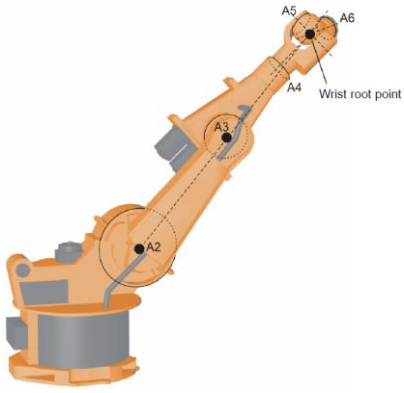

Figure 2-5 : Extended position singularity

2.3.3 Wrist axis singularity

When axis 4 and axis 6 are aligned, inverse kinematics cannot present a clear solution, as there will be an infinite number of axis positions for A4 and A6 for which the sum of the axis angles is identical. Figure 2-6 shows the wrist axis singularity [20].

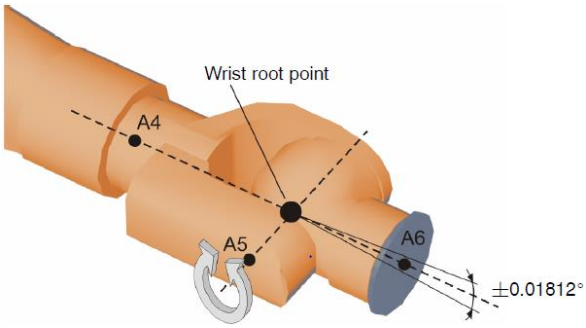

Figure 2-6 : Wrist axis singularity

If the end point of a PTP motion result in this singularity and axis5 is $\pm 0.01812°$, the controller can provide two options, either axis 4 is moved to default position during PTP motion or axis 4 angle remains the same for the start and end points [20].

2.4 KUKA ROBOT OPERATIONS

2.4.1 Robot Controller

A controller is a device that receives input and adjusts the output of the device it controls. KRC4 is a Kuka controller that have long time efficiency and flexibility. The KRC4 software architecture integrates four control processes in one controller, it integrates robot control, PLC control, motion control, and safety control. Besides, the KRC4 can understand PLC and CNC (G-code) languages. It features intelligent, flexible and scalable application potential. Figure 2-7 shows the architecture of the KRC4 [21].

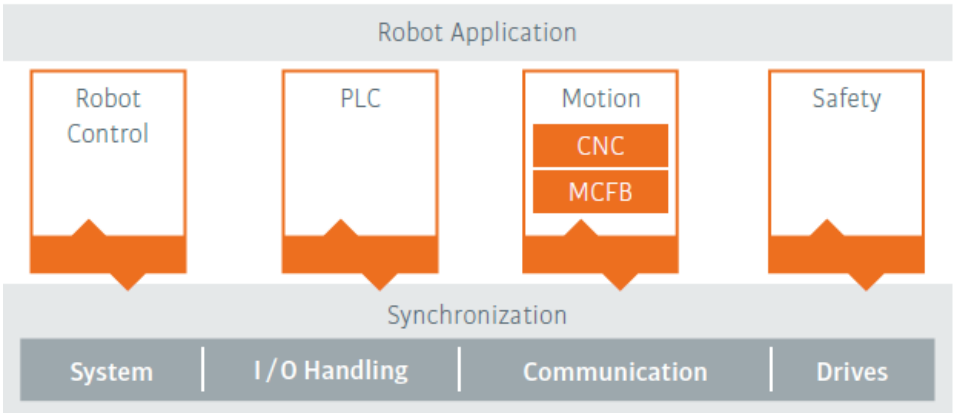

Figure 2-7 : KRC4 software architecture

The high-end Soft PLC option provides full access to the entire controller the I / O system and has high performance. It allows I / O to handle a robot, a complete robot cell or a line of robots. In addition, variables such as axis positions and velocities can be read and processed through function blocks. Furthermore, KUKA.CNC enables direct programming and operating the robot by G-code. It can process complex CAD/CAM systems with high accuracy. Safety function and safety-oriented communication are executed via Ethernet protocols [21].

KRC4 consists of many elements such as control PC and the smart pad. The control PC is responsible for many functions such as the graphical user interface, program creation, correction, archiving, and maintenance, monitoring and communication with external devices (other controllers, PC's or network) [21].

The smart pad allows the user to operate and program the robot. The user can move the robot manually or by programming the robot in KRL. The user operates and program the robot on the Human Machine Interface (HMI). However, the KRC4 have a windows interface to allow communication with other devices. Unlike previous versions of the Kuka controllers, the KRC4 only has Ethernet communication interface, it does not have a serial communication [21].

2.4.2 Kuka robot motions

Kuka robots have three types of motions. Each type can be programmed by the user and they differ from each other either by the speed of the motion or by the way they are programmed in the controller. The motions are Point-To-Point motion (PTP), Linear motion and Circular motion.

2.4.2.1 Point-To-Point motion

The PTP motion is the quickest way to move the tool because the robot guide the tool in the fastest path to the end point. However, the fastest path is not the shortest because the robot axes are rotational thus a curved path is faster than a straight path. Therefore, while using PTP motion it is necessary to reduce the robot speed especially in the presence of obstacles and objects [22]. Figure 2-8 shows the path of the PTP motion.

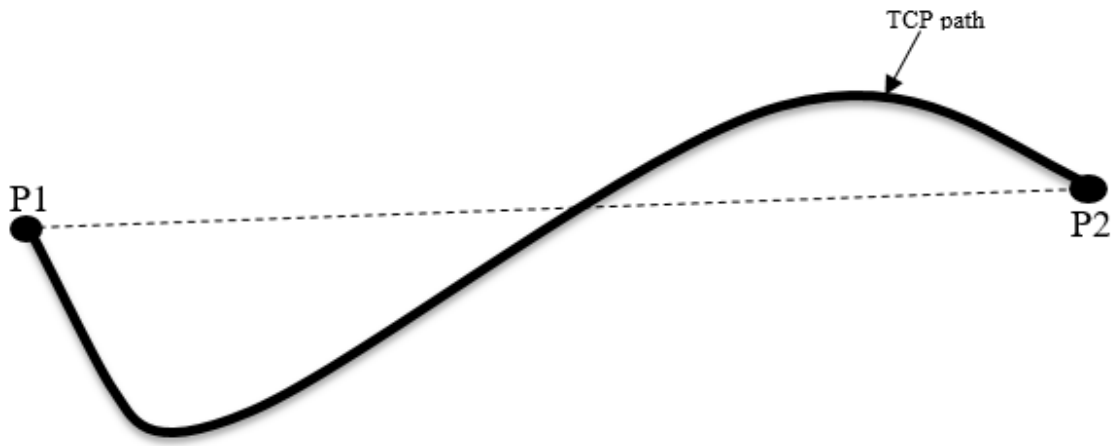

Figure 2-8 : PTP motion path

2.4.2.2 Linear motion

In linear motion, the controller will guide the tool at a defined velocity along a straight line from the starting position to the end point [22]. Figure 2-9 shows the linear motion.

Figure 2-9 : Figure 2 9: Linear motion path

2.4.2.3 Circular motion

In circular motion, the robot will guide the tool in circular path, the circular motion needs three points to be programmed start point, auxiliary point and end point [22]. As seen in figure 2-10.

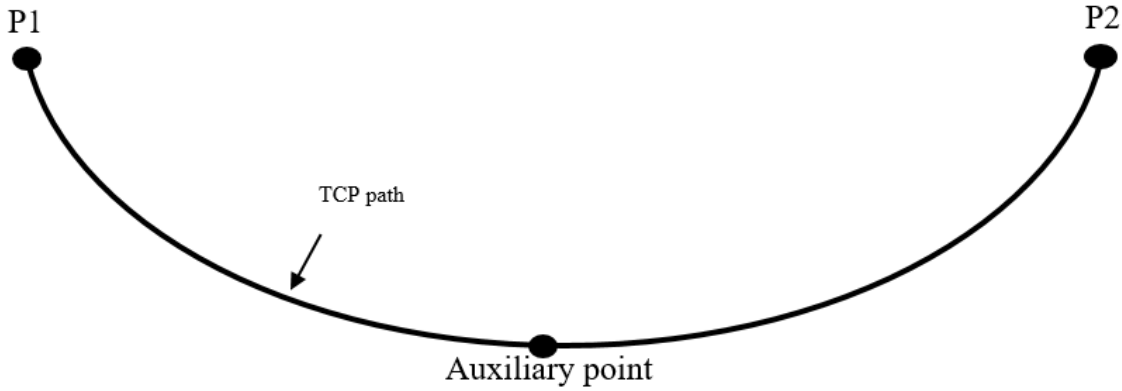

Figure 2-10 : Circular motion

2.5 ROBOT PROGRAMING

Kuka robot can be programmed by either teaching the robot the position or orientation of the tool or by programming the robot using KRL language. However, to avoid any confliction and avoid accident, the user has to specify the tool and the base that will be used. Any KRL code consists of two files: movement command file, which contains

movement commands that the robot will use to move, and a data file which stores the data to be used in the movement commands file. KRL is similar to other programming languages regarding the data types, it has four common data types which are shown in table 4 below [4, 5].

Table 1 : Data types in KRL

Data type	Keyword	Meaning	Range of values
Integer	INT	Integer	$-2^{31} - 1 \ldots 2^{31} - 1$
Real	REAL	Floating point number	±1.1E-38 ... ± 3.4E+38
Boolean	BOOL	Logic state	TRUE, FALSE
Character	CHAR	character	ASCII character

For motion programming, KRL have data types AXIS, E6AXIS, E6POS, POS, and FRAME. To declare a new variable in KRL the user has to define the data type and the data value preceded by DECL. However, for structured data type (motion programming data types) the user has to define the motion data type with the data type for the variable and the variable data preceded by STRUC.

In E6POS and POS the user can defined the axis positions by using the status and turn bits. They represent a specific known robot position. This option can help in some singularity situations where several axis positions are possible for the same point. It is important to program the status and turn for the first motion to define a clear initial position. The status and turn bits are specified only for PTP motion with axis coordinates because they are not taken in consideration in continuous path motion [20].

The turn is a 6-bit integer number written in binary form (0 and 1), each bit describes the sign of the axis individually. If the bit value is 0 then the axis angle is greater or equal to 0°, on the other hand, if the bit is 1 then the axis angle is less than 0°. Table 5 shows what the turn bits means [20].

Table 2 : Turn bits meaning

Value	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	A6 ≥ 0°	A5 ≥ 0°	A4 ≥ 0°	A3 ≥ 0°	A2 ≥ 0°	A1 ≥ 0°
1	A6 < 0°	A5 < 0°	A4 < 0°	A3 < 0°	A2 < 0°	A1 < 0°

Therefore, the entry T 'B 101100' means that axes 6, 3 and 2 are negative and axes 4, 2 and 0 are positive.

On the other hand, status bit describes the state and position of the robot, the robot can take a position other than what the user wants due to the multiple solution problem in inverse kinematics (Figure 2-11) [20].

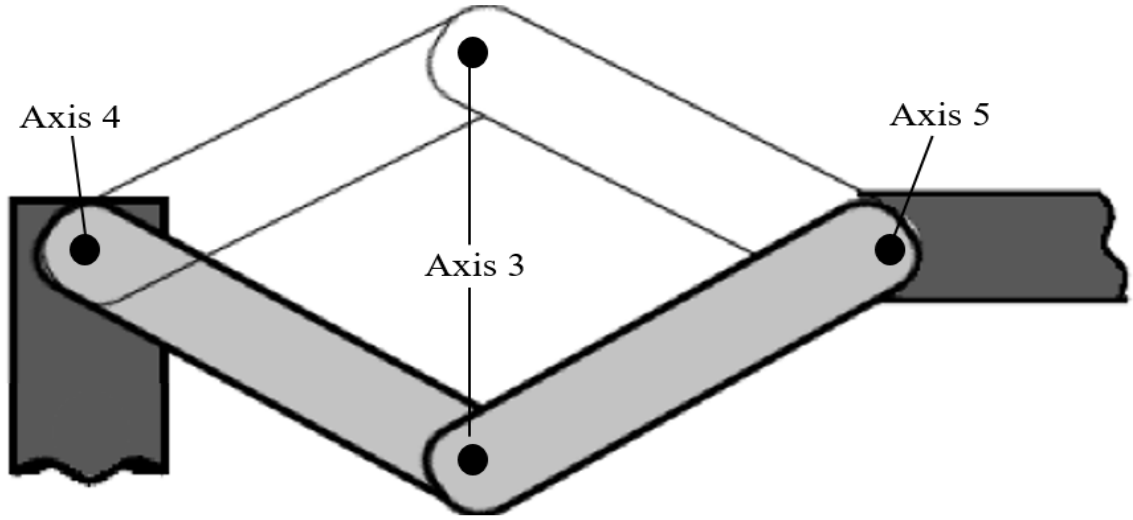

Figure 2-11 : Unclear robot kinematics

Status is a 3-bit integer number written in binary form, each status bit represents different things. Bit 0 represents the position of the wrist root point, Bit 1 represents the arm configuration, and Bit 2 represent the wrist configuration. Table 6 shows the function for each bit [20].

Table 3 : Status bits meaning

Value	Bit 2	Bit 1	Bit 0
0	$0° \leq A5 < 180°$ $A5 < -180°$	$A3 < \emptyset$ ($\emptyset$ depends on robot model)	Basic area
1	$-180° \leq A5 < 0°$ $A5 \geq 180°$	$A3 \geq \emptyset$ ($\emptyset$ depends on robot model)	Overhead area

Kuka robots have three types of motions: Point-to-point (PTP), linear (LIN) and circular (CRC). The user can program these motions either by moving the robot manually and use the touch down option and then the coordinates will be saved as a point. This method can be used to program PTP and linear motions but for circular motions the user has to teach an extra point because circular motions need an auxiliary point in order to be programmed.

The other method is to program these motions by using the point coordinates, any point can be considered as a variable even if the point is a structured data type. So, if the user programmed a motion for a specific variable the controller will understand that the variable is a point and will move towards it. To program a motion, the user only have to declare the motion type and the end point of the motion.

Motions could also be programmed by using another method, the user can declare a variable (i.e. position) with the type POS, then it is possible to declare the coordinates separately.

This means that the x-coordinates for the point has this specific value and it could be used for all other coordinates, after declaring this the user can use the motion commands to move to the point.

CHAPTER 3: KUKA PACKAGE

In this chapter, the package design will be reviewed as well as the elements used for communication and programming will be reviewed with the method of communication between the Kuka robot and MATLAB. The design of this package was based on the Jopenshowvar, which is a java cross-platform communication interfaces to Kuka robots that uses KUKAVARPROXY (KVP) to write and read Kuka variables.

The architecture for the package is shown in figure 3-1. It is a client-server network where MATLAB works as a client and the KVP works as server. The MATLAB send commands to KVP via TCP/IP, according to these commands the KVP change a set of global variables in the Kuka's $CONFIG.DAT file, at the same time a KRL program will be running so based on the changes of the variables the robot will move.

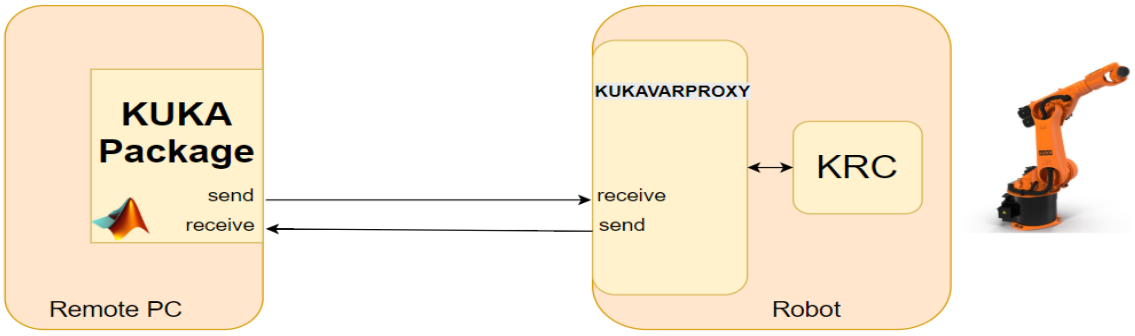

Figure 3-1: Package architecture

In order to be able to program the robot from a remote PC, it was necessary to develop a KRL program to move the robot. A new set of global variables had been introduced to avoid any errors that can occur when using any robot variables. The added variables set were divided into two sets, one for the motion commands and the other is for the coordinates that the user will enter for the movement.

3.1 COMMUNICATION PROTOCOL

The communication protocol is a TCP/IP connection, in order to connect the (KVP) with MATLAB, the user needs to define the robot IP and the communication port. Generally, KVP listens on TCP port 7000. KVP receives a formatted message from the user to access the variable. The message is sent as an array where each element of the array stands for a specific data. For the reading command, the user has to specify two parameters in the message the type of the desired function and the variable name. However, for the writing command, the user needs to specify one more parameter, which is the variable value.

3.2 THE KRL PROGRAM

Kuka robot can only be programmed using KRL so it was necessary to write a KRL program to be able to move the robot. Also, Kuka robots have many motion types such as point-to-point motion or linear motion, so when designing this package, it was necessary to take the type of movements in consideration.

The KRL program consists of IF loops that uses global variables as the loop's conditions, each global variable has a specific task. It either describes a movement or it is used to describe the position of the robot. For example, the variable $MPT is the condition for the loop that is responsible for the point-to-point motion. Table 7 shows the motion's global variables and their descriptions. The KRL program can be seen in Appendix B.

Table 4 : Description of the motion's global variables

Variable	Variable description
$MPT	Point-to-point motion without tool rotation
$MPS	Point-to-point motion with tool rotation and status and turn
$MPT3	Point-to-point motion with tool rotation
$MLN	Linear motion without tool rotation
$MLN3	Linear motion with tool rotation
$MCR	Circular motion without tool rotation
$CRC3	Circular motion with tool rotation

3.3 MATLAB FUNCTIONS

To give the user the freedom to program the robot, it was necessary to produce functions similar to those used in KRL, these functions will be reviewed in detail in the next parts. While developing these functions the connections errors had been taken in accord, so the connection had to be opened at the start of the command and closed when the instructions are finished. The user will simply use the functions from the package to program the robot from MATLAB. Figure 3-2 shows the general layout of the package. Table 5 shows the functions and their descriptions.

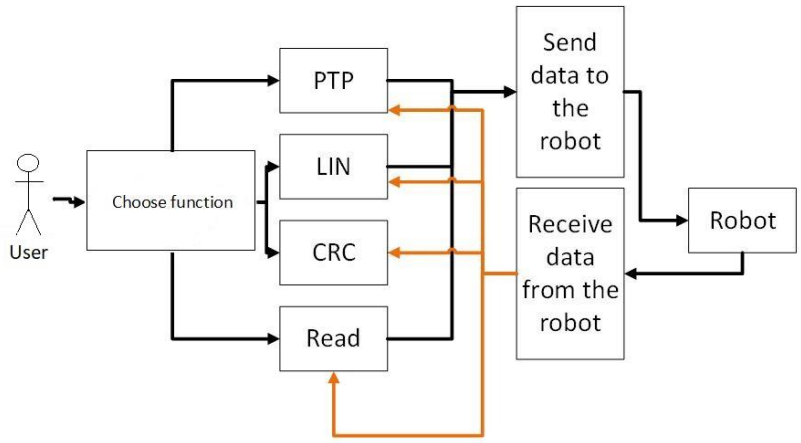

Figure 3-2 : Package general layout

Table 5 : The MATLAB functions description

Function	Function description	NO. of inputs
KUKACNCT	Establish communication with the robot.	-
Kukaerr	Calculate the error between the intended position and the current position.	-
Kukastrt	Allows the KRL program in the robot controller to run.	-
Kukaclose	Stop the KRL program in the controller.	-
Ptp	Move the robot in Point-To-Point motion.	3, 6, 8
Lin	Move the robot in linear motion.	3, 6
Crc	Move the robot in circular motion.	6, 12
Kukard	Read the current position of the robot.	-

3.3.1 KUKACNCT

This function uses the TCP/IP command in MATLAB to establish the connection between the MATLAB and the robot. This function is an auxiliary function and it was developed to facilitate the process of connecting the robot and the MATLAB to the user, the user only needs to change the IP address in the function to the robot's IP address and save the changes. It is important that the user enters the right IP address because this function is used in every function of the package.

3.3.2 Kukaerr

This function calculates the error between the entered coordinates and the robot coordinates. It is used to make sure that the robot goes to the specified point and to avoid skipping the robot for any movement especially for programs that contain many movements. This function is invoked in the movement functions so the user does not need to use it. The acceptable error is 1 mm, this error was selected based on the speed of the robot. Because when a smaller error is chosen, the robot will take longer to reach the requested point and the robot may stuck, especially if the error is less than 0.5mm.

3.3.3 kukastrt and kukaclose

The robot cannot sense the objects around him so if the user changes the base of the robot and run the program before specifying a point, the robot will move to a previous point, which could cause harm to the user or the robot. To avoid this problem these functions were developed. These functions reset the global variables that are used as movement conditions. Therefore, the robot will not move until the user enter kukastrt and will not stop until the user enters the kukaclose.

3.3.4 ptp, lin and crc

These are the movement functions; the user will use these functions to move and program the robot. Those functions accept variable input arguments this is because the robot can move by specifying the coordinates or by specifying the coordinates with the tool rotation.

The ptp function stands for point-to-point and it accepts three, six and eight inputs. Three inputs mean that the user want the robot to move according to X, Y and Z coordinates,

but six means that the robot will move according to X, Y, Z A, B, and C, while eight inputs means the robots will take in consideration the status and turn bit with the coordinates and the tool rotation.

The lin function stands for linear movement and it is similar to the ptp function, but the lin function does not have a status and turn option so it only accepts three and six inputs.

The crc function stand for circular motion and it is different from the previous two functions because it needs two points auxiliary point and a final point. It accepts six and twelve inputs, six means that the two points entered as coordinates only, but twelve means that the points are entered as coordinates and the tool rotation.

At first, a connection to the robot will be made using the KUKACNCT function. Then the user will choose the type of motion along with the exact coordinates. The function then will construct the array and prepare it to be send to the KUKAVARPROXY. After that the connection will open and the data array will be sent to the robot. According to the coordinates the robot will start moving. However, during the robot movement the error function (kukaerr) will be reading the robot position and calculating the error between the robot position and the intended position. When the error is less than 1 mm the loop will be break and the robot will stop. Figure 3-3 shows the flow chart for the three functions.

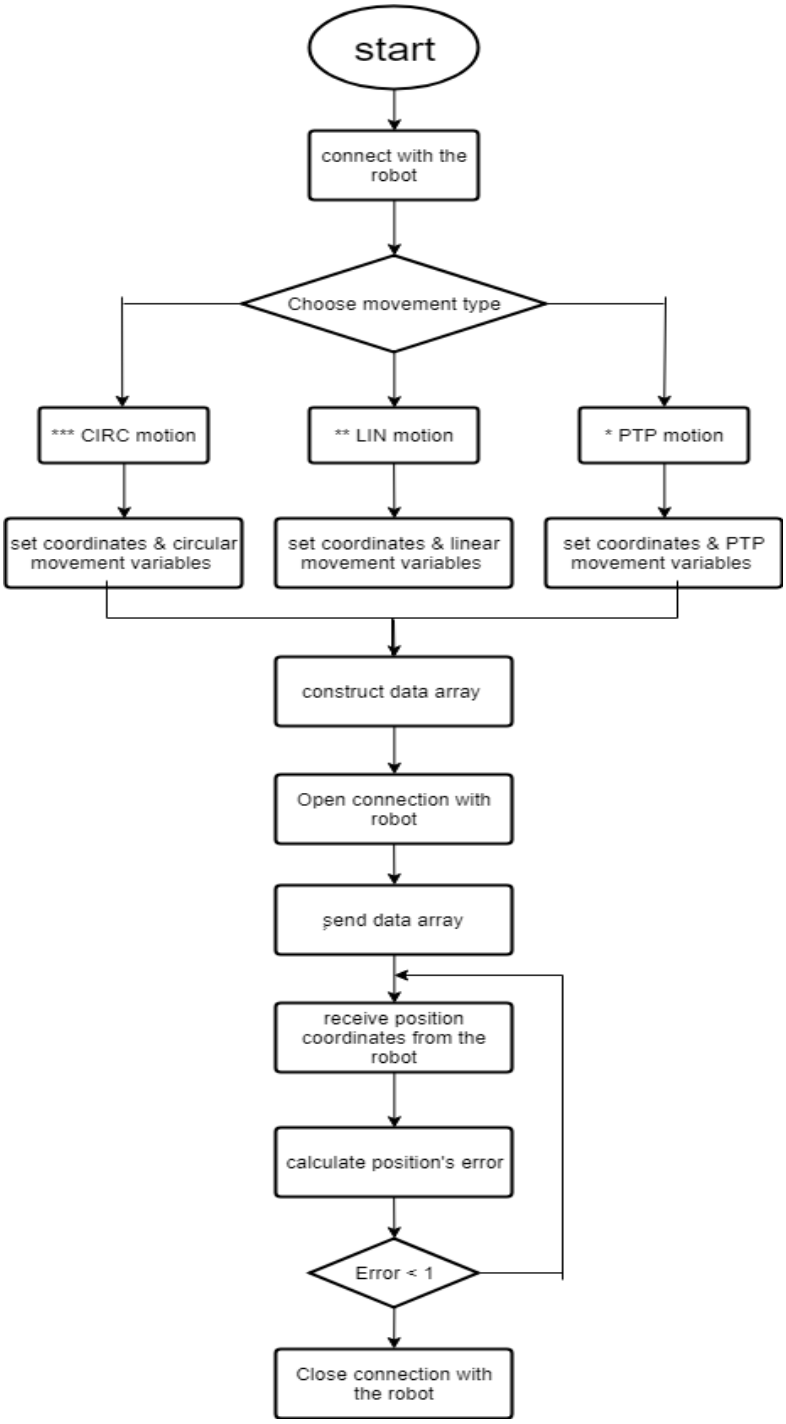

Figure 3-3 : Flow chart for the movement functions

3.3.5 kukard

This function reads the position of the robot by reading the global variable $POS_ACT and sends it to the MATLAB. The user can only read $POS_ACT and cannot write it this is a restriction from the robot controller [10], so it could not be used to move the robot. It

is better to read $POS_ACT than reading the coordinates send by the user to make sure that the robot actually reached the needed position, also sometimes the user can move the robot manually so if the user read the sent coordinates it will give a wrong position for the robot. Figure 3-4 shows the read function flow chart.

The read function is similar to the movement function, at first it uses the KUKACNCT function to connect with the robot, the array to read the $POS_ACT will be prepared to send, then the connection will open and the array will be sent to the robot. After that the MATLAB will receive the exact position of the robot from the KUKAVARPROXY.

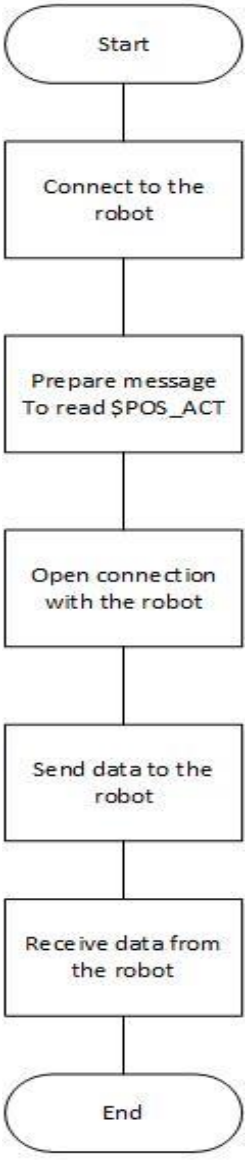

Figure 3-4 : Read function flow chart

3.4 GRAPHICAL USER INTERFACE

The graphical user interface (GUI) allows the user to interface with the package, and make it easier for the user to use the package. The user can choose any motion type and enter the needed coordinates. Figure 3-5 shows the GUI of the package.

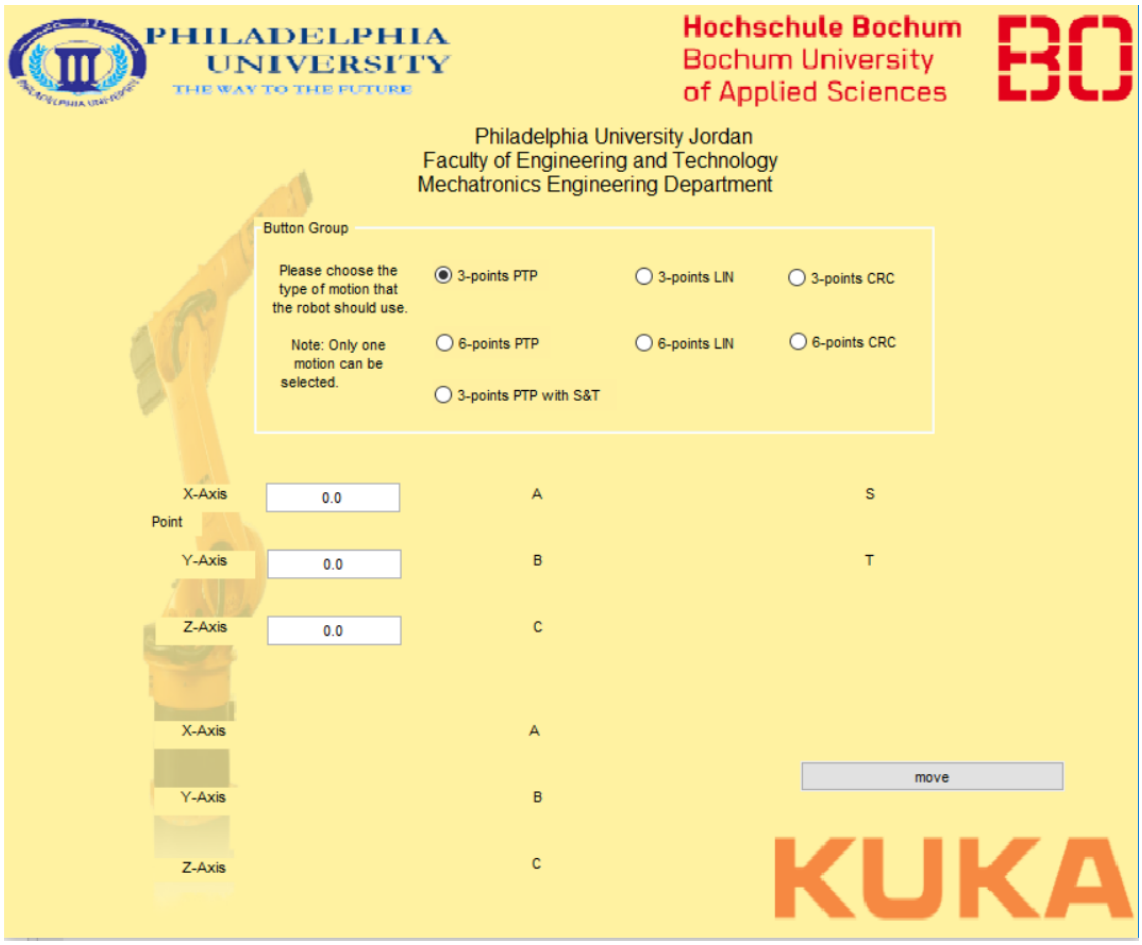

Figure 3-5 : Package GUI

Using the GUI, the user can move the robot without writing the functions on the command window. However, the GUI can be used to move the robot. As seen the GUI includes all motion types. After choosing the motion type and setting the coordinates the user can click on the “move” button to make the robot move to the needed coordinates.

3.5 PACKAGE CONFIGURATION AND USAGE

This section shows how the user has to use the package to program the robot from the PC. As mentioned before the package consist of three parts: KUKAVARPROXY, KRL program and the MATLAB functions.

3.5.1 KVP configuration

1. First the user has to put the KUKAVARPROXY on the windows layer of the robot, then run the program. To enter the windows layer, the user has to follow the path: Kuka → Start-UP → Service → Minimize HMI. Then copy the KVP to the needed folder (desktop is recommended to make accessing it easier).
2. Open port 7000 (KVP normally uses port 7000). To open the port:
    a) Select the HMI.
    b) KUKA →Start-up→Network configuration→Advanced
    c) NAT→Add port→Port number 7000.
    d) Set permitted protocols: tcp/udp
3. Start the KUKAVARPROXY.EXE program on the robot controller (running on Windows).

3.5.2 KRL configuration

The Kuka can only be programmed using KRL. To prepare the KRL program follow these steps:

1. Copy the Matlab.dat and Matlab.src to the needed location. The user will only use the src file but it is important to copy the dat file because it contains all the needed data.
2. Copy the text in the Global.txt (could be found in the package folder) in the end of the $config.dat file to prepare the global variables. Note: any changes in the global variables requires changes in the MATLAB functions.

3.5.3 MATLAB functions syntax

The package has eight functions. However, the user only has to deal with six functions three of them are movement functions, two of them are safety functions and the read function. The functions are the ptp, lin and crc.

1. The ptp function is the function responsible of the PTP motion for the robot, this function accepts either three, six or eight inputs as follows:
    a) ptp (x, y, z) this syntax uses the three input as the point coordinates.
    b) ptp (x, y, z, a, b, c) the six inputs in this syntax are the point coordinates and the tool orientation.
    c) ptp (x, y, z, a, b, c, s, t) the eight inputs in this syntax are the point coordinates and the tool orientation with status and turn bits.
2. The lin function is responsible of the LIN motion for the robot, this function is as same as the ptp function. Unlike the ptp function the lin function do not have a status and turn bits. The lin syntax is:
    a) lin (x, y, z) the three inputs are the point coordinates
    b) lin (x, y, z, a, b, c) the six inputs in this syntax are the point coordinates and the tool orientation.
3. The crc function is responsible for the CRC motion, unlike the previous functions crc requires two point because the CRC motion needs two points and end point and an auxiliary point. The user will have to specify two points either by the point coordinates or by the coordinates and the tool orientation. The crc syntax are as follow:
    a) crc (xa, ya, za, x, y, z) the first three points are the auxiliary point coordinates, the last three are the end point coordinates.
    b) crc (xa, ya, za, aa, ba, ca, x, y, z, a, b, c) the first six inputs are the auxiliary point coordinates and the tool orientation, the last six are the end point coordinates and tool orientation.
4. The safety functions kukastrt and kukaclose are used to reset the global variables used in the program to avoid any problem when starting the robot. Beside the while loop in the Matlab.src file will not break unless the user uses the kukaclose command.

5. Kukard is a function to read the actual position of the robot it does not require any input. However, if the user uses the function will the robot is moving the function will represent the position of the robot at that specific time.

3.6 ROBOT PROGRAMMING

After configuring the KVP and the KRL, the robot is ready to be programmed from the MATLAB. To start programming the user has to follow some steps:

1. Open the KUKAVARPROXY.EXE in the window layer in the robot. The KVP will detect the connection with the PC automatically.
2. Use the kukastrt to reset the global variables.
3. Run the Matlab.src file on the robot. The user has to change the base in the Matlab.src file otherwise the robot might not work as intended.
4. Start programming the robot from the MATLAB, the user has to start by running the kukastrt function to reset the global variable to avoid running into any problems.
5. After finishing the robot programming, the user has to use the kukaclose function. Because the Matlab.src program has a while loop and the loop will not break until the user run the kukaclose function.

CHAPTER 4: RESULTS AND DISCUSSION

In this chapter the Kuka package will be tested by using it to program a robot, the quality of the package will be determined by certain criteria, the criteria are:

- The robot will be able to perform the motions exactly as they are executed in normal KRL program.
- The robot will be able to execute a full program not only one movement command.
- The robot will move to the exact position given in the program.

To determine the quality of the robot, the experiment will be divided into two tests. First, the motions will be tested separately, the test will only determine whether the robot can move as it is intended or not. Then a full program will be tested and will be compared to an exact KRL program. The data from the experiment will be collected by using the trace option in the KRC, which records the robot movement. Then the data will be viewed in MATLAB by using a premade function in [23].

4.1 MOTIONS TEST

It is necessary that the robot moves in the exact motion needed, in this test the motions will be tested separately, the objective of this test is to make sure that the robot can perform the motions as intended. Before starting the experiments, the base and the tool to be used were calibrated.

4.1.1 PTP motion test

As mentioned in chapter 2 during the PTP motion the robot will guide the tool in the fastest path and because the axes are rotational a straight line is not necessarily the fastest. The PTP test will be performed by programming the robot to move from the origin point (0, 0, 0) to the point (544,113,1). Figure 4-1 shows the tool path using ptp motion.

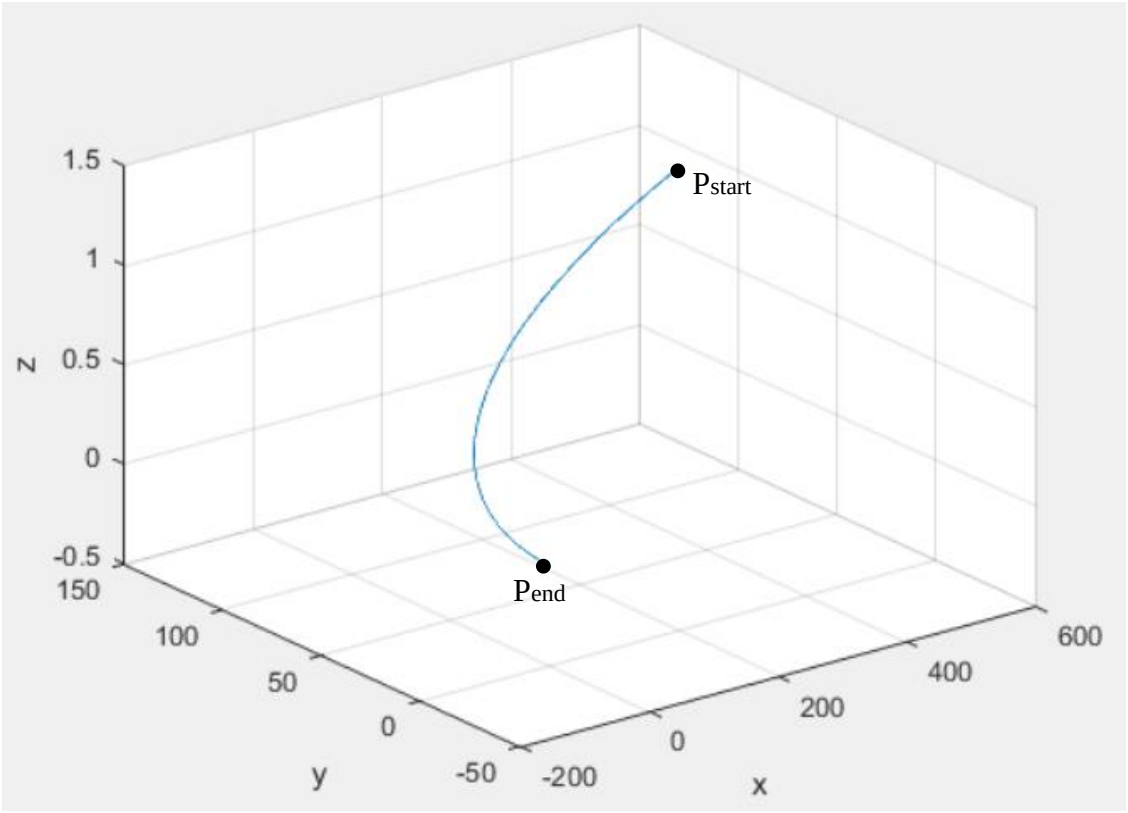

Figure 4-41: Tool path using ptp motion

As seen in the figure the ptp function from the package works exactly as the ptp motion using KRL. It is clear that the robot does not move in a straight line and it reached the needed point.

4.1.2 Linear motion test

The linear motion is slower than the PTP motion, during the motion the robot will guide the tool in a linear path. The test will be performed by moving the robot to the same points in the PTP test (0,0,0) and (544,113,20). Figure 4-2 shows the tool path using linear motion.

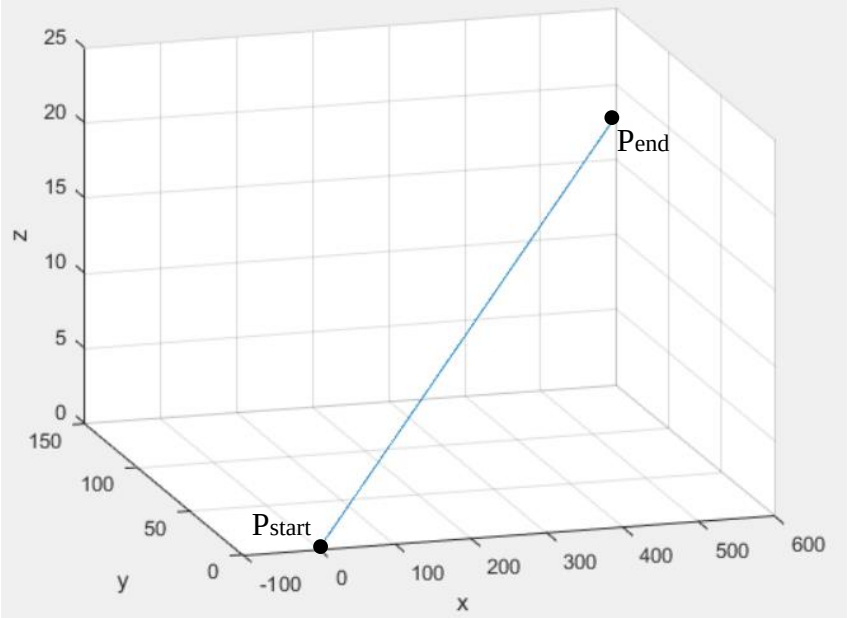

Figure 4-2: Tool path using linear motion

According to the figure above the linear motion function works fine however sometimes MATLAB sends the coordinates separately, so sometimes the robot will move along the x-axis and will not wait for the other coordinates. So, the robot will move in x-axis until the robot receives the other coordinates.

4.1.3 Circular motion test

Circular motion is different than the previous motions because the robot needs an auxiliary point to move through. The test will be performed by moving from point (2.55, 196.68,0) to point (121,314,0) with the auxiliary point (38.11,280,0). Figure 4-3 shows the tool path using circular motion.

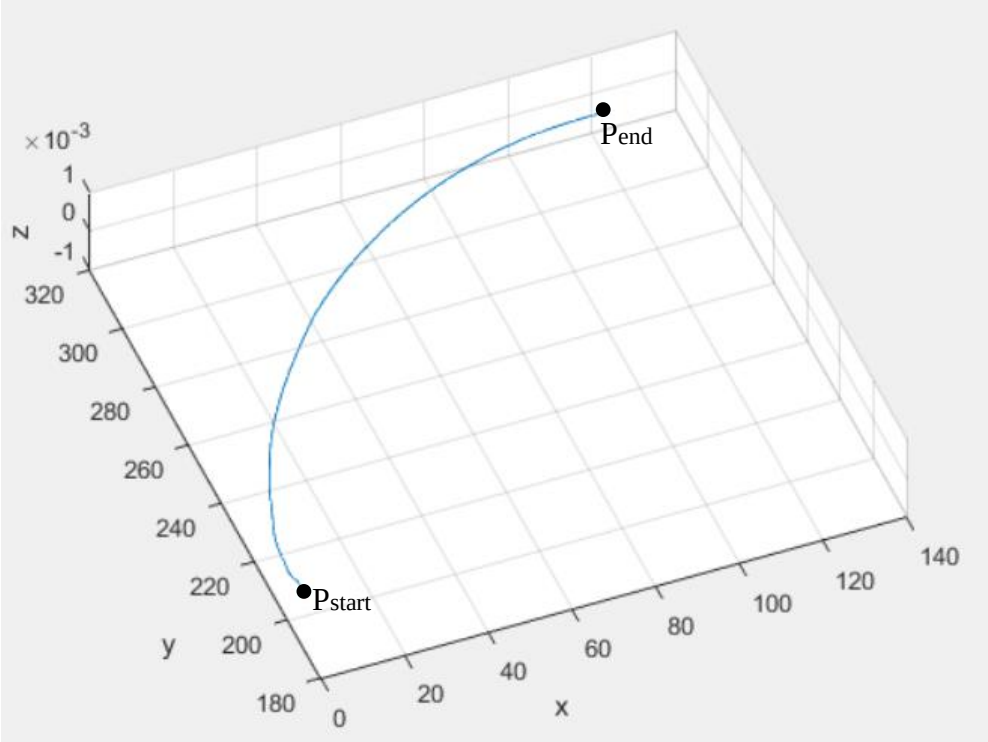

Figure 4-3: Tool path using circular motion

The circular motion function moves the robot in a circular path, however, there was an error with the function, where after completing the movement the program stops and forces the user to close the connection and ends the movement process. So, it is not possible to use this function in a program.

4.2 PROGRAMING TEST

This test is made to make sure that the package can be used to execute large programs, the robot will be programmed using the smart pad first then the package will be tested. The robot will try to follow the path shown in figure 4-4. Unfortunately, the circular motion will not be implemented in the program because after executing the function the robot will stuck and the user will be required to close the connection and run the program again. Table 8 shows the points coordinates.

Figure 4-4: Intended robot path

Table 6 : Test points coordinates

Point	X	Y	Z	A	B	C
P1	0	0	0	0	0	180
P2	2.55	196.68	0	0	0	180
P3	38.11	280	0	0	0	180
P4	121	314	0	0	0	180
P5	504.49	312.4	0	-0.45	-0.705	-156.3
P6	544.46	113	0	0	0	180
P7	426.09	-4.41	0	0	0	180

The points were chosen according to the calibrated base and the tool that was used for this experiment. To test the tool orientation point P5 has been placed outside the robot workspace so it had to be reached be rotating the tool. Figure 4-5 shows the tool path using a normal KRL program.

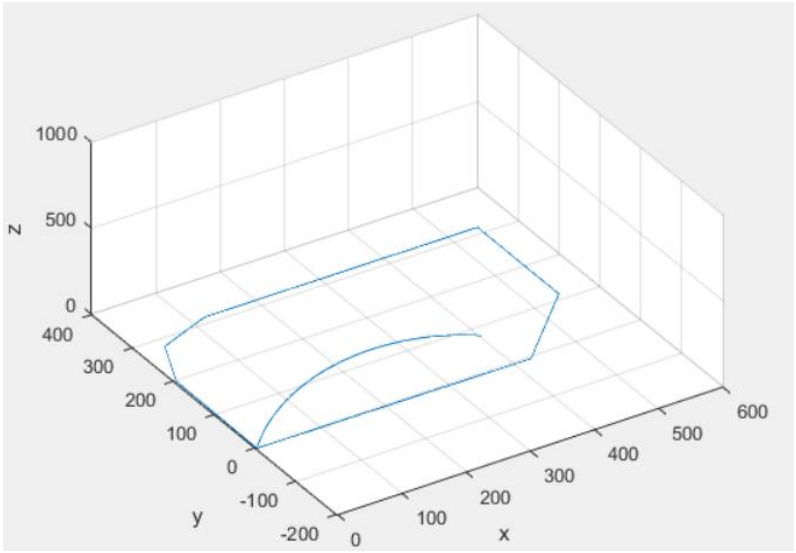

Figure 4-5: Tool path using a normal KRL program

To program the robot using the package, the program was written as a script which is shown in figure 4-6. To make sure that the package works fine the test was held for three times. Figure 4-7 shows the result for the robot motion when programmed using the package.

```
1 -    ptp(0.78,-2.25,0,0,0,180)
2 -    lin(2.55,196.68,0,0,0,180)
3 -    lin(38.11,280,0,0,0,180)
4 -    lin(121,312.4,0,0,0,180)
5 -    lin(450,312.4,0,-0.45,-0.705,-156.3)
6 -    lin(504.49,312.4,0,-0.45,-0.705,-156.3)
7 -    lin(504.49,312.4,0,0,0,180)
8 -    lin(544.46,113,1,0,0,180)
9 -    lin(426.09,-4.41,0,0,0,180)
10 -   lin(0.78,-2.25,0,0,0,180)
11 -   kukaclose
```

Figure 4-6: Robot movement program written in MATLAB

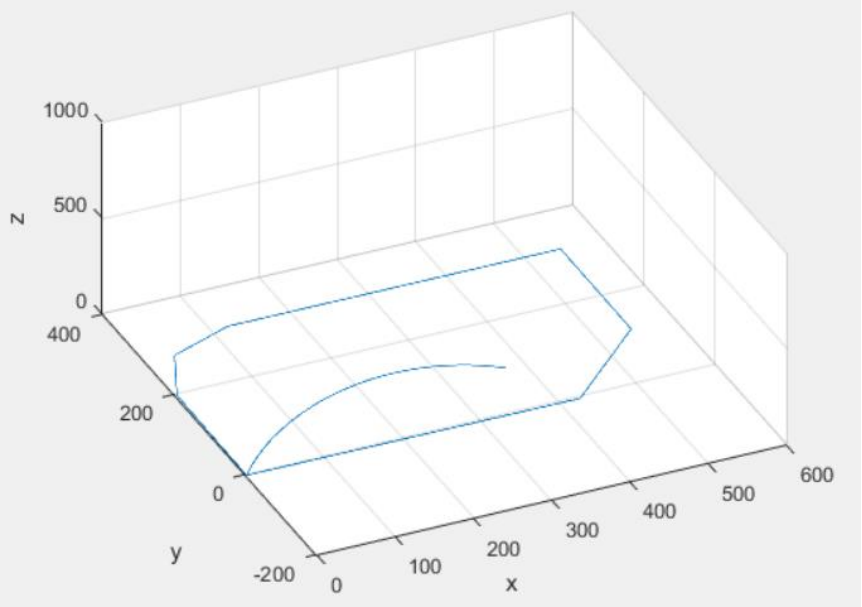

Figure 4-7: Tool path using the package

The result of the program was the same as the normal KRL program. However, due to the error function, the robot will take longer time than normal. Because the robot will not perform the next motion until it reaches the needed position. This is represented in figure 4-8 which shows the time it took the robot to reach the needed positions and end the program.

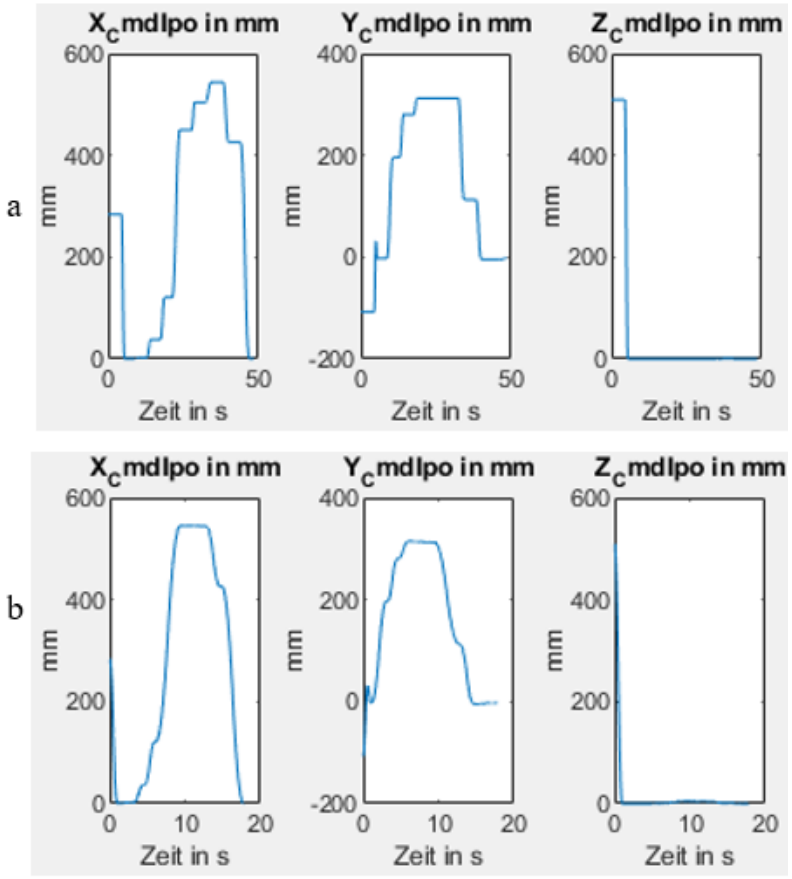

Figure 4-8: (a) Robot axis movement when programmed using the package.
(b) Robot axis movement when programmed using KRL.

As seen when programming the robot using KRL the robot will be faster, the robot took less than 20 seconds to finish the program, unlike the MATLAB program which takes up to 50 seconds to finish the program. It is also clear that when using the MATLAB package, the robot does not move in a continues path it has to stop to make sure that it reached the right position then continue to the next position.

4.3 DISCUSSION

The package was able to perform motions and perform full programs, so based on the pre-set criteria the package can be considered as acceptable. However, the program execution in the package is longer than the execution and some problems with the circular and linear motions. While using the linear function with low speed, sometimes the robot will not move in a linear motion instead it will move in axis motion means it will move

to the point by moving in the axes separately. This can be solved by sending the coordinates before the sitting the linear motion global variable, this way the robot will receive all the coordinates before moving. However, this solution can make it longer to execute the movement because the robot will have to receive the new coordinates before moving.

Also, in order to change the tool orientation, the user must declare a mid-point and make the robot changes ton. As seen on figure 4-6 commands in line 5 and 7 were only added for the tool orientations. This will not be a problem if the points are located on the work space, but in the experiment point, P5 is not in the work space so mid points had to be added. Figure 4-9 shows an approach without using mid-points.

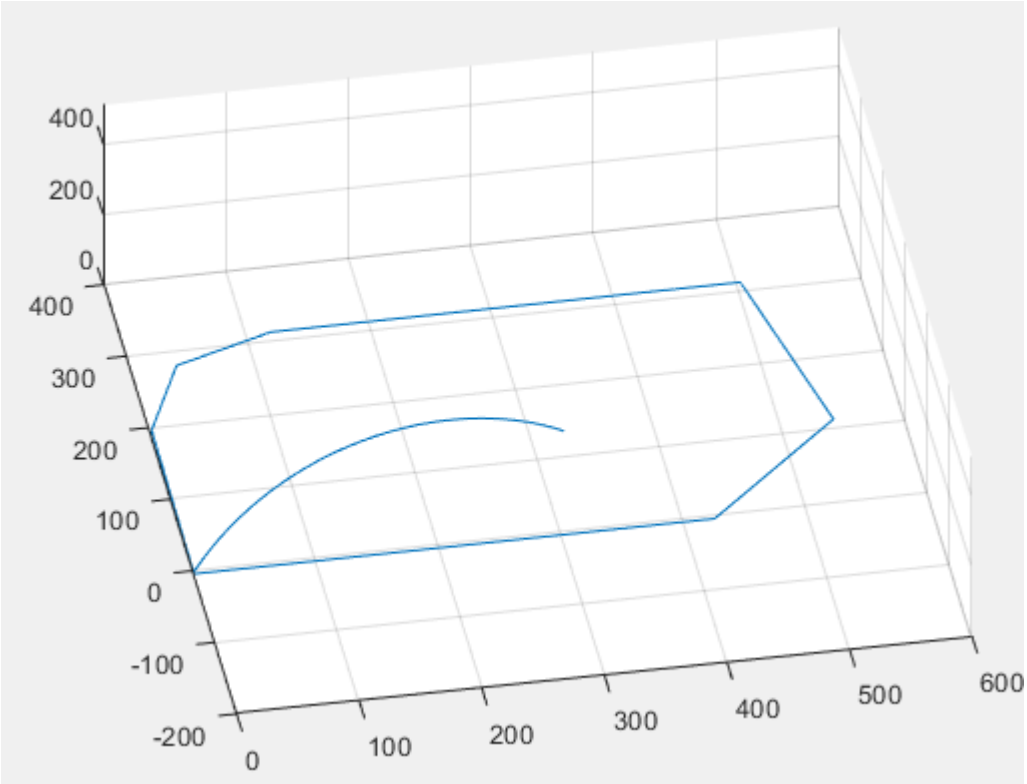

Figure 4-9: Tool path without using mid-points

CHAPTER 5: CONCLUSION AND FUTURE WORK

The aim of this Thesis was to find a way to program a Kuka robot from a remote PC and eliminate the need of additional software from Kuka and eliminate the need of the robot kinematics. The package was tested and it can be used to program the robot the result was compared with the results of a normal KRL program and the results were nearly the same

The package is easy to use, and unlike RSI based connections it does not need the robot kinematics and does not require any knowledge in other programming language. And because the KVP is compatible with every controller since KRC2 the package can work on various types of Kuka robots that uses KRC2, KRC3 and KRC4.

The package was tested for the three motions, the point-to-point and linear motions were tested and were successful. However, the circular motion showed a Problem where after executing the motion the robot get stuck, the developed package was validated by comparing the performance of the package and the performance of a normal KRL. The results shows that the developed package will move the robot to a specific point s same as a normal KRL. However, a normal KRL is faster than the developed package.

The program execution takes longer time than normal KRL because the error function will keep calculating the error until the robot reaches the intended position. Also, the robot motions will be unpredictable until the movement is over, because the controller is the one responsible for calculating the kinematics, so the user cannot predict the movement especially in linear and circular motions because they have no status and turn bits. Beside circular motion had a problem were the robot was stuck after execution the motion.

There are some suggestions for future works and adjustments need to be done:

- Improving the error function to decrease the execution time of the robot can improve the performance of the package.
- Solving the problem with the circular motion.
- Adding a function to control the speed of the robot during the movement execution.
- Developing a new set of functions to control the axes of the robot.

REFERENCES

[1] L. Takayama, W. Ju and C. Nass, "Beyond dirty, dangerous and dull: What everyday people think robots should do," *2008 3rd ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Amsterdam, 2008, pp. 25-32.

[2] M. Ben-Ari and F. Mondada, *Elements of Robotics*. Cham: Springer International Publishing AG, 2017.

[3] K. R. Gmbh,"www.kuka.com.".

[4] Braumann, J. and Brell-Cokcan, S., 2011. *Parametric robot control: integrated CAD/CAM for architectural design*.

[5] Mühe, H., Angerer, A., Hoffmann, A., & Reif, W. (2010). On reverse engineering the KUKA Robot Language. *arXiv preprint arXiv:1009.5004*.

[6] F. Chinello, S. Stefano, M. Fabio, and D. Prattichizzo, "KCT: a MATLAB toolbox for motion control of KUKA robot manipulators," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 4603–4608.

[7] G. Maletzki, T. Pawletta, S. Pawletta, and B. Lampe. A ModelBased Robot Programming Approach in the MATLAB-Simulink Environment. In Int. Conf. Manuf. Res., pages 377–382, 2006. [Online]. http://www.mb.hs-wismar.de/˜gunnar/ software/KukaKRLTbx.html.

[8] H. Elshatarat "*MATLAB Toolbox Implementation and Interface for Motion Control of KUKA KR6-R900-SIXX Robotic Manipulator*". Master Thesis, Bochum, 2016.

[9] X. Shi, F. Zhang, X. Qu, and B. Liu, "*An online real-time path compensation system for industrial robots based on laser tracker*," no. October, pp. 1–14, 2016.

[10] T. Wrütz, J. Golz, H. L. Elshatarat, and R. Biesenbach, "*Robot Offline Programming Tool (RoBO-2L) for Model-Based Design with Robot Offline Programming Tool (RoBO-2L) for Model-Based Design with MATLAB*," no. April 2016.

[11] Sanfilippo, F., Hatledal, L. I., Zhang, H., Fago, M., & Pettersen, K. Y. (2014, July). JOpenShowVar: an open-source cross-platform communication interface to kuka robots. In *Proc. of the IEEE International Conference on Information and Automation (ICIA), Hailar, China* (pp. 1154-1159).

[12] I. Eriksen, "Setup and Interfacing of a KUKA Robotics," Norwegian University of Science and Technology, 2017.

[13] Simulator for industrial robots and offline programming - robodk, "www.robodk.com.".

[14] "Definition of 'robot'. Oxford English Dictionary. Retrieved August 25, 2018."

[15] NASA, "www.jpl.nasa.gov/missions/details.php?id=5917," *Mars Exploration Rover*.

[16] Cubero, S. (2006). *Industrial robotics: Theory, modelling and control*. Pro Literatur Verlag.

[17] S. Briot and W. Khalil, *Dynamics of parallel robots*. 2015.

[18] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning, and control*. 2010.

[19] Adept Technology, "*Six-Axis Robot Configuration Singularities Use of the V + MV. SL _ MOVE Routine and the SPEED. LIMIT Parameter*," 2007.

[20] K. R. Gmbh, "*SOFTWARE KR C2 / KR C3 Expert Programming KUKA System Software (KSS)*," 2003.

[21] K. R. Gmbh, "*The control system of the future _KR C4*," 2016.

[22] K. R. Gmbh, "KUKA System Software 8 . 3: Operating and Programming Instructions for End Users," 2013.

[23] T.Beutler, J.Bolinger, C.Reisiger, "MATLAB-zur Auswertung und Darstellung von KUKA Traces", 2018.

[24] K. R. Gmbh, "KR 6 R900 sixx manual.", KUKA Robotics.

[25] K. Enien, "*Remote Motion Control of a KUKA industrial robot via Matlab by deriving its forward and inverse kinematics*," 2017.

APPENDIX A: KUKA KR-R900-SIXX KINEMATICS

- **D-H parameters**

To determine the D-H parameters for the robot it is necessary to represent the robot's frames. The frames are shown in figure 6-1 [6].

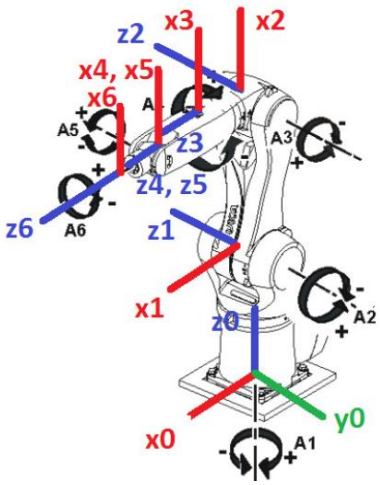

Figure 6-1: Manipulator frames

The D-H parameters for the Kuka KR-R900-Sixx are presented in table 3 [6].

Table 7 : D-H parameters for the KUKA KR-R900-Sixx

i+1	θ_{i+1}	d_{i+1} [mm]	a_{i+1} [mm]	α_{i+1} [rad]
1	θ_1	400	25	$\frac{\pi}{2}$

2	θ_2	0	455	0
3	θ_3	0	35	$\frac{\pi}{2}$
4	θ_4	420	0	$-\frac{\pi}{2}$
5	θ_5	0	0	$\frac{\pi}{2}$
6	θ_6	80	0	0

KR-R900-Sixx is a manipulator produced by Kuka robotics and belongs to the KR AGILUS series and its controller is KRC4, KR-R900-Sixx is a 6-DOF manipulator and it is the used manipulator in this work. Figure 6-2 shows the KR-R900-Sixx [24].

Figure 6-2: Kuka KR-R900-Sixx

Table 2 shows the specification of the KR-R900-Sixx [24].

Table 8 : Specification of the KR-R900-Sixx

Specification	Value
Max reach	901.5mm
Max payload	6kg
Pose repeatability	±0.03mm
Number of axes	6
Mounting position	Floor
Robot footprint	320mm x 320mm
Weight approx.	52 kg

The robot's axes cannot rotate for a full 360° turn as its axes have limitations, table 2 shows the axis limitations for the manipulator [24].

Table 9 : Axis and Axis Range for the manipulator

Axis	Axis Range
(A1)	+/- 170°
(A2)	- 190°/+45°
(A3)	-120°/+156°
(A4)	±185°
(A5)	±120°
(A6)	±350°

- **Forward Kinematics**

To calculate the forward kinematics for the robot it is necessary to find the transformation matrix of the tool with respect to the reference frame [6]. The tool transformation matrix is:

$$\boldsymbol{T_e^0 = T_6^0 T_e^6 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 T_6^5 T_e^6} \quad (8)$$

The result of these matrices multiplication will be a 4x4 matrix and can be expressed as [6].

$$\boldsymbol{T_e^0 = \begin{bmatrix} l_x & m_x & n_x & p_x \\ l_y & m_y & n_y & p_y \\ l_z & m_z & n_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}} \quad (9)$$

Where:

$$l_x = s_1(s_4c_5c_6 + c_4s_6) + c_1(-s_{23}s_5c_6 + c_{23}(c_4c_5c_6 - s_4s_6))$$

$$l_y = -c_1(s_4c_5c_6 + c_4s_6) + s_1(-s_{23}s_5c_6 + c_{23}(c_4c_5c_6 - s_4s_6))$$

$$l_z = -c_6(s_{23}c_4c_5 + c_{23}s_5) + s_{23}s_5s_6$$

$$m_x = c_6(s_1c_4 - c_1c_{23}s_4) - s_6(s_1s_4c_5 + c_1(c_{23}c_4c_5 - s_{23}s_5))$$

$$m_y = c_1(-c_4c_6 + c_5s_4s_6) - s_1(-s_{23}s_5s_6 + c_{23}(s_4c_6 + c_4c_5s_6))$$

$$m_z = s_{23}s_4c_6 + s_6(s_{23}c_4c_5 + c_{23}s_5)$$

$$n_x = -s_1s_4s_5 - c_1(s_{23}c_5 + c_{23}c_4s_5)$$

$$n_y = -s_1s_{23}c_5 + s_5(-s_1c_{23}c_4 + c_1s_4)$$

$$n_z = -c_{23}c_5 + c_4s_{23}s_5$$

$$p_s = -d_6s_1s_4s_5 + c_1(a_1 + a_2c_2 - s_{23}(d_4 + d_6c_5) + c_{23}(a_3 - d_6c_4s_5))$$

$$p_y = d_6c_1s_4s_5 + s_1(a_1 + a_2c_2 - s_{23}(d_4 + d_6c_5) + c_{23}(a_3 - d_6c_4s_5))$$

$$p_z = d_1 - c_{23}(d_4 + d_6c_5) - a_2s_2 + s_{23}(-a_3 + d_6c_4s_5)$$

Where: $s_i = \sin\theta_i$, $c_i = \cos\theta_i$, $s_{23} = \sin(\theta_2 + \theta_3)$ and $c_{23} = \cos(\theta_2 + \theta_3)$.

- **Inverse Kinematics**

Inverse kinematics is used for the control of manipulators. Solving the inverse kinematics takes a very long time in the real time control of manipulators. However, when singularity exists and the kinematics equations coupled it is difficult to solve inverse kinematics problem. Manipulator's tasks are in the Cartesian space, which includes orientation matrix and position vector. However, actuators work in joint space, which is represented by joint angles. The conversion of the position and orientation of a manipulator end-effector from Cartesian space to joint space is called as inverse kinematics problem [16].

Kuka KR-900-Sixx is a 6-DOF manipulator with a spherical wrist, with these characteristics the inverse problem could be divided into two sub problems: inverse position and inverse orientation. [25], This is called kinematics decoupling. This technique gives a close solution for the inverse kinematics problem and makes it easier to find the joint angles for a desired position, the first three joint could be found with the position information and the last three could be found using Euler angles and the knowledge of the desired orientation.

The first three joint angles for the KR-900-Sixx can be found by simplifying the robot inverse kinematics into a two-link manipulator kinematic figure 6-3 shows the top and front view of the robot [25].

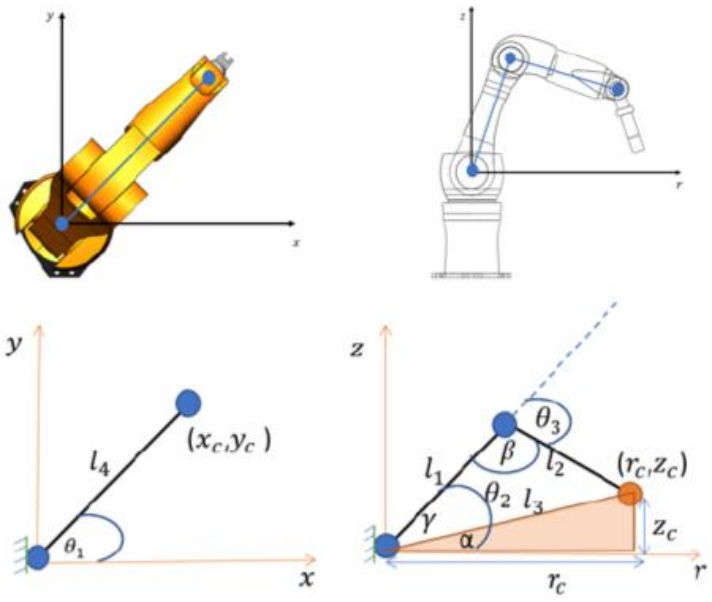

Figure 6-3: Kinematic analysis

The center of the wrist is represented in the Cartesian point $[x_c, y_c, z_c]$, and the plane

(r, z) is a plane parallel to z-axis and passing throw l_4. The last three angles can be found by using the Euler angles. Only the result for the inverse kinematics will be represented in this thesis, a more detailed solution for the inverse kinematics could be found in [25]. The robot inverse kinematics as expressed in [25] are:

$$\boldsymbol{\theta_1} = \boldsymbol{atan2}(\boldsymbol{y_{C,}}\, \boldsymbol{x_C}) \tag{10}$$

$$\boldsymbol{\theta_2} = -(\boldsymbol{\alpha} + \boldsymbol{\gamma}) \tag{11}$$

Where: $\alpha = atan2(z_c, r_c)$ & $\beta = \text{acos}\left(\frac{L_3^2+L_1^2-L_2^2}{2*L_1*L_2}\right)$

$$\boldsymbol{\theta_3} = \boldsymbol{\pi} - \boldsymbol{\beta} \tag{12}$$

Where: $\beta = \text{acos}\left(\frac{L_1^2+L_2^2-L_3^2}{2*L_1*L_2}\right)$

After finding the first three angles using the position of the end-effecter the last three angles could be found by using the inverse orientation. As mentioned, robot orientation could be represented using Euler angles. The Euler angles has different representation. The representation used to find the last three angles is the Z-Y-Z representation [25].

$$\boldsymbol{\theta_4} = \boldsymbol{atan2}(\boldsymbol{s_{23}}, \boldsymbol{s_{13}})\ \boldsymbol{atan2(s_{23}, s_{13})} \tag{13}$$

$$\boldsymbol{\theta_5} = \boldsymbol{atan2}(\sqrt{\boldsymbol{s_{13}^2} + \boldsymbol{s_{23}^2}}, \boldsymbol{s_{33}}) \tag{14}$$

$$\boldsymbol{\theta_6} = \boldsymbol{atan2}(\boldsymbol{s_{32}} - \boldsymbol{s_{31}}) \tag{15}$$

APPENDIX B: KRL CODE FOR KUKA MANIPULATOR

```
&ACCESS RVO
&REL 209
&PARAM EDITMASK = *
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
&PARAM DISKPATH = KRC:\R1
; ================================================
DEF matlab( )
; ================================================
; Declaration of KRL variables
DECL POS POSITION
DECL POS PAUX
;FOLD INI;%{PE}
  ;FOLD BASISTECH INI
    GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
    INTERRUPT ON 3
    BAS (#INITMOV,0 )
  ;ENDFOLD (BASISTECH INI)
;FOLD SPOTTECH INI
USERSPOT(#INIT)
;ENDFOLD (SPOTTECH INI)
;ENDFOLD (INI)

; Move to Home position
```

```
;FOLD PTP HOME Vel=100 % DEFAULT;%{PE}%R
8.3.34,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:HOME, 3:, 5:100,
7:DEFAULT
$BWDSTART=FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS(#PTP_PARAMS,100)
$H_POS=XHOME
PTP XHOME
;ENDFOLD

WHILE $STR==1
$BASE= BASE_DATA[11]
$TOOL= TOOL_DATA[11]
; Move Point-To-Point with tool Orintation and status & ;turn
IF $MPS == 1 THEN
POSITION.X= $XAC
POSITION.Y= $YAC
POSITION.Z= $ZAC
POSITION.a= $RAA
POSITION.b= $RBA
POSITION.c= $RCA
POSITION.s= $RSS
POSITION.t= $RST
PTP POSITION
ENDIF

; Move Point-To-Point with tool Orintation
IF $MPT == 1 THEN
POSITION.X= $XAC
POSITION.Y= $YAC
POSITION.Z= $ZAC
POSITION.a= $RAA
POSITION.b= $RBA
POSITION.c= $RCA
PTP POSITION
ENDIF

; Move Point-To-Point
IF $MPT3 == 1 THEN
POSITION.X= $XAC
POSITION.Y= $YAC
POSITION.Z= $ZAC
PTP POSITION
ENDIF

; Move Linear Motion with tool Orintation
IF $MLN == 1 THEN
POSITION.X= $XAC
POSITION.Y= $YAC
POSITION.Z= $ZAC
POSITION.a= $RAA
POSITION.b= $RBA
POSITION.c= $RCA
```

```
LIN POSITION
ENDIF

; Move Linear Motion
IF $MLN3 == 1 THEN
POSITION.X= $XAC
POSITION.Y= $YAC
POSITION.Z= $ZAC
LIN POSITION
ENDIF

; Move Circular Motion with tool Orintation
IF $MCC == 1 THEN
POSITION.X= $XAC
POSITION.Y= $YAC
POSITION.Z= $ZAC
POSITION.a= $RAA
POSITION.b= $RBA
POSITION.c= $RCA
PAUX.X=$XAX
PAUX.Y=$YAX
PAUX.Z=$ZAX
PAUX.A=$AAX
PAUX.B=$BAX
PAUX.c=$CAX
CIRC PAUX, POSITION
ENDIF

; Move Circular Motion
IF $CRC3 == 1 THEN
POSITION.X= $XAC
POSITION.Y= $YAC
POSITION.Z= $ZAC
PAUX.X=$XAX
PAUX.Y=$YAX
PAUX.Z=$ZAX
CIRC PAUX, POSITION
ENDIF

IF $GRP == 1 THEN
;FOLD SET GRP 1 State=OPN GDAT1;%{PE}%R
8.3.1,%MKUKATPGRP,%CGRP,%VGRP,%P 2:1, 4:1, 5:#NO, 6:GDAT1, 8:0,
10:0
H50(GRP,1,1,GGDAT1)
;ENDFOLD
ENDIF
IF $GRP ==0 THEN
;FOLD SET GRP 1 State=CLO GDAT2;%{PE}%R
8.3.1,%MKUKATPGRP,%CGRP,%VGRP,%P 2:1, 4:2, 5:#NO, 6:GDAT2, 8:0,
10:0
H50(GRP,1,2,GGDAT2)
;ENDFOLD
ENDIF
ENDWHILE
```

```
;FOLD PTP HOME  Vel= 100 %
DEFAULT;%{PE}%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:HOME, 3:,
5:100, 7:DEFAULT
$BWDSTART = FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS (#PTP_PARAMS,100 )
$H_POS=XHOME
PTP  XHOME
;ENDFOLD
END
```

APPENDIX C: MATLAB PACKAGE FUNCTIONS

❖ KUKACNCT.m

```
function [t] = KUKACNCT()
%This function is used to connect matlab to KUKAVARPROXY on the
robot
%Please enter the robot ip address to establish connection between
matlab
%and the robot
%This function is invoced in the movment functions so the user only
have to
%change the ip address.
%----------Robot Ip------port
%           ||            ||
%           \/            \/
t= tcpip('172.31.1.147', 7000,'Timeout', 1);
t.OutputBufferSize = 100;
t.InputBufferSize = 100;
end
```

❖ Kukaerr.m

```
function [err] = kukaerr(varargin)
```

```
%This function calculates the error between the needed position and
the
%actual position of the robot.
M1=kukard;
        x= varargin{1};
        y= varargin{2};
        z= varargin{3};
pos=[x, y, z];
for i=1:3
    posi=['X','Y','Z'];
    Str = [M1];
Str(strfind(Str, '=')) = [];
Key   = posi(1,i);
Index = strfind(Str, Key);
Value(i) = sscanf(Str(Index(1) + length(Key):end), '%g', 1);
end
err1=abs(Value(1)-pos(1,1));
err2=abs(Value(2)-pos(1,2));
err3=abs(Value(3)-pos(1,3));
err= (err1 +err2+err3) /3;
end
```

❖ Kukastrt.m

```
function [open] = kukastrt()
%this function is used to reset all the commands global variables
%this function must be used before moving the robot to avoid any
problems
%with the robot.
[t] = KUKACNCT();
mpt=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 84, 0, 4, 48];
mps=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 83, 0, 4, 48];
mln=[0, 99, 0, 10, 1, 0, 4, 36, 77, 76, 78, 0, 4, 48];
mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
mpt3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 80, 84, 51, 0, 5, 48];
mln3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 76, 78, 51, 0, 5, 48];
crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5, 48];
str=[0, 99, 0, 10, 1, 0, 4, 36, 83, 84, 82, 0, 4, 49];
MOV={mpt,mps,mln,mcc,mpt3,mln3,crc3,str};
for MV=1:8
    fopen(t);
    fwrite(t,MOV{1,MV});
    fclose(t);
end


end
```

❖ Kukaclose.m

```
function [dis] = kukaclose()
%this function is used to reset all the commands global variables
%this function must be used after working on the robot to avoid any
%problems with the robot when used again
[t] = KUKACNCT();
mpt=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 84, 0, 4, 48];
mps=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 83, 0, 4, 48];
mln=[0, 99, 0, 10, 1, 0, 4, 36, 77, 76, 78, 0, 4, 48];
mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
mpt3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 80, 84, 51, 0, 5, 48];
mln3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 76, 78, 51, 0, 5, 48];
crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5, 48];
str=[0, 99, 0, 10, 1, 0, 4, 36, 83, 84, 82, 0, 4, 48];
MOV={mpt,mps,mln,mcc,mpt3,mln3,crc3,str};
for MV=1:8
    fopen(t);
    fwrite(t,MOV{1,MV});
    fclose(t);
end
end
```

❖ Ptp.m

```
function [pos] = ptp(varargin)
%This function is for the point to point motion command
%   this function can work with three inputs  were in this case the
inputs are X
%   Y & Z
%Also it can work with six inputs were in this case the inputs are
X
%   Y Z A B & C
%Also it can work with eight inputs were in this case the inputs
are X
%   Y Z A B C and the status and turn bits
%note : status and turn bits needs to be enterd as decimal values
not
%binary.
[t] = KUKACNCT();
t.OutputBufferSize = 2000;
fclose(t);
error=1;
while error == 1
x= num2str(varargin{1});
y= num2str(varargin{2});
z= num2str(varargin{3});
pos=[x,y,z];
[err] = kukaerr(varargin{1},varargin{2},varargin{3});
if err>1
error=1;
switch nargin
```

```
case 0
fprintf(2,'\nNo inputs have been given\n')
fprintf(2,'\nFunction can not procced\n')
case 3
%sending the motion variables values
disp('PTP with three points')
x= num2str(varargin{1});
y= num2str(varargin{2});
z= num2str(varargin{3});
mpt=[0, 1, 0, 10, 1, 0, 4, 36, 77, 80, 84, 0, 4, 48];
mps=[0, 2, 0, 10, 1, 0, 4, 36, 77, 80, 83, 0, 4, 48];
mln=[0, 3, 0, 10, 1, 0, 4, 36, 77, 76, 78, 0, 4, 48];
mcc=[0, 4, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
mpt3=[0, 5, 0, 11, 1, 0, 5, 36, 77, 80, 84, 51, 0, 5, 49];
mln3=[0, 6, 0, 11, 1, 0, 5, 36, 77, 76, 78, 51, 0, 5, 48];
crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5, 48];
MOV={mpt,mps,mln,mcc,mpt3,mln3,crc3};
for MV=1:7
    fopen(t);
    fwrite(t,MOV{1,MV});
    fclose(t);
end
%Sending the coordinates
for g=1:3
    if g==1
        bx = unicode2native(x);
        h=numel(bx);
        f=9+h;
        X=[0, 90, 0, f, 1, 0, 4, 36, 88, 65, 67, 0, 4,bx];
        BS=f+4;
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,X);
    end
    if g==2
        by = unicode2native(y);
        h=numel(by);
        f=9+h;
        BS=f+4;
        Y=[0, 80, 0, f, 1, 0, 4, 36, 89, 65, 67, 0, 4,by];
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,Y);
    end
    if g==3
        bz = unicode2native(z);
        h=numel(bz);
        f=9+h;
        Z=[0, 70, 0, f, 1, 0, 4, 36, 90, 65, 67, 0, 4,bz];
        BS=f+4;
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,Z);
    end
    fclose(t);
end
pos=[x,y,z];
case 6
disp('PTP with no S&T')
```

```
x= num2str(varargin{1});
y= num2str(varargin{2});
z= num2str(varargin{3});
a= num2str(varargin{4});
b= num2str(varargin{5});
c= num2str(varargin{6});
mpt=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 84, 0, 4, 49];
mps=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 83, 0, 4, 48];
mln=[0, 99, 0, 10, 1, 0, 4, 36, 77, 76, 78, 0, 4, 48];
mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
mpt3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 80, 84, 51, 0, 5, 48];
mln3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 76, 78, 51, 0, 5, 48];
crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5, 48];
MOV={mpt,mps,mln,mcc,mpt3,mln3,crc3};
for MV=1:7
    fopen(t);
    fwrite(t,MOV{1,MV});
    fclose(t);
end

for g=1:6
    if g==1
        bx = unicode2native(x);
        h=numel(bx);
        f=9+h;
        X=[0, 90, 0, f, 1, 0, 4, 36, 88, 65, 67, 0, 4,bx];
        BS=f+4;
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,X);
    end
    if g==2
        by = unicode2native(y);
        h=numel(by);
        f=9+h;
        BS=f+4;
        Y=[0, 80, 0, f, 1, 0, 4, 36, 89, 65, 67, 0, 4,by];
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,Y);
    end
    if g==3
        bz = unicode2native(z);
        h=numel(bz);
        f=9+h;
        Z=[0, 70, 0, f, 1, 0, 4, 36, 90, 65, 67, 0, 4,bz];
        BS=f+4;
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,Z);
    end
    if g==4

        ba = unicode2native(a);
        h=numel(ba);
        f=9+h;
        S=[0, 99, 0, f, 1, 0, 4, 36, 82, 65, 65, 0, 4, ba];
        BS=f+4
        t.OutputBufferSize = BS;
```

```
        fopen(t);
        fwrite(t,S);
    end
    if g==5
        bb = unicode2native(b);
        h=numel(bb);
        f=9+h;
        S=[0, 99, 0, f, 1, 0, 4, 36, 82, 66, 65, 0, 4,bb];
        BS=f+4;
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,S);
    end
    if g==6
        bc = unicode2native(c);
        h=numel(bc);
        f=9+h;
        S=[0, 99, 0, f, 1, 0, 4, 36, 82, 67, 65, 0, 4,bc];
        BS=f+4;
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,S);
    end
    fclose(t);
end
pos=[x,y,z,a,b,c];

case 8
disp('PTP with S&T')
x= num2str(varargin{1});
y= num2str(varargin{2});
z= num2str(varargin{3});
a= num2str(varargin{4});
b= num2str(varargin{5});
c= num2str(varargin{6});
st= num2str(varargin{7});
tr= num2str(varargin{8});
mpt=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 84, 0, 4, 48];
mps=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 83, 0, 4, 49];
mln=[0, 99, 0, 10, 1, 0, 4, 36, 77, 76, 78, 0, 4, 48];
mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
mpt3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 80, 84, 51, 0, 5, 48];
mln3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 76, 78, 51, 0, 5, 48];
crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5, 48];
MOV={mpt,mps,mln,mcc,mpt3,mln3,crc3};
for MV=1:7
    fopen(t);
    fwrite(t,MOV{1,MV});
    fclose(t);
end
for g=1:8
    if g==1
        bx = unicode2native(x);
        h=numel(bx);
        f=9+h;
        X=[0, 90, 0, f, 1, 0, 4, 36, 88, 65, 67, 0, 4,bx];
        BS=f+4;
        t.OutputBufferSize = BS;
        fopen(t);
```

```
    fwrite(t,X);
end
if g==2
    by = unicode2native(y);
    h=numel(by);
    f=9+h;
    BS=f+4;
    Y=[0, 80, 0, f, 1, 0, 4, 36, 89, 65, 67, 0, 4,by];
    t.OutputBufferSize = BS;
    fopen(t);
    fwrite(t,Y);
end
if g==3
    bz = unicode2native(z);
    h=numel(bz);
    f=9+h;
    Z=[0, 70, 0, f, 1, 0, 4, 36, 90, 65, 67, 0, 4,bz];
    BS=f+4;
    t.OutputBufferSize = BS;
    fopen(t);
    fwrite(t,Z);
end
if g==4
    ba = unicode2native(a);
    h=numel(ba);
    f=9+h;
    S=[0, 99, 0, f, 1, 0, 4, 36, 82, 65, 65, 0, 4, ba];
    BS=f+4;
    t.OutputBufferSize = BS;
    fopen(t);
    fwrite(t,S);
end
if g==5
    bb = unicode2native(b);
    h=numel(bb);
    f=9+h;
    S=[0, 99, 0, f, 1, 0, 4, 36, 82, 66, 65, 0, 4,bb];
    BS=f+4;
    t.OutputBufferSize = BS;
    fopen(t);
    fwrite(t,S);
end
if g==6
    bc = unicode2native(c);
    h=numel(bc);
    f=9+h;
    S=[0, 99, 0, f, 1, 0, 4, 36, 82, 67, 65, 0, 4,bc];
    BS=f+4;
    t.OutputBufferSize = BS;
    fopen(t);
    fwrite(t,S);
end
if g==7
    bs = unicode2native(st);
    h=numel(bs);
    f=9+h;
    S=[0, 60, 0, f, 1, 0, 4, 36, 82, 83, 83, 0, 4,bs];
    BS=f+4;
    t.OutputBufferSize = BS;
```

```
        fopen(t);
        fwrite(t,S);
    end
    if g==8
        btr = unicode2native(tr);
        h=numel(btr);
        f=9+h;
        S=[0, 99, 0, f, 1, 0, 4, 36, 82, 83, 84, 0, 4, btr];
        BS=f+4;
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,S);
    end
    fclose(t);
end
pos=[x,y,z,a,b,c,st,tr];
end
else
error=0;
end
end
end
```

❖ lin.m

```
function [pos] = lin(varargin)
%this function is for the linear motion command
%   this function can work with three inputs  were in this case the
inputs are X
%   Y & Z
%also it can work with six inputs were in this case the inputs are
X
%   Y Z A B & C

[t] = KUKACNCT();
t.OutputBufferSize = 2000;
fclose(t);
error=1;
while error == 1
x= num2str(varargin{1});
y= num2str(varargin{2});
z= num2str(varargin{3});
pos=[x,y,z];
[err] = kukaerr(varargin{1},varargin{2},varargin{3});
if err>1
error=1;
switch nargin
case 0
    fprintf(2,'\nNo inputs have been given\n')
```

```
    fprintf(2,'\nFunction can not procced\n')
case 3
    %sending the motion variables values
    disp('LIN with three points')
    x= num2str(varargin{1});
    y= num2str(varargin{2});
    z= num2str(varargin{3});
    mpt=[0, 99, 0,10, 1, 0, 4, 36, 77, 80, 84, 0, 4, 48];
    mps=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 83, 0, 4, 48];
    mln=[0, 99, 0, 10, 1, 0, 4, 36, 77, 76, 78, 0, 4, 48];
    mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
    mpt3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 80, 84, 51, 0, 5, 48];
    mln3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 76, 78, 51, 0, 5, 49];
    crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5, 48];
    MOV={mpt,mps,mln,mcc,mpt3,mln3,crc3};
    for MV=1:7
        fopen(t);
        fwrite(t,MOV{1,MV});
        fclose(t);
    end
    %sending the coordinates
    for g=1:3
        if g==1
            bx = unicode2native(x);
            h=numel(bx);
            f=9+h;
            X=[0, 90, 0, f, 1, 0, 4, 36, 88, 65, 67, 0, 4,bx];
            BS=f+4;
            t.OutputBufferSize = BS;
            fopen(t);
            fwrite(t,X);
        end
        if g==2
            by = unicode2native(y);
            h=numel(by);
            f=9+h;
            BS=f+4;
            Y=[0, 80, 0, f, 1, 0, 4, 36, 89, 65, 67, 0, 4,by];
            t.OutputBufferSize = BS;
            fopen(t);
            fwrite(t,Y);
        end
        if g==3
            bz = unicode2native(z);
            h=numel(bz);
            f=9+h;
            Z=[0, 70, 0, f, 1, 0, 4, 36, 90, 65, 67, 0, 4,bz];
            BS=f+4;
            t.OutputBufferSize = BS;
            fopen(t);
            fwrite(t,Z);
        end
        fclose(t);
    end
    pos=[x,y,z];
case 6
    disp('LIN with six points')
    x= num2str(varargin{1});
    y= num2str(varargin{2});
```

```
z= num2str(varargin{3});
a= num2str(varargin{4});
b= num2str(varargin{5});
c= num2str(varargin{6});
mpt=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 84, 0, 4, 48];
mps=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 83, 0, 4, 48];
mln=[0, 99, 0, 10, 1, 0, 4, 36, 77, 76, 78, 0, 4, 49];
mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
mpt3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 80, 84, 51, 0, 5, 48];
mln3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 76, 78, 51, 0, 5, 48];
crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5, 48];
MOV={mpt,mps,mln,mcc,mpt3,mln3,crc3};
for MV=1:7
    fopen(t);
    fwrite(t,MOV{1,MV});
    fclose(t);
end
for g=1:6
    if g==1
        bx = unicode2native(x);
        h=numel(bx);
        f=9+h;
        X=[0, 90, 0, f, 1, 0, 4, 36, 88, 65, 67, 0, 4,bx];
        BS=f+4;
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,X);
    end
    if g==2
        by = unicode2native(y);
        h=numel(by);
        f=9+h;
        BS=f+4;
        Y=[0, 80, 0, f, 1, 0, 4, 36, 89, 65, 67, 0, 4,by];
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,Y);
    end
    if g==3
        bz = unicode2native(z);
        h=numel(bz);
        f=9+h;
        Z=[0, 70, 0, f, 1, 0, 4, 36, 90, 65, 67, 0, 4,bz];
        BS=f+4;
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,Z);
    end
    if g==4

        ba = unicode2native(a);
        h=numel(ba);
        f=9+h;
        S=[0, 99, 0, f, 1, 0, 4, 36, 82, 65, 65, 0, 4, ba];
        BS=f+4;
        t.OutputBufferSize = BS;
        fopen(t);
        fwrite(t,S);
    end
```

```
        if g==5

            bb = unicode2native(b);
            h=numel(bb);
            f=9+h;
            S=[0, 99, 0, f, 1, 0, 4, 36, 82, 66, 65, 0, 4,bb];
            BS=f+4;
            t.OutputBufferSize = BS;
            fopen(t);
            fwrite(t,S);
        end
        if g==6

            bc = unicode2native(c);
            h=numel(bc);
            f=9+h;
            S=[0, 99, 0, f, 1, 0, 4, 36, 82, 67, 65, 0, 4,bc];
            BS=f+4;
            t.OutputBufferSize = BS;
            fopen(t);
            fwrite(t,S);
        end
        fclose(t);
    end
    pos=[x y z a b c];

end
else
error=0;
end
end
end
```

❖ **crc.m**

```
function [fpos,auxpos] = crc(varargin)
%this function is for the circular motion command
%   this function works with 6 were the first three inputs are the
final
%   position coordinates and the last three inputs are the auxulary
point
%   coordinats
%also it can work with 12 inputs were the first six inputs are the
final
%   position coordinates and the last six inputs are the auxulary
point
%   coordinats
[t] = KUKACNCT();
fclose(t);
error=1;
while error == 1
x= num2str(varargin{1});
y= num2str(varargin{2});
z= num2str(varargin{3});
pos=[x,y,z];
[err] = kukaerr(varargin{1},varargin{2},varargin{3});
if err>1
error=1;
switch nargin
```

```
    case 6
        %sending the motion variables vaues
        disp('CRC with three points')
        x= num2str(varargin{1});
        y= num2str(varargin{2});
        z= num2str(varargin{3});
        xax= num2str(varargin{4});
        yax= num2str(varargin{5});
        zax= num2str(varargin{6});
        mpt=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 84, 0, 4, 48];
        mps=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 83, 0, 4, 48];
        mln=[0, 99, 0, 10, 1, 0, 4, 36, 77, 76, 78, 0, 4, 48];
        mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
        mpt3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 80, 84, 51, 0, 5, 48];
        mln3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 76, 78, 51, 0, 5, 48];
        crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5,
48];
        MOV={mpt,mps,mln,mcc,mpt3,mln3,crc3};
        for MV=1:7
            fopen(t);
            fwrite(t,MOV{1,MV});
            fclose(t);
        end
        %Sending the coordinates
        for g=1:6
            if g==1
                bx = unicode2native(xax);
                h=numel(bx);
                f=9+h;
                X=[0, 90, 0, f, 1, 0, 4, 36, 88, 65, 88, 0, 4,bx];
                BS=f+4;
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,X);
            end
            if g==2
                by = unicode2native(yax);
                h=numel(by);
                f=9+h;
                BS=f+4;
                Y=[0, 80, 0, f, 1, 0, 4, 36, 89, 65, 88, 0, 4,by];
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,Y);
            end
            if g==3
                bz = unicode2native(zax);
                h=numel(bz);
                f=9+h;
                Z=[0, 70, 0, f, 1, 0, 4, 36, 90, 65, 88, 0, 4,bz];
                BS=f+4;
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,Z);
            end
            if g==4
                bx = unicode2native(x);
                h=numel(bx);
                f=9+h;
```

```
                X=[0, 90, 0, f, 1, 0, 4, 36, 88, 65, 67, 0, 4,bx];
                BS=f+4;
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,X);
            end
            if g==5
                by = unicode2native(y);
                h=numel(by);
                f=9+h;
                BS=f+4;
                Y=[0, 80, 0, f, 1, 0, 4, 36, 89, 65, 67, 0, 4,by];
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,Y);
            end
            if g==6
                bz = unicode2native(z);
                h=numel(bz);
                f=9+h;
                Z=[0, 70, 0, f, 1, 0, 4, 36, 90, 65, 67, 0, 4,bz];
                BS=f+4;
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,Z);
            end
            fclose(t);
        end
        mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
        crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5,
49];
        MOV={mcc,crc3};
        for MV=1:2
            fopen(t);
            fwrite(t,MOV{1,MV});
            fclose(t);
        end
        fpos=[x,y,z];
        auxpos=[xax,yax,zax];
    case 12
        disp('CRC with three points')
        x= num2str(varargin{1});
        y= num2str(varargin{2});
        z= num2str(varargin{3});
        a= num2str(varargin{4});
        b= num2str(varargin{5});
        c= num2str(varargin{6});
        xax= num2str(varargin{7});
        yax= num2str(varargin{8});
        zax= num2str(varargin{9});
        aax= num2str(varargin{10});
        bax= num2str(varargin{11});
        cax= num2str(varargin{12});
        mpt=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 84, 0, 4, 48];
        mps=[0, 99, 0, 10, 1, 0, 4, 36, 77, 80, 83, 0, 4, 48];
        mln=[0, 99, 0, 10, 1, 0, 4, 36, 77, 76, 78, 0, 4, 48];
        mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
        mpt3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 80, 84, 51, 0, 5, 48];
        mln3=[0, 99, 0, 11, 1, 0, 5, 36, 77, 76, 78, 51, 0, 5, 48];
```

```
        crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5,
48];
        MOV={mpt,mps,mln,mcc,mpt3,mln3,crc3};
        for MV=1:7
            fopen(t);
            fwrite(t,MOV{1,MV});
            fclose(t);
        end
        for g=1:12
            if g==1
                bx = unicode2native(xax);
                h=numel(bx);
                f=9+h;
                X=[0, 90, 0, f, 1, 0, 4, 36, 88, 65, 88, 0, 4,bx];
                BS=f+4;
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,X);
            end
            if g==2
                by = unicode2native(yax);
                h=numel(by);
                f=9+h;
                BS=f+4;
                Y=[0, 80, 0, f, 1, 0, 4, 36, 89, 65, 88, 0, 4,by];
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,Y);
            end
            if g==3
                bz = unicode2native(zax);
                h=numel(bz);
                f=9+h;
                Z=[0, 70, 0, f, 1, 0, 4, 36, 90, 65, 88, 0, 4,bz];
                BS=f+4;
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,Z);
            end

            if g==4

                ba = unicode2native(aax);
                h=numel(ba);
                f=9+h;
                S=[0, 99, 0, f, 1, 0, 4, 36, 65, 65, 88, 0, 4, ba];
                BS=f+4;
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,S);
            end
            if g==5

                bb = unicode2native(bax);
                h=numel(bb);
                f=9+h;
                S=[0, 99, 0, f, 1, 0, 4, 36, 66, 65, 88, 0, 4,bb];
                BS=f+4;
                t.OutputBufferSize = BS;
```

```
    fopen(t);
    fwrite(t,S);
end
if g==6

    bc = unicode2native(cax);
    h=numel(bc);
    f=9+h;
    S=[0, 99, 0, f, 1, 0, 4, 36, 67, 65, 88, 0, 4,bc];
    BS=f+4;
    t.OutputBufferSize = BS;
    fopen(t);
    fwrite(t,S);
end
if g==7
    bx = unicode2native(x);
    h=numel(bx);
    f=9+h;
    X=[0, 90, 0, f, 1, 0, 4, 36, 88, 65, 67, 0, 4,bx];
    BS=f+4;
    t.OutputBufferSize = BS;
    fopen(t);
    fwrite(t,X);
end
if g==8
    by = unicode2native(y);
    h=numel(by);
    f=9+h;
    BS=f+4;
    Y=[0, 80, 0, f, 1, 0, 4, 36, 89, 65, 67, 0, 4,by];
    t.OutputBufferSize = BS;
    fopen(t);
    fwrite(t,Y);
end
if g==9
    bz = unicode2native(z);
    h=numel(bz);
    f=9+h;
    Z=[0, 70, 0, f, 1, 0, 4, 36, 90, 65, 67, 0, 4,bz];
    BS=f+4;
    t.OutputBufferSize = BS;
    fopen(t);
    fwrite(t,Z);
end
if g==10

    ba = unicode2native(a);
    h=numel(ba);
    f=9+h;
    S=[0, 99, 0, f, 1, 0, 4, 36, 82, 65, 65, 0, 4, ba];
    BS=f+4;
    t.OutputBufferSize = BS;
    fopen(t);
    fwrite(t,S);
end
if g==11

    bb = unicode2native(b);
    h=numel(bb);
```

```
                f=9+h;
                S=[0, 99, 0, f, 1, 0, 4, 36, 82, 66, 65, 0, 4,bb];
                BS=f+4;
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,S);
            end
            if g==12

                bc = unicode2native(c);
                h=numel(bc);
                f=9+h;
                S=[0, 99, 0, f, 1, 0, 4, 36, 82, 67, 65, 0, 4,bc];
                BS=f+4;
                t.OutputBufferSize = BS;
                fopen(t);
                fwrite(t,S);
            end
            fclose(t);
        end
        mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 49];
        crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5,
48];
        MOV={mcc,crc3};
        for MV=1:2
            fopen(t);
            fwrite(t,MOV{1,MV});
            fclose(t);
        end
        fpos=[x,y,z,a,b,c];
        auxpos=[xax,yax,zax,aax,bax,cax];

    case 0
        fprintf(2,'\nEnter inputs as array\n')
        fprintf(2,'\nFunction can not procced\n')
end
fclose(t);
else
error=0;
mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5, 48];
MOV={mcc,crc3};
for MV=1:2
    fopen(t);
    fwrite(t,MOV{1,MV});
    fclose(t);
end
end
mcc=[0, 99, 0, 10, 1, 0, 4, 36, 77, 67, 67, 0, 4, 48];
crc3=[0, 99, 0, 11, 1, 0, 5, 36, 99, 114, 99, 51, 0, 5, 48];
MOV={mcc,crc3};
for MV=1:2
fopen(t);
fwrite(t,MOV{1,MV});
fclose(t);
end
end
end
```

❖ **kukard.m**

```
function [POS] = kukard(varargin)
%This function reads the actual position of the robot
% This function reads the global variable $POS_ACT
switch nargin
case 0
    [t] = KUKACNCT();
    t.OutputBufferSize = 200;
    t.InputBufferSize = 200;
    fopen(t);
    x=[0, 99, 0, 11, 0, 0, 8, 36, 80, 79, 83, 95, 65, 67, 84];
    fwrite(t,x);
    c=fread(t);
    fclose(t);
    n=c(4,1);
    b=(n+4)-3;
    h=c(8:b);
end
POS = native2unicode(h');
```