Off-line Programming of Industrial Robot

End-term report submitted for
BACHELOR OF TECHNOLOGY PROJECT (PART-1)
IN
MECHANICAL ENGINEERING

Submitted by
B Nalin (2013ME10652)
Yash Kumar Bansal (2013ME10742)

Under the guidance of
Prof. S. K. Saha

Department of Mechanical Engineering
Indian Institute of Technology Delhi
New Delhi, 110016

Abstract

Advances in robotics and its applications are gaining pace each year. An essential aspect of robotics is robot programming. Off-line programming is a robot programming method where the robot program is created independent from the actual robot cell. The robot cell is represented through a graphical 3D model in a simulator. The task that we want to accomplish through this project is to make a kinematic simulator for KUKA KR5 Arc Robot and then generate KRL code for the corresponding off-line program automatically.

The motivation for this project comes from an indigenous software called *RoboAnalyzer*, developed at IITD. It is essentially an educational software to study kinematics and dynamics of some important industrial robots and other custom-made robots. The KUKA KR5 Arc robot, which is also available in IITD's PAR Lab is one of the robots which can be studied using the software. But, the link between the software and the real-world robot was absent. Although the motivation for our project has been derived from *RoboAnalyzer*, we haven't used the software for uses other than for verification purposes, since the source code wasn't available. Hence, our task here is two-fold: (i) to simulate the kinematic motion of the KUKA KR5 Arc robot on a virtual platform and (ii) to develop a code generator such that the real-world robot may emulate this motion.

This report chronicles the methodology adopted and challenges faced during the completion of the above task.

Nomenclature

List of Abbreviations

KR : KUKA Robot
KRC : KUKA Robot Controller
KRL : KUKA Robot Language
6R : 6 revolute joints
2R : 2 revolute joints
IK : Inverse Kinematics
FK : Forward Kinematics
STL : STereo Lithography (file format)
DOF : Degree of Freedom
JV : Joint Variable

List of Symbols

θ_i : Joint space co-ordinate of the i^{th} axis
θ_o : Initial angle
θ_f : Final angle
ω : Angular velocity
T^i_j : Transformation matrix of i^{th} link w.r.t j^{th} link
R^i_j : Rotation matrix of i^{th} link w.r.t j^{th} link

Contents

List of Figures

List of Tables

Chapter 1

Introduction

According to the ISO definition, an industrial robots is an automatically controlled, reprogrammable, multi-purpose manipulator, programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications[1].

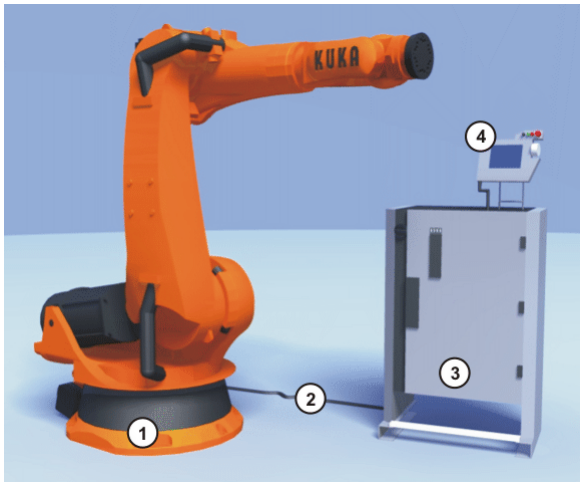

Figure 1.1: Example of an industrial robot
1-Robot, 2-Cable set 3-Robot Controller, 4-Teach Pendant

The KR5 Arc ('KR' is short for KUKA robot; '5' denotes 5kg rated payload; 'Arc' denotes that it is typically used in arc-welding applications) fits the above definition perfectly. The following are the basic specifications of KR5 Arc[2]:

- 6 axis serial chain (6R robot)
- Volume of working envelope: $8.4m^3$

- Weight: approx. 127kg
- Max. total load: 37kg
- KRC embedded in Windows XP OS
- Designated teach pendant
- Programming Language: KRL

KR5 arc can be programmed online using the teach pendant or it can be programmed offline, independent of the actual robot. Offline programming offers several advantages over the online method[3]:

1. Programming can be carried out in parallel with production rather than in series with it, hence minimising the robot down-time.
2. Offline programs are more flexible. Changes can be quickly incorporated and previously developed routines can be re-used.
3. Simulation, an essential part of OLP can be used to pre-check robot movements and therefore improve safety and productivity.

The important steps in OLP, which we have achieved in our project, are:

- **CAD Model Generation**: A 3D CAD model of the robot (and its environment) is required for subsequent simulation. The STL files for the KR5 arc parts can be imported from the RoboAnalyzer package[4].
- **Trajectory Planning**: In this step, a function is generated for each joint space coordinate (w.r.t time), according to the required end-effector coordinates and joint space profiles. In the project, we experimented with trapezoidal and cycloidal profiles.
- **Simulation**: Simulation is used to verify the robot movements and parameters on a virtual platform. In the project, we used MATLAB SimMechanics for kinematic simulation of the robot links.
- **Calibration**: In this step, we take care of the off-sets or deviations between the actual geometry of the robot and that used in the virtual environment.

Chapter 2

Literature Review

2.1 DH Parameters

For a serial chain robot manipulator, we may assume a co-ordinate frame at the tip of each link. The transformation matrix connecting any two frames usually has 6 independent variables. But in the case of single-DOF links, we assume these coordinate frames according to a set convention, and this allows us to represent the transformation matrices between consecutive links using just 4 parameters called the Denavit-Hatenberg (DH) parameters[5].

These parameters are joint offset (b), joint angle (θ), link length (a) and twist angle (α). *RoboAnalyzer*[4] has an excellent feature to visualize these DH parameters. The DH parameters for the KUKA KR5 Robot are tabulated below[6].

Joint No (i)	b_i (m)	θ_i (degrees)	a_i (m)	α_i (degrees)
1	0.4	θ_1 (JV)	0.18	90
2	0.135	θ_2 (JV)	0.6	180
3	0.135	θ_3 (JV)	0.12	-90
4	0.62	θ_4 (JV)	0	90
5	0	θ_5 (JV)	0	90
6	0.115	θ_6 (JV)	0	0

Table 2.1: DH Parameters of KR5 Arc Robot

2.2 Forward Kinematics

In forward kinematics[5, 7], we are given the joint space coordinates, that is all DH parameters (including the joint variables) are known. Our goal is to find the position and orientation of all the links, which can be represented by the transformation matrix T_0^i for link i. Link 0 is assumed to be the fixed link. T_0^i can be recursively calculated using these two equations:

$$T_{i-1}^i = \begin{pmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & a_i C\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & b_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

$$T_0^i = T_0^{i-1} T_{i-1}^i \tag{2.2}$$

2.3 Trajectory Generation

Trajectory refers to the time history of position, velocity, and acceleration of each degree of freedom[8, 9]. If each joint variable is known as a function of time, then all positions and velocities can be calculated using forward kinematics.

Given initial and final joint angles, θ_o and θ_f, we are interested in a function $\theta(t)$, such that $\theta(t_o) = \theta_o$ and $\theta(t_f) = \theta_f$.

There are several such commonly used functions. The KUKA KR5 robot implements the trapezoidal trajectory (also known as 'bang-bang' trajectory) for its PTP motions. Trapezoidal trajectory is so called since the plot of the time derivative of the joint variable is a trapezium, i.e. the velocity increases linearly, stays constant, and then decreases linearly. Though this is a fast way of reaching the final joint value, trapezoidal trajectory is subject to jerks and hence parabolic blend trapezoidal profiles are often used.

RoboAnalyzer gives us the option to choose between cycloid, cosine, quintic and cubic joint space trajectories. In the project, we have worked on the cycloid profile. For a cycloid trajectory, velocity and acceleration curves are continuous (hence eliminating jerk), and start and end at zero. The time-function for cycloid profile is given as[10]:

$$\theta(t) = \theta_o + \left(\frac{t}{t_f} - \frac{1}{2\pi} sin \frac{2\pi t}{t_f} \right) (\theta_f - \theta_o) \tag{2.3}$$

Researchers are experimenting with new trajectories. In 2013, an exponential function was used as a basis for the trajectory profile and implemented on a 6-dof robot KUKA KR5 sixx R650[11].

2.4 Inverse Kinematics

The inverse kinematics problem is of finding the joint variables in terms of the end-effector position and orientation. There could be more than one solutions to this problem. One of these solutions could be selected on the basis of minimum deviation of joint angles[12].
Several techniques have been developed to solve inverse kinematics problem:

- **Numerical approach**: There is a plethora of literature discussing various numerical techniques to solve the IK problem. One of the earliest numerical algorithm to find 16 different solutions for general 6R manipulators in real time was presented in 1992[13].
- **Quaternion approach**: Quaternions, also known as versors, provide a convenient mathematical notation for representing rotations in three dimensions. Dual quaternions can be used to represent transformations in a more compact form[14]. They are numerically more stable and algorithmically more efficient.
- **Geometric approach**: In this project, a geometric approach was used for doing IK of the 6-dof KR5 Arc robot. Using kinematic decoupling, we can consider the position and orientation problems independently if the last three axes intersect at a single point[7]. For a kinematic-decoupled 6R manipulator, at most 8 IK solutions are possible.

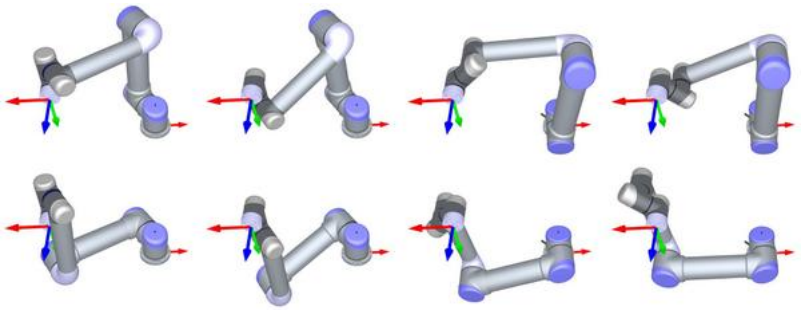

Figure 2.1: 8 Inverse Kinematics solutions for the 6-dof robot

Chapter 3

Objectives

The primary objective of this project is to make a module for off-line programming of an industrial robot, specifically the KUKA KR5 robot. The objective consists of the following tasks:

1. **Forward and Inverse Kinematics of KR5 robot**: A MATLAB function is to be developed for FK (using DH Parameters) and IK (by using the wrist de-coupled approach) for finding all solutions for a given end-effector configuration.

2. **Kinematic Simulation in SimMechanics**: To visualize and verify the rotation of joints and trajectories of the links, a kinematic simulation needs to be done. In order to do this, a CAD model needs to be assembled in SimMechanics using suitable transformation matrices for each link, and actuate the revolute joints according to a suitable joint space trajectory function.

3. **Automatic Code Generation**: Since coding in the robot language is a difficult task, and writing a corresponding offline program after each simulation is a strenuous task, it is desirable to have an application that generates the KUKA program automatically.

4. **Experimenting with Cycloidal Trajectory**: The PTP motions of the KUKA robot inherently follows trapezoidal joint-space trajectory. We have to develop and implement a technique to make it follow cycloidal trajectory.

5. **Integration**: We need to integrate the code generator with the simulation environment and take care of the off-sets and joint limits of the actual robot.

Chapter 4

Progress Summary and Results

4.1 Work Done prior to December 2016

4.1.1 Kinematics of 2R robot in SimMechanics

SimMechanics provides a simulation environment[15, 16] in which links and joints can be modelled similar to a solid modelling software or imported from elsewhere (in STL format) and assembled by specifying suitable transformation matrices. Parameters like link lengths, velocities, angles, etc. can be controlled from an external MATLAB code.

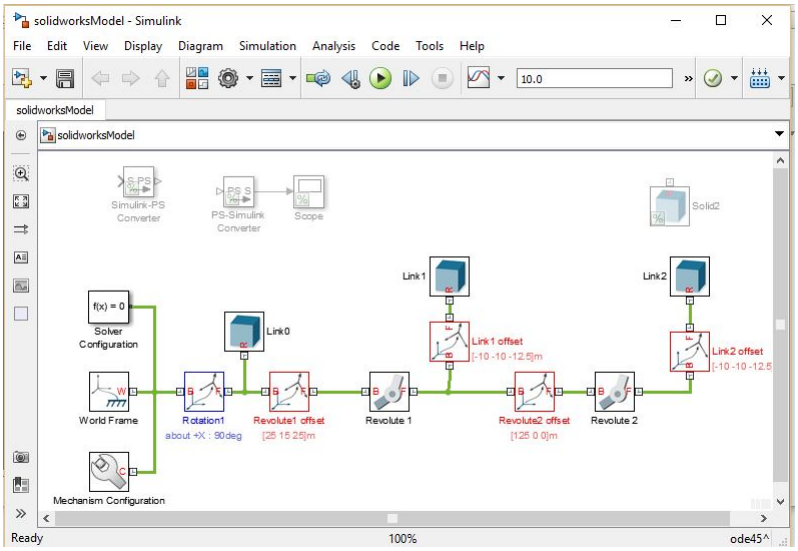

Figure 4.1: Simulink Block diagram for the 2R planar robot

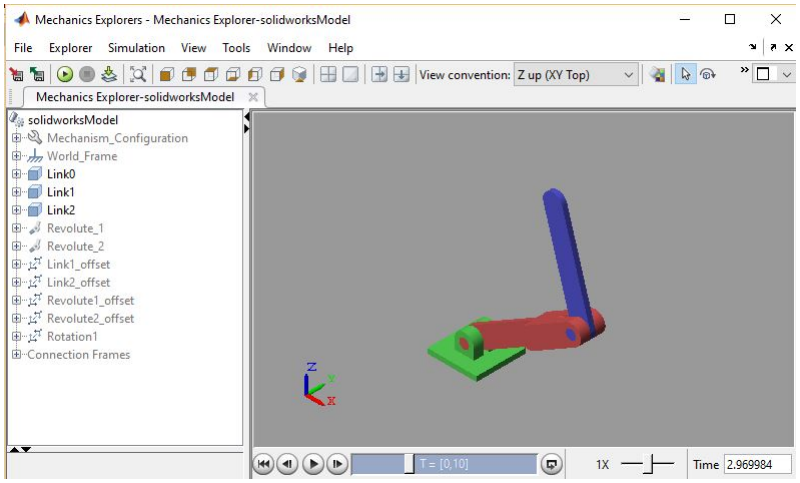

Figure 4.2: SimMechanics Animation for the 2R planar robot

The relations used for forward kinematics of the 2R planar robot were:

$$x = l_1 cos(\theta_1) + l_2 cos(\theta_1 + \theta_2) \tag{4.1}$$

$$y = l_1 sin(\theta_1) + l_2 sin(\theta_1 + \theta_2) \tag{4.2}$$

where l_1 and l_2 are the link lengths and θ_1 and θ_2 are the joint angles, measured from positive x-axis in counter-clockwise direction.
The relations used for inverse kinematics were:

$$\theta_2 = \pm\ cos^{-1}\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right) \tag{4.3}$$

$$\theta_1 = atan2(y, x) - tan^{-1}\left(\frac{l_2 sin\theta_2}{l_1 + l_2 cos\theta_2}\right) \tag{4.4}$$

4.1.2 Path Planning Application in C#

Since the source-code of RoboAnalyzer is written in C# , and it is highly preferable to integrate our code generator with RoboAnalyzer later, we needed practice in developing applications in C# . Keeping this in mind, we took on the task of making a user-friendly application for implementation of bug algorithm.

Bug algorithm is the simplest type of path-planning algorithm, where the virtual robot has only tactile sensors and no prior information about the obstacles in its environment. Such algorithms can be used by robots in unexplored environments, and come in handy if visual sensors are damaged.

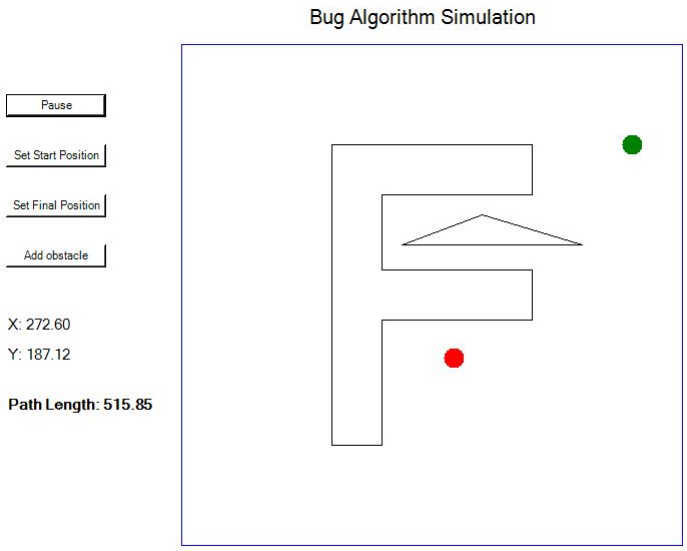

Figure 4.3: Bug Algorithm Application

Our application has the following features:

- Takes input for start and end-point of a point robot, from the user and simulates the motion of the robot from the start to the end. Simulation can be paused or resumed at any time.
- Position of robot at each instant and total path length is displayed.
- User may change or add more obstacles.

4.1.3 KRL Training Session and Practice

The KUKA robot language is not just a trivial programming language. A good understanding of the robot's link parameters and coordinate systems is a pre-requisite for programming the robot. The training sessions were held in three phases:

1. In the first session, we received the basic safety training and hardware handling instructions. Apart from that, we learnt how to set the tool and base coordinates systems. We also made few basic path-traversal programs using online teaching methods.
2. Next, we learnt about various in-built structures and variables which could be used while giving motion commands. We practised off-line programming.

3. Next, we learnt about RSI communication protocols. These were later used to get the data about the joint angles of the robot as a function of time.

Even after formal training, several intricacies were unknown to us. The KUKA manuals[17, 18, 19, 20] had to be read thoroughly and several commands had to be practised before we could proceed to our actual coding job.

As stated in our earlier presentation, the reasons for delay in our B.Tech Project were the delays in training sessions and the loss of robot mastering during practice.

PTP Motion and Common Joint Velocities

The PTP (Point-to-Point) command is a motion command used to move the robot manipulator from one point to the other. The points can be specified either in terms of 6 joint angles,, i.e. PTP AXIS (A1, A2, A3, A4, A5, A6) or end-effector configuration,i.e. PTP POS (X, Y, Z, A, B, C, S, T). Here, A,B,C denote the ZYX euler angles and S,T are used to identify one of the eight possible inverse kinematics solutions.
Unless velocity of each axis is specified separately, the PTP command is executed with common joint velocity, meaning that all axes cover the joint angle from initial to final in the same time.

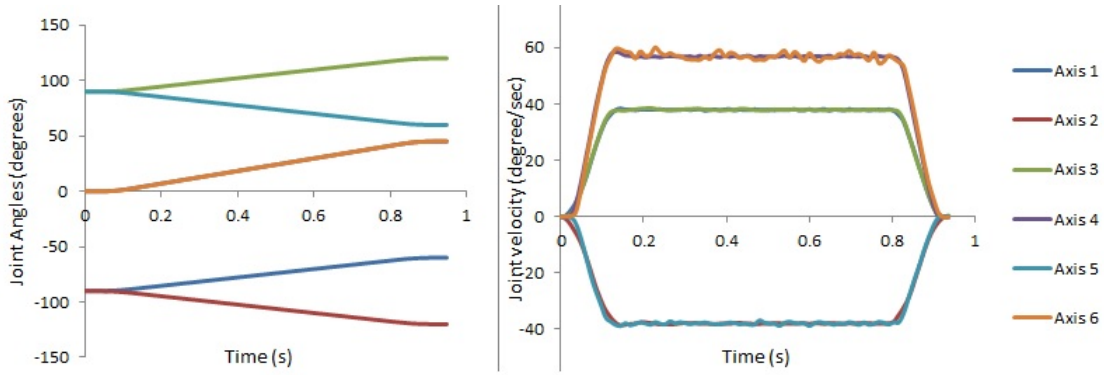

Figure 4.4: Plots for PTP AXIS command from (-90,-90,90,0,90,0) to (-60,-120,120,45,60,45)

Joint Limits and Maximum Speeds

While doing OLP, it must be ensured that joint limits or speed limits aren't exceeded. Otherwise, robot may get damaged and/ or the programs may not run. Also, it is essential to know the maximum speeds since the velocities need to be programmed as a percentage of the maximum speeds.

When each axis is independently moved through 30^o at 20% of its respective speed , the following velocity plot is obtained:

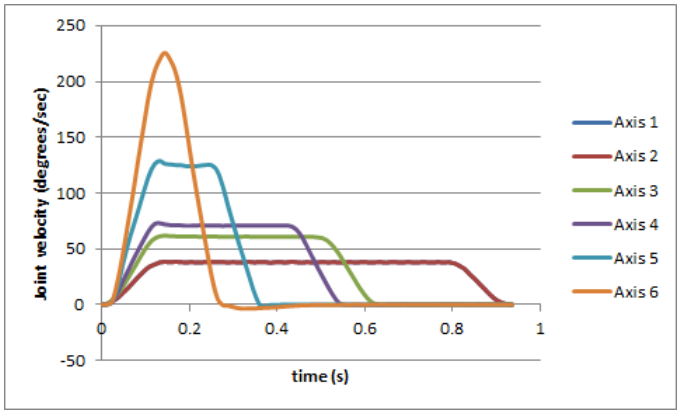

Figure 4.5: Veocity plot when each axis is independently moved through 30^o

Since the current payload (a gripper) is lighter than the rated payload (5kg), the maximum speeds are higher. Just like joint velocities, even joint angles have a maximum and minimum value.
The joint limits[2] and maximum speeds have been summarized in the table below:

Axis	Range of motion (in degrees)	Speed with rated payload (degrees/s)	Speed with current payload (degrees/s)
1	-155 to 155	154	192
2	-180 to 65	154	192
3	-15 to 158	228	310
4	-350 to 350	343	365
5	-130 to 130	384	644
6	-350 to 350	721	1128

Table 4.1: Joint limits and Maximum speeds of KR5 Arc Robot

C_PTP Command
The PTP command follows a trapezoidal joint-space profile, i.e. the speed starts at zero, increases linearly, stays constant for a while, decreases linearly and finally ends at zero. The C_PTP or the continuous PTP command is used to club two or more PTP commands such that velocity doesn't become zero at the intermediate points. This command was later exploited to mimic a cycloidal trajectory through a series of short C_PTP commands.

DH model to KUKA Joint Angle offsets
While practising on the KUKA robot, it was realized that the KUKA angles didn't follow the DH Parameter model. They were actually offset by a fixed angle. Also, a few of the angles were measured in the opposite direction. The following table gives a conversion of joint angles from the DH model (also used in SimMechanics simulation later) to KUKA angles:

Axis	DH model(in degrees)	KUKA (in degrees)
1	θ_1	$-\theta_1$
2	θ_2	$-\theta_2$
3	θ_3	$90 + \theta_3$
4	θ_4	$-\theta_4$
5	θ_5	θ_5
6	θ_6	$-\theta_6$

Table 4.2: DH model to KUKA Joint Angle Offsets

4.2 Work Done during December 2016

4.2.1 Kinematic Simulation of KR5 in SimMechanics

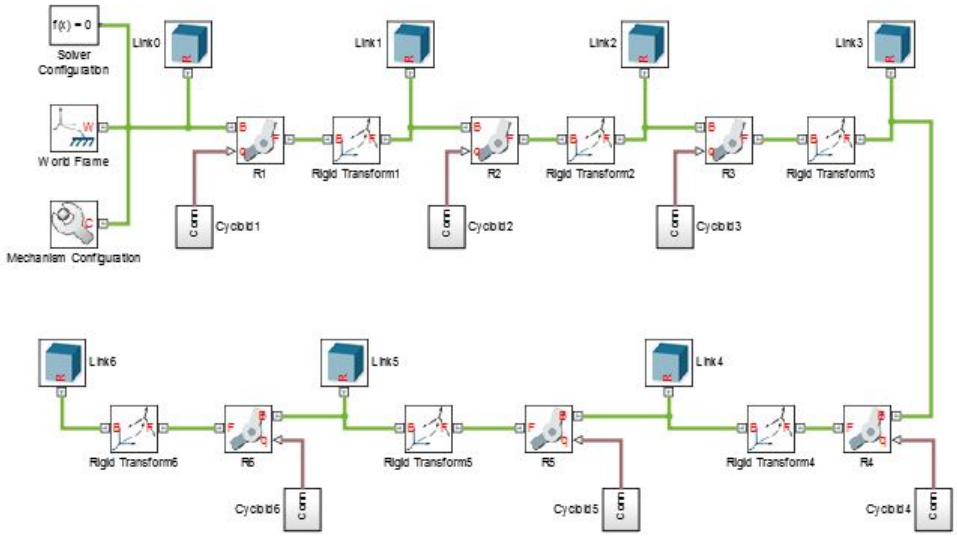

Figure 4.6: Simulink Block Diagram for KR5 links and joints

The 3D CAD model of KUKA KR5 Arc robot is available on KUKA website[21]. But, we found it easier to import the STL files corresponding to each link separately from the RoboAnalyzer Package[4].
Next, these links were joined using corresponding transformation matrices. The transformation matrices for the base case (i.e. all joint angles equal to zero) can be found using substituting the values of DH parameters from Table 2.1 in Eqn. 2.1.

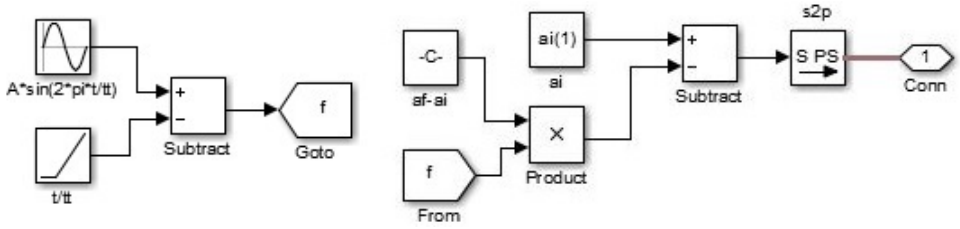

Figure 4.7: Simulink Block Diagram for Cycloidal trajectory input for revolute joint actuators

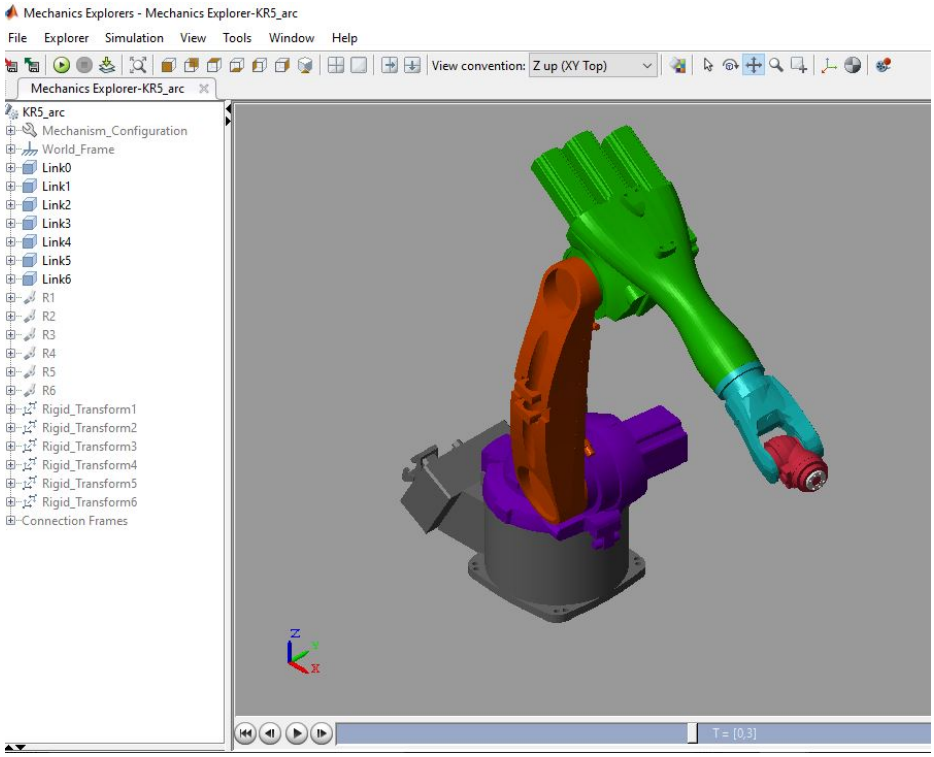

Figure 4.8: SimMechanics Animation for KR5 link movements

Next, a cycloidal function was constructed using basic simulink blocks and fed into the actuators of the revolute joints. The data used to generate the cycloidal function (angle vs time for each time-step) was collected in .xlsx file and verified by comparing it with data from RoboAnalyzer.

Finally, the kinematic simulation was done for KR5 using the cycloidal joint space trajectories to actuate the system of links and joints.

4.2.2 KRL Code Generation and Preliminaries

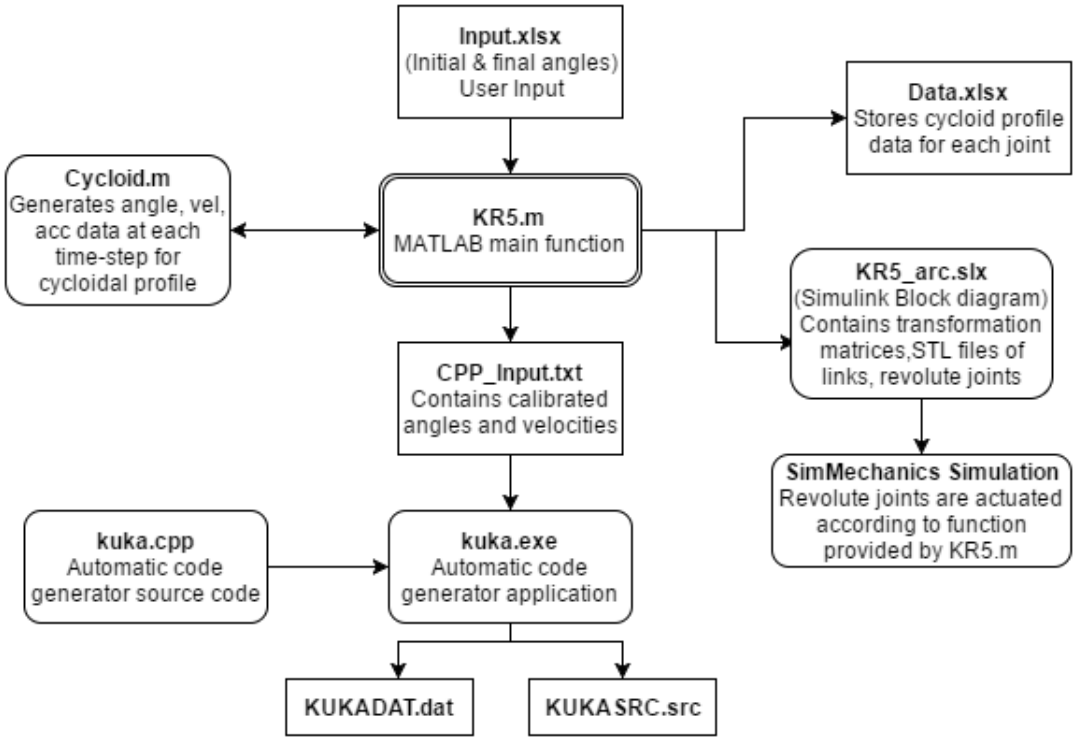

Figure 4.9: Control flow of different program and data files

The parent program is a MATLAB function, which interacts with the code generator as well as the SimMechanics blocks. The main function takes input from user through a .xlsx file and simulates the motion in the SimMechanics virtual environment using another helper MATLAB function which generates cycloidal profiles. After the simulation the main program also checks whether all joint angles are within joint limits and joint velocities are in within the maximum speed limits.

The code generator program is basically a C++ file which writes KRL code into two files: "KUKADAT.dat" and "KUKASRC.src". The .dat file

stores information to be accessed by the .sr file, for e.g. the initial and final joint angles for the PTP motion. The .src file is the KUKA source code file which contains all the motion commands. Apart from that, it also takes care of the RSI protocols so that relevant axis and position data is sent to another computer.

An executable version (.exe file) of the C++ program is accessed from the MATLAB main function. Hence, the .dat and .src files can be generated directly by the MATLAB main function itself.

Some of the KUKA commands that be written by our code generator are:

1. **PTP AXIS**: Takes 6 parameters (joint angles) and follows trapezoidal trajectory to the new configuration.

2. **PTP POS**: Takes 8 parameters (end-effector configuration), KUKA does IK inherently and follows trapezoidal joint-space profile to the new configuration.

3. **C_PTP**: A series of PTP commands can be given with C_PTP such that the robot manipulator doesn't stop at intermediate points.

4. **LIN**: The end-effector follows a linear path in cartesian-space to the next point. KUKA does the necessary IK inherently.

5. **CIRC**: Takes 3 points as input and end-effector follows a circular path in cartesian-space. KUKA does the necessary IK inherently.

6. **HOME**: This command takes the robot to the home position. The code generator can set the home position as well as implement the HOME command.

7. **WAIT**: The robot motion is paused for a given number of seconds.

8. **RSI setup**: A set of commands to setup the RSI communication to export axis and end-effector data to an external RSI monitor.

9. **RSI_ON and RSI_OFF**: Commands to switch 'on' and 'off' the RSI mode so that only the essential data is obtained.

4.2.3 Cycloid Trajectory Generation using CPTP

When we generated cycloid data using MATLAB, we generated an array of tuples (θ, t), i.e tuple of joint angle and time. But in KUKA, we can't possibly feed in the time. But, we can surely feed in the time derivative of joint angles, i.e. the axis velocities.

Note that, if there exists a function f such that $f(\theta, t) = c_1$ for some constant c_1, then there must exist a function g and a constant c_2 such that $g(\theta, \dot{\theta}) = c_2$

So, instead of feeding the tuples (θ, t), we can very well feed the corresponding tuples $(\theta, \dot{\theta})$ to the KUKA robot and theoretically obtain the same trajectory.

For making the robot follow a cycloidal trajectory, the time interval was discretized into small time-steps. for each time step $(\theta, \dot{\theta})$ was generated using the formulae:

$$\theta = \theta_o + \left(\frac{t}{t_f} - \frac{1}{2\pi} sin \frac{2\pi t}{t_f}\right)(\theta_f - \theta_o) \tag{4.5}$$

$$\dot{\theta} = \frac{1}{t_f}\left(1 - cos \frac{2\pi t}{t_f}\right)(\theta_f - \theta_o) \tag{4.6}$$

where t is the time-step, t_f is the total time, and θ_o and θ_f are initial and final angles.

Now, these tuples (joint angles and velocities) were fed into the program and C_PTP (continuous Point-to-Point) motion command was used between each tuple.

The graphs below compare the joint angles and angular velocities obtained by the the theoretical approach (using MATLAB) with those obtained from RSI feedback of the actual robot, when the above algorithm was implemented on it. The plots correspond to motion of Axis 3 from 60^o to 120^o in 3s via cycloidal profile.

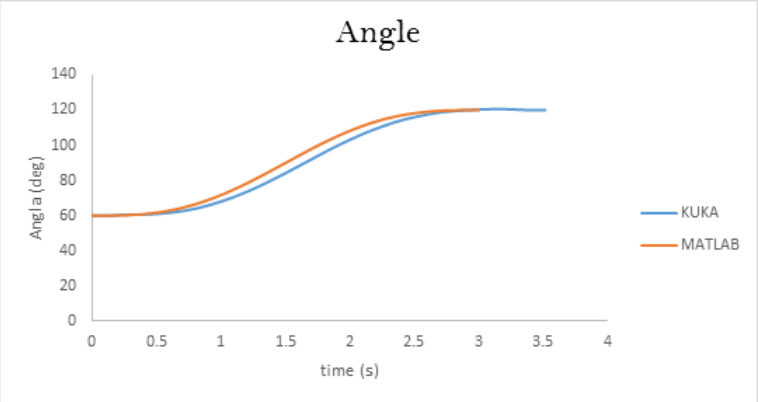

Figure 4.10: Joint Angle plots for KUKA and MATLAB

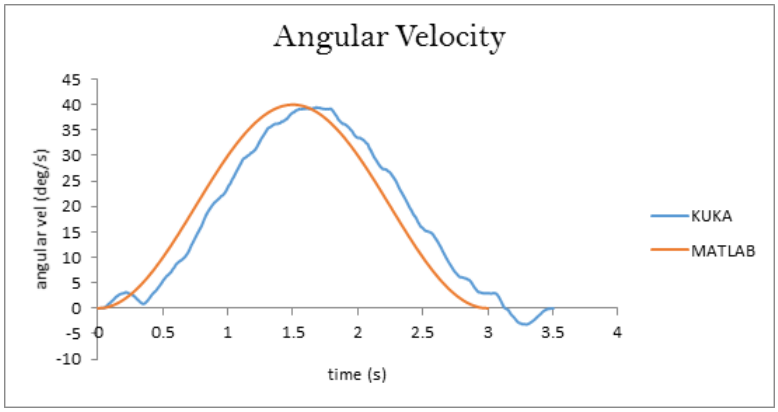

Figure 4.11: Joint Velocity plots for KUKA and MATLAB

Discussions

1. There is an observable rightwards shift in the KUKA graph, when compared to MATLAB, probably due to an initial jerk. Similar jerk could be observed at the end of the motion. If these jerky parts be removed both plots will align in a better fashion.

2. The velocity input needs to be given as percentage of the maximum velocity. This percentage needs to be an integer. Hence, all floating point numbers are rounded to their nearest integer before being accepted as a legitimate axis velocity. This definitely contribute to errors, especially when the velocities are low.

3. Having about 100 time-steps spread over the 3 seconds gave optimal results. Too few time steps wouldn't produce a cycloidal velocity profile as velocity is constant between two consecutive time steps. Too many time steps is even more dangerous as it somehow increases the total time taken, probably due to increase in processing time.

4.2.4 Inverse Kinematics of 6 dof Wrist-partitioned Robot

Problem Statement

The rectangular coordinate system is same as that shown in Fig. 4.8. The end-effector position is given: (x,y,z). Its orientation is specified in terms of the ZYX euler angles: (α,β,γ). The goal is to find all possible combinations of the 6 joint angles such that the given end-effector configuration is reached.

Wrist Centre Position

The rotation matrix of the end-effector can be calculated using the formula:

$$R = R_z(\alpha)R_y(\beta)R_x(\gamma) = \begin{pmatrix} C\alpha & -S\alpha & 0 \\ S\alpha & C\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} C\beta & 0 & S\beta \\ 0 & 1 & 0 \\ -S\beta & 0 & C\beta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & C\gamma & -S\gamma \\ 0 & S\gamma & C\gamma \end{pmatrix} \tag{4.7}$$

The wrist-center (P_W) is the point where the last three axes meet. It can be calculated as follows:

$$P_W = P - b_6 R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.8}$$

where $P = [x\ y\ z]^T$ and b_6 is the join toffset of the 6th link. For KR5 Arc, $b_6 = 0.115m$

Articulated 3R Joint Angles

The first three links of the robot can be treated as an articulated robot. The joints of an articulate robot can be thought of as waist, shoulder and elbow. Let $P_W = [x_w\ y_w\ z_w]^T$. Then, we can easily find the first joint angle:

$$\theta_1 = tan^{-1}\left(\frac{y_w}{x_w}\right) \tag{4.9}$$

$\theta_1 + \pi$ is also a valid solution. Now the second and third links denote a planar 2R robot. Hence, θ_2 and θ_3 can be found using Eqns 4.3 and 4.4. (The offsets need to be taken care of). For each solution of θ_1, there will be two solutions for (θ_2, θ_3).

Wrist Joint Angles

Since we have already determined $(\theta_1, \theta_2, \theta_3)$, we can calculate R_0^3. The rest of the joint angles can be calculated using these equations:

$$R_3^6 = (R_0^3)^{-1} R \tag{4.10}$$

$$R_3^6 = \begin{pmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5s_6 - s_4c_6 & -c_4s_5 \\ s_4c_5c_6 + c_4s_6 & -s_4c_5c_6 + c_4c_6 & -s_4s_5 \\ s_5c_6 & -s_5s_6 & c_5 \end{pmatrix} \tag{4.11}$$

Chapter 5

Conclusions and Future Scope

We have been able to achieve the following in our B.Tech project Part-I:

1. Forward kinematics of KR5 arc robot using its DH parameters.
2. Kinematic simulation of the robot motion in SimMechanics. The simulation follows cycloidal joint space trajectories, given the initial and final joint angles, total time, and the time step.
3. Inverse Kinematics of KR5 arc using geometric (wrist-partitioned) approach. The input is taken in the KUKA format - X,Y,Z,A,B,C.
4. Cycloidal joint space trajectory for the actual robot using a series of PTP commands.
5. We have made an application (.exe file) for the code generator. This can be run from MATLAB. Hence, we have integrated the simulation and code generation part. The generated KUKA codes (.dat and .src) have been tested on the robot.

The following are the future prospects of our project:

1. Our code generator application (coded in C++) can be integrated with the RoboAnalyzer package. This will increase the educational software's usability.
2. We can experiment with other joint space trajectories, for e.g. those based on polynomial functions, cosine functions, etc.
3. The features of the simulation environment can be expanded to include forward and inverse dynamics, similar to RoboAnalyzer.

Appendix A : Generated KRL Codes (Example)

The following code snippets have been generated by C++ code. The C++ takes input from a .txt file. The input file is the one shown below:
Input.txt

```
HOME AXIS -90.000000 -90.000000 90.000000 0.000000
90.000000 0.000000
TRAP AXIS -90.000000 -90.000000 60.000000 0.000000
90.000000 0.000000 25.000000
CYLCOID AXIS -90.000000 -90.000000 60.000000 0.000000
90.000000 0.000000 25.000000 3.000000 100
```

- The line starting with 'HOME AXIS' (followed by 6 floating point numbers) is used to generate commands to set the home position of the robot and bring the robot to the home position.
- The line starting with 'TRAP AXIS' (followed by 7 floating point numbers) is used to take the robot to a new position through trapezoidal joint space profile. The first six floating point numbers specify the final joint angles and the seventh one is the maximum angular velocity as a percentage of the maximum velocity that could be attained by that axis.
- The line starting with 'CYCLOID AXIS' (followed by 7 floating point numbers) is used to take the robot to a new position through cycloidal joint space profile. The first six floating point numbers specify the final joint angles, the seventh one specifies the total time to be taken, and the eighth one specifies the number of steps.

Two files, a .dat file (contains object declarations) and a .src file (contains functions for motion commands) are generated.

KUKADAT.dat

```
&ACCESS RVP
&REL 8
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
&PARAM EDITMASK = *
DEFDAT KUKADAT

;FOLD EXTERNAL DECLARATIONS;%{PE}%MKUKATPBASIS,%CEXT
;FOLD BASISTECH EXT;%{PE}%MKUKATPBASIS,%CEXT,%VEXT,%P
EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )
DECL INT SUCCESS
;ENDFOLD (BASISTECH EXT)
;FOLD USER EXT;%{E}%MKUKATPUSER,%CEXT,%VEXT,%P
;Make your modifications here

;ENDFOLD (USER EXT)
;ENDFOLD (EXTERNAL DECLARATIONS)
DECL BASIS_SUGG_T LAST_BASIS={
POINT1[] "P1                                    ",
POINT2[] "P1                                    ",
CP_PARAMS[] "CPDAT0                                ",
PTP_PARAMS[] "PDAT1                                ",
CONT[] "                                  ",
CP_VEL[] "2.0                                   ",
PTP_VEL[] "100                                  ",
SYNC_PARAMS[] "SYNCDAT                             ",
SPL_NAME[] "S0                                    "}
DECL AXIS XHOME = {A1 -90.000,A2 -90.000,A3 90.000,
A4 0.000,A5 90.000,A6 0.000}
DECL AXIS XP1 = {A1 -90.000,A2 -90.000,A3 60.000,
A4 0.000,A5 90.000,A6 0.000}
DECL AXIS XP2 = {A1 -90.000,A2 -90.000,A3 90.000,
A4 0.000,A5 90.000,A6 0.000}
ENDDAT
```

KUKASRC.src

```
&ACCESS RVP
&REL 8
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
```

```
&PARAM EDITMASK = *
DEF KUKASRC( )

DECL INT I,J,N1
DECL REAL TMP_RL1,TMP_RL2, N,T
DECL AXIS CYC_PT,CYC_DEL

DECL RSIERR RSI_RET
DECL INT hMonitor,hAxis,hPos,PORT,CYCLE
DECL CHAR IP[16]

;FOLD INI
  ;FOLD BASISTECH INI
    GLOBAL INTERRUPT DECL 3 WHEN
$STOPMESS==TRUE DO IR_STOPM ( )
    INTERRUPT ON 3
    BAS (#INITMOV,0 )
  ;ENDFOLD (BASISTECH INI)
  ;FOLD USER INI
    ;Make your modifications here

  ;ENDFOLD (USER INI)
;ENDFOLD (INI)

;FOLD RSI SETUP % IP 192.168.1.60  % PORT 6000
  IP[] = "192.168.1.60"
  PORT = 6000
  CYCLE = 1

  RSI_RET = ST_MONITOR(hMonitor,0,IP[],PORT,CYCLE)
  RSI_RET = ST_ACTAXIS(hAxis,0)
  RSI_RET = ST_ACTPOS(hPos,0)

  RSI_RET = ST_SETPARAM(hMonitor,1,1.0)
  RSI_RET = ST_NEWLINK(hPos,1,hMonitor,2)
  RSI_RET = ST_NEWLINK(hPos,2,hMonitor,3)
  RSI_RET = ST_NEWLINK(hPos,3,hMonitor,4)
  RSI_RET = ST_NEWLINK(hPos,4,hMonitor,5)
  RSI_RET = ST_NEWLINK(hPos,5,hMonitor,6)
  RSI_RET = ST_NEWLINK(hPos,6,hMonitor,7)
  RSI_RET = ST_NEWLINK(hAxis,1,hMonitor,8)
```

```
   RSI_RET = ST_NEWLINK(hAxis,2,hMonitor,9)
   RSI_RET = ST_NEWLINK(hAxis,3,hMonitor,10)
   RSI_RET = ST_NEWLINK(hAxis,4,hMonitor,11)
   RSI_RET = ST_NEWLINK(hAxis,5,hMonitor,12)
   RSI_RET = ST_NEWLINK(hAxis,6,hMonitor,13)
;ENDFOLD (RSI SETUP)

;FOLD PTP HOME Vel=25 % DEFAULT;
   $BWDSTART=FALSE
   PDAT_ACT=PDEFAULT
   FDAT_ACT=FHOME
   BAS(#PTP_PARAMS,25.000)
   PTP XHOME
;ENDFOLD
;FOLD PTP P1 Vel=25 % DEFAULT;
   $BWDSTART=FALSE
   PDAT_ACT=PDEFAULT
   FDAT_ACT=FHOME
   BAS(#PTP_PARAMS,25.000)
   PTP XP1
;ENDFOLD
RSI_RET = ST_ON()

N=100
T=3
N1 = N

;FOLD PTP P2 % PDAT1 % CYCLOID
   CYC_DEL.A1 = XP2.A1-XP1.A1
   CYC_DEL.A2 = XP2.A2-XP1.A2
   CYC_DEL.A3 = XP2.A3-XP1.A3
   CYC_DEL.A4 = XP2.A4-XP1.A4
   CYC_DEL.A5 = XP2.A5-XP1.A5
   CYC_DEL.A6 = XP2.A6-XP1.A6
   FOR I = 1 TO N1 STEP 2
      TMP_RL1 = (I+1)/(N-1)-(SIN(I*360.0/(N-1)))/6.2831853
      TMP_RL2 = (1-COS(I*360.0/(N-1)))/T
      CYC_PT.A1 = XP1.A1 + CYC_DEL.A1*TMP_RL1
      J =(CYC_DEL.A1*TMP_RL2)/1.920417
      IF J==0 THEN
           J=1
```

```
      ENDIF
      $VEL_AXIS[1]=J
      CYC_PT.A2 = XP1.A2 + CYC_DEL.A2*TMP_RL1
      J =(CYC_DEL.A2*TMP_RL2)/1.920417
      IF J==0 THEN
           J=1
      ENDIF
      $VEL_AXIS[2]=J
      CYC_PT.A3 = XP1.A3 + CYC_DEL.A3*TMP_RL1
      J =(CYC_DEL.A3*TMP_RL2)/3.095942
      IF J==0 THEN
           J=1
      ENDIF
      $VEL_AXIS[3]=J
      CYC_PT.A4 = XP1.A4 + CYC_DEL.A4*TMP_RL1
      J =(CYC_DEL.A4*TMP_RL2)/3.646813
      IF J==0 THEN
           J=1
      ENDIF
      $VEL_AXIS[4]=J
      CYC_PT.A5 = XP1.A5 + CYC_DEL.A5*TMP_RL1
      J =(CYC_DEL.A5*TMP_RL2)/6.438575
      IF J==0 THEN
           J=1
      ENDIF
      $VEL_AXIS[5]=J
      CYC_PT.A6 = XP1.A6 + CYC_DEL.A6*TMP_RL1
      J =(CYC_DEL.A6*TMP_RL2)/11.28703
      IF J==0 THEN
           J=1
      ENDIF
      $VEL_AXIS[6]=J
      PTP CYC_PT C_PTP
   ENDFOR
   PTP XP2
;ENDFOLD (PTP CYCLOID)

RSI_RET = ST_OFF()

END
```

References

[1] ISO 8373. *Robots and robotic devices-Vocabulary*, 2012. Webpage: `https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en`.

[2] KUKA Robot Group. *KR 5 arc Specification*, 2016. Download from: `https://www.kuka.com/en-de/services/downloads`.

[3] Z. Pan, J. Polden, N. Larkin, and S. van Duin J. Norrish. Recent progress on programming methods for industrial robots. *Robotics and Computer Integrated Manufacturing*, 2(28):87–94, 2012.

[4] Roboanalyzer software package. Download from: `http://www.roboanalyzer.com/downloads.html`.

[5] S. K. Saha. *Introduction to Roboticsl*, Chap. 5–6, pages 116–208. 2^{nd} edition.

[6] A. D. Udai, C. G. Rajeevlochana, and S. K. Saha. Dynamic Simulation of a KUKA KR 5 Industrial Robot using MATLAB SimMechanics. In *National Conference on Machines and Mechanisms (NaCoMM2011)*, Chennai, India, November 2012.

[7] M. W. Spong, S.Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*, Chap. 1–3, pages 1–302. 1^{st} edition.

[8] J. J. Craig. *Introduction to Robotics Mechanics and Control*, Chap. 7, pages 201–229. 3^{rd} edition.

[9] D. L. Pieper. The Kinematics of Manipulators Under Computer Control. Research project report, Stanford University, Oct 1968.

[10] A. A. N. Aljawi, H. Diken, and S. A. Alshahrani. Selection of a trajectory function for minimum energy requirements of a spherical robot. *JKAU: Eng. Sci*, 15(2):99–110, 2004.

[11] Z. Rymansaib, P. Iravani, and M.N.Sahinkaya. Exponential Trajectory Generation for Point to Point Motions. In *International Conference on Advanced Intelligent Mechatronics (AIM)*, Wollongong, Australia, July 2013.

[12] A. A. Hayat, R. Sadanad, and S. K. Saha. Robot Manipulation through Inverse Kinematics. In *Advances in Robotics, Int. Conf. of Robotics Society of India, (AiR 2015)*, Goa, India, July 2014.

[13] D. Manocha and J. F. Canny. Real Time Inverse Kinematics for General 6R Manipulators. In *International Conference on Robotics and Automation*, Nice, France, May 1992.

[14] S. Kucuk and Z. Bingul. *Industrial Robotics: Theory, Modelling and Control*, Chap. Robot Kinematics, pages 117–148. Pro Literatur Verlag, Germany, 2006. Available from: `http://www.intechopen.com/books/industrial_robotics_theory_modelling_and_control/robot_kinematics__forwar`.

[15] Matlab simmechanics official website. `https://in.mathworks.com/products/simmechanics.html`.

[16] Gordon Parker. Simmechanics series. Watch on YouTube: `https://www.youtube.com/watch?v=iVrjuHYtr_o`.

[17] KUKA Robot Group. *Reference Manual*, 4.1 edition, 2003. Hard-copy available in PAR lab, IITD.

[18] KUKA Robot Group. *Basic Programming Manual*, 2003. Hard-copy available in PAR lab, IITD.

[19] KUKA Robot Group. *Advanced Programming Manual*, 2003. Hard-copy available in PAR lab, IITD.

[20] KUKA Robot Group. *Expert Programming Manual*, 5.2 edition, 2003. Hard-copy available in PAR lab, IITD.

[21] Kuka download center. Website:`https://www.kuka.com/en-de/services/downloads`.